



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Diseño de un sistema de
identificación de personas**



Presentado por Víctor de Castro Hurtado
en Universidad de Burgos — 22 de junio
de 2018

Tutor: César Represa Pérez



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. César Represa Pérez, profesor del departamento de Ingeniería Electromecánica, área de Tecnología Electrónica.

Expone:

Que el alumno D. Víctor de Castro Hurtado, con DNI 71289378G, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Diseño de un sistema de identificación de personas.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 22 de junio de 2018

Vº. Bº. del Tutor:

D. César Represa Pérez

Resumen

Las tecnologías avanzan continuamente, al igual que hacen las amenazas que acompañan a dichas tecnologías. Por ello, y para mantener una sociedad estable, la seguridad debe ir ligada a estos avances.

Con este proyecto se pretende abordar la temática de la seguridad en lugares públicos (como pueden ser aeropuertos o centros comerciales) desde el punto de vista del reconocimiento facial.

Para ello, se ha diseñado un prototipo de un sistema de identificación de personas a través de un programa de reconocimiento facial, desarrollado en *Python* y utilizando técnicas de **Machine-Learning**.

En dicho sistema se parte de una base de datos que contiene imágenes de personas en una lista de busca y captura. De este modo, a partir de una imagen capturada en el lugar de interés, el sistema reconocerá el rostro y lo identificará si se encuentra en la base de datos.

Descriptores

Python, inteligencia artificial (visión artificial), reconocimiento facial, aprendizaje automático, entrenamiento de redes

Abstract

Technology is continuously advancing, so do the threads attached with this technology. For that, and to sustain a stable society, security must suit those advances.

With this project, we intend to approach the security topic in public places (such as airports or shopping centers) from a facial recognition point of view.

For that, we have designed a prototype of a person identifier system through a facial recognition program, developed in *Python* and using *Machine-Learning* technologies.

In that system, we start with a database with images of people in a search and capture list. This way, from an image captured in-place, the system will recognize the face and identify if it exists in the database.

Keywords

Python, artifical intelligence (computer vision), facial recognition, machine learning, network training

Índice general

Índice general	7
Índice de figuras	10
Introducción	1
1.1. El problema	1
1.2. La solución	2
1.3. Funcionamiento general	3
Aplicación real a gran escala	3
Aplicación real con los recursos disponibles	6
1.4. Estructura de la memoria	7
1.5. Material complementario	8
1.6. Estructura del repositorio	9
Objetivos del proyecto	15
2.1. Objetivos generales	15
2.2. Objetivos técnicos	16
2.3. Objetivos personales	17
Conceptos teóricos	19
3.1. Inteligencia Artificial	19
Aprendizaje Automático	19
Analítica predictiva	19
Aprendizaje profundo	20
Visión Artificial	20
3.2. Redes neuronales	21
Red neuronal artificial	21

Red neuronal convolucional	22
Tipos de Entrenamiento	23
3.3. Reconocimiento Facial	24
Eigenvectores	25
Algoritmos de Reconocimiento Facial	25
Eigenfaces	25
Fisherfaces	26
Local binary patterns	26
LBP-Histograms (LBPH)	27
Técnicas y herramientas	29
4.1. Técnicas metodológicas	29
4.2. Patrones	31
4.3. Herramientas	33
Herramientas de desarrollo	33
Herramientas de planificación y diseño	34
Aspectos relevantes del desarrollo del proyecto	35
5.1. Explicación general	35
5.2. Requisitos técnicos	36
5.3. Fases de desarrollo	36
Definir el proyecto	36
Identificar información, recursos y requisitos	36
Instalación y configuración	37
Captura y manipulación de imágenes	37
Reconocimiento facial I: Localizar rostros	39
Reconocimiento facial II: Identificar rostros	41
¿Por qué OpenCV?	44
¿Por qué este sistema de entrenamiento?	44
Base de datos	46
Interfaz	47
Revisión y control continuo	55
Release	55
Trabajos relacionados	57
Conclusiones y Líneas de trabajo futuras	59
7.1. Posibilidades de mejora del proyecto	59
Base de datos	59
Entrenamiento	60
Reconocimiento a gran escala	61

<i>ÍNDICE GENERAL</i>	9
<i>¿Es viable?</i>	62
7.2. Conclusiones	63
Bibliografía	67

Índice de figuras

1.1.	Esquema de como se organizaría el sistema en una aplicación a gran escala.	4
1.2.	Estructura general del contenido del CD y del repositorio de GitHub.	8
2.3.	Objetivos generales del proyecto.	16
3.4.	[6] Google Cloud Vision.	20
3.5.	Red neuronal con las diferentes capas implicadas.	21
3.6.	Red neuronal convolucional con las diferentes capas implicadas y cómo se relacionan los nodos de cada una entre si.	23
3.7.	Proceso de agrupación e identificación de características en una imagen según el algoritmo LBP.	27
4.8.	Ejemplo de patrón Adaptador utilizado al comienzo del desarrollo del proyecto.	31
4.9.	Ejemplo de patrón Abstract Factory utilizado al comienzo del desarrollo del proyecto.	32
4.10.	Ejemplo de patrón Singleton utilizado en la clase GUI.	32
5.11.	Imagen del rostro escalada para la interfaz.	38
5.12.	Imagen original convertida a escala de grises.	38
5.13.	Método que se encarga de inicializar la configuración de la cámara ('cv2' corresponde a la librería de OpenCV).	39
5.14.	Entrenando red y generando fichero con características.	42
5.15.	Mensaje avisándonos de que en esa imagen no hay rostros.	43
5.16.	Tiempo de entrenamiento con 2 imágenes.	45
5.17.	Tiempo de entrenamiento con 15 imágenes.	45
5.18.	Tiempo de entrenamiento con 70 imágenes.	45
5.19.	Tiempo de entrenamiento con 200 imágenes.	45

5.20. Muestreo de los resultados de varios entrenamientos de la red sucesivos en función del número de imágenes.	46
5.21. Interfaz inicial.	47
5.22. GetInstance de la interfaz GUI.	48
5.23. Creamos la ventana principal con tamaño fijo.	48
5.24. Añadimos las imágenes a la ventana.	49
5.25. Añadimos la barra de progreso a la ventana.	49
5.26. Interfaz con barra de progreso.	50
5.27. Botón para crear el pop-up.	50
5.28. Pop-up con la información de la persona.	51
5.29. Información de la persona que se va a mostrar en el pop-up.	51
5.30. Pop-up con información.	51
5.31. Interfaz final.	52
5.32. Programa preguntándonos si queremos una imagen desde la cámara, desde un fichero o en tiempo real.	53
5.33. Programa preguntándonos si queremos introducir una nueva imagen en la base de datos.	53
5.34. Programa preguntándonos si queremos entrenar la red de nuevo.	53
5.35. Mensaje de aviso de que no hay rostros en la imagen.	53
5.36. Interfaz final con color añadido y otras mejoras en elementos visuales (bordes y profundidad en botones y barra de progreso).	54
7.37. Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: mala iluminación).	64
7.38. Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: muy mala iluminación).	64
7.39. Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: solo visibles los ojos).	65
7.40. Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: foto de casi perfil).	65
7.41. Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: sin gafas).	66

Introducción

1.1. El problema

Las tecnologías avanzan continuamente, al igual que hacen las amenazas que acompañan a dichas tecnologías. Por ello, y para mantener una sociedad estable, la seguridad debe ir ligada a estos avances.

Últimamente escuchamos hablar mucho sobre ataques terroristas, tiroteos, asesinatos, violaciones, etc., y ciertas medidas que se están tomando en algunos países (como EEUU) para intentar impedir dichos ataques antes de que sucedan.

Pero estas medidas llevan un coste asociado: la pérdida de privacidad. Es un coste que mucha gente no está dispuesta a pagar, y es un tema muy interesante que podría dar para una reflexión más larga, pero no es el objetivo del proyecto, de modo que se tratará brevemente en el apartado *7_Conclusiones_Líneas_de_trabajo_futuras 5.3* y en el anexo *A_Plan_proyecto*.

Otro hecho es que, a día de hoy, todo está conectado, desde nuestros teléfonos móviles, hasta toda nuestra información en las redes sociales, así como los nuevos vehículos de conducción automática, etc. Por ello, y como indicábamos antes, se pierde privacidad, pero también se gana en seguridad: al estar todo conectado, es más fácil controlarlo.

1.2. La solución

Con este proyecto se pretende abordar la temática de la seguridad en lugares públicos (como pueden ser aeropuertos o centros comerciales) desde el punto de vista del reconocimiento facial. Para ello, se parte de una base de datos que contiene imágenes de personas en una lista de **busca y captura**. A partir de estas imágenes se pretende identificar a las personas de la base de datos que se encuentren en dichos lugares públicos.

Como se mencionaba antes, al estar todo conectado, encontrar a estas personas entre la multitud o incluso entre las redes sociales es más sencillo que nunca, y en este proyecto se pretende diseñar un sistema de **identificación de personas** a través de un programa de reconocimiento facial, desarrollado en *Python* y utilizando técnicas de **Machine Learning**.

Y, ¿por qué machine learning? 3.1

Bueno, siempre ha estado ahí, aunque ha sido cuando lo han empezado a usar las grandes compañías como *Google* o *Apple* cuando realmente se ha visto su verdadero potencial.



Consiste, en resumidas cuentas, en que ya no tenemos que decirle al ordenador lo que tiene que hacer en cada ocasión con instrucciones, sino que se le indica un modelo de comportamiento que debería seguir (como se enseña a los niños pequeños), y es él el encargado de actuar en cada caso como entienda que es mejor. Es una técnica muy útil y versátil, y puede ser utilizado desde reconocimiento de objetos en imágenes (como es nuestro caso), a aprender a formar frases completas y con sentido, como es el caso de la nueva funcionalidad de *Google*: **Google Duplex** [75].

Una de las muchas aplicaciones que tiene el *machine learning* es el *facial recognition* que vamos a usar en nuestro caso, el cual tiene a su vez muchas aplicaciones, cada día más utilizadas, destacando las relacionadas con la **seguridad**, como la utilizada en los aeropuertos para llevar un control de los pasajeros, o en las cámaras públicas de las ciudades, para obtener información sobre atracos, ataques terroristas, etc.

También destacan (aunque no nos interesan en la realización de este proyecto) los usos en reconocimiento y seguimiento de clientes por parte de empresas y tiendas, así como el reciente *desbloqueo facial* de los teléfonos inteligentes como los de *Apple*.

En el siguiente enlace [13] se pueden ver distintas aplicaciones desarrolladas por *tensorflow* (herramienta para desarrollar aplicaciones de machine learning) [70], no sólo para el reconocimiento facial, sino para el reconocimiento de imágenes en general.

1.3. Funcionamiento general

Como hemos mencionado, el proyecto pretende servir como forma de identificación de personas que, teniendo una imagen de la persona en cuestión previamente obtenida en la base de datos, y su información básica (nombre, edad, lugar de nacimiento y profesión), se puede identificar a dicha persona cuando se pone delante de la cámara utilizando técnicas de machine-learning y computer vision.

Aplicación real a gran escala

Para la identificación de estas personas se tienen sus imágenes almacenadas en la base de datos, todas ellas con las mismas características (tomadas de un registro oficial, como pueden ser imágenes de DNI, registros de su paso por comisaría en otras detenciones, organizaciones internacionales, etc).

Teniendo las personas que queremos encontrar ya introducidas en nuestra base de datos, se procedería a ejecutar un **rastreo**, ejecutando el programa en un **servidor** con capacidad suficiente como para analizar cada rostro de todas las imágenes en tiempo real, sobre las cámaras de ayuntamientos, bancos, semáforos, comisarías, aeropuertos, estaciones de autobús o tren o complejos deportivos de la ciudad. El esquema que se podría seguir en este caso se pueve observar en la imagen 1.1.

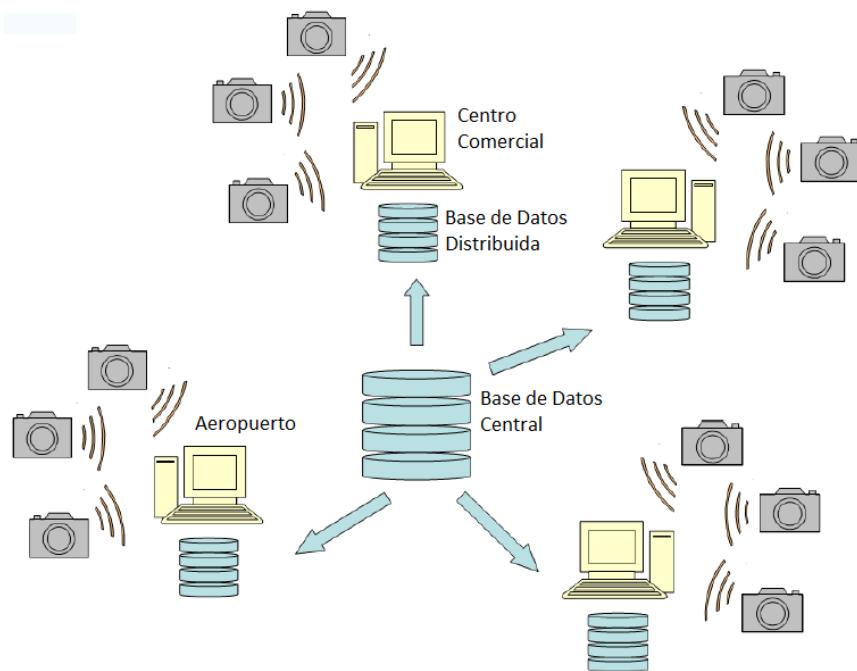


Figura 1.1: Esquema de como se organizaría el sistema en una aplicación a gran escala.

Por supuesto, este sistema de reconocimiento a **gran escala** y en **tiempo real** (recordemos que está recopilando información de todas las cámaras disponibles en la ciudad, a la vez que está obteniendo las características de los rostros que encuentra en cada una de ellas, comparando dichas características con la base de datos), tiene un **alto coste computacional** y de recursos, de manera que sería necesario ejecutar el programa principal en un servidor con unos requisitos técnicos muy altos, además de una infraestructura adecuada para poder transmitir toda esa información sin retrasos (lo que implica, de nuevo, unos altos requisitos técnicos).

(Se entrará más en detalle en este aspecto en la sección *7_Conclusiones_Líneas_de_Trabajo_futuras: Reconocimiento a Gran Escala 7.1*).

Si alguna de estas personas aparece en alguna de las cámaras, el sistema le reconoce y muestra una interfaz con la imagen capturada desde la cámara y la imagen de la base de datos con la que la relaciona, mostrando además la cámara con la que ha realizado el reconocimiento, para que sea más sencillo localizar a esta persona en la zona de la ciudad que corresponda.

Estamos hablando de una aplicación que debería ser llevada a cabo a **nivel local** (en un municipio o ciudad no demasiado grande), porque, como hemos comentado, cuanto mayor sea la zona a cubrir (y por lo tanto el número de cámaras utilizadas), mayores serán los requisitos técnicos necesarios.

Otra posible aplicación que se podría obtener de este proyecto sería el buscar personas desaparecidas. Utilizando el mismo método, lo único que habría que variar sería la base de datos utilizada: de criminales a personas desaparecidas.

Lo mismo que se podría adaptar a otros ámbitos simplemente teniendo una base de datos diferente para cada uno de los casos. En ese sentido, el **coste de adaptación** a la variedad de casos de usos sería **mínimo**, siendo el coste inicial el mayor de todos: un servidor principal que se encargue de ejecutar el programa, recopilando los datos obtenidos de todas las cámaras.

Aplicación real con los recursos disponibles

En la realización de este proyecto no se ha llegado al nivel de desarrollo descrito en el apartado anterior, primero porque no se disponía de los medios necesarios para cumplir con los requisitos del hardware (el servidor principal con requisitos técnicos elevados que comentábamos anteriormente), y segundo porque no se disponía de una base de datos suficientemente amplia como para mostrar esta funcionalidad completa que se mencionaba en párrafos anteriores.

Por lo tanto, se ha optado por describir sus posibles y futuras implicaciones en esta memoria, mientras se implementaba una primera versión del proyecto, acorde al presupuesto, tiempo y recursos de los que se disponía.

En esta primera versión se tiene una base de datos con unas cuantas imágenes de personajes famosos (representaría en la aplicación final a los criminales), entre los que se incluye una foto nuestra para poder obtener un resultado positivo en el análisis en tiempo real.

Con esta base de datos improvisada, se entrena la red, identificando las características de las personas que tenemos, almacenando dichas características para evitar entrenar la red cada vez que se trate de identificar un rostro (ahorriendo tiempo y recursos).

Tras esto, ejecutamos la parte del programa encargada del reconocimiento propiamente dicho, por lo que nos colocamos delante de la cámara y tomamos una captura (esta parte también se ha visto simplificada con respecto a la aplicación final en la que se tomarían dichas capturas con varias cámaras, distribuidas a lo largo de la ciudad).

Por último, el programa reconoce automáticamente las características del rostro de la imagen capturada, compara dichas características con las que teníamos guardadas de los famosos, y nos da una estimación de la persona a la que más nos parezcamos.

En una aproximación más real y comercial, bastaría con adaptar los parámetros de comparación y muestreo de los resultados para evitar mostrar los resultados que no superen un umbral, de tal manera que, si no coincide en un alto porcentaje con la persona que estamos buscando, no muestre nada, evitando de esta manera falsos positivos.

1.4. Estructura de la memoria

La presente memoria se ha estructurado en los siguientes apartados:

- **Introducción:** descripción del problema a resolver y la solución propuesta. Estructura de la memoria y del proyecto, así como los materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que persigue el proyecto, tanto técnicos como personales.
- **Conceptos teóricos:** explicación de los conceptos teóricos clave para la comprensión del proyecto.
- **Técnicas y herramientas:** técnicas metodológicas y herramientas utilizadas para el desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** explicación del desarrollo del proyecto, destacando los aspectos más relevantes.
- **Trabajos relacionados:** proyectos relacionados.
- **Conclusiones y líneas de trabajo futuras:** conclusiones obtenidas y posibilidades de mejora del proyecto.

Junto a la memoria se proporcionan los siguientes anexos:

- **Plan del proyecto software:** planificación temporal y estudio de viabilidad del proyecto.
- **Especificación de requisitos del software:** fase de análisis del proyecto, objetivos generales, requisitos funcionales y no funcionales.
- **Especificación de diseño:** fase de diseño, tanto del software como de los datos.
- **Manual del programador:** aspectos relevantes relacionados con el código fuente, localizado en la carpeta *src*.
- **Manual de usuario:** guía de usuario con instrucción y aspectos destacables que puedan facilitar el correcto manejo de la aplicación.

1.5. Material complementario

- CD/DVD

El contenido del CD es la carpeta completa del proyecto, conteniendo la siguientes estructura 1.6: las carpetas *src*, *sources*, *Other* y los ficheros *run.bat* y *readme.md*. El CD tiene además la memoria con sus anexos, y un vídeo explicativo del funcionamiento básico del programa. Se puede observar dicho contenido en la siguiente imagen 1.2

- Repositorio de *Github*

<https://github.com/victorcas04/TFG-FacialRecognition>

Este repositorio sigue una estructura similar a la presentada en la figura 1.2.

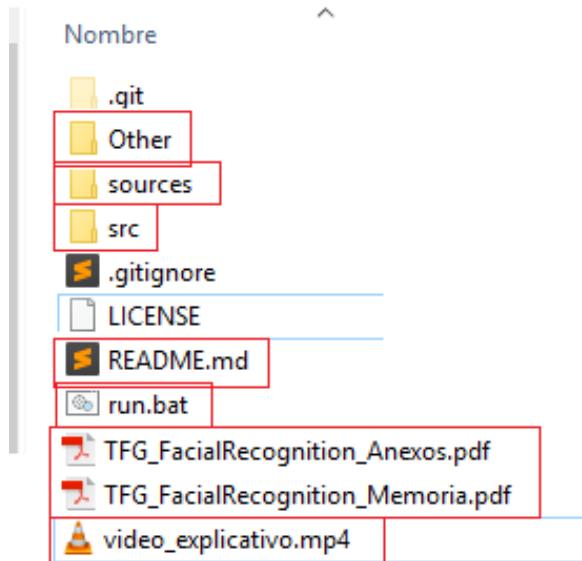


Figura 1.2: Estructura general del contenido del CD y del repositorio de GitHub.

1.6. Estructura del repositorio

La estructura que sigue nuestro proyecto y que se puede encontrar en Github [1.5](#) es la siguiente:

- Fichero **run.bat**

Este es el fichero que deberemos ejecutar para ver como funciona el programa sin necesidad de importar el proyecto entero en un editor de python. Al darle doble click (depende de los permisos que tengamos, puede ser necesario ejecutarlo como administrador), se encargará de ejecutar el programa desde el principio.

- Carpeta **Other**

En esta carpeta se encuentra el material ajeno al proyecto en python, y que, por lo tanto, no afecta a la ejecución del programa. En dicha carpeta se encuentra:

- Carpeta **Images**

Contiene diversas imágenes relacionadas con el desarrollo del proyecto, explicaciones sobre algunos aspectos cuya descripción con palabras podía no quedar demasiado clara, etc.

La mayoría de estas imágenes se han realizado de forma 'casera' por nuestra cuenta, aunque pueden encontrarse también algunas imágenes obtenidas de internet (en cuyo caso se especificará su origen).

- Fichero **TFG_FacialRecognition_Memoria.pdf**.

Fichero **TFG_FacialRecognition_Anexos.pdf**.

Carpeta comprimida **TFG_FacialRecognition.zip**

Contienen esta memoria junto con sus anexos, tanto en formato **.pdf** como los ficheros **.tex** necesarios para generarla.

- Fichero **videoexplicativo.mp4**

Un vídeo que se ha realizado explicando el funcionamiento general del programa.

- Carpeta **sources**

En esta carpeta se puede encontrar material que afecta a la ejecución del programa, pero que no es código ejecutable como tal. En dicha carpeta encontramos:

- Carpeta **dataset**

En esta carpeta podemos encontrar las imágenes originales que guardamos al principio en nuestra base de datos. Como se puede observar, son imágenes de cuerpo entero la mayoría de ellas, por lo tanto no están ajustadas a la posición de la cara de la persona, por lo que necesitamos crear (interna y automáticamente) la siguiente carpeta:

- Carpeta **facesDataset**

Contiene las imágenes anteriores, pero exclusivamente con la cara que haya podido encontrar en la imagen, recortando dicha imagen para eliminar elementos ajenos al rostro que puedan dificultar el reconocimiento.

- Carpeta **xml**

Contiene dos ficheros .xml que son los encargados de encontrar las partes identificables de un rostro en la imagen. Dichos ficheros se proporcionan gratuitamente con fines educativos en un repositorio [50].

- Carpeta **recognizer**

Contiene ficheros que utilizamos durante la ejecución del programa para visualizar datos sobre la persona reocnoscida o cargar la imagen correcta de la base de datos a partir de un diccionario.

- Fichero **dictionary_ID_labels.txt**

Este fichero se crea automáticamente tras entrenar la red, y contiene el diccionario que relaciona una ID automática a cada imagen de la que ha conseguido obtener un rostro. Se podría evitar usar este fichero si se utilizara directamente el diccionario en el programa, pero eso supondría tener que entrenar la red todas las ejecuciones, y si no se han añadido imágenes nuevas es tiempo y recursos perdidos, de modo que se ha optado por crear un fichero para que el diccionario persista a cada ejecución.

- Fichero **info.txt**

Este fichero se ha creado sólo para mostrar información de la persona que se haya reconocido durante la ejecución. Para mostrar esta información consultar el anexo *E_Manual_usuario*. Esta información se puede actualizar a mano actualmente, aunque conviene no hacerlo, ya que es el propio programa el encargado de actualizarla en caso de añadir nuevas imágenes.

- Fichero **trainedData.yml**

Este fichero contiene las principales características de los rostros detectados cuando entrenamos la red. Es el resultado de dicho entrenamiento, y por lo tanto se crea automáticamente. Las características que obtiene son las mismas que las que se pueden encontrar en los ficheros .xml de dicha carpeta.

- Fichero **default.png**

Esta imagen es la que carga el programa por defecto cuando no se ha conseguido superar el umbral de reconocimiento de un rostro en una imagen. Normalmente se conseguirá superar dicho umbral, aunque el reconocimiento falle, pero en caso de que no se hayan podido encontrar rostros, no se haya entrenado la red, o se le pase un fichero que no sea una imagen, el programa cargará esta imagen por defecto.

- Carpeta **src**

En esta carpeta se encuentra el código de nuestro programa. Está dividido en diferentes clases que vemos a continuación:

- Fichero **Camera.py**

Esta clase tiene métodos relacionados con la cámara, ya sea obtener una imagen o inicializar la propia cámara. Dentro de esta clase se especifica si la imagen que estamos obteniendo la vamos a usar para una cosa (compararla) o para otra (guardarla en la base de datos).

- Fichero **CompareImages.py**

Realiza la comparación entre la imagen que acabamos de obtener (ya sea mediante captura con la cámara o desde fichero), con todas las de la base de datos. Se puede ver el funcionamiento exacto de esta función en el anexo *D_Manual_programador*. Se ha reutilizado esta clase tanto para el reconocimiento de una imagen 'estática' (un solo frame) como de la imagen en tiempo real (vídeo o varios frames).

- Fichero **Files.py**

En esta clase están todos los métodos relacionados con ficheros, desde obtener una simple ruta hasta cargar todos los datos almacenados en él (*loadInfo()*).

- Fichero **GUI.py**

Contiene todo lo relacionado con la interfaz gráfica, desde el panel central, donde se muestra toda la información y las imágenes obtenidas, hasta el menú que nos permite seleccionar una imagen desde fichero en vez de desde la cámara. Dicha interfaz se ha realizado con tKinter. Se puede ver cada uno de los métodos y su funcionalidad en el anexo *D_Manual_programador*.

- Fichero **Main.py**

Contiene el método principal de la ejecución del programa. Es donde se le pregunta al usuario si quiere entrenar la red, seleccionar una imagen desde fichero, desde la cámara o realizar el reconocimiento en tiempo real, llama a todos los demás métodos de otras clases para comparar las imágenes, generar la interfaz y mostrar resultados. Se pueden ver la estructura en el anexo *D_Manual_programador*, y el flujo de programa en el anexo *C_Diseno*.

- Fichero **RecognizerRealTime.py**

Esta clase contiene el flujo de programa encargado sólo de la captura y comparación de las imágenes en tiempo real, así como su respectiva interfaz.

- Fichero **TextInterface.py**

Esta clase contiene todo lo relacionado con la interfaz en modo texto (por consola de comandos) de nuestro programa, así como un par de clases internas que se encargan de almacenar variables de tipo *Enum*, las cuales vamos a usar para mostrar los mensajes por pantalla, como si de los mensajes de excepciones se tratara.

- Fichero **Trainer.py**

Esta clase se encarga exclusivamente de entrenar la red a partir de unas imágenes fuente, que se encuentran en la carpeta *faces-Dataset*. La funcionalidad completa se puede observar en el anexo *D_Manual_programador*.

- Fichero **Util.py**

Contiene algunos de los métodos de utilidad que no se encuentran ya en otros ficheros (*Files.py*), como pasar una imagen de RGB a escala de grises o recortar la parte de la imagen correspondiente al rostro. Se pueden ver todos los métodos en el anexo *D_Manual_programador*.

- Fichero **README.txt**

Fichero con información general sobre el proyecto, instrucciones básicas y modo de empleo.

Objetivos del proyecto

2.1. Objetivos generales

El objetivo principal de este trabajo consiste en identificar el rostro de una persona presente en una imagen (ya sea grabada o capturada en tiempo real) comparándolo con una lista de rostros almacenados en una base de datos.

Para conseguir este objetivo nuestra aplicación deberá ser capaz de:

- Obtener una imagen, ya sea almacenada o en tiempo real.
- Reconocer un rostro partir de una imagen.
- Comparar dicho rostro con otros almacenados en la base de datos.
- Mostrar el rostro almacenado con mayor coincidencia con el primero en una interfaz sencilla.

Estos objetivos generales se pueden observar en el siguiente esquema 2.3.

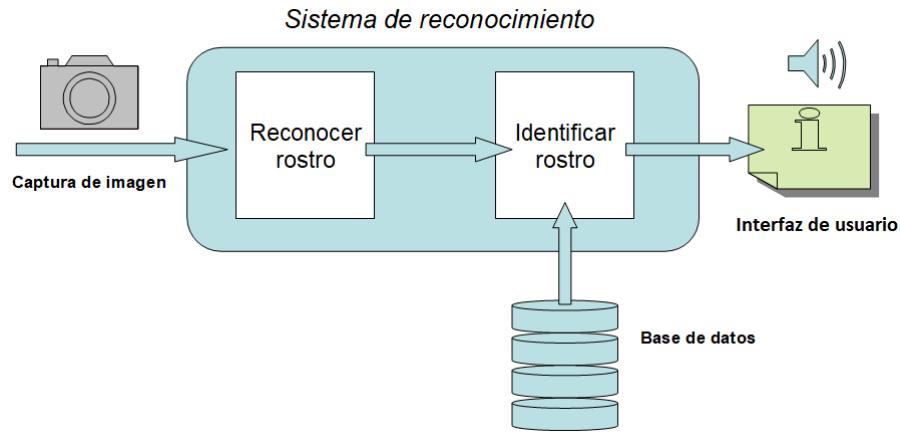


Figura 2.3: Objetivos generales del proyecto.

2.2. Objetivos técnicos

- Obtener, manipular y guardar imágenes con las herramientas *OpenCV 4.3* y *Pillow 4.3*.
- Reconocer un rostro dentro de la imagen utilizando un fichero 1 con las principales características de una cara.
- Entrenar una red mediante técnicas de *machine learning* y *computer vision* utilizando como entrada las imágenes de la base de datos, y como herramienta *OpenCV 4.3* y una técnica *LBPH 3.3*.
- Hacer que se reconozca el rostro de la persona de la imagen capturada a partir de los resultados de la red entrenada, comparando características de rostros.
- Crear una interfaz adecuada para mostrar los resultados finales utilizando como herramientas principales *Pillow 4.3* y *tKinter 4.3*.
- Realizar el informe con la herramienta *LaTex 4.3*.

2.3. Objetivos personales

- Aplicar varios de los conocimientos obtenidos durante la carrera.
- Familiarizarme más con el lenguaje de programación *Python*, y sus herramientas más comunes.
- Utilizar y conocer de primera mano aquello de lo que habla todo el mundo acerca de la última tecnología (*machine-learning*).
- Desarrollar un proyecto que pueda servir de carta de presentación a la hora de buscar empleo o empresas que estén interesadas en seguir adelante con dicho proyecto.
- Aprender a desarrollar una interfaz para una aplicación desarrollada en *Python*, aunque sencilla, que permita ir profundizando poco a poco en la 'puesta en escena' de nuestro programa.

Conceptos teóricos

Algunos de los conceptos relacionados con el presente proyecto son:

3.1. Inteligencia Artificial

La inteligencia artificial (AI) según *searchdatacenter* [58], es: 'la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos. Estos procesos incluyen el aprendizaje (la adquisición de información y reglas para el uso de la información), el razonamiento (usando las reglas para llegar a conclusiones aproximadas o definitivas) y la autocorrección.'

Aprendizaje Automático

El aprendizaje automático (o más conocido por su nombre en inglés: Machine-Learning). es la parte de la inteligencia artificial que consiste en conseguir que un ordenador actúe de manera automática, sin necesidad de programarlo para ello.

Analítica predictiva

La analítica predictiva es la ciencia que analiza los datos para intentar predecir datos futuros relacionados. En nuestro caso, a partir de unos datos originales, se analizan mediante el aprendizaje automático para crear nuevos modelos predictivos.

Aprendizaje profundo

El aprendizaje profundo (o más conocido por su nombre en inglés: Deep-Learning), es una rama del aprendizaje automático que automatiza la analítica predictiva.

En el siguiente artículo de *Christian S. Perone* [53], se puede encontrar un ejemplo de una red convolucional desarrollada en python.

Visión Artificial

La visión artificial [76] (o más conocido por su nombre en inglés: ComputerVision) pretende simular el comportamiento del ojo humano, permitiendo adquirir, procesar, analizar y comprender las imágenes mediante la clasificación y segmentación de las mismas utilizando técnicas de machine learning.

Las grandes empresas de tecnología están desarrollando su propio software para aprovechar al máximo sus aplicaciones, una de estas aplicaciones, y de las que mejores resultados proporciona (además de no necesitar instalar nada), es de *Google* y se llama *Google Cloud Vision* [25].



Figura 3.4: [6] Google Cloud Vision.

3.2. Redes neuronales

Red neuronal artificial

Las redes neuronales artificiales [82] son un modelo computacional basado en nodos y enlaces, donde **cada nodo corresponde con una neurona**, y los enlaces corresponden a las relaciones entre dichos nodos. Cada nodo almacena información independiente del resto de nodos, de manera que nuestro sistema se divide en función de la información independiente que tenga.

Dados unos **datos de entrada** (nodos *input*), se forman relaciones entre ellos para establecer los aspectos comunes y las diferencias.

De estas relaciones nos quedamos con los nodos intermedios de la red (nodos *hidden*), que son los que no ve el usuario (programador), y que puede haber tantas capas de nodos *hidden* como hagan falta.

Por último, la última de estas capas se conforma de los nodos de salida que vamos a obtener como **resultado del entrenamiento** de la red (nodos *output*).

Este proceso se puede observar en la siguiente imagen 3.5.

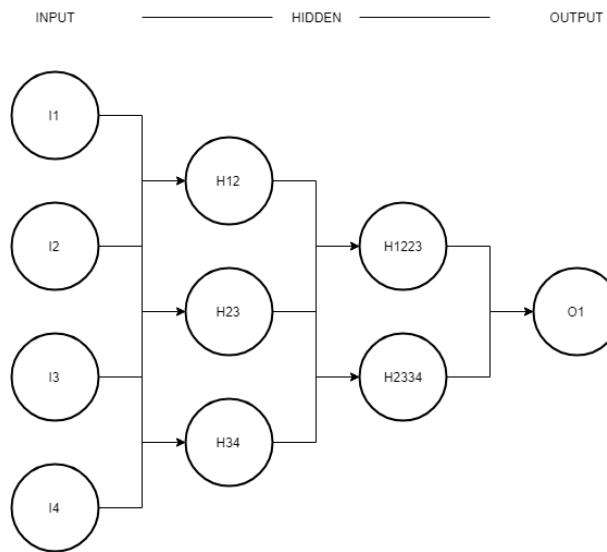


Figura 3.5: Red neuronal con las diferentes capas implicadas.

El objetivo de la red neuronal es resolver los problemas de la misma manera que el cerebro humano, aunque de forma más abstracta y con variantes de entre miles a millones de neuronas (nodos).

Algo importante cuando hablamos de redes neuronales, inteligencia artificial y machine learning, es que **no se puede garantizar nunca su grado de éxito** después de realizar el auto-aprendizaje. Para obtener unos resultados más o menos homogéneos, se necesitan miles y miles de ejecuciones.

Red neuronal convolucional

Las redes neuronales convolucionales [83] son un subtipo de las redes neuronales artificiales, que se basan en las versiones biológicas de los perceptrones multicapa (Multi Layer Perceptron en inglés) [80], y consisten en **varias capas** o *layers*, en cada una de las cuales existen una serie de nodos con información.

Estos nodos se conectan con cada uno de los nodos de la capa siguiente (a diferencia de las redes neuronales normales, que simplemente crean un nodo de la capa siguiente a partir de dos de la capa anterior), quedando **todos los nodos interconectados entre una capa y otra**.

Este tipo de redes son especialmente útiles en sistemas de computer vision y deep learning, ya que los nodos se retroalimentan entre si, dando lugar a mejores soluciones.

Para obtener más información sobre este tipo de redes se pueden consultar las siguientes referencias [60], [35], aunque se proporciona un esquema con el proceso básico de estas redes en la siguiente imagen 3.6.

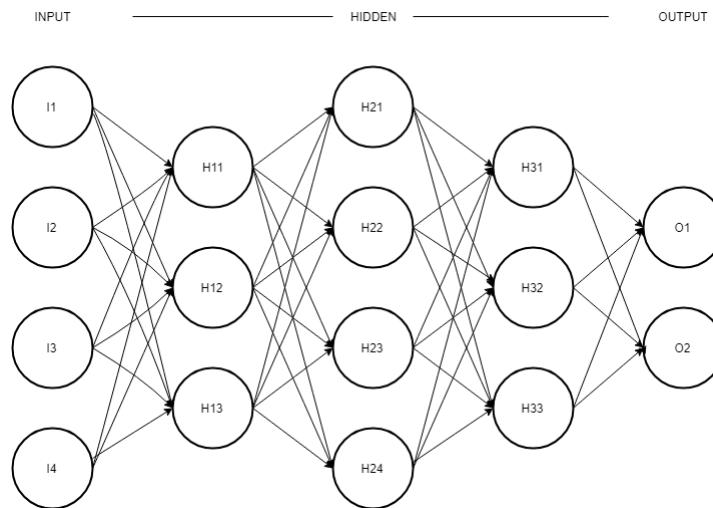


Figura 3.6: Red neuronal convolucional con las diferentes capas implicadas y cómo se relacionan los nodos de cada una entre si.

Tipos de Entrenamiento

Los diferentes tipos de entrenamiento de una red son:

- Aprendizaje supervisado

En este tipo de aprendizaje los datos tienen etiquetas, las cuales se usan para obtener nuevos patrones relacionados con los datos de dichas etiquetas.

- Aprendizaje no supervisado

En este tipo de aprendizaje los datos no tienen etiquetas, de manera que los nuevos patrones se obtienen y clasifican en función de similitudes y/o diferencias con el resto de datos.

- Aprendizaje por refuerzo

En este sistema de aprendizaje, como en el no supervisado, no hay etiquetas, pero a diferencia de este, tras realizar un modelo o patrón, el sistema recibe retroalimentación para ajustar dicho patrón.

3.3. Reconocimiento Facial

El reconocimiento facial [15], [20] es una de las principales aplicaciones de la visión artificial, y consiste en localizar y reconocer rostros de gente en imágenes.

El proceso de este reconocimiento se compone de dos partes:
(Para la información completa consultar el siguiente artículo [21].)

1. Detección de rostros

En esta primera parte del proceso se identifican los píxeles de la imagen que pertenezcan a un rostro. Para realizar dicha identificación, hay varios algoritmos, aunque uno de los más utilizados y extendidos es el llamado *Haar Cascade face detection* [2]. Todas las versiones de este algoritmo se pueden encontrar en el siguiente repositorio [50].

Hay muchas aplicaciones [37], [36] que se aprovechan de esta detección de rostros sin llegar a usar el reconocimiento facial, ya que no supone tanto trabajo implementar esta primera parte como seguir con el proceso.

2. Reconocimiento de rostros

En esta segunda parte se utilizan diversas técnicas para evitar los problemas derivados de la calidad de la imagen (iluminación, posición de la persona, cambios en los rasgos faciales, etc) como pueden ser las funciones que realizan cortes o *thresholds* de nuestra imagen a partir de ciertos valores umbrales, y se obtiene como resultado una segunda imagen más *estandarizada*, la cual comparamos (usando en nuestro caso los *CascadeClassifier* [1]) con las otras imágenes que teníamos preparadas para ser comparadas, y como resultado final obtenemos una de estas imágenes, que será la que contenga el rostro que queríamos reconocer en la primera imagen (o la que más se acerque).

Eigenvectores

Un eigenvector o valor propio de una imagen es, según wikipedia [85]: 'un vector no nulo que, cuando es transformado por el operador, da lugar a un múltiplo escalar de sí mismo, con lo que no cambia su dirección.'

Esta definición a nosotros no nos interesa mucho, aunque si nos interesa su aplicación, que viene a significar lo siguiente:

De una imagen, obtenemos todos sus píxeles, y hacemos que cada uno de ellos represente un elemento de una matriz de tamaño MxN (tamaño de la imagen). Empezamos por un píxel (generalmente al azar), y realizamos la operación que hemos visto en la definición más arriba con cada uno de los píxeles en los alrededores. De esta manera estamos **comparando su dirección con la de cada uno de los píxeles cercanos**, para saber cuales de ellos hacen que cambie su dirección.

Si todos los píxeles en los alrededores mantienen la dirección del pixel actual, significa que todos ellos pertenecen al mismo elemento dentro de la imagen. Si hay alguno que hace cambiar de dirección el eigenvector del pixel actual, significa que esos dos píxeles pertenecen a elementos diferentes de la imagen.

De esta manera se pueden obtener las diferentes características de una imagen a partir de los eigenvectores.

Hay multitud de aplicaciones de los eigenvectores relacionados con imágenes por internet, en la siguiente página se muestran algunas de ellas [29].

Algoritmos de Reconocimiento Facial

Los diferentes algoritmos que se barajaban para realizar el reconocimiento facial fueron:

Eigenfaces

Las eigencaras o eigenfaces [87] es el nombre que se les da a los eigenvectores cuando son usados en el reconocimiento facial de las personas con computer vision.

Es la forma más básica y una de las primeras técnicas conocidas para el reconocimiento facial, que se empezó a usar a partir de 1991.

Fisherfaces

Las fisherfaces [45] se basan en el mismo modelo que las eigenfaces, aunque en este caso se tiene también en cuenta la luz y los espacios entre características del rostro, dando más importancia a las expresiones faciales que a los propios rostros.

A partir de 1997 se le empezó a dar un poco más de importancia a este método, aunque nunca acabó de tener mucho éxito.

Local binary patterns

Según *Kelvin Salton* [61], el algoritmo LBP (Local Binary Pattern) es un sistema simple pero eficaz, que consiste en etiquetar los píxeles de una imagen a partir del umbral que se le asigna a cada uno de sus vecinos. Esto significa que le da un valor a cada uno de los píxeles en función de la intensidad de color de los píxeles cercanos [59]. Para ello es necesario tratar la imagen en **escala de grises**.

Si la intensidad de color de uno de los píxeles es mayor o igual, a ese pixel se le asigna un *1*, en caso contrario un *0*. A continuación, se juntan los 8 bits obtenidos de los 8 píxeles vecinos al pixel que estemos comparando y se forma un número en binario (el orden en que se leen los bits es irrelevante, mientras se use el mismo orden en toda la imagen, aunque lo más común suele ser empezar por una esquina y seguir el sentido de las agujas del reloj, aunque no hay ningún standard), que corresponderá con la nueva **intensidad de color** del pixel (de 0 -negro- a 255 -blanco-).

Con esta nueva intensidad de color definida para todos los píxeles de la imagen, se puede obtener una relación entre ellos, saber qué pixeles están más relacionados entre ellos que con otros, etc. A partir de cada una de estas agrupaciones **se obtienen las características** que vamos a utilizar en nuestro sistema de reconocimiento facial.

Este método se empieza a estudiar antes que las fisherfaces, aunque hasta 1996 no se pone de moda. A partir de entonces se ha convertido en el más utilizado por su sencillez y relativa eficacia.

Se puede observar todo el proceso en la siguiente imagen 3.7.

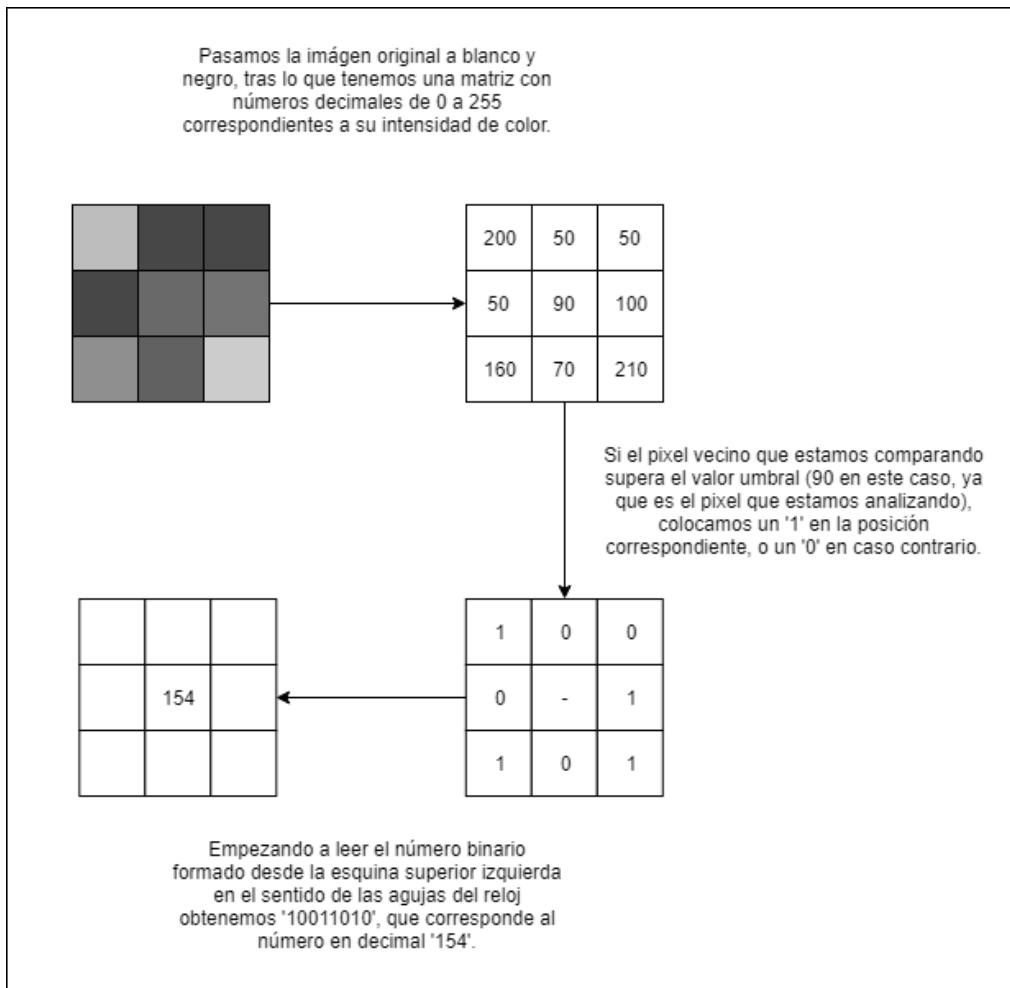


Figura 3.7: Proceso de agrupación e identificación de características en una imagen según el algoritmo LBP.

LBP-Histograms (LBPH)

Basado en la imagen resultado del algoritmo anterior (recordemos que habíamos obtenido una imagen en escala de grises), se obtienen una serie de histogramas en función de las regiones que definamos. Cada uno de estos histogramas representan la relación de intensidad de cada píxel.

A continuación, se concatenan todos los histogramas para formar un histograma más grande, el cual contiene las características de la imagen original.

Técnicas y herramientas

4.1. Técnicas metodológicas

Aunque nos hemos basado en un método de **desarrollo en cascada** [77], hemos incluido algunos elementos de la **metodología scrum** [84] como las reuniones mensuales.

Por otro lado, se puede decir que se ha seguido un **diseño planeado**, en lugar de evolutivo, ya que no se evalúa el coste de la actual flexibilidad del código ni el coste de posibles refactorizaciones más tarde, sino que se ha diseñado para minimizar los cambios con futuras necesidades desde el principio. Esto se puede observar en algunos métodos más generales de lo que nos hacen falta en un principio, o en cómo está organizada la estructura general del proyecto.

En cuanto al desarrollo en cascada, hemos tomado la base de este método, ya que teníamos unos **requisitos iniciales**, que eran los objetivos primarios (*2_Objetivos_del_proyecto 2.1*) que teníamos que cumplir, a partir de los cuales se hizo un primer **diseño del proyecto**.

Aquí es donde entra la metodología scrum, y es que incluimos la primera reunión con el tutor, para ver que estructura tenía el proyecto y que posibles cambios iniciales se podrían realizar.

Esta primera fase del proyecto (identificación de requisitos, planteamiento del proyecto y diseño de la estructura básica del mismo) duró aproximadamente un mes, desde principio de Enero hasta mediados de Febrero.

A continuación, seguimos con el siguiente paso de la metodología en cascada: el **desarrollo e implementación**.

Si bien es cierto que el desarrollo ha continuado hasta casi la fecha de entrega, casi todo el código se implementó desde mediados de Febrero a principios de Mayo, algo más de dos meses. A partir de esa fecha el desarrollo del código fue más de corregir algunos errores, mejorar la apariencia, etc.

Durante toda esta fase de desarrollo se tuvieron varias reuniones mensuales con el tutor para ver cómo iba avanzando el proyecto, lo que necesitaba ser mejorado o cambiado, o asignar nuevas tareas hasta la próxima reunión (en nuestro caso la mayoría fueron semanales o quincenales, en vez de mensuales).

En el apartado *5_Aspectos_relevantes_del_desarrollo_del_proyecto 4.3* y en el anexo *A_Plan_proyecto* se explican las fases del desarrollo de manera detallada, asignando las tareas o *issues* principales a cada uno de ellos.

Además, en los anexos correspondientes a estas fases (*A_Plan_proyecto* para la fase de Planteamiento del proyecto, *B_Requisitos* para la fase de Identificación de requisitos, *C_Diseño* para la fase de Diseño del proyecto y *D_Manual_programador* para la fase de Desarrollo) se explica y detalla cada una de ellas.

4.2. Patrones

A la hora de desarrollar el proyecto, no se hizo pensando en utilizar unos u otros patrones de manera intencionada, sino que se fueron añadiendo en función de las necesidades.

Al principio se utilizaron más patrones [40], como los **Adaptadores** 4.8, que utilizaban clases intermedias para comunicar dos elementos diferentes del proyecto (la base de datos con la interfaz, por ejemplo).

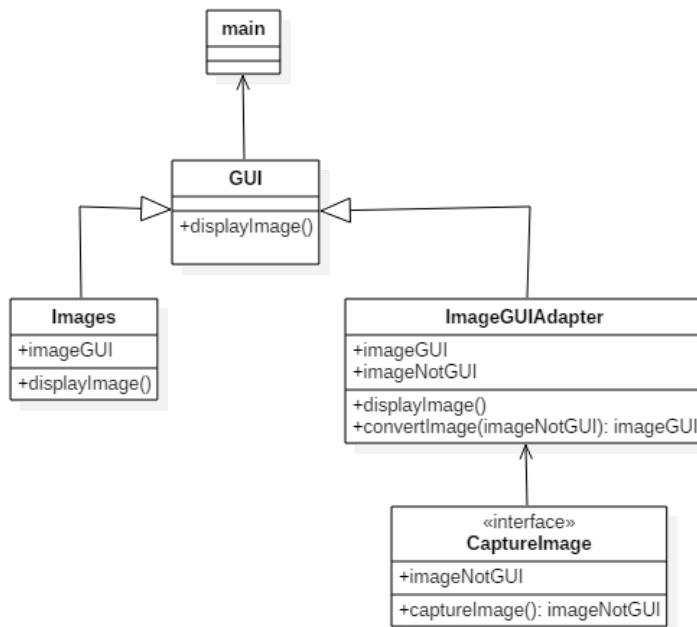


Figura 4.8: Ejemplo de patrón Adaptador utilizado al comienzo del desarrollo del proyecto.

O en el caso de obtener las imágenes que íbamos a analizar, en una primera versión se utilizaba el patrón **Abstract Factory** 4.9, ya que existía una interfaz *CaptureImage* que instanciaba una clase concreta *CaptureImageFromFile* o *CaptureImageFromCamera* en función del origen de la imagen.

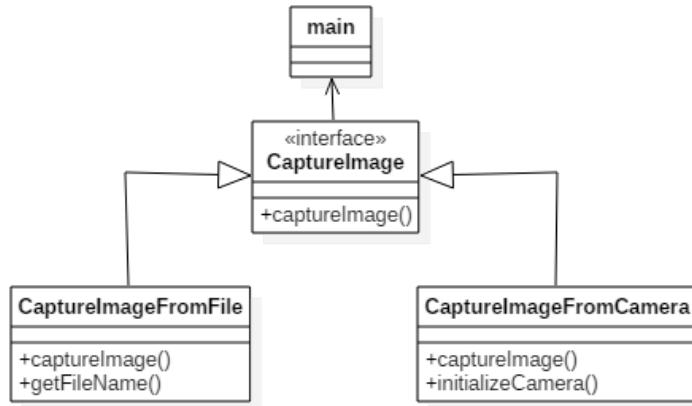


Figura 4.9: Ejemplo de patrón Abstract Factory utilizado al comienzo del desarrollo del proyecto.

Aunque posteriormente se unificaron varias clases, se hizo refactorización de muchos de los métodos anteriores, y se consiguió eliminar complejidad de código, referencias y clases, por lo que algunos de estos patrones ya no eran necesarios.

Al final, el patrón más utilizado y que más claramente podemos encontrar es el de **Singleton 4.10**, especialmente útil cuando queremos instanciar una nueva interfaz gráfica, que nos ayuda a asegurar que no se creen y sobreesciban varias interfaces (ya que la utilizamos de varias maneras diferentes, no nos interesa crear una interfaz para cada uno de estos usos, sino una común que vaya cambiando en función del uso que le queramos dar).

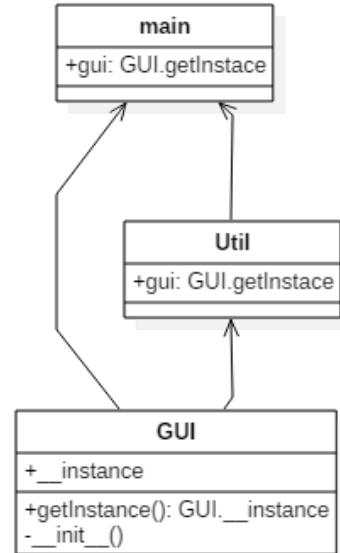


Figura 4.10: Ejemplo de patrón Singleton utilizado en la clase GUI.

4.3. Herramientas

Algunas de las herramientas que hemos utilizado para el desarrollo del proyecto han sido los siguientes:

Herramientas de desarrollo

- El lenguaje de programación ha sido *Python* [16].



- Para realizar la instalación del entorno virtual utilizado durante el desarrollo del trabajo se ha utilizado *Anaconda* [5].



- Se ha utilizado la herramienta *OpenCV* [51] para realizar el reconocimiento facial con computer-vision.



- Para tratar y modificar imágenes se ha utilizado *Pillow* (PIL para python) [7].
- Para el desarrollo de la interfaz se ha utilizado *tKinter* para python [17].

- El editor de texto utilizado ha sido *pycharm* [32].



- Se han obtenido respuestas y soluciones a varios de los problemas encontrados durante la realización del proyecto en *stackoverflow* [30].



- Se ha utilizado la aplicación *image.net* [68] para la obtención de algunas imágenes de muestra.

Herramientas de planificación y diseño

- Para las tareas semanales/mensuales y llevar un control de las versiones del proyecto se ha utilizado *Github* [23] y su cliente nativo para el escritorio de *Windows Github Desktop* [24].



- Para los diagramas de clases se ha usado la herramienta gratuita *starUML* [48].



- Para los flujogramas y otras imágenes de esquemas y diagramas se ha utilizado la herramienta *draw.io* [43].



- Para la memoria se han utilizado la herramienta *LaTex* [79] que permite crear documentos con extensas posibilidades de formateo, y la herramienta *sharelatex* [64], que permite la edición de ficheros en LaTex online.



Aspectos relevantes del desarrollo del proyecto

5.1. Explicación general

El TFG consiste en una aplicación que, dadas dos imágenes, una obtenida en el momento de ejecución, ya sea mediante fichero (accediendo a la foto que tengamos almacenada en nuestro equipo), o mediante una captura que realicemos con nuestra cámara, y la otra alojada localmente en una base de datos: sea capaz de distinguir si en ambas imágenes se encuentra la misma persona utilizando técnicas de machine-learning.

A la hora de comparar ambas imágenes, si la persona que se pone delante de la cámara no estaba registrada en la base de datos, aparecerá la persona que esté registrada que más se parezca. En caso de que no se supere un umbral de coincidencia con ninguna de las personas registradas, mostrará una imagen por defecto avisando de que no se ha podido obtener ningún resultado satisfactorio.

En cualquier caso, junto con la imagen que se obtenga como resultado, se mostrará una barra de progreso, que indica el porcentaje de acierto que ha obtenido al encontrar dicho resultado. Además, se muestra un botón que nos permite crear otra ventana extra con información de la persona obtenida como resultado (nombre, edad, lugar de nacimiento y profesión).

5.2. Requisitos técnicos

Es necesario tener instalado *python* (preferiblemente en su versión 2.7, o un entorno virtual con dicha versión ya que es el entorno en el que se ha desarrollado) y diferentes librerías: *cv2* (machine-learning orientado a imágenes), *tKinter* (interfaz), *numpy* (utilidad), *Pillow* (operaciones con imágenes). Todas ellas se pueden encontrar en la sección 4.3. En nuestro caso se ha probado en un entorno controlado con *Windows 10*, no se asegura su funcionamiento para otras distribuciones.

En nuestro entorno de pruebas se ha utilizado tanto la cámara integrada del portatil (*Sony Vaio VPCF13A4E* con resolución [640 X 480]) como una cámara externa (*Logitech WEBCAM C170* con resolución [1024 X 768]) [41]. Para leer las especificaciones completas del equipo utilizado consultar el anexo *B_Requisitos*.

5.3. Fases de desarrollo

Aunque se ha intentado seguir cierto orden en las fases de desarrollo del proyecto [65], las últimas se iban solapando de vez en cuando, ya que los resultados de unas (*Reconocimiento facial*) afectaban sobre otras (*Interfaz*), a continuación se presentan todas ellas.

Definir el proyecto

Al principio se tuvo que escoger el tipo de proyecto que se iba a realizar. En nuestro caso un proyecto medio, ya que no se tenían ni el tiempo ni los recursos necesarios para que fuera un proyecto demasiado grande, y tampoco podía ser un proyecto demasiado pequeño dado el objetivo del mismo.

También se tuvo que definir la idea principal, que en nuestro caso iba a ser reconocer gente.

Identificar información, recursos y requisitos

Juntando este apartado con el anterior de definición del proyecto, se optó por adaptar la base de datos a un sistema más accesible y rápido de usar, almacenando las imágenes en una carpeta reservada para tal fin.

A continuación, se definió qué tipo de información iba a tratar: íbamos a trabajar con imágenes en sus diferentes formatos, y con redes neuronales, las cuales íbamos a entrenar para que reconocieran a la gente almacenada en nuestra base de datos.

Los recursos de los que íbamos a disponer eran: la imagen que obtuviéramos con la cámara (o desde un fichero llegado el caso), otras imágenes de gente a la que necesitáramos reconocer (almacenadas como hemos dicho en nuestra carpeta), y por último, información básica sobre cada una de las personas de nuestro programa.

Instalación y configuración

Antes de empezar a programar, fueron necesarias varias instalaciones de prueba en varios entornos virtuales diferentes (esto fue llevado a cabo en -relativamente- muy poco tiempo gracias a Anaconda 4.3 y Pycharm 4.3) y con varias configuraciones distintas, tanto del mismo entorno como de las librerías necesarias especificadas en la sección *Requisitos técnicos* 5.2.

Mientras se llevaba a cabo este proceso, se fueron diseñando diversos diagramas de clases y flujoogramas básicos, que eran la primera idea de cómo debería quedar nuestro programa, aunque como veremos más tarde, el diseño cambió mucho con el tiempo. Se proporcionan estas figuras iniciales en el anexo *A_Plan_proyecto*, sección *Flujoogramas y diagramas de clases iniciales*.

Durante esta fase empezaron a surgir dudas, algunas de las cuales se pudieron resolver consultando las referencias asociadas en el anexo *D_Manual_programador*.

Captura y manipulación de imágenes

Como mencionamos en la sección *2_Ojetivos_del_proyecto*, nuestro primer objetivo era *Obtener imágenes en tiempo real*, de manera que fue una de las primeras tareas en llevarse a cabo. Aunque primero, se aprendió a abrir imágenes desde un fichero determinado, a trabajar con ellas de diversas maneras (entre las que destacan: cambiarlas el tamaño 5.11 -para mostrarlas todas con el mismo en la interfaz de usuario- y pasarlas a blanco y negro 5.12 -para que el sistema de reconocimiento tuviera unas características fijas de color-).

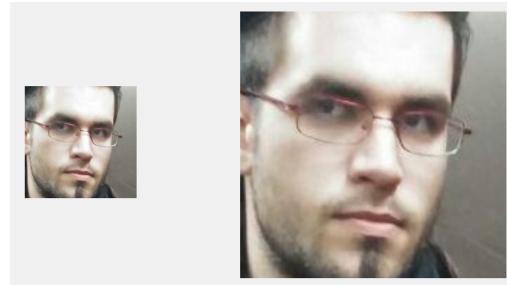


Figura 5.11: Imagen del rostro escalada para la interfaz.

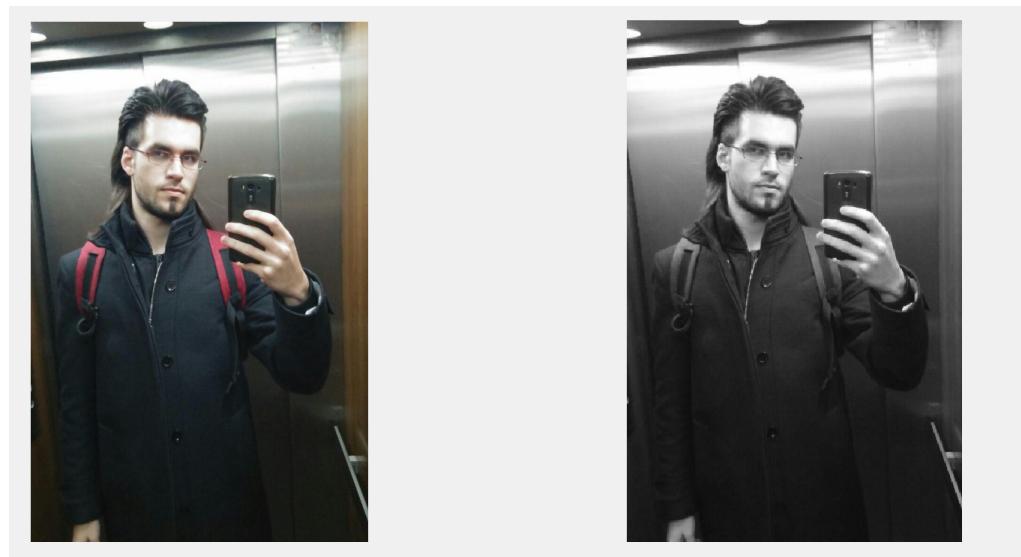


Figura 5.12: Imagen original convertida a escala de grises.

A continuación, se desarrolla la captura de imágenes en tiempo real (se puede observar su funcionamiento en el vídeo proporcionado 1.6). Ayudándonos de la herramienta OpenCV 4.3, podemos tanto obtener como mostrar la imagen de la cámara en nuestra pantalla de forma muy sencilla, siendo la parte que más **problemas** produjo la **inicialización de la cámara**, ya que a veces no llegaba a cargar los valores que debía tener, obteniendo imágenes completamente en negro. Para ello, antes de empezar a capturar los frames, se inicializaron explícitamente todos los valores como se muestra en la imagen 5.13, esperando dos (2) segundos en todos los casos, de esta manera nos ahorraremos problemas de este tipo.

```

def initializeCamera(indexCamera):
    maxInt = sys.maxsize

    print("Initializing camera...")

    cap = cv2.VideoCapture(indexCamera)

    if not cap.isOpened():
        print("Camera couldn't be turned on.")
        return None

    cap.set(cv2.CAP_PROP_FPS, maxInt)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, maxInt)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, maxInt)

    # Esperamos dos segundos para que la cámara termine de inicializarse y no tomar datos basura
    time.sleep(2)
    return cap

```

Figura 5.13: Método que se encarga de inicializar la configuración de la cámara ('cv2' corresponde a la librería de OpenCV).

Durante esta fase se encontraron alguno problemas referentes a las imágenes leídas desde fichero, algunos de los cuales se pudieron resolver consultando las referencias asociadas en el anexo *D_Manual_programador*.

Reconocimiento facial I: Localizar rostros

Con las imágenes ya obtenidas (de una manera u otra), lo siguiente es realizar el reconocimiento facial 1, para lo cual, necesitábamos una cara, por lo que el siguiente paso en nuestro desarrollo fue identificar un rostro en una imagen, quedándonos sólo con esa parte.

Utilizamos openCV para cargar un fichero clasificador en cascada (Haar Cascade face detection 1), el cual se encuentra en la carpeta *xml* dentro de *sources* 1.6 utilizando la función *CascadeClassifier* (cuya información se puede encontrar en el manual oficial [1]). Este fichero se llama *haarcascade_frontalface_default* porque estamos identificando rostros, si quisieramos identificar otros objetos como coches, señales, etc tendríamos que cambiar este fichero por el correspondiente, que se puede encontrar en el repositorio [50].

cv2.CascadeClassifier('/sources/xml/haarcascade_frontalface_default.xml')

Con el fichero *.xml* cargado, pasamos a detectar los rostros, para ello utilizamos la función proporcionada por el manual de OpenCV, la cual se encarga de hacer exactamente esto. El funcionamiento consiste en coger una parte de la imagen, comprobando si existe o no un rostro en esa parte. Si se detecta rostro se guarda en una lista de rectángulos, y el proceso se repite hasta que se ha recorrido la imagen entera. En el siguiente artículo de *Adrian Rosebrock* se explica cada uno de los parámetros de dicha función [57].

faceCascade.detectMultiScale(gray, scaleFactor, minNeighbors, minSize)

Donde *faceCascade* corresponde con el fichero cargado previamente y *gray* corresponde con la imagen de la cual queramos obtener los rostros pasada a escala de grises 5.12.

El parámetro *scaleFactor* representa qué cantidad de la imagen es analizada en cada iteración del proceso (cuanto mayor sea el valor, menos tiempo tardará el proceso, ya que estará tomando trozos de la imagen más grandes, por lo que realizará menos iteraciones).

El parámetro *minNeighbors* representa en cuantos rectángulos a la vez (como mínimo) debe aparecer el mismo rostro.

El parámetro *minSize* representa el tamaño mínimo que puede tener un rostro, esto significa que, si hay objetos que haya detectado como rostros, que sean menores que el tamaño especificado, se ignorarán.

Este parámetro es importante y se utiliza para **evitar** tomar caras de gente que se encuentre de fondo o **falsos positivos**. En nuestro caso se ha creído conveniente ignorar todo aquello que reconozca como rostro que no ocupe al menos un décimo de la imagen (*int(gray.shape/10)*), ya que las imágenes van a estar capturadas siguiendo unos cánones 1.3 y por lo tanto los rostros van a estar centrados ocupando la mayor parte de la imagen.

Como era la primera vez que utilizamos esta herramienta (OpenCV), surgieron problemas que se fueron solucionando poco a poco, algunos de los cuales se pueden encontrar en las referencias asociadas en el anexo *D_Manual_programador*.

Una herramienta alternativa que se tuvo en cuenta a la hora de realizar el proyecto fue DLib. DLib es una herramienta para reconocimiento básico de rostros [38], pero carecía de alguna de las opciones que necesitábamos en nuestro proyecto, de manera que se ignoró esta opción.

Reconocimiento facial II: Identificar rostros

Una vez que tenemos la parte de la imagen que queremos analizar (el rostro), pasamos a comparar las características de ese rostro con las características de todos los rostros almacenados en nuestra pequeña base de datos 2.

Por supuesto, pensamos que extraer esas características en tiempo real supondría mucha carga de trabajo, no en nuestro caso, pero si con una base de datos con muchas imágenes (+1000), por lo que, cuando entrenamos la red, se extraen esas características y se guardan en un fichero *.yml* llamado *trainedData*, de manera que sólo hace falta actualizarlo si se han cambiado cosas en la base de datos.

Esto tiene sus pros y sus contras, que se explican más a fondo en la sección *7_Conclusiones_Líneas_de_trabajo_futuras 7.1*.

Aún así, se decidió dejarle la opción abierta al usuario, por si quería ver cuánto tiempo tardaba o cómo lo hacía internamente, de manera que se le pregunta, junto con otras opciones en el menú principal que decidimos crear, como se puede observar en la siguiente imagen 5.14 (lo explicaremos más tarde en la sección *Interface 5.3*).

Como se puede ver en la imagen 5.14, se imprime además el tiempo de entrenamiento por si fuera del interés del usuario.

```

Creating face-focused images from original-database images...
Training network...

Loading file ..\sources\dataset\alexandra_daddario.jpg...
Loading file ..\sources\dataset\cas.jpg...
Loading file ..\sources\dataset\cas_abii.jpg...
Loading file ..\sources\dataset\cesar_sala_juntas.jpg...
Loading file ..\sources\dataset\chris_pratt.jpg...
Loading file ..\sources\dataset\dwayne_the_rock_johnson.jpg...
Loading file ..\sources\dataset\elizabeth_olsen.jpg...
Loading file ..\sources\dataset\gaben.jpg...
Loading file ..\sources\dataset\gal_gadot.jpg...
Loading file ..\sources\dataset\john_cena.jpg...
Loading file ..\sources\dataset\katheryn_winnick.jpg...
Loading file ..\sources\dataset\madds_mikkelsen.jpg...
Loading file ..\sources\dataset\natalie_dormer.jpg...
Loading file ..\sources\dataset\new_image_name.jpg...
Loading file ..\sources\dataset\robert_downey_jr.jpg...
Loading file ..\sources\dataset\shohreh_aghdashloo.jpg...
Loading file ..\sources\dataset\stan_lee.jpg...
Loading file ..\sources\dataset\will_smith.jpg...

Training time with 18 images: 6.33 seconds.

```

Figura 5.14: Entrenando red y generando fichero con características.

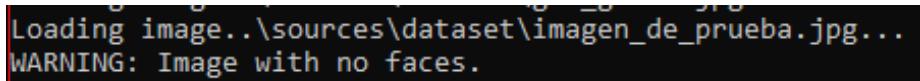
Para comparar nuestras características con las de la base de datos, necesitamos hacerlo siguiendo un algoritmo, que podríamos haberlo hecho manualmente, lo que habría supuesto una pérdida de tiempo importante, ya que los principales vienen implementados por defecto en la librería *cv2*.

```
recognizer=cv2.face.LBPHFaceRecognizer_create
```

Con el algoritmo de comparación preparado, se cargan los datos a los cuales se va a acceder posteriormente a partir del fichero *.yml* mencionado anteriormente. Por suerte para nosotros, la misma librería cv2 nos proporciona dicho método:

```
recognizer.read(yml)
```

Pero además le hemos añadido un control de errores de modo que, si la red no está entrenada nos avisa y nos pide amablemente que la entrenemos. También tenemos otro control de errores a la hora de entrenar la red, como por ejemplo, si tomamos una imagen que no tiene rostros en ella, nos avisará que no ha podido reconocer ninguno y, por lo tanto, no podrá extraer las características ni entrenar la red con dicha imagen, pasando a la siguiente.



```
 Loading image..\sources\dataset\imagen_de_prueba.jpg...
WARNING: Image with no faces.
```

Figura 5.15: Mensaje avisándonos de que en esa imagen no hay rostros.

Como hemos comentado en la sección anterior 5.3, la función *detectMultiScale* nos devuelve una lista con todas las caras detectadas, más concretamente una lista en la que cada uno de sus items contiene cuatro parámetros, correspondientes al rectángulo que determina cada rostro (*x*-posición en el eje X donde se encuentra el primer vértice del rectángulo, *y*-posición en el eje Y donde se encuentra el primer vértice del rectángulo, *w*-anchura del rectángulo, *h*-altura del rectángulo).

En nuestro caso, como la aplicación va a servir para reconocer el rostro de una sola persona a la vez, se ha acotado la forma de comparación, de manera que, si hay más de una entrada en esa lista (una cara), no se tienen en cuenta ninguna excepto la primera.

Para realizar la comparación propiamente dicha, el propio método algoritmo (*LBP**H* en nuestro caso) proporciona una función que se dedica a ello, dando como resultado un número (ID) de la imagen de la base de datos con más coincidencia. Esta coincidencia es el segundo parámetro que nos devuelve la función por defecto.

$$id, conf = recognizer.predict(gray[y:y + h, x:x + w])$$

El parámetro *conf* nos da un valor entre 0 y 100, que nos dice cuan diferentes son ambas imágenes, representando el 0 ninguna diferencia, y el 100 totalmente diferentes.

Al principio se pensaba que *conf* se correspondía con el tanto por ciento (%) de coincidencia de la imagen, pero dados los bajos resultados que obteníamos siempre, se siguió investigando y se observó que era justo el comportamiento opuesto, de manera que se optó por hacer una simple operación: *100-conf*, para facilitar la lectura de los datos.

¿Por qué OpenCV?

La idea original del proyecto era desarrollar un sistema de reconocimiento facial utilizando *Tensorflow*, una herramienta muy poderosa y multiusos basada en machine-learning [11], la cual está a disposición de cualquiera en su repositorio oficial [71]. Tras estudiar un poco su funcionamiento y realizar algunos tutoriales sobre su uso general [39], se optó por una opción más cómoda y menos engorrosa, como era el caso de OpenCV.

También, aunque en menor medida y ya con el proyecto más adelantado, se probaron otras opciones que pudieran facilitar su desarrollo como *Keras* [69] (API de redes neuronales de alto nivel en Python donde se pueden ejecutar programas en Tensorflow de manera más rápida y eficiente) o *Google colab* [26] (proyecto de Google que simula un entorno de notebooks Jupyter, en el que no hace falta cargar nada, simplemente programamos nuestro código y lo probamos en la nube). Ambas eran muy buenas opciones para haber desarrollado el proyecto desde cero con ese enfoque, pero se probó a finales de abril o principio de mayo, de manera que ya no daba tiempo a cambiar todo el proyecto.

¿Por qué este sistema de entrenamiento?

Nuestro sistema de entrenamiento consiste en coger una sola imagen de cada persona, a ser posible con una calidad y características parecidas (fondo blanco, iluminación parecida, etc.), extraer las características de cada una y decirle a nuestra red que esos datos corresponden a una persona diferente cada vez. De esta manera él es capaz de diferenciar una de esas personas, pero no un desconocido.

Esto puede diferir de la idea que se tiene en general sobre entrenar una red, pero es que nosotros no queremos reconocer a todas las personas de la calle, solamente queremos reconocer a aquellas que necesitemos. Por ello otros sistemas de aprendizaje, como pasarle muchas imágenes de la misma persona alteradas levemente no nos son eficaces, ya que de esa manera se le enseña a la red cómo diferenciar dos tipos de *entidades* diferentes, como perros y gatos, o coches y peatones, etc.

Si este hubiera sido el caso se podrían haber utilizado técnicas para generar esas imágenes modificadas de manera automática en vez de tomar varias capturas a la misma persona, a esto se le llama *image augmentation* [34], [4].

Para más información sobre los distintos tipos de entrenamientos se puede consultar la sección *3_Conceptos_teoricos: Tipos de Entrenamiento 3.2*.

A continuación se dejan los tiempos de entrenamiento para cada uno de los casos estudiados (hasta donde se ha podido escalar) en función del número de imágenes: si hay menos de dos imágenes no se entrena y se utiliza el fichero anterior (este es un caso especial explicado en el anexo *D_Manual_programador*, con dos imágenes (el mínimo necesario) 5.16, con 15 imágenes (nuestra base de prueba original) 5.17, con 70 5.18 y con 200 imágenes 5.19. Además se incluye una gráfica final 5.20 (realizada con la herramienta *plotly* [54]).

```
Training time with 2 images: 0.3 seconds.
```

Figura 5.16: Tiempo de entrenamiento con 2 imágenes.

```
Training time with 15 images: 2.83 seconds.
```

Figura 5.17: Tiempo de entrenamiento con 15 imágenes.

```
Training time with 70 images: 10.07 seconds.
```

Figura 5.18: Tiempo de entrenamiento con 70 imágenes.

```
Training time with 200 images: 32.04 seconds.
```

Figura 5.19: Tiempo de entrenamiento con 200 imágenes.

Por último, para una base de datos con 1000 imágenes se ha hecho una estimación del tiempo que se tardaría en entrenarla ahora mismo, y ha sido de *155.86* segundos, lo que equivale a más de dos minutos y medio.

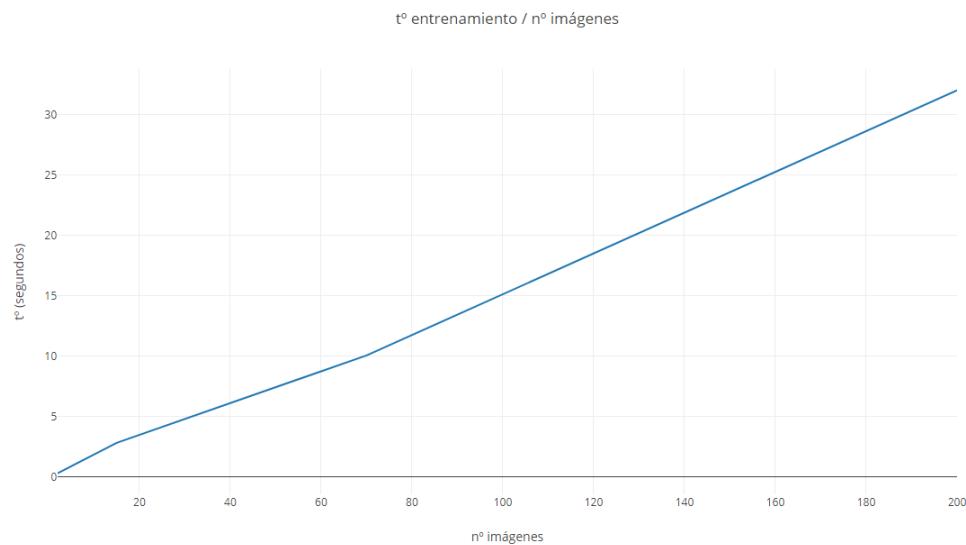


Figura 5.20: Muestreo de los resultados de varios entrenamientos de la red sucesivos en función del número de imágenes.

Base de datos

Al principio la idea era diseñar una base de datos que, aunque sencilla, tuviera algún tipo de persistencia para poder asignar los datos a cada imagen. Se hicieron pruebas con *SQLite* [67], aunque al final se decidió utilizar una carpeta predefinida como sistema de base de datos provisional.

Interfaz

Por último, y dado que necesitábamos información o recursos de apartados anteriores, se desarrolló la interfaz de usuario con *tKinter*. Dicha interfaz ha pasado por muchos rediseños, aunque desde el principio se tenía claro cual sería su función: mostrar imágenes resultado e información sobre las mismas.

Al principio simplemente mostraba ambas imágenes: la capturada y la ocurrencia de la base de datos 5.21, aunque se fueron añadiendo funcionalidades extra, como una barra de progreso 5.26 o un botón que nos permitía abrir una nueva ventana (en forma de pop-up) con información de la imagen 5.31, 5.30.

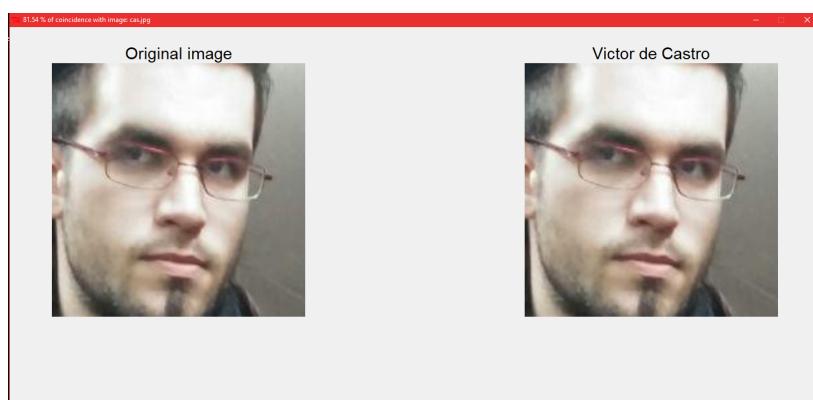


Figura 5.21: Interfaz inicial.

Como se menciona en el apartado *4_Tecnicas_y_herramientas* 4.2, se utilizó el patrón de diseño singleton para evitar problemas de tener abiertas más de una ventana de la interfaz a la vez 5.22.

```

def __init__(self):
    if GUIClass.__instance != None:
        raise Exception("This is a Singleton class. "
                        "Access it through getInstance()")
    else:
        GUIClass.__instance = self

    @staticmethod
def getInstance():
    if GUIClass.__instance == None:
        GUIClass()
    return GUIClass.__instance

```

Figura 5.22: GetInstance de la interfaz GUI.

Además, el desarrollo de la interfaz se basó en una estructura modular, creando primero la ventana principal 5.23, a la cual se la irían añadiendo elementos uno a uno. El método *fixedSize()* nos permite establecer un tamaño fijo para nuestra interfaz (en función del tamaño de la pantalla), eliminando la posibilidad de que el usuario lo modifique manualmente.

```

def createMainWindow(self, title=None, realTime=False):
    windowObject = tk.Tk()
    self.title = title if title is not None else "GUI - Default title"
    windowObject.title(self.title)
    self.window = windowObject
    self.window.config(bg=self.COLORS.BACKGROUNDGENERAL.value)
    self.realTime = realTime
    return windowObject

def fixedSize(self):
    hDisplay, wDisplay = self.getDisplaySize()
    self.width = wDisplay-wDisplay//10
    self.height = hDisplay-hDisplay//10
    self.window.resizable(width=False, height=False)
    self.window.minsize(width=self.width, height=self.height)
    self.window.maxsize(width=self.width, height=self.height)

```

Figura 5.23: Creamos la ventana principal con tamaño fijo.

Primero se añaden las imágenes 5.24 mediante la herramienta *Frame*. Hubo problemas al comienzo ya que la interfaz no aceptaba cierto tipo de imágenes, de manera que se tuvo que hacer una conversión del tipo mediante *Image.fromarray*. Esta sería la primera versión más básica de la interfaz, exclusivamente con las imágenes y nada más, como se puede observar en la imagen 5.21.

```
def createTop_BottomPanel(self, photoFromCamera, photoFromDatabase, p):
    myFrame = tk.Frame(self.window)
    imgCRecolor = cv2.cvtColor(photoFromCamera, cv2.COLOR_BGR2RGB)
    imgCFromArray = Image.fromarray(imgCRecolor)
    imgC = ImageTk.PhotoImage(imgCFromArray)

    myLabelC = tk.Label(myFrame, compound=tk.BOTTOM, image=imgC, text="Original image")
    myLabelC.grid(row=0, column=0, sticky=tk.W)
    myLabelC.image = imgC
    myLabelC.config(font=(self.textFont, self.textSize))
```

Figura 5.24: Añadimos las imágenes a la ventana.

A continuación se añadió la barra de progreso mediante la herramienta *Progressbar*. Como podemos observar en la imagen 5.25, la barra se adapta dinámicamente en función del resultado.

```
progress = ttk.Progressbar(myFrame, orient="horizontal", length=self.width-int(self.width/10),
                           mode="determinate", style="TProgressbar")
progress.grid(row=3, columnspan=2, sticky=tk.S, pady=(self.height/30, 0))
progress["value"] = p
progress["maximum"] = 100
```

Figura 5.25: Añadimos la barra de progreso a la ventana.

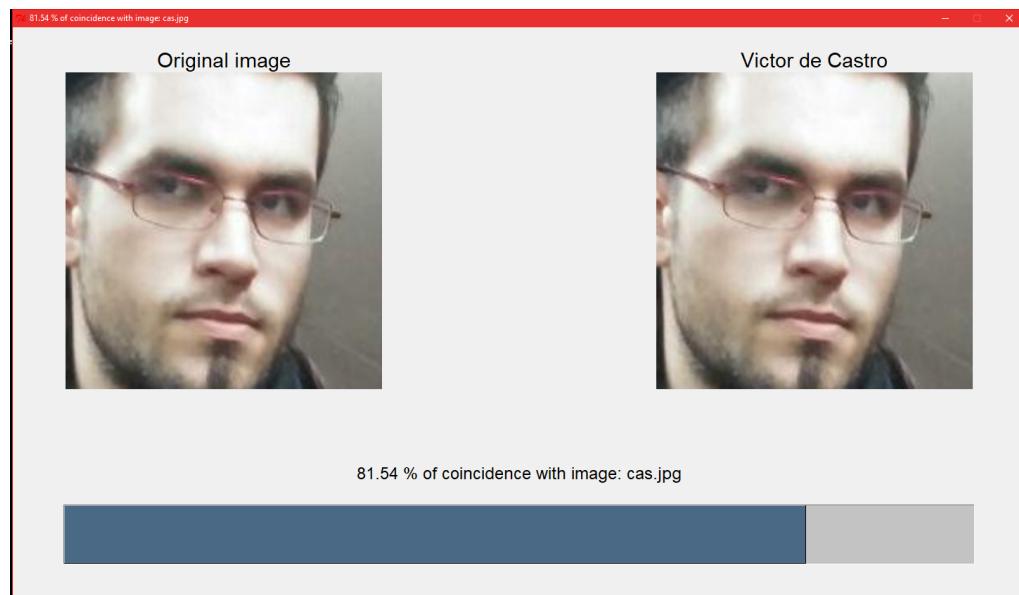


Figura 5.26: Interfaz con barra de progreso.

Y por último el botón de *More Information...* se crea como se muestra en la imagen 5.27. Al pulsar dicho botón, se accede al método *createPopUpInfo*, que crea el pop-up mediante la funcionalidad *TopLevel* 5.28, al cual se le añade la información adecuada 5.29.

```
buttonInfoLabel = tk.Button(myFrame, text="More information...", command=self.createPopUpInfo)
buttonInfoLabel.config(font=(self.textFont, self.textSizeProgressBar))
buttonInfoLabel.grid(row=1, columnspan=2, pady=(self.height/30, 0))
```

Figura 5.27: Botón para crear el pop-up.

```
def createPopUpInfo(self):  
    popup = tk.Toplevel(self.window)  
    popup.resizable(width=False, height=False)  
  
    # Nos permite evitar usar la ventana principal mientras esté esta abierta  
    popup.grab_set()  
  
    backButton = tk.Button(popup, text="Back", command=popup.destroy)  
    backButton.config(font=(self.textFont, self.textSizeProgressBar))  
  
    popup.mainloop()
```

Figura 5.28: Pop-up con la información de la persona.

```
popup.wm_title("More information about image " + str(self.namePhoto) + ".jpg")  
nameL = tk.Label(popup, text="Name: " + self.getInfo("name"), font=(self.textFont, self.textSize))  
ageL = tk.Label(popup, text="Age: " + self.getInfo("age"), font=(self.textFont, self.textSize))  
bpL = tk.Label(popup, text="Birthplace: " + self.getInfo("birth_place"), font=(self.textFont, self.textSize))  
jobL = tk.Label(popup, text="Occupation: " + self.getInfo("job"), font=(self.textFont, self.textSize))
```

Figura 5.29: Información de la persona que se va a mostrar en el pop-up.



Figura 5.30: Pop-up con información.

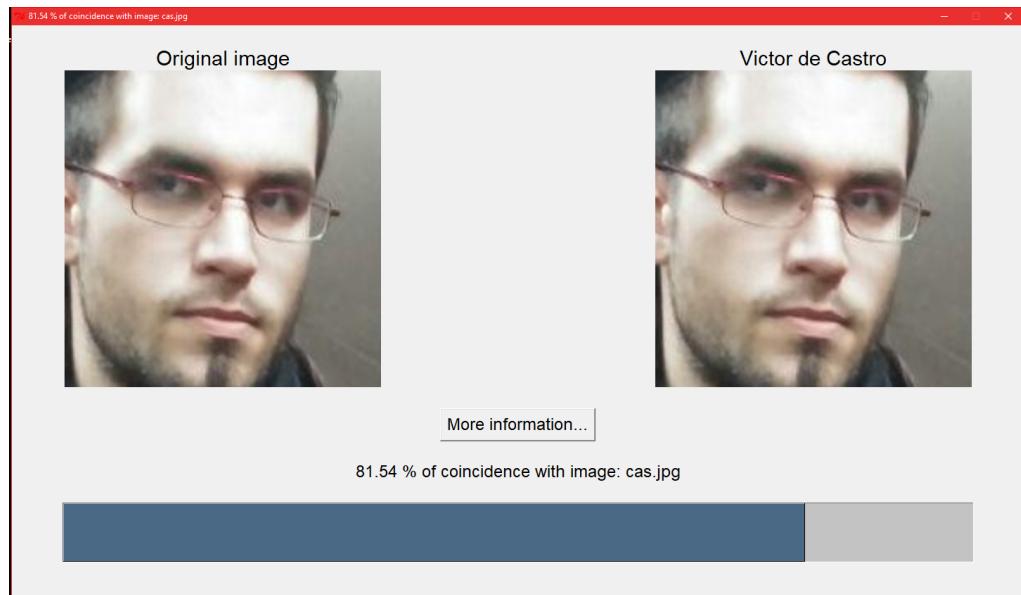


Figura 5.31: Interfaz final.

Al principio, y mientras se desarrollaban las primeras partes del proyecto se utilizaba la interfaz nativa de *OpenCV*, que era simple con casos sencillos como mostrar una única imagen, pero era más complicada de operar para nuestro caso final, al que había que añadir que no tenía suficientes opciones de personalización para nuestro gusto. También se intentó utilizar una especie de interfaz básica proporcionada por *Matplotlib* [33] y otra de *Pillow* [7], pero ninguna nos convenció hasta que encontramos tKinter.

Como elemento adicional a la interfaz principal gráfica (realizada con tKinter), también se muestra información sobre el progreso del programa en la consola de comandos. Esta información es más útil para poder estudiar el comportamiento general del programa por parte del desarrollador que para el usuario medio, sin embargo, este también necesita estar atento a dicha consola de comandos, ya que de vez en cuando se le preguntarán cosas al usuario, como qué desea hacer a continuación, si quiere introducir una nueva imagen en la base de datos 5.33, cargar la imagen desde fichero, desde la cámara o realizar el reconocimiento en tiempo real 5.32, entrenar la red de nuevo 5.34, además de mostrar mensajes de aviso si ha habido algún problema 5.35. Estos casos especiales de comportamiento se verán en el anexo *D_Manual_programador*.

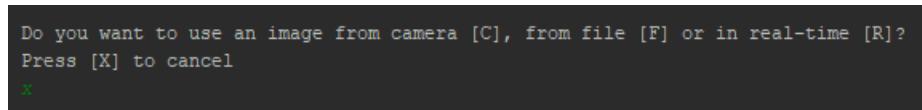


Figura 5.32: Programa preguntándonos si queremos una imagen desde la cámara, desde un fichero o en tiempo real.

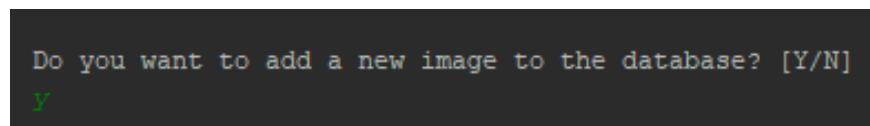


Figura 5.33: Programa preguntándonos si queremos introducir una nueva imagen en la base de datos.

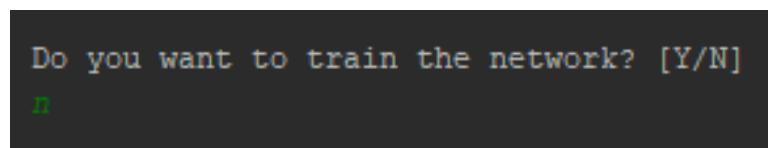


Figura 5.34: Programa preguntándonos si queremos entrenar la red de nuevo.

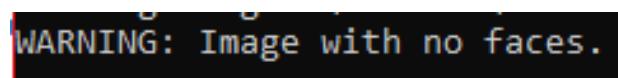


Figura 5.35: Mensaje de aviso de que no hay rostros en la imagen.

Durante esta fase se encontraron varios problemas, algunos de los cuales se pudieron resolver consultando las referencias asociadas en el anexo *D_Manual_programador*.

Como la interfaz final nos parecía un poco falta de vida, se decidió añadir algo de color, para ello se consultaron las tablas de color proporcionadas por *Rapid Tables* [56] y un manual on-line sencillo sobre personalización de botones y elementos visuales [31].

Además, la barra que indica el tanto por ciento de coincidencia cambia de color en función de dicho porcentaje. En la siguiente imagen 5.36 se puede observar el resultado final.



Figura 5.36: Interfaz final con color añadido y otras mejoras en elementos visuales (bordes y profundidad en botones y barra de progreso).

Además de utilizar esta interfaz gráfica (GUI) para mostrar resultados, se ha utilizado también para encontrar la imagen que queremos leer desde fichero en el sistema. En dicha interfaz se pueden filtrar los resultados por ficheros *.png* y *.jpg* para facilitar la búsqueda.

Se puede observar dicha interfaz en el anexo *E_Manual_usuario: 4.1.- Obtener la imagen desde fichero*.

Revisión y control continuo

Con el proyecto en *Github*, ha sido sencillo seguir el progreso y desarrollo del mismo, sabiendo en todo momento qué tareas estábamos realizando, cuales nos quedaban por realizar, cuales eran los objetivos generales, etc.

Además de las tareas principales descritas en la tabla *Distribución del desarrollo del proyecto según Milestones e Issues* del anexo A_Plan_proyecto, también fueron surgiendo tareas secundarias, como mejorar algún método que faltaba de pulir, corregir errores o bugs, etc.

A la hora de editar y formatear el fichero readme.md de github, se han utilizado herramientas y recursos varios [22].

Release

Se intentó crear un ejecutable con todas las librerías, referencias y dependencias con la herramienta py2exe[47], la herramienta pyinstaller [9], usando un .bat [18] y mediante otras soluciones [44], aunque finalmente se optó por no crear ejecutable como tal (comprimido con todos los recursos), sino un .bat que ejecute la sección principal de nuestro programa (*Main.py*), de manera que hace falta tener las librerías instaladas 5.2.



Trabajos relacionados

Algunos de los proyectos relacionados con el presente que se han consultado han sido:

- TFG: *Sistema de Reconocimiento Facial*, por Germán Matías Scarel [63].
- TFG: *Reconocimiento facial mediante el Análisis de Componentes Principales*, por Sara Domínguez Pavó [14].
- TFG: *Reconocimiento de imágenes utilizando redes neuronales artificiales*, por Pedro Pablo García García [19].
- TFG: *Interfaces Hombre Máquina Basados en Hardware Libre*, por José Ramón Cuevas Diez [10].
- TFG: *GoBees - Monitorización del estado de una colmena mediante la cámara de un smartphone*, por David Miguel Lozano [42].
- TFG: *Reconocimiento de señales de tráfico mediante Raspberry Pi*, por Alberto Miguel Tobar [72].
- TFG: *Visión Artificial Aplicada a la Clasificación basada en Color*, por María Viyuela Fernández [74].
- *How to train a Tensorflow face object detection model*, por Dion van Velde [73].
- *Deep Learning CNN's in Tensorflow with GPUs*, por Cole Murray [49].
- *Detecting facial features using Deep Learning*, por Peter Skvarenina [66].

Otros repositorios relacionados:

- Clasificación de imágenes usando Tensorflow [86].
- Detección de rostros con la api de Tensorflow [55], [52].
- Google cloud platform: Computer Vision [27].
- Microsoft Cognitive Toolkit [46].
- COCO: reconocimiento de objetos a gran escala [8].
- Facenet: Face recognition using Tensorflow [12].
- Face recognition: The world's simplest facial recognition api for Python and the command line [3].
- Red neuronal desarrollada en python con numpy [28].
- Tutoriales sobre computer vision [62].

Conclusiones y Líneas de trabajo futuras

7.1. Posibilidades de mejora del proyecto

Base de datos

Ahora mismo hay una cantidad de imágenes en la base de datos suficiente para testear y probar el funcionamiento del programa a nuestro nivel de desarrollo, pero eso en un futuro habría que ampliarlo, dependiendo de la gente que queramos reconocer. No se han querido meter demasiadas imágenes para no tener una carpeta que ocupe demasiado solo con imágenes de gente. Por supuesto, para una aplicación real tendría que tener tantas como se tengan almacenadas en el registro correspondiente.

El formato de dichas imágenes es muy irregular, cada una está tomada desde un ángulo diferente, con una calidad y tamaño variables, debido a que están tomadas de internet, y a partir de estas fotos aleatorias de internet se han guardado en otra carpeta la parte exclusiva del rostro, descartando el resto de la imagen, haciendo que todas tengan el mismo tamaño y una calidad (salvando las distancias) similar. Por supuesto, van a seguir estando cada una tomada desde un ángulo y con una influencia de iluminación y color variable.

Este aspecto, si se tuviera la posibilidad (como en el registro que comentaba antes), lo mejor sería que todas tuvieran el mismo formato (sacadas del registro del dni, registro en instituciones públicas, anteriores condenas, etc), pero estamos hablando de un entorno de prueba, en el cual se ha desarrollado el proyecto, por lo que es complicado obtener un standard a partir de imágenes aleatorias obtenidas de internet.

Una posible solución para esto podría haber sido intentar conseguir un foco, una cámara de fotos profesional (o de mayor calidad que una del móvil o una webcam del ordenador), y preguntar a amigos y conocidos que se pusieran ropa similar (camiseta oscura) sobre un fondo claro para obtener los mejores resultados posibles. Esto requeriría de mucho tiempo y recursos (de los cuales no se disponía), y tiene más importancia a la hora de poner en marcha de manera más comercial el proyecto.

Entrenamiento

En el estado de desarrollo en el que se encuentra el proyecto, cada vez que se introduzca una imagen nueva, se tiene que entrenar la red, en caso contrario se trabajará sobre las imágenes de la base de datos sin actualizar, de manera que se pueden dar lecturas fantasma o falsas identificaciones.

Esto se debe a que, cuando se entrena la red, lo que se hace realmente es extraer las características del rostro de cada una de las imágenes y se guardan en un fichero, por lo que si alguna imagen cambia o se añaden o eliminan es necesario repetir el proceso para recoger los nuevos datos.

Esto podría adaptarse si se quiere conseguir una mejora en el rendimiento cuando se insertan imágenes a menudo, poniendo las funcionalidades de entrenamiento de la red en un programa separado del principal, de manera que, al insertar una nueva imagen, se ejecute solamente ese programa, y al ejecutar el programa principal ya se ha entrenado la red previamente, por lo que tiene que hacer simplemente es el reconocimiento propiamente dicho, sin perder tiempo ni recursos en entrenar y extraer las características.

Otra opción sería extraer las características de cada imagen por separado, de manera que no se debería entrenar la red con todas las imágenes cada vez que se introduce una, sino entrenar la red para la nueva imagen. El problema con este último método, es que necesitaríamos "desentrenar" la red para las imágenes que elimináramos, de manera que tendríamos que llevar un registro de las imágenes antiguas y de las nuevas, algo que, en una fase inicial del proyecto se descartó, ya que implicaría desarrollar una base de datos compleja, para la que no se tenía tiempo.

Un ejemplo del problema anterior es el siguiente:

Si hay una imagen que se llama *persona1.png* para una persona [1], se crea el fichero *entrenamiento_persona1.yml*, después de lo cual se borra la imagen y se introduce otra imagen, de otra persona diferente (persona [2]), con el mismo nombre de fichero *persona1.png*. Al ir a comprobar el control de versiones mencionado en el apartado anterior, nos va a decir que si que se ha entrenado ya esa imagen, pero realmente el fichero *entrenamiento_persona1.yml* lo que tiene es el entrenamiento de la persona [1] que ya no está en las imágenes, así que volvería a dar errores y falsos positivos.

Reconocimiento a gran escala

Ahora mismo las dos opciones que tenemos de reconocer a alguien están muy limitadas por las características de nuestro equipo, de manera que habría que acudir a *clusters* o *servers* de empresas para poder alcanzar el máximo rendimiento de la aplicación.

Una de las opciones actuales es reconocer a alguien en una sola imagen, sin tiempo real, bien siendo esta una imagen obtenida desde un fichero o uno de los frames que hemos capturado con la cámara, pero sigue siendo una sola imagen, de modo que tiene tiempo para compararla con la base de datos y mostrar los resultados. La otra opción es comparar los *frames* que va capturando la cámara, tanto si queremos como si no, con la base de datos, pero esto es muy costoso, de manera que se limita el número de frames por segundo que puede capturar nuestra imagen, para que le de tiempo a obtener los resultados y mostrarlos antes de capturar el siguiente frame. Esta última opción habría que sustituirla por una similar, pero sin limitar el número de frames que puede captar la cámara, aumentando su eficacia.

Otra posible mejora de cara al futuro podría ser reconocer varios rostros en la misma imagen, acercándolo un poco más a esa idea original que se tenía del proyecto. Por supuesto, esto dependería del lugar en el que se encuentre la cámara, pero en un sitio con mucha gente, seguramente tenga que estar analizando simultáneamente hasta cien rostros, por lo que volvemos de nuevo a los costes de un material más potente.

Como ya se comentó en la sección *1_Introducción: Aplicación Real a Gran Escala* 1.3, este sistema de reconocimiento a **gran escala** y en **tiempo real** (recordemos que está recopilando información de todas las cámaras disponibles en la ciudad, a la vez que está obteniendo las características de los rostros que encuentra en cada una de ellas, comparando dichas características con las obtenidas al entrenar previamente la red), tiene un **alto coste computacional** y de recursos, de manera que sería necesario ejecutar el programa principal en un servidor con unos requisitos técnicos muy altos, además de una infraestructura adecuada para poder transmitir toda esa información sin retrasos (lo que implica, de nuevo, unos altos requisitos técnicos).

¿Es viable?

Esta pregunta es un poco ambigua, y vamos a entrar más en detalle en el anexo *A_Plan_proyecto*, pero la respuesta breve es SI en el ámbito tecnológico y DEPENDE en el ámbito moral, legal y jurídico.

Es viable **tecnológicamente** si se tiene un *sistema distribuido* [78] en el cual se tenga un servidor (que sea capaz de realizar procesamiento en paralelo [81]) donde se realicen todas las operaciones de comparación, predicción, muestreo de resultados, etc., y varios clientes, concretamente uno por cada cámara (o zona de cámaras), que se encargarán de obtener las imágenes de la calle y enviárselas al servidor junto con información de qué cámara ha realizado esa captura, en qué zona está, a qué hora, etc. Como podemos observar, ya se tiene una estructura similar hoy en día, de modo que se podría aprovechar ese sistema, ahorrando costes.

En cuanto al **apartado legal**, es una cuestión complicada, ya que, depende de países, unos tienen unas normas y leyes de privacidad más o menos estrictas. En el caso de España, ya se controla el acceso a determinados lugares públicos como aeropuertos, de manera que sería extender ese control y mejorarlo.

Una cuestión que mucha gente no está dispuesta a dejar pasar es el **riesgo moral**: ¿merece la pena la pérdida de privacidad por un poco más de seguridad? Desde luego, es una cuestión muy importante, pero es demasiado compleja como para resolverlo aquí, ya que cada uno tendrá su opinión y tampoco es el objetivo que persigue este proyecto.

7.2. Conclusiones

Con este proyecto se han logrado cumplir los objetivos impuestos al principio del desarrollo: a partir de una imagen, reconocer un rostro y saber si pertenece a alguna de las personas almacenadas en nuestra base de datos, además de mostrar el rostro almacenado con mayor coincidencia con el primero en una interfaz sencilla.

Por supuesto, hay algunas cosas mejorables, como hemos comentado en los apartados anteriores: En cuanto a la gestión de la base de datos, se podría haber implementado un sistema sencillo de persistencia que almacenase la información de cada persona ligada a su imagen. El tipo de entrenamiento utilizado o las herramientas necesarias para ello (Tensorflow como era la idea inicial) podría haber variado para adquirir más especialización en nuestro proyecto, pudiendo haber obtenido mejores resultados.

Además de esos objetivos generales (completados con pequeñas modificaciones), se han conseguido completar los objetivos técnicos que eran un pequeño desglose de los anteriores: Obtener, manipular y guardar imágenes. Reconocer un rostro dentro de la imagen utilizando las principales características de la cara. Entrenar una red mediante técnicas de machine-learning haciendo que se reconozca el rostro de la persona de la imagen capturada a partir de los resultados de la red entrenada, comparando características de rostros. Crear una interfaz adecuada para mostrar los resultados finales.

Por último, también se han aplicado varios de los conocimientos obtenidos durante la carrera y nos hemos familiarizado más con el lenguaje de programación Python. Hemos conseguido utilizar y conocer de primera lo último en tecnología: el machine-learning. Como reto personal, he conseguido desarrollar una interfaz para una aplicación en Python. Sencilla, pero que me ha permitido ir profundizando poco a poco en la parte más visual de nuestro programa.

Se han obtenido buenos resultados en entornos controlados [7.41](#), aunque en lugares con mucha variación de luz (lugares públicos) habría que reconsiderar la utilización de otras herramientas. Se añaden a continuación algunas imágenes de resultados de ejecución favorables obtenidos.

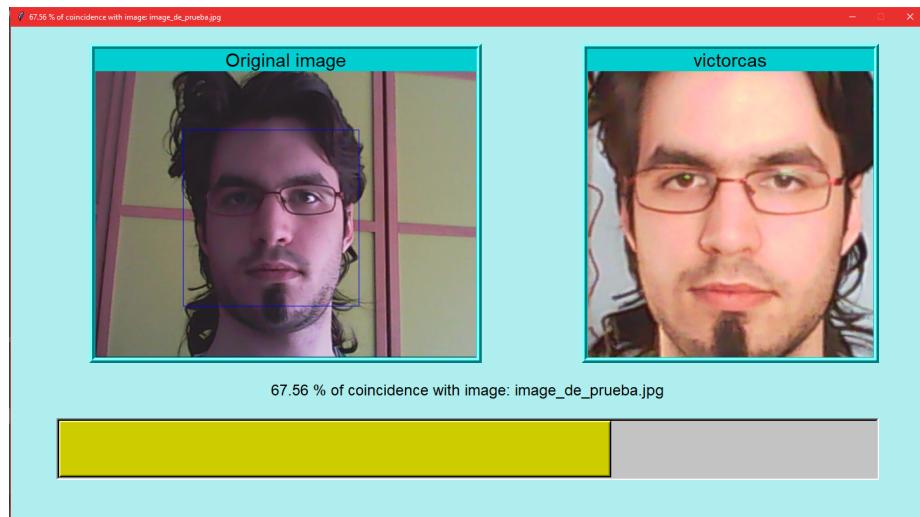


Figura 7.37: Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: mala iluminación).

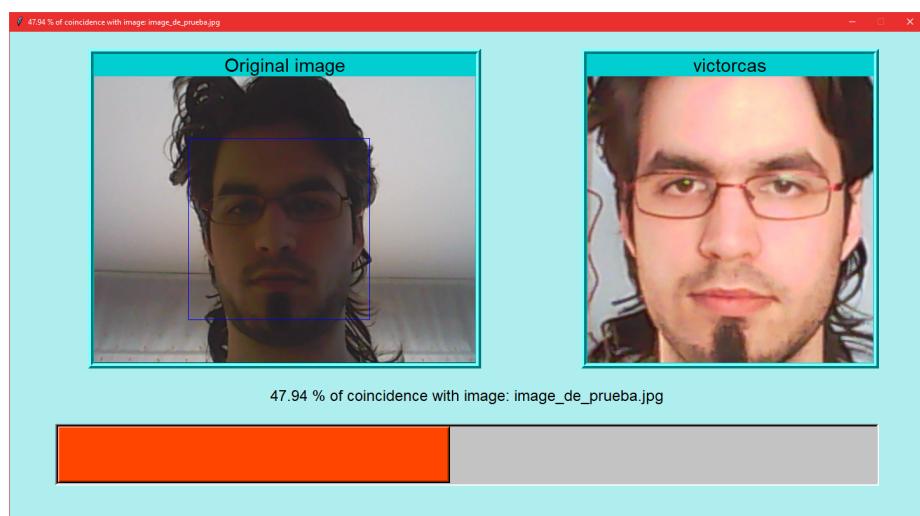


Figura 7.38: Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: muy mala iluminación).

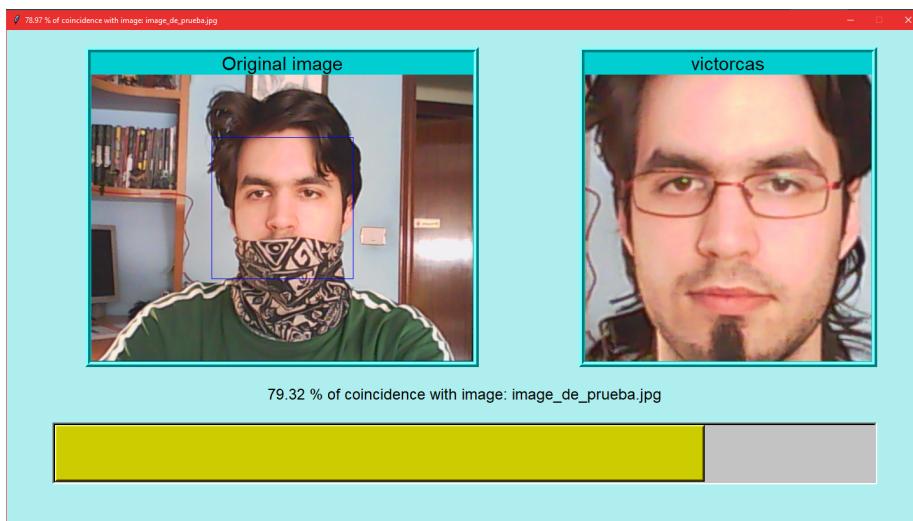


Figura 7.39: Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: solo visibles los ojos).

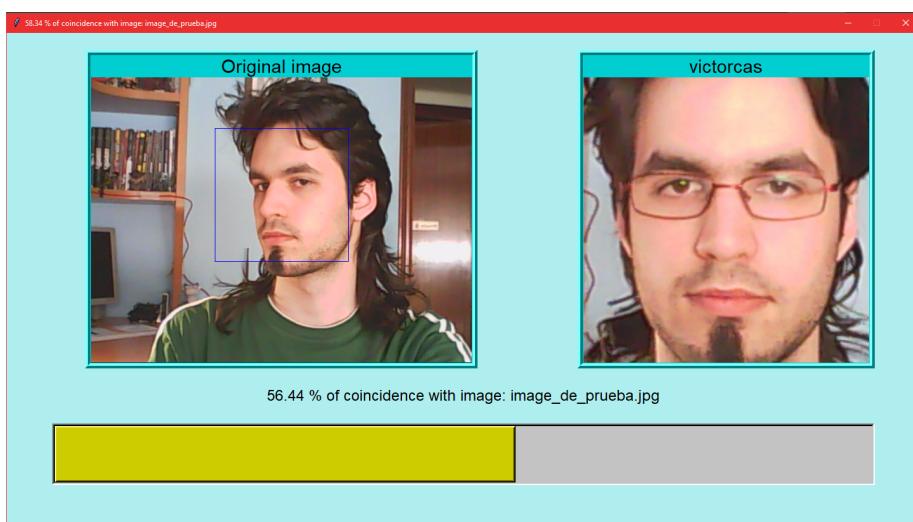


Figura 7.40: Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: foto de casi perfil).

Como se puede observar, el elemento más influyente es la iluminación [7.37](#), pudiendo bajar hasta un 30 % el valor de confianza de una imagen que en otro entorno sería perfectamente válida [7.38](#).

A continuación, tenemos los ojos como elemento predominante sobre el resto, ya que, haciendo pruebas, si se tapaban los ojos completamente no nos reconocía con nadie de la base de datos y nos mostraba la imagen por defecto. Sin embargo, dejando los ojos descubiertos y tapando el resto (como en la imagen [7.39](#)), seguimos obteniendo unos resultados aceptables.

También está el factor de que la persona debe estar de frente a la cámara, ya que si se encuentra de lado se obtienen unos resultados bastante bajos [7.40](#), llegando incluso a no obtenerse ningún resultado satisfactorio si la persona se coloca totalmente de lado.

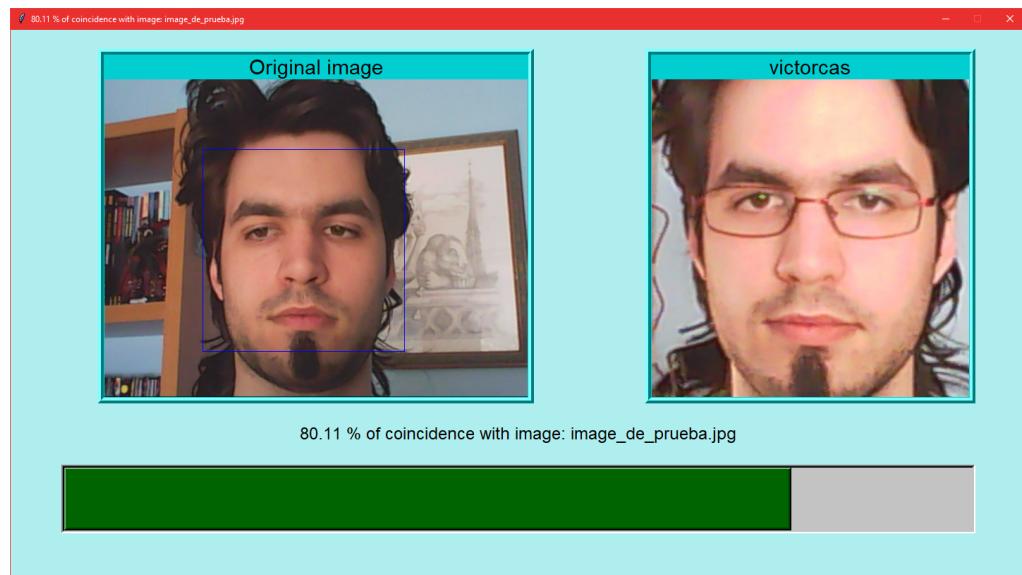


Figura 7.41: Comparación entre una imagen guardada en la base de datos, y otra desde la cámara en tiempo real (rasgo distintivo: sin gafas).

Bibliografía

- [1] OpenCV Version 2.4.13. Haar feature-based cascade classifier for object detection. https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html, 2018.
- [2] OpenCV Version 3.4.1. Face detection using haar cascades. https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html, 2018.
- [3] ageitgey. face_recognition. https://github.com/ageitgey/face_recognition, 2018.
- [4] Ryan Allred. Image augmentation for deep learning using keras and histogram equalization. <https://towardsdatascience.com/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085>, 2017.
- [5] Inc Anaconda. Anaconda. <https://anaconda.org/anaconda/python>, 2018.
- [6] The Digital Bridges. How google cloud vision api uses machine learning to build metadata. <http://www.thedigitalbridges.com/google-cloud-vision-api-build-metadata>, 2017.
- [7] Alex Clark and Contributors. Pillow. <https://pillow.readthedocs.io/en/5.1.x/>, 2018.
- [8] cocodataset. Coco: Common objects in context. <http://cocodataset.org/#home>, 2018.

- [9] David Cortesi, Giovanni Bajo, William Caban, and Gordon McMillan. Pyinstaller manual - version 3.3.1. <http://pyinstaller.readthedocs.io/en/v3.3.1/>.
- [10] José Ramón Cuevas Diez. *Interfaces Hombre Máquina Basados en Hardware Libre*. Universidad de Burgos, Burgos, España, 2017.
- [11] Melissa Dale. Python: Tensorflow on windows 10 in pycharm. <https://www.youtube.com/watch?v=83vR1Nz3dHA>, 2017.
- [12] davidsandberg. facenet. <https://github.com/davidsandberg/facenet>, 2018.
- [13] Google Developers. Image recognition. https://www.tensorflow.org/tutorials/image_recognition, 2018.
- [14] Sara Domínguez Pavón. *Reconocimiento facial mediante el Análisis de Componentes Principales*. Universidad de Sevilla, Sevilla, España, 2017. http://bibing.us.es/proyectos/abreproj/91426/fichero/TFG_SARA_DOMINGUEZ_PAVON.pdf.
- [15] Jan Fajfr. The basics of face recognition. <https://blog.octo.com/en/basics-face-recognition>, 2011.
- [16] Python Software Foundation. Python - version 2.7. <https://www.python.org/downloads/release/python-2715>, 2018.
- [17] Python Software Foundation. Tkinter — python interface to tcl/tk. <https://docs.python.org/2/library/tkinter.html>, 2018.
- [18] FrustratedWithFormsDesigner and Alvin SIU. Bat file to open cmd in current directory. <https://stackoverflow.com/questions/4451668/bat-file-to-open-cmd-in-current-directory/23633328>, 2010.
- [19] Pedro Pablo García García. *Reconocimiento de imágenes utilizando redes neuronales artificiales*. Universidad Complutense de Madrid, Madrid, España, 2013. <http://eprints.ucm.es/23444/1/ProyectoFinMasterPedroPablo.pdf>.
- [20] Adam Geitgey. Machine learning is fun! part 4: Modern face recognition with deep learning. <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>, 2016.

- [21] Adam Geitgey. Machine learning is fun! part 4: Modern face recognition with deep learning. <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>, 2016.
- [22] github. Basic writing and formatting syntax. <https://help.github.com/articles/basic-writing-and-formatting-syntax/#styling-text>, 2018.
- [23] Github. Github. <https://github.com/>, 2018.
- [24] Github. Github desktop. <https://desktop.github.com/>, 2018.
- [25] Google. Google cloud vision api documentation. <https://cloud.google.com/vision/docs>.
- [26] Google. Google colab. <https://colab.research.google.com/notebooks/welcome.ipynb>, 2018.
- [27] GoogleCloudPlatform. cloud-vision. <https://github.com/GoogleCloudPlatform/cloud-vision>, 2018.
- [28] greydanus. pythonic_ocr. https://github.com/greydanus/pythonic_ocr, 2017.
- [29] Bharath Hariharan. How are eigenvectors and eigenvalues used in image processing? <https://www.quora.com/How-are-Eigenvectors-and-Eigenvalues-used-in-image-processing>, 2013.
- [30] Stack Exchange Inc. Stackoverflow. <https://stackoverflow.com>, 2018.
- [31] java2s.com. Button border : Button - gui tk - python. <http://www.java2s.com/Code/Python/GUI-Tk/ButtonBorder.htm>.
- [32] JetBrains. Pycharm. <https://www.jetbrains.com/pycharm>, 2018.
- [33] Eric Firing Michael Droettboom John Hunter, Darren Dale and the Matplotlib development team. Matplotlib - version 2.2.2. <https://matplotlib.org>, 2012 - 2018.
- [34] Alexander Jung. imgaug. <https://github.com/aleju/imgaug>, 2018.

- [35] Andrej Karpathy Justin Johnson. Convolutional neural networks for visual recognition. <http://cs231n.github.io/convolutional-networks>, 2018.
- [36] kaboomfox. Overlay a smaller image on a larger image python opencv. <https://stackoverflow.com/questions/14063070/overlay-a-smaller-image-on-a-larger-image-python-opencv>, 2012.
- [37] KP Kaiser. Deal with it in python with face detection. <https://dev.to/burnington/deal-with-it-in-python-with-face-detection-chi>, 2017.
- [38] Davis King. Dlib 18.7 released: Make your own object detector in python! <http://blog.dlib.net/2014/04/dlib-187-released-make-your-own-object.html>, 2014.
- [39] LearningTensorFlow.com. Beginner-level tutorials for a powerful framework. <https://learningtensorflow.com>, 2016.
- [40] Antonio Leiva. Patrones de diseño de software. <https://devexperto.com/patrones-de-diseno-software/>, 2016.
- [41] Logitech. Webcam c170 - plug-and-play video calls. <https://www.logitech.com/en-gb/product/webcam-c170>, 2018.
- [42] David Miguel Lozano. *GoBees - Monitorización del estado de una colmena mediante la cámara de un smartphone*. Universidad de Burgos, Burgos, España, 2017.
- [43] JGraph Ltd. Draw-io. <https://www.draw.io>, 2018.
- [44] mamcx, Caleb Hattingh, Jason Baker, and Eli Bendersky. How to deploy python to windows users? <https://stackoverflow.com/questions/1646326/how-to-deploy-python-to-windows-users>, 2009.
- [45] A. Martinez. Fisherfaces. *Scholarpedia*, 6(2):4282, 2011. revision #91266.
- [46] Microsoft. Microsoft cognitive toolkit - cntk. <https://github.com/Microsoft/CNTK>, 2018.
- [47] MikeFox. python: py2exe tutorial. <http://www.py2exe.org/index.cgi/Tutorial>, 2014.

- [48] Ltd MKLab Co. Staruml. <http://staruml.io>, 2018.
- [49] Cole Murray. Deep learning cnn's in tensorflow with gpus. <https://hackernoon.com/deep-learning-cnns-in-tensorflow-with-gpus-cba6efe0acc2>, 2017.
- [50] OpenCV. haarcascades. <https://github.com/opencv/opencv/blob/master/data/haarcascades>, 2018.
- [51] OpenCV. Opencv - releases. <https://opencv.org/releases.html>, 2018.
- [52] Rohit Patil. How to use tensorflow object detection api on windows. <https://medium.com/@rohitrpatal/how-to-use-tensorflow-object-detection-api-on-windows-\102ec8097699>, 2018.
- [53] Christian S. Perone. Deep learning – convolutional neural networks and feature extraction with python. <http://blog.christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-python>, 2015.
- [54] Plotly. Modern visualization for the data era: Plotly. <https://plot.ly>, 2017.
- [55] qdraw. tensorflow-face-object-detector-tutorial. <https://github.com/qdraw/tensorflow-face-object-detector-tutorial>, 2017.
- [56] RapidTables.com. Rgb color codes chart. https://www.rapidtables.com/web/color/RGB_Color.html, 2018.
- [57] Adrian Rosebrock. Hog detectmultiscale parameters explained. <https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained>, 2015.
- [58] Margaret Rouse. Inteligencia artificial, o ai. <https://searchdatacenter.techtarget.com/es/definicion/Inteligencia-artificial-o-AI>, 2017.
- [59] Cesar Troya S. Lbp y ulbp – local binary patterns y uniform local binary patterns. <https://cesartroyasherdek.wordpress.com/2016/02/26/deteccion-de-objetos-vi/>, 2016.

- [60] ANKIT SACHAN. Convolutional neural network (cnn). <http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification>, 2017.
- [61] Kelvin Salton. Face recognition: Understanding lbph algorithm. <https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b>, 2017.
- [62] sankit1. cv-tricks.com. <https://github.com/sankit1/cv-tricks.com>, 2018.
- [63] Germán Matías Scarel. *Sistema de Reconocimiento Facial*. Universidad Nacional del Litoral, Santa Fe, Argentina, 2010. http://pdi-fich.wdfiles.com/local--files/investigacion/PF_Scarel_SistemaReconocimientoFacial.pdf.
- [64] ShareLaTeX. Sharelatex. <https://es.sharelatex.com>, 2018.
- [65] sinnaps. Metodología de un proyecto. <https://www.sinnaps.com/blog-gestion-proyectos/metodologia-de-un-proyecto>, 2018.
- [66] Peter Skvarenina. Detecting facial features using deep learning. <https://towardsdatascience.com/detecting-facial-features-using-deep-learning-2e23c8660a7a>, 2017.
- [67] sqlitebrowser. Db browser for sqlite. <http://sqlitebrowser.org/>, 2017.
- [68] Princeton University Stanford Vision Lab, Stanford University. Image-net. <http://www.image-net.org/index>, 2016.
- [69] Keras Team. Keras: The python deep learning library. <https://keras.io>, 2018.
- [70] Tensorflow. Tensorflow. <https://www.tensorflow.org>, 2018.
- [71] tensorflow. tensorflow. <https://github.com/tensorflow/tensorflow>, 2018.
- [72] Alberto Miguel Tobar. *Reconocimiento de señales de tráfico mediante Raspberry Pi*. Universidad de Burgos, Burgos, España, 2017.

- [73] Dion van Velde. How to train a tensorflow face object detection model. <https://towardsdatascience.com/how-to-train-a-tensorflow-face-object-detection-model-\3599dcd0c26f>, 2017.
- [74] María Viyuela Fernández. *Visión Artificial Aplicada a la Clasificación basada en Color*. Universidad de Burgos, Burgos, España, 2016.
- [75] Chris Welch. Google just gave a stunning demo of assistant making an actual phone call. <https://www.theverge.com/2018/5/8/17332070/google-assistant-makes-phone-call-demo-duplex-io-2018>, 2018.
- [76] Wikipedia. Computer vision — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [77] Wikipedia. Desarrollo en cascada — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [78] Wikipedia. Distributed computing, 2018. [Internet; descargado 10-junio-2018].
- [79] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [80] Wikipedia. Multilayer perceptron — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [81] Wikipedia. Parallel computing, 2018. [Internet; descargado 10-junio-2018].
- [82] Wikipedia. Red neuronal artificial — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [83] Wikipedia. Redes neuronales convolucionales — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [84] Wikipedia. Scrum (desarrollo de software) — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [85] Wikipedia. Vector propio y valor propio — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].
- [86] wolfib. Python.gitignore. <https://github.com/wolfib/image-classification-CIFAR10-tf>, 2017.

- [87] S. Zhang and M. Turk. Eigenfaces. *Scholarpedia*, 3(9):4244, 2008.
revision #128015.

