UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un sistema de reconocimiento facial utilizando *Deep Learning* con OpenCV

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Francisco Morcillo Vizuete

Tutor: Manuel Agustí Melchor

Curso 2019-2020

Resumen

El reconocimiento facial ha existido durante décadas, pero ha evolucionado rápidamente con la incorporación de técnicas de *Deep Learning* y su uso se ha vuelto más notable y accesible en los últimos años, ya que ahora se utiliza con soluciones innovadoras, como aplicaciones de fotos personales, autenticación en dispositivos móviles, etc.

El objetivo de este trabajo es explorar y desarrollar un sistema de reconocimiento facial utilizando la biblioteca de visión por computador y aprendizaje automático, OpenCV, a partir de un programa básico de reconocimiento facial que utiliza la biblioteca Face Recognition.

El sistema de reconocimiento facial desarrollado permite administrar (añadir o eliminar) los usuarios registrados en el sistema, la persistencia de la información de los mismos y la detección de personas con vida frente a artefactos inanimados. Dispone de tres modos de uso: usuario, administrador y desarrollador.

Finalmente, se realizan pruebas de rendimiento del sistema de reconocimiento facial, tanto de tiempos de ejecución como de detección e identificación de caras y se muestran y analizan los resultados.

Palabras clave: *Deep Learning*, detección de *liveness*, Inteligencia Artificial, OpenCV, reconocimiento facial, visión por computador

Resum

El reconeixement facial ha existit durant dècades, però ha evolucionat ràpidament amb la incorporació de tècniques de *Deep Learning* i el seu ús s'ha tornat més notable i accessible en els darrers anys, ja que ara s'utilitza amb solucions innovadores, com aplicacions de fotos personals, autenticació en dispositius mòbils, etc.

L'objectiu d'aquest treball és explorar i desenvolupar un sistema de reconeixement facial utilitzant la biblioteca de visió per computador i aprenentatge automàtic, OpenCV, a partir d'un programa bàsic de reconeixement facial que utilitza la biblioteca Face Recognition.

El sistema de reconeixement facial desenvolupat permet administrar (afegir o eliminar) els usuaris registrats al sistema, la persistència de la informació dels mateixos i la detecció de persones amb vida davant artefactes inanimats. Disposa de tres maneres d'ús: usuari, administrador i desenvolupador.

Finalment, es realitzen proves de rendiment del sistema de reconeixement facial, tant de temps d'execució com de detecció i identificació de cares i es mostren i analitzen els resultats.

Paraules clau: *Deep Learning*, detecció de *liveness*, Intel·ligència Artificial, OpenCV, reconeixement facial, visió per computador

Abstract

Facial recognition has been around for decades, but it has evolved rapidly with the incorporation of Deep Learning techniques and its use has become more remarkable and accessible in the past few years as it is now powers innovative solutions such as personal photo applications, authentication on mobile devices, etc.

The objective of this work is to explore and develop a facial recognition system using the computer vision and machine learning library, OpenCV, from a basic facial recognition program that uses the Face Recognition library.

The developed facial recognition system allows to manage (add or delete) the registered users in the system, the persistence of their information and the liveness detection. It has three modes of use: user, administrator and developer.

Finally, performance tests of the face recognition system evaluate execution times and face detection and identification, and then, the results are displayed and analyzed.

Key words: Artificial Intelligence, computer vision, Deep Learning, face recognition, liveness detection, OpenCV

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Las aplicaciones de reconocimiento biométrico juegan un papel cada vez más importante en los sistemas comerciales, administrativos y de seguridad modernos. El reconocimiento facial es el campo de la biometría que ha atraído un mayor interés durante los últimos años en la mayoría de países, ya que proporciona un medio discreto y no intrusivo de detección, identificación y verificación.

La incorporación de técnicas de aprendizaje profundo o *Deep Learning*, que usan una cascada de múltiples capas de unidades de procesamiento para la extracción y transformación de características faciales, ha contribuido a la rápida evolución de esta tecnología.

Actualmente, los sistemas de reconocimiento facial están presentes en aplicaciones como la seguridad aeroportuaria y han sido ampliamente adoptados por las agencias de aplicación de la ley, debido a la mejora de la precisión y la capacidad de implementación de los sistemas y al tamaño cada vez mayor de las bases de datos faciales [58]. Como consecuencia de la capacidad probada de los sistemas basados en redes neuronales profundas para superar el desempeño humano en tareas de verificación facial, se prevé que el sector gubernamental internacional sea el mayor usuario de sistemas de reconocimiento facial, durante los próximos años. Las tecnologías de reconocimiento facial también están a la vanguardia de una amplia gama de aplicaciones y dispositivos de consumo, desde tareas de verificación de usuarios que permiten el acceso a cuentas, hasta aplicaciones de cámaras y álbumes de fotos digitales, y etiquetado de redes sociales. Los consumidores y la industria requieren aplicaciones rápidas, asequibles y eficientes para satisfacer la demanda en las funciones comerciales, laborales y educativas, que se extienden al monitoreo de los empleados, pasar lista y seguridad, reducir los costos administrativos y la eficiencia de los procedimientos.

En este trabajo se presenta el diseño y desarrollo de un sistema de reconocimiento facial por computador, utilizando la biblioteca de visión por computador y de aprendizaje automático, OpenCV, tomando como base un programa elemental de reconocimiento facial que usa la biblioteca Face Recognition. El sistema desarrollado es capaz de determinar si está interactuando con un ser humano físicamente presente y no con un artefacto inanimado, además de permitir la administración y la persistencia de los usuarios registrados.

1.1 Motivación

Actualmente, hay una creciente inversión en inteligencia artificial por parte de empresas y gobiernos para conseguir una mayor eficiencia en sus actividades o fines.

Desde hace mucho tiempo me ha llamado la atención la Inteligencia Artificial, gracias, por ejemplo, a películas como las de Iron Man (J.A.R.V.I.S.). La asignatura de Sistemas Inteligentes cursada en el grado de Ingeniería Informática despertó en mí un mayor interés por este tema. Dentro del campo de la Inteligencia Artificial, he escogido la temática del reconocimiento facial porque me impresionan aplicaciones como Google Fotos, capaz de detectar y reconocer objetos y personas en fotografías con mucha precisión, y como el Face Id implementado en los iPhone, capaz de desbloquear dispositivos identificando caras incluso a oscuras usando un proyector de luz infrarroja. Me interesaba aprender sobre el tema y entender cómo funcionan las aplicaciones que usan esta tecnología. Además, considero que adquirir conocimientos relacionados con la Inteligencia Artificial puede abrirme puertas en mi carrera profesional.

1.2 Objetivos

El principal objetivo de este trabajo es desarrollar un sistema de reconocimiento facial. Este objetivo global se descompone en los siguientes objetivos:

- detectar e identificar caras de personas (reconocimiento facial);
- añadir y eliminar usuarios del sistema de reconocimiento facial (administración de usuarios);
- implementar tres modos de uso del sistema de reconocimento facial (administración de usuarios);
- permitir la persistencia de la información de los usuarios añadidos al sistema (persistencia);
- registrar los usuarios detectados por el sistema de reconocimiento facial en un fichero de *log*;
- determinar si la persona con la que está interactuando el sistema de reconocimiento facial está físicamente presente (*liveness*).

Es necesario la consecución de todos los objetivos mencionados para finalmente alcanzar el objetivo global.

1.3 Estructura de la memoria

El presente trabajo se divide en los siguientes siete capítulos:

- **Capítulo 1. Introducción:** se introduce la temática de la memoria, se expone el problema global y se describen las motivaciones que han llevado a realizar el trabajo y los objetivos que se pretenden alcanzar.

- **Capítulo 2. Contexto tecnológico:** se explican algunos de los primeros avances del reconocimiento facial por computador, los principales enfoques de este campo que han surgido a lo largo de la historia y las tecnologías actuales, para dar una visión general de la evolución de esta tecnología. También se describen los componentes de un sistema de reconocimiento facial con el objetivo de comprender su funcionamiento.
- **Capítulo 3. Análisis del problema:** se realiza la especificación de requisitos y se analizan las posibles soluciones de diseño y desarrollo del sistema de reconocimiento facial teniendo en cuenta estos requisitos. Finalmente se propone una solución al problema escogiendo un programa elemental que se utiliza como base para desarrollar el sistema de reconocimiento facial.
- **Capítulo 4. Diseño de la solución:** se describen las tecnologías seleccionadas para desarrollar el sistema de reconocimiento facial y se explica brevemente para qué y cómo se va a usar cada una de ellas.
- **Capítulo 5. Desarrollo del sistema de reconocimiento facial:** se detalla cómo se ha desarrollado cada uno de los componentes del sistema de reconocimiento facial así como su utilidad y funcionamiento.
- **Capítulo 6. Pruebas de rendimiento:** se comentan las pruebas de rendimiento del sistema de reconocimiento facial y se muestran y analizan sus resultados.
- **Capítulo 7. Conclusiones:** se describen las conclusiones del trabajo realizado. También se presentan los trabajos futuros y se reflexiona sobre la relación del trabajo con los estudios cursados.

CAPÍTULO 2
Contexto tecnológico

Este capítulo presenta los avances más relevantes en el campo del reconocimiento facial a lo largo de la historia para tratar de dar una visión general de la evolución del reconocimiento facial hasta la actualidad. Para ello, se describen los primeros avances y los cuatro principales enfoques de estudio de este campo, mencionando sus ventajas y desventajas, y explicando las principales contribuciones de los estudios más representativos de cada enfoque. Asimismo, el capítulo presenta los componentes de un sistema de reconocimiento facial y expone algunos de los SDK (*Software Develoment Kit*), API (*Application Programming Interface*), plataformas y compañías de reconocimiento facial más reconocidos actualmente [4, 7].

2.1 Primeros avances

En los siguientes subapartados de esta sección del capítulo se explican algunos de los primeros avances más relevantes en reconocimiento facial automático.

2.1.1. Hombre-máquina

Woody Bledsoe, Helen Chan Wolf y Charles Bisson, fueron pioneros del reconocimiento facial automático [18]. Entre 1964 y 1965, Bledsoe, junto con Helen Chan y Charles Bisson, trabajó en un proyecto que fue etiquetado como hombre-máquina, ya que fue el humano quien extrajo manualmente, usando una tableta gráfica RAND[1] (véase Figura 2.1), las coordenadas de un conjunto de características de las fotografías tales como el centro de las pupilas, la esquina interna de los ojos, la esquina externa de los ojos, el punto del pico de viuda, etc. (v. Figura 2.2). A partir de estas coordenadas, se calculó una lista de 20 distancias, como el ancho de los ojos, de pupila a pupila, y el ancho de la boca. Este conjunto de distancias se normalizó para representar la cara en una orientación frontal, usando un programa que deshacía el efecto de la inclinación, rotación y escala de la cabeza de la fotografía. Al construir la base de datos, el nombre de la persona de la fotografía se asoció con la lista de distancias calculadas y se almacenó en la computadora. En la fase de reconocimiento, el conjunto de distancias de la cara a reconocer se comparaba con las distancias correspondientes para cada fotografía de la base de datos, produciendo una nueva distancia. El programa devuelve los registros más cercanos de la base de datos, es decir, con la menor distancia respecto a la fotografía de la cara que se quería reconocer.

[1] https://www.rand.org

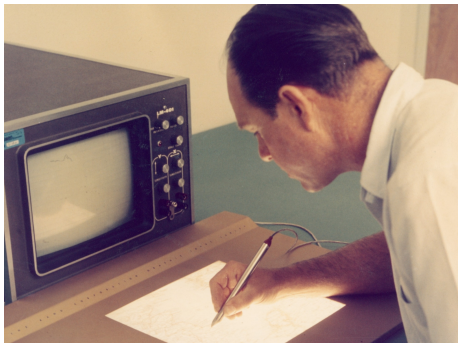

Figura 2.1: Tableta gráfica RAND.

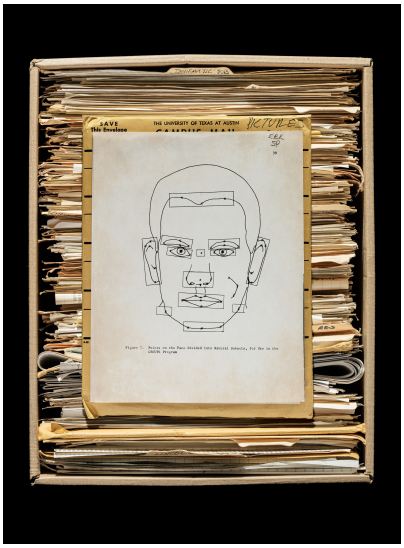

Figura 2.2: Características faciales de Bledsoe.

2.1.2. Caras propias

En 1991, Matthew A. Turk y Alex P. Pentland descubrieron la manera de detectar y reconocer caras automáticamente en imágenes de rostros humanos [65], logrando muy buenos resultados, y, también, en tiempo real, en un entorno favorable, restringido, pero de forma rápida y relativamente sencilla. Para ello, partiendo de un conjunto de imágenes de caras (conjunto de entrenamiento), transformaban las imágenes de las caras en un pequeño conjunto de imágenes de rasgos característicos (que no tenían por qué corresponder con las nociones intuitivas que las personas tienen de las partes y características faciales). En términos matemáticos, y considerando las imágenes de caras como vectores con valores de intensidad y dimensión igual al número de píxeles de la imagen, extraían los vectores propios con mayores valores propios de la matriz de covarianza del conjunto de imágenes de rostros de entrenamiento. Dichos vectores propios son los componentes principales de la distribución de caras, tienen aspecto de caras fantasmales y estos dos investigadores los denominaron caras propias (*eigenfaces*). El reconocimiento facial se realiza proyectando una nueva imagen en el subespacio definido por las caras propias («espacio facial») (v. Figura 2.5) y luego clasificando la cara comparando su posición en el espacio facial con la posición de individuos conocidos. En la Figura 2.5 se muestran las cuatro opciones que existen para un simple ejemplo con dos *eigenfaces*.

Turk y Pentland se basaron, entre otros, en los trabajos de Bledsoe [8], Kanade [30] y Yuille *et al.* [74]. La idea de usar *eigenfaces*, según se describe en [66], fue motivada por

una técnica desarrollada por L. Sirovich y M. Kirby [55] para representar eficientemente fotografías de caras utilizando el análisis de componentes principales. Ellos argumentaron que un conjunto de imágenes de caras puede ser reconstruida aproximadamente guardando una pequeña colección de pesos para cada imagen y un pequeño conjunto de fotografías estándar.

Figura 2.3: Siete caras propias o *eigenfaces*, sin fondo eliminado. Recuperada de [66].

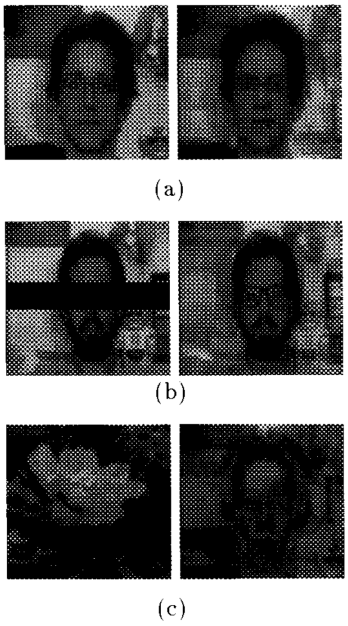

Figura 2.4: Tres imágenes y sus proyecciones en el espacio facial definido por las caras propias de la Figura 2.3. Recuperada de [66].

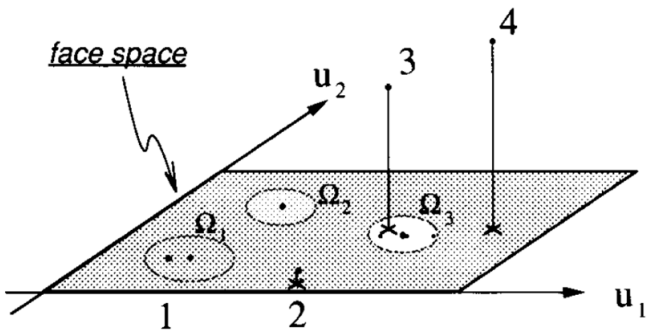

Figura 2.5: Versión simplificada del espacio facial para ilustrar los cuatro resultados de proyectar una imagen en el espacio facial. En este caso, hay dos *eigenfaces* (u_1 y u_2) y tres individuos conocidos (Ω_1, Ω_2 y Ω_3). Recuperada de [66].

2.1.3. Algoritmo Viola-Jones

En 2001, Paul Viola y Michael Jones desarrollaron el algoritmo Viola-Jones, un marco de reconocimiento de objetos que permite la detección rápida de características de imágenes en tiempo real usando características simples, de tipo Haar (*Haar-like Features*) (v. Figura 2.6 y Figura 2.7), que toman el nombre del matemático húngaro Alfred Haar, quien desarrolló en el siglo XIX el concepto de ondículas de Haar o *Haar wavelets* (antecesor de las características tipo Haar) [25]. La primera aportación de [69] fue una nueva representación de imágenes, llamada imágenes integrales (v. Figura 2.8), que permite una muy rápida evaluación de dichas características. La segunda contribución, es una variante del algoritmo AdaBoost (*Adaptative Boosting*) [19] utilizado para seleccionar un pequeño conjunto de características importantes y para entrenar el clasificador. La tercera contribución principal, es un método para combinar sucesivamente clasificadores más complejos en una estructura de cascada la cual aumenta drásticamente la velocidad de detección enfocándose en las regiones más prometedoras de la imagen. Se trata de una cascada de clasificadores para descartar con clasificadores más simples la mayoría de subventanas de una imagen antes de utilizar los clasificadores más complejos (v. Figura 2.9). Este nuevo enfoque minimiza el tiempo de cálculo a la vez que logra una alta precisión de detección.

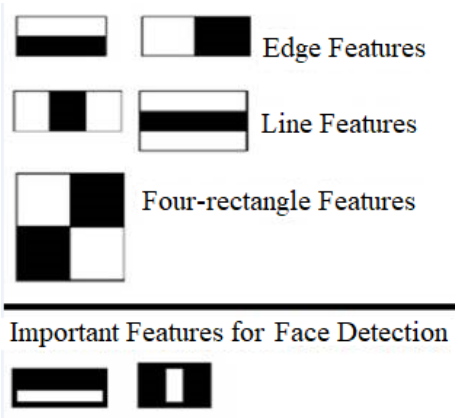

Figura 2.6: Características tipo Haar (*Haar-like features*). Recuperada de [25].

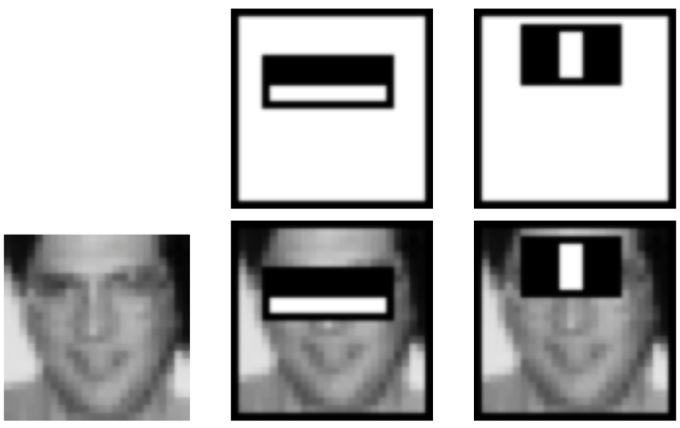

Figura 2.7: Características de tipo Haar seleccionadas por Adaboost. Recuperada de [69].

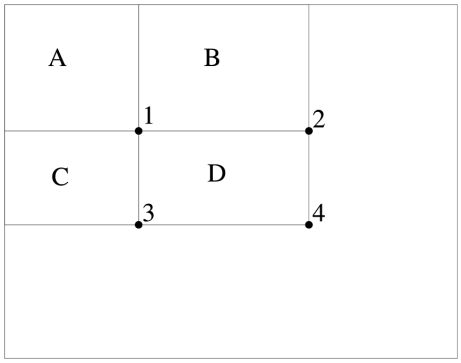

Figura 2.8: Imagen integral simplificada. El sumatorio de los píxeles dentro del rectángulo *D* puede ser calculado con cuatro referencias de matriz. El valor de la imagen integral en la localización 1 es la suma de los píxeles en el rectángulo *A*. El valor en la localización 2 es *A* + *B*, en la localización 3, *A*+*C*, y en la 4, *A* + *B* + *C* + *D*. La suma dentro de *D* puede ser calculada como 4 + 1 - (2 + 3). Recuperada de [69].

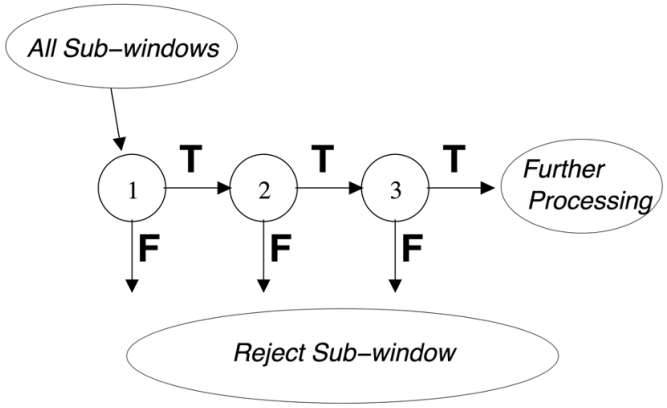

Figura 2.9: Representación esquemática de la cascada de detección. Se aplican una serie de clasificadores a cada subventana. El clasificador inicial elimina una gran cantidad de ejemplos negativos con muy poco procesamiento. Recuperada de [69].

2.2 Enfoques de reconocimiento facial

A partir de los trabajos de [65] y [69], explicados brevemente en las secciones 2.1.2 y 2.1.3 de esta memoria, hubo un significante incremento del interés en las tecnologías de reconocimiento facial [58]. El progreso técnico en este campo se puede dividir en cuatro principales áreas de interés de estudio o enfoques: **reconocimiento por aprendizaje supervisado mediante el uso de características globales** o por aprendizaje holístico, **reconocimiento por aprendizaje supervisado mediante el uso de características locales** o manual local (*local handcraft*), **reconocimiento por aprendizaje no supervisado mediante clasificadores automáticos con redes neuronales** o por aprendizaje superficial (*shallow learning*) y **reconocimiento por aprendizaje profundo no supervisado mediante clasificadores automáticos** o *Deep Learning*.

2.2.1. Reconocimiento por aprendizaje supervisado mediante el uso de características globales

Estos enfoques holísticos de reconocimiento facial toman toda la imagen del rostro humano para el reconocimiento [56]. Se utiliza información global de las caras (que está

fundamentalmente representada por un pequeño número de características derivadas directamente de la información de los píxeles de las imágenes de caras) para identificar a las personas que aparecen en las imágenes. Uno de los mejores ejemplos de método holístico es las *eigenfaces* o caras propias de [65, 66], ya descritas en la sección 2.1.2 de esta memoria [44].

Los enfoques holísticos emplean conceptos de distribución como representación múltiple [27] y dispersa [73, 75], y subespacio lineal [6, 38] para crear representaciones de baja dimensión. Sin embargo, estos enfoques están limitados por las incontrolables variaciones de la apariencia de las caras. Se podría decir que la ventaja de los enfoques holísticos es la sencillez y la eficiencia para reconocer caras en entornos restringidos mientras que la desventaja sería que es relativamente ineficaz reconociendo rostros en condiciones sin restricciones debido a la falta de robustez frente a los cambios de iluminación, pose, expresión y calidad de imagen.

2.2.2. Reconocimiento por aprendizaje supervisado mediante el uso de características locales

En estos enfoques de reconocimiento manual local se localizan varias características faciales, diseñadas manualmente, y luego, se clasifican las caras comparando y combinando las estadísticas locales correspondientes.

La detección de características locales puede realizarse mediante métodos basados en apariencia local o mediante métodos basados en características locales. Los métodos basados en la apariencia local detectan puntos de características al segmentar la imagen en subregiones (v. Figura 2.10), pero están muy limitados debido a que el rendimiento de las técnicas de segmentación es muy limitado. Sin embargo, este problema se puede resolver fácilmente con métodos basados en características locales (los picos, valles y crestas de ojos, nariz, boca, y otras características sobresalientes, como por ejemplo, los hoyuelos) (v. Figura 2.11), ya que la base de datos de características locales se monta con cada característica representando a un único objeto, y en la fase de reconocimiento, las características locales de un objeto son comparadas con las características almacenadas en la base de datos. Las imágenes del mismo objeto pueden tomarse en diferentes condiciones ambientales e instrumentales. Por lo tanto, el éxito de los métodos basados en características locales depende en gran medida de la detección correcta de las características locales que son altamente distintivas e invariables para diferentes condiciones de imagen [43].

Figura 2.10: Método basado en la apariencia de las regiones locales. Recuperada de [3].

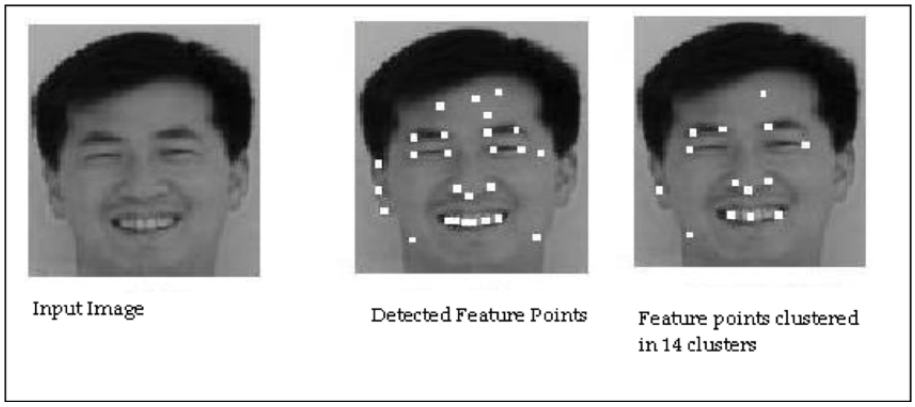

Figura 2.11: Detección de puntos para método basado en características locales. Resultado de la aplicación del detector de esquinas de Harris (*Harris corner detector*). Recuperada de [43].

Las limitaciones de los enfoques holísiticos fueron las que dieron lugar a las técnicas basadas en características locales tales como las ondículas o wavelets de Gabor (*Gabor wavelets*) [33, 57, 61] (v. Figura 2.12) y Patrones Locales Binarios (*Local Binary Patterns* o LBP) [2] (v. Figura 2.13 y 2.14), las cuales proporcionaron una mayor robustez debido a una mayor invariancia a las transformaciones y a los efectos ambientales.

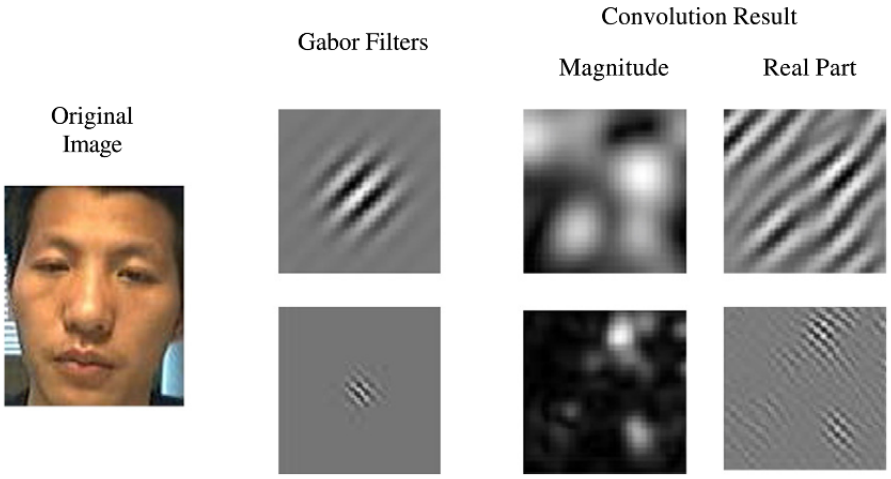

Figura 2.12: Resultados de convolución de una imagen facial con dos filtros Gabor. Recuperada de [57].

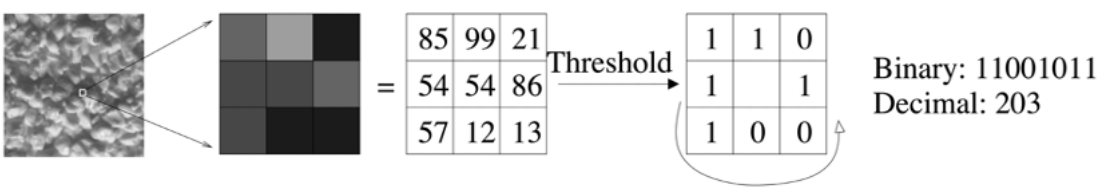

Figura 2.13: El operador LBP básico. Recuperada de [2].

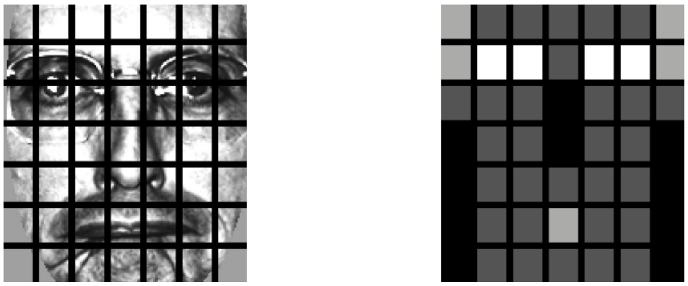

Figura 2.14: Descripción de la cara basada en LBP. Recuperada de [2].

2.2.3. Reconocimiento por aprendizaje no supervisado mediante clasificadores automáticos con redes neuronales

El concepto de reconocimiento por aprendizaje superficial hace referencia al reconocimiento que, a diferencia del reconocimiento por aprendizaje profundo o *Deep Learning*, utiliza redes neuronales con pocas capas ocultas (una o dos capas ocultas) (v. Figura 2.15). Los descriptores locales basados en el aprendizaje superficial [10, 11, 35] solucionaron parcialmente el problema de la falta de distinción de caras del reconocimiento manual local y mejoraron la compactibilidad e introdujeron la codificación automatizada. Sin embargo, estos métodos superficiales no lograron un rendimiento óptimo debido a su incapacidad para representar con precisión las complejas variaciones no lineales en la apariencia facial.

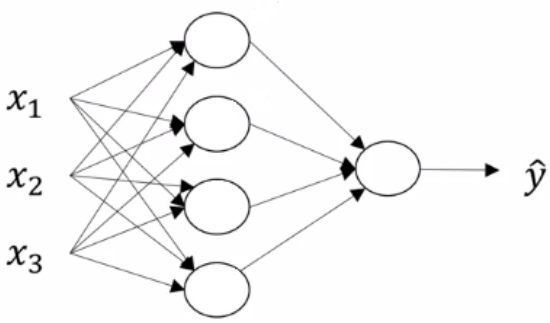

Figura 2.15: Representación gráfica de una *shallow neural network* con una capa oculta, una capa de entrada y una capa de salida. Recuperada de [1].

En [10] se presenta un método de codificación basado en el aprendizaje (*learning-based encoding method*) que utiliza métodos de aprendizaje no supervisado para codificar las microestructuras locales de la cara en un conjunto de códigos discretos. Los códigos aprendidos están distribuidos de manera más uniforme y el histograma del código resultante puede lograr mucho mejor equilibrio entre poder discriminativo y robustez (o invarianza) que los métodos de codificación manuales existentes. Además, para conseguir un descriptor de cara compacto, se aplica la técnica denominada Análisis de Componentes Principales (*Principal Component Analysis* o PCA) al histograma de código para reducir dimensiones. Después de aplicar PCA, un mecanismo de normalización simple puede mejorar la capacidad discriminativa del histograma de código. Utilizando dos métodos simples de aprendizaje no supervisado, se obtiene el descriptor basado en el aprendizaje (LE), una representación facial altamente discriminativa y compacta.

PCANet, propuesta en [11], es una red de aprendizaje profundo para la clasificación de imágenes, muy simple, que puede ser diseñada y enseñada de manera extremadamente sencilla y eficiente, y que comprende solamente los siguientes componentes básicos de

procesamiento de datos (v. Figura 2.16): análisis de componentes principales en cascada (PCA), *hashing* binario e histogramas de bloques.

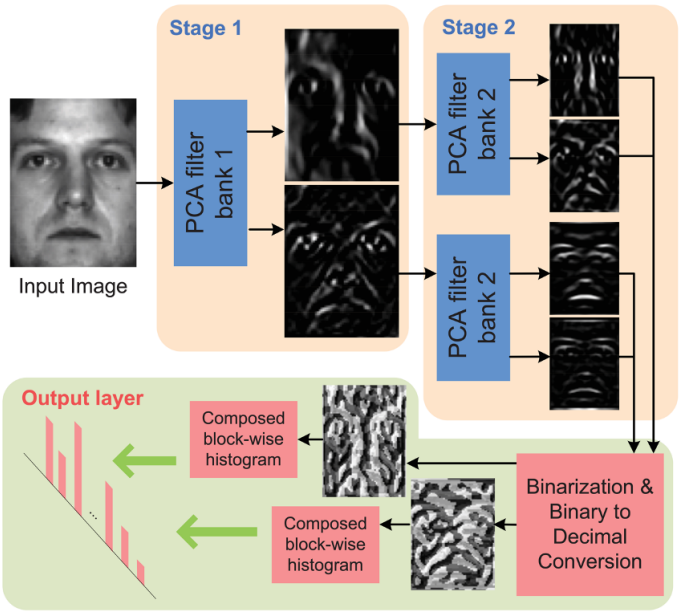

Figura 2.16: Ilustración de cómo PCANet extrae características de una imagen a través de tres componentes de procesamiento: filtros PCA, *hashing* binario e histograma. Recuperada de [11].

2.2.4. Reconocimiento por aprendizaje profundo no supervisado mediante clasificadores automáticos

Los métodos superficiales de reconocimiento solamente consiguieron mejorar la precisión del banco de pruebas o *benchmark* LFW (*Labeled Face in the Wild*) sobre el 95 % [12], lo que indica que no son suficientes para extraer la característica de identidad estable, invariante a los cambios del mundo real. Debido a esta insuficiencia técnica, se reportaron innumerables falsas alarmas en aplicaciones del mundo real [71].

Pero todo esto cambió en 2012 cuando AlexNet ganó la competición de reconocimiento visual a gran escala ImageNet (ILSVRC, *ImageNet Large-Scale Visual Recognition Competition*) superando el anterior mejor resultado con amplio margen usando una técnica llamada aprendizaje profundo o *deep learning* [32]. Los métodos de aprendizaje profundo, como las redes neuronales convolucionales, usan una cascada de múltiples capas de unidades de procesamiento (por lo que, hay muchas capas ocultas o *hidden layers*) para la extracción y transformación de características. Estas capas aprenden múltiples niveles de representaciones que corresponden a diferentes niveles de abstracción. Los niveles forman una jerarquía de conceptos, mostrando una fuerte invariabilidad frente a los cambios de postura, iluminación y expresión de la cara. Las capas iniciales aprenden automáticamente las características diseñadas durante años o incluso décadas, como por ejemplo Gabor y SIFT, y las capas posteriores aprenden un nivel superior de abstracción. La combinación de las abstracciones de nivel superior representa la identidad facial con una estabilidad sin precedentes. Es decir, el modelo de aprendizaje profundo (v. Figura 2.17) consta de múltiples capas de neuronas simuladas que convolucionan y agrupan la entrada, durante las cuales el tamaño del campo receptivo de las neuronas simuladas se amplía continuamente para integrar los elementos primarios de bajo nivel en atributos faciales múltiples, finalmente alimentando los datos hacia una o más capas completamente conectadas en la parte superior de la red. La salida es un vector de características comprimido que representa la cara. AlexNet consiste en cinco capas convolucionales y

tres capas totalmente conectadas (v. Figura 2.18), y también integra varias técnicas como por ejemplo el aumento de datos, *dropout* [16] y la unidad lineal rectificada (ReLU, *Rectified Linear Unit*). ReLU, fue considerado como el componente más importante para hacer posible el aprendizaje profundo.

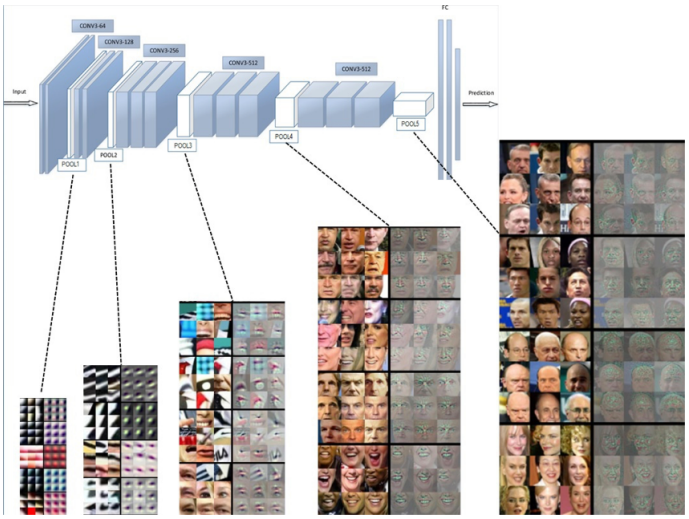

Figura 2.17: La arquitectura jerárquica del modelo de aprendizaje profundo. Recuperada de [71].

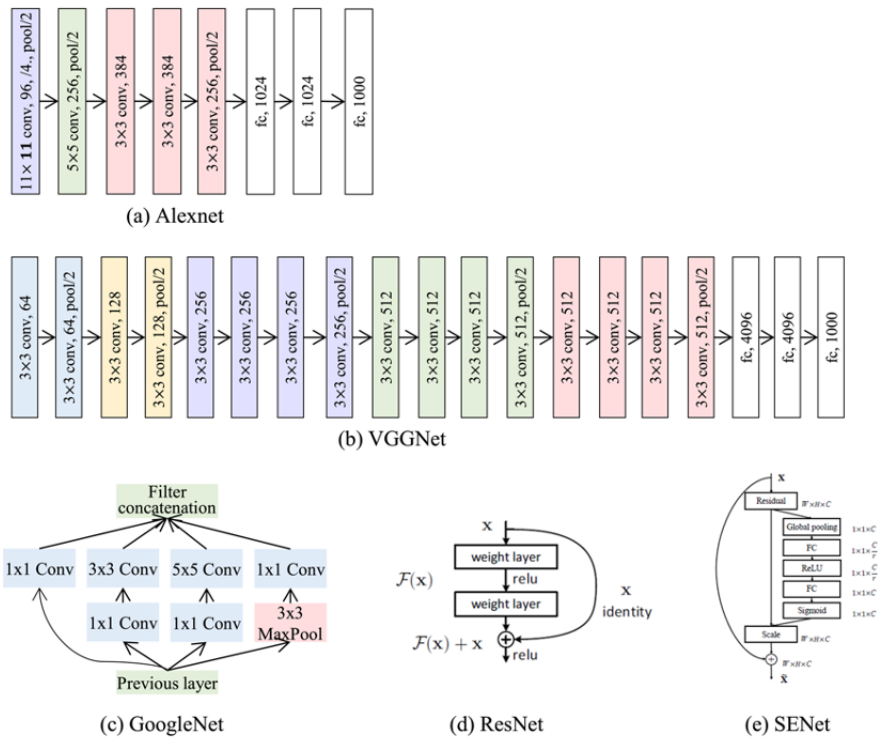

Figura 2.18: La arquitectura de Alexnet, VGGNet, GoogleNet, ResNet y SENet. Recuperada de [71].

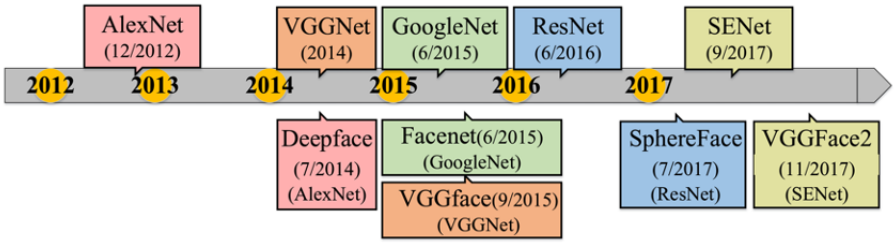

Figura 2.19: Las arquitecturas de red típicas en la clasificación de objetos (fila superior) y los algoritmos de reconocimiento facial profundo que utilizan las arquitecturas típicas y que logran un buen rendimiento (fila inferior). Los rectángulos del mismo color significan que usan la misma arquitectura. Recuperada de [71].

Figura 2.20: Estructura de FaceNet. Recuperada de [52].

En los siguientes años se siguieron haciendo avances en clasificación de objetos y surgieron arquitecturas de red como VGGNet [59] en 2014, GoogleNet [62] en 2015 y ResNet [26] en 2016 (v. Figura 2.18), entre otras. La comunidad de reconocimiento facial profundo, motivada por estos grandes avances en clasificación de objetos, siguió estas arquitecturas [71] (v. Figura 2.19). Así, en 2014, DeepFace [63] logró una precisión de 97,35 % en el famoso banco de pruebas LFW (v. Figura 2.21), acercándose por primera vez al rendimiento humano (97,53 %) en condiciones sin restricciones, entrenando un modelo de nueve capas con cuatro millones de imágenes. En 2015, FaceNet [52] (v. Figura 2.20) utilizó un gran conjunto de datos privados para entrenar a GoogleNet con una función de pérdida de tripleta (v. Figura 2.22) basada en tripletas de parches de cara coincidentes y no coincidentes, generadas usando un novedoso método de minería de tripletas en línea, y logró un nuevo récord de precisión (99,63 %) en el conjunto de datos LFW con solo 128 bytes por cara. En ese mismo año, VGGface [45] entrenó la VGGNet en un conjunto de datos de Internet y luego ajustó las redes a través de una función de pérdida de tripleta similar a FaceNet. VGGface consiguió una precisión del 98,95 %. En 2017, SphereFace [34] usó una arquitectura ResNet de 64 capas y propuso la pérdida angular softmax (A-Softmax) (v. Figura 2.23) para aprender características faciales discriminatorias con margen angular (99.42 % en LFW). En 2018, Arcface [15] (v. Figura 2.24) consiguió un 99,78 % en LFW.

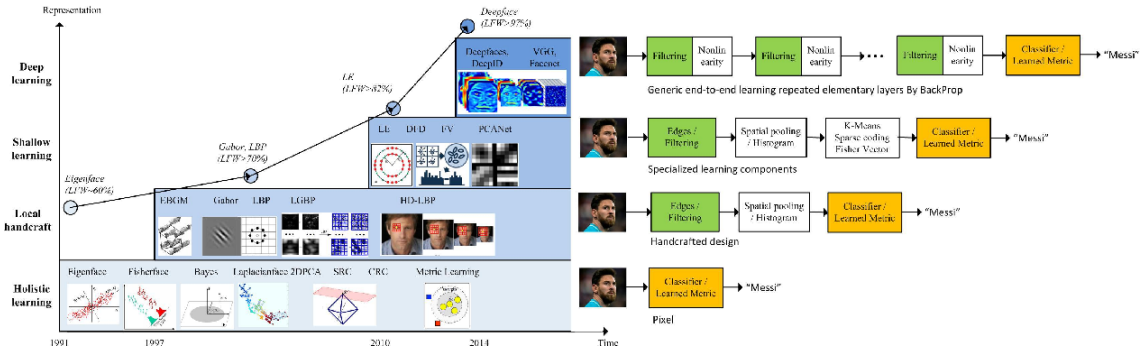

Figura 2.21: Hitos de la representación facial para el reconocimiento. Recuperada de [71].

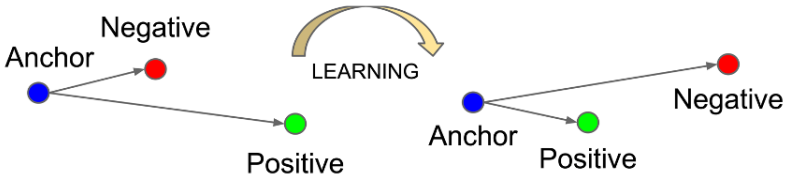

Figura 2.22: La pérdida de tripleta (*Triplet Loss*) minimiza la distancia entre un ancla y un positivo, los cuales tienen la misma identidad, y maximiza la distancia entre el ancla y un negativo de una identidad diferente. Recuperada de [52].

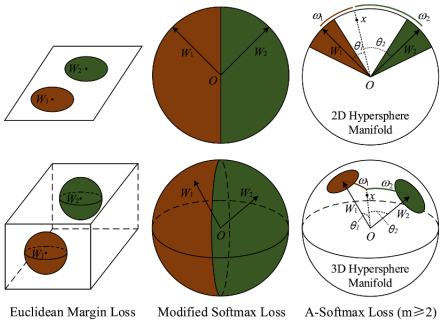

Figura 2.23: Interpretación de la geometría de la pérdida de margen euclidiana, pérdida de Softmax modificada y pérdida de A-Softmax. La primera fila es la restricción de características 2D, y la segunda fila, la de características 3D. La región marrón indica la restricción discriminativa para la clase 1 y la verde, para la clase 2. Recuperada de [34].

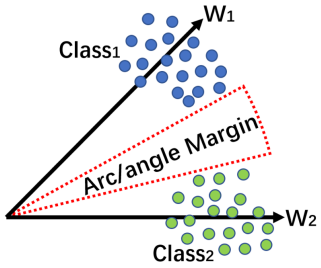

Figura 2.24: Interpretación geométrica de ArcFace. Los puntos azules y verdes representan características de incrustación de dos clases diferentes. ArcFace puede imponer directamente un margen angular (arco) entre clases. Recuperada de [15].

En la Figura 2.25 se muestra el desarrollo de las funciones de pérdida. Después de la introducción de Deepface y DeepID, la pérdida basada en la distancia euclidiana jugó un papel importante en la función de pérdida, como la pérdida por contracción, la pérdida de tripleta y la pérdida central. En 2016 y 2017, L-softmax y A-softmax promovieron aún más el desarrollo del aprendizaje de características de gran margen. En 2017, la normalización de características y peso también comenzó a mostrar un excelente rendimiento, lo que llevó al estudio sobre variaciones de softmax. Los rectángulos rojo, verde, azul y amarillo representan métodos profundos con softmax, pérdida basada en la distancia euclidiana, pérdida basada en el margen angular o coseno y variaciones de softmax, respectivamente.

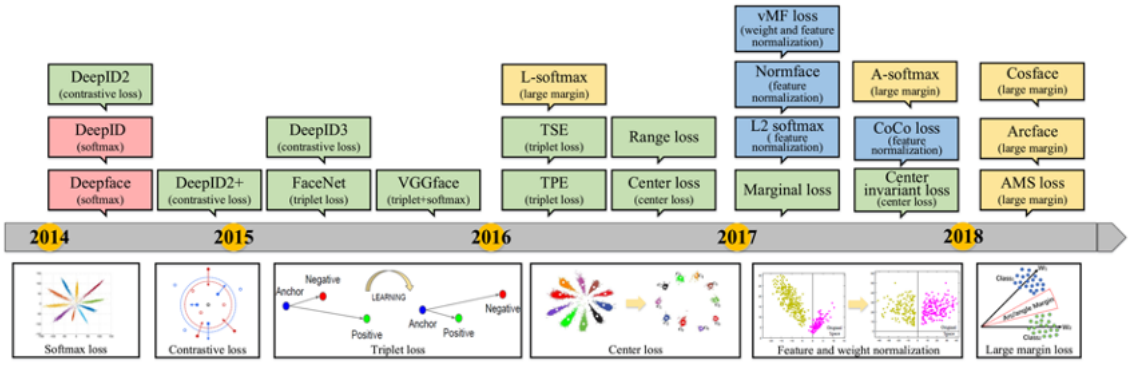

Figura 2.25: El desarrollo de las funciones de pérdida. Recuperada de [71].

2.3 Componentes de un sistema de reconocimiento facial

Un sistema completo de reconocimiento facial está compuesto de tres módulos: el módulo de **detección facial**, el de **detección de puntos de referencia faciales** (alineación para normalizar) y el de **reconocimiento facial**.

2.3.1. Detección facial

La detección de caras es un paso fundamental del reconocimiento facial [58]. Se usa también en aplicaciones como reconocimiento de expresiones faciales, seguimiento facial para fines de vigilancia, etiquetado digital en redes sociales y autoenfoque de cámaras o cámaras de teléfonos inteligentes. El desarrollo del método de Viola-Jones [69] para detectar caras (explicado en la sección 2.1.3 de esta memoria) permitió superar el obstáculo que había hasta la fecha de detectar rostros con alta precisión en entornos no controlados, y la detección de caras se volvió muy común. A partir de esa contribución se hicieron significantes avances debido al desarrollo de potentes técnicas de extracción de características. Existen las metodologías de detección de caras tradicionales y métodos basados en el aprendizaje profundo. A continuación se hablará solamente de estos últimos.

La creación de grandes bases de datos como MegaFace Challenge, LFW y WIDER FACE ha provocado que las redes neuronales convolucionales profundas (DCNN, *Deep Convolutional Neural Network*) mejoren significativamente su rendimiento en las tareas de detección de objetos y caras. Los métodos de reconocimiento facial que usan redes neuronales convolucionales profundas (DCNN) pueden ser basados en regiones o basados en ventanas deslizantes.

El enfoque basado en regiones utiliza un generador de propuestas de objetos, como por ejemplo la Búsqueda Selectiva (*Selective Search*) [68], para generar un conjunto de regiones que pueden incluir una o más caras. Estas propuestas son después las entradas de la DCNN, que las clasifica según si hay una cara o no, y devuelve las coordenadas del cuadro delimitador de caras de la imagen, incluyendo el mínimo fondo de fotografía. La mayoría de los métodos actuales utilizan este enfoque. Las redes neuronales convolucionales (CNN) más eficientes basadas en la región (R-CNN) utilizan una DCNN para generar propuestas, realizar regresiones y clasificaciones de cuadro delimitador. A pesar de los resultados impresionantes de los detectores de cara profunda basados en la región, estos son computacionalmente caros debido al requisito de la generación de propuestas.

Un método alternativo y mucho más eficiente para la detección de rostros es el método basado en la ventana deslizante, que calcula las coordenadas precisas del cuadro delimitador en cada ubicación en un mapa de características de una escala específica, uti-

lizando una operación de convolución. La invariancia de escala se logra al generar una pirámide de imagen que contiene múltiples escalas.

Los métodos de detección facial con más precisión en el banco de pruebas WIDER FACE son ScaleFaces (76,4 %), HR (81,9 %), SSH (84,4 %), S3FD (85,8 %) y FAN (88,5 %) [58]. Pero a pesar de la creciente precisión y velocidad de estos métodos, se siguen requiriendo técnicas de detección de caras CNN más eficientes.

2.3.2. Detección de puntos de referencia faciales

La extracción de características generalmente ocurre inmediatamente después de la detección de rostros y puede considerarse como una de las etapas más importantes en los sistemas de reconocimiento facial, ya que su efectividad depende de la calidad de las características extraídas [58]. Esto se debe a que los puntos de referencia faciales identificados por una red determinada determinan la precisión con la que se representan las características. Los localizadores de puntos de referencia tradicionales se basan en modelos, mientras que muchos métodos recientes se basan en regresiones en cascada. Últimamente, se han realizado mejoras clave con el desarrollo de métodos profundos de doble vía y otras soluciones basadas en mapas de confianza. Las metodologías tradicionales de puntos de referencia basadas en modelos incluyen el modelo de forma activa (ASM, *Active Shape Model*), con baja precisión, parcialmente rectificada por el modelo de apariencia activa (AAM, *Active Appearance Model*) y los modelos locales restringidos (CLM, *Constrained Local Model*), entre otros. Los modelos CLM generalmente son superados por la regresión en cascada. Sin embargo, cabe señalar que se han desarrollado métodos muy eficaces basados en CLM. El reconocimiento facial se diferencia del reconocimiento de objetos en que implica la alineación (o normalización de rosotros) antes de la extracción. Un aumento en la disponibilidad de datos ha dado como resultado el desarrollo de métodos basados en el aprendizaje en lugar de características diseñadas debido a su capacidad inherente para descubrir y optimizar características específicas de una tarea. En consecuencia, los métodos de aprendizaje han superado las características de ingeniería. En las CNN, se emplea un detector de puntos de referencia para localizar rasgos faciales importantes como el centro de los ojos, las comisuras de la boca y la punta de la nariz. Una vez que se han identificado estos puntos de referencia, la cara se alinea de acuerdo con las coordenadas canónicas normalizadas. Posteriormente, se extrae un descriptor de características que codifica la información de identidad.

2.3.3. Reconocimiento facial

El módulo de reconocimiento facial (RF) puede ser categorizado como verificación e identificación facial. En cualquiera de los dos casos, un conjunto de imágenes de sujetos conocidos se introduce inicialmente en el sistema (la galería), y durante las pruebas, se presenta una de un nuevo sujeto (la sonda). La verificación de rostros calcula la similitud uno a uno entre la galería y la sonda para determinar si las dos imágenes son del mismo sujeto, mientras que la identificación de rostros calcula la similitud uno a muchos para determinar la identidad específica de una cara de sonda. Se conoce como identificación de conjunto cerrado cuando la sonda aparece en las identidades de la galería y, de conjunto abierto, cuando las sondas incluyen a aquellos que no están en la galería.

Antes de que una imagen de una cara pase al módulo de RF, el *anti-spoofing* puede reconocer si la cara está viva o no. Un término muy relacionado con este último es el *liveness* o *liveness detection* [67], que es, en biometría, la capacidad de un sistema informático de inteligencia artificial para determinar que está interactuando con un ser humano físicamente presente y no con un artefacto inanimado (*spoof artifact*). Como se muestra

en la Figura 2.26, un módulo de RF está formado por el **procesamiento de caras**, que se usa para facilitar el reconocimiento antes del entrenamiento y las pruebas, la **extracción de características profundas**, donde se utilizan diferentes arquitecturas y funciones de pérdida para extraer características profundas discriminatorias durante el entrenamiento y las **coincidencias faciales**, para hacer la clasificación de características cuando se extrae la característica profunda de los datos de prueba [71].

Procesamiento de caras

Aunque los enfoques basados en el aprendizaje profundo se han utilizado ampliamente debido a su poderosa representación, se ha demostrado que las posturas, las iluminaciones, las expresiones y las oclusiones, todavía afectan al rendimiento del RF profundo y que el procesamiento de la cara es beneficioso, particularmente para las posturas. Dado que la variación de pose es ampliamente considerada como el mayor desafío en las aplicaciones de RF autómatico, se expondrán a continuación los métodos profundos de procesamiento facial para poses. Otras variaciones pueden resolverse por métodos similares. Los métodos de procesamiento facial se clasifican como **aumento de uno a muchos** y **normalización de muchos a uno** [71].

- **Aumento de uno a muchos:** genera muchos parches o imágenes de la variabilidad de la postura a partir de una sola imagen para permitir que las redes profundas aprendan representaciones invariantes de la pose. Se usan no solamente para aumentar los datos de entrenamiento sino que también la galería de datos para test. Se pueden clasificar en cuatro clases: aumento de información, modelo 3D, modelo de autocodificador y modelo GAN.

- **Normalización de muchos a uno:** recupera la vista canónica de las imágenes faciales de una o muchas imágenes de una vista no frontal. Luego, el RF puede realizarse como si estuviera bajo condiciones controladas. Es decir, produce caras frontales y reduce la variabilidad de la aparenciencia de la información de test para hacer que las caras se alineen y poder compararlas fácilmente. Se pueden categorizar como: modelo de autocodificador, modelo GAN y modelo CNN.

Extracción de características profundas

Las arquitecturas de red se pueden clasificar como redes troncales y redes ensambladas. Inspiradas por el éxito extraordinario en el desafío ImageNet, las arquitecturas típicas de CNN, como AlexNet, VGGNet, GoogleNet, ResNet y SENet son introducidas y ampliamente utilizadas como modelo de referencia en RF (directamente o ligeramente modificadas). Además de la corriente principal, todavía hay algunas arquitecturas novedosas diseñadas para RF para mejorar la eficiencia. Además, al adoptar redes troncales como bloques básicos, los métodos de RF a menudo entrenan redes ensambladas con múltiples entradas o múltiples tareas. Una red es para un tipo de entrada o un tipo de tarea. Esto proporciona un aumento en el rendimiento después de acumular los resultados de las redes ensambladas.

En cuanto a la función de pérdida, la pérdida softmax se usa comúnmente como señal de supervisión en el reconocimiento de objetos, y fomenta la separabilidad de las características. Sin embargo, para RF, cuando las variaciones internas pudieran ser mayores que las inter-diferencias, la pérdida de softmax no es suficientemente efectiva para RF. Muchos trabajos se centran en crear nuevas funciones de pérdida para hacer que las características no solo sean más separables sino también discriminatorias [71].

- **Pérdida basada en la distancia euclidiana:** la pérdida basada en la distancia euclidiana es un método de aprendizaje de métricas que introduce imágenes en el espacio Euclidiano en el que se reduce la intravarianza y se amplia la intervarianza. La pérdida de contraste y la pérdida de triplete son las funciones de pérdida más comúnmente utilizadas. Sin embargo, la pérdida de contraste y la pérdida de triplete ocasionalmente encuentran inestabilidad de entrenamiento debido a la selección de muestras de entrenamiento efectivas. La pérdida central y sus variantes son buenas opciones para reducir la intravarianza pero consumen mucha memoria de GPU en la capa de clasificación.

- **Pérdida basada en el margen angular o coseno:** es un método de aprendizaje de características faciales discriminatorias en términos de similitud angular, lo que conduce a una separabilidad angular / coseno potencialmente mayor entre las características aprendidas. La pérdida basada en el margen angular / coseno puede lograr mejores resultados en un conjunto de datos limpio, pero es vulnerable al ruido y se vuelve peor que la pérdida central y softmax en la región de alto ruido.

- **Pérdida de softmax y sus variaciones:** utilizando directamente la pérdida de softmax o modificándola para mejorar el rendimiento, por ejemplo, normalización de L2 en características o pesos, así como inyección de ruido.

Coincidencias faciales

Durante la fase de test, la distancia coseno y la distancia L2 (distancia euclidiana) se utilizan generalmente para medir la similitud entre características profundas. Luego, se utliza una comparación con un umbral y un clasificador de vecinos más cercanos (*nearest neighbor*, NN) para realizar la verificación y la identificación. Aparte de estos métodos, que son los más comunes, hay algunos otros.

- **Verificación facial:** aprendizaje de métricas, que tiene como objetivo encontrar una nueva métrica para hacer que dos clases sean más separables. También se puede usar para la coincidencia de caras en función de las características profundas extraídas. El modelo JB (*Joint Bayesian*) es uno de los métodos de aprendizaje de métricas más conocido [71].

- **Identificación facial:** los sistemas modernos de reconocimiento facial que usan DCNN implican una extracción profunda de características y, por último, una comparación de similitud. Más específicamente, la verificación implica la comparación de la similitud uno a uno entre una imagen de la sonda y una galería de una identidad conocida, mientras que la identificación determina una a muchas similitudes para determinar la identidad de la sonda. Ambos procesos requieren una representación de características robusta y un modelo de clasificación discriminativa o una medida de similitud. Los métodos tradicionales utilizados para la representación de características incluyen LBP, HoGs y Fisher Vector. Los métodos de aprendizaje métrico relevantes incluyen el aprendizaje métrico coseno, el aprendizaje métrico Mahalanobis y el núcleo de similitud de una sola vez. Otros incluyen un gran margen vecino más cercano, conjuntos bayesianos y clasificadores basados en atributos [58].

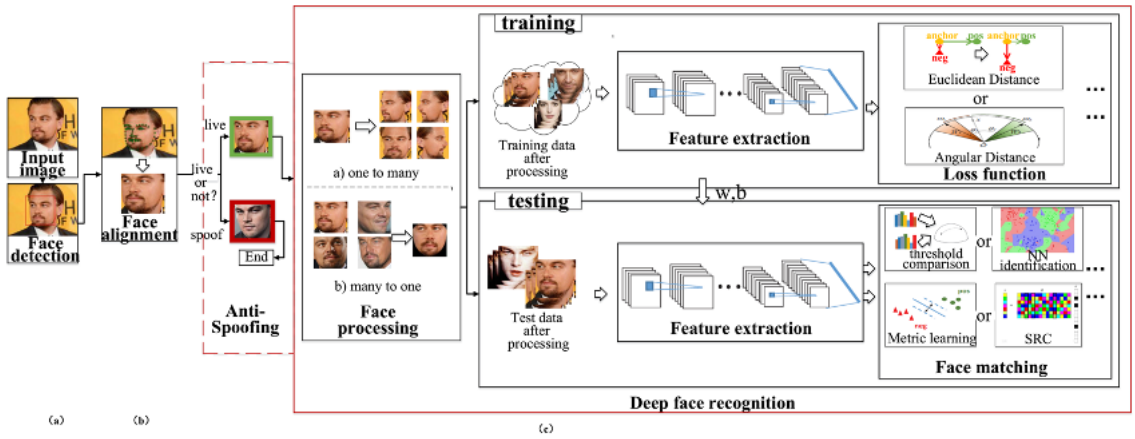

Figura 2.26: Sistema de reconocimiento facial profundo con detector facial y alineación. Recuperada de [71].

2.4 Tecnologías actuales de reconocimiento facial

A continuación se van a presentar algunos de los SDK (*Software Develoment Kit*), API (*Application Programming Interface*), plataformas y compañías de reconocimiento facial más reconocidos en la actualidad [4, 7].

2.4.1. Deep Vision AI

Deep Vision AI [14] es una empresa de software de visión por computador que destaca en reconocimiento facial. Utiliza tecnología avanzada propia de visión por computadora para comprender imágenes y vídeos automáticamente, convirtiendo el contenido visual en análisis en tiempo real y conocimientos de valor.

Con más de 500 millones de cámaras existentes en todo el mundo en la actualidad, Deep Vision AI permite analizar transmisiones de cámara a través de diferentes módulos de software basados en inteligencia artificial en una única plataforma de «conectar y usar» (*plug-and-play*), permitiendo a los usuarios tener alertas en tiempo real y una respuesta más rápida. Su software de reconocimiento facial monitorea continuamente las zonas objetivo para identificar individuos, que se comparan con una lista de seguimiento de personas de interés con tasas de alta precisión. El software es independiente de la cámara, se conecta a cualquier infraestructura de cámara existente y se puede implementar en la nube o en el *edge*.

Las diferentes capacidades de análisis facial son aplicadas a los casos de uso de inteligencia empresarial para ayudar a reconocer a los clientes importantes en tiempo real, cuantificar la frecuencia de los visitantes o mejorar la seguridad general. Otros atributos, como el recuento, la edad y el sexo de las personas, se utilizan para comprender los cambios demográficos a lo largo del tiempo, el comportamiento del cliente y medir los esfuerzos de marketing. Los clientes generalmente combinan las capacidades de reconocimiento facial con el reconocimiento avanzado de vehículos (v. Figura 2.27) y otras características basadas en inteligencia artificial que ofrece Deep Vision AI.

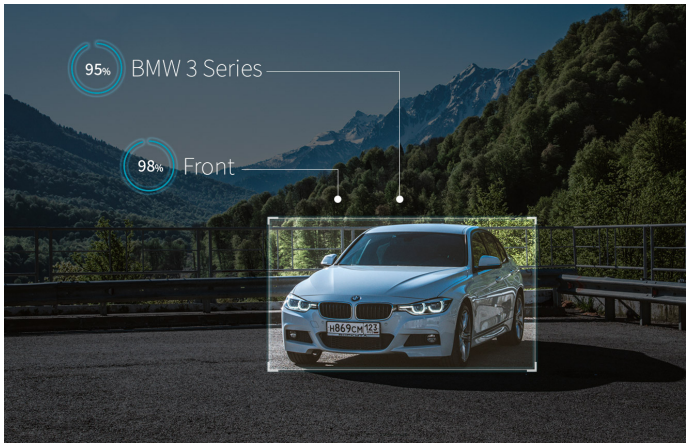

Figura 2.27: Reconocimiento avanzado de vehículos de Deep Vision AI. Recuperada de [14].

Los mercados principales incluyen ciudades y lugares públicos, grandes tiendas minoristas, energía e infraestructura, transporte público, campus escolares, lugares de juego y arenas para eventos. Deep Vision AI es un socio certificado de NVIDIA, Dell Digital Cities, Amazon AWS, Microsoft, Red Hat y otros.

2.4.2. SenseTime

SenseTime [53], desarrollador líder de plataformas tecnológicas, se dedica a crear soluciones para la industria a través de innovaciones en inteligencia artificial y análisis de *big data*. La tecnología multifuncional de SenseTime se está expandiendo rápidamente y ya incluye reconocimiento facial, reconocimiento de imágenes, análisis de video inteligente, conducción autónoma y reconocimiento de imágenes médicas. SenseTime ha prestado servicios a más de 400 empresas y agencias gubernamentales reconocidas, incluidas Honda, Qualcomm, China Mobile, UnionPay, Huawei, Xiaomi, OPPO, Vivo y Weibo. Entre su software de plataforma se encuentran:

- **SensePortrait-S** es un servidor de reconocimiento facial estático, que proporciona funciones de detección de rostros desde una fuente de imagen, así como extracción de características, análisis de atributos, comparación de atributos y recuperación de objetivos de una amplia base de datos de imágenes faciales.
- **SensePortrait-D** es un servidor de reconocimiento facial dinámico que proporciona funciones de detección de rostros en múltiples transmisiones de videovigilancia, así como seguimiento facial, extracción de funciones y comparación.
- **SenseFace** es una plataforma de vigilancia de reconocimiento facial. Basada en la tecnología de reconocimiento facial, utilizando un algoritmo de aprendizaje profundo, SenseFace proporciona soluciones integradas de análisis de vídeo inteligente, que funciona en vigilancia de objetivos, análisis de trayectoria, gestión de población y análisis de datos relevantes, etc.

2.4.3. Amazon Rekognition

Amazon Rekognition [5] facilita la adición de análisis de imagen y vídeo a sus aplicaciones con tecnología probada, altamente escalable y de aprendizaje profundo que no requiere experiencia en aprendizaje automático para su uso. Con Amazon Rekognition se puede identificar objectos, personas, texto, escenas y actividades en imágenes y vídeos,

además de detectar cualquier contenido inapropiado. Amazon Rekognition también proporciona análisis faciales de alta precisión y capacidades de búsqueda facial que puede usar para detectar, analizar y comparar rostros. Es posible implementar estos recursos en una amplia variedad de casos de uso vinculados con la verificación de usuarios, el conteo de personas y la seguridad pública.

Figura 2.28: Detección y análisis de rostros de Amazon Rekognition. Recuperada de [5].

Las características principales de Amazon Rekognition son las etiquetas para identificar todo tipo de objetos, la detección de texto, la moderación de contenido multimedia, la detección y análisis de rostros (v. Figura 2.28), la búsqueda y verificación de rostros, el reconocimiento de famosos y el seguimiento del recorrido de personas (para, por ejemplo, identificar las jugadas de los deportistas en un partido y realizar un análisis posterior, como muestra la Figura 2.29).

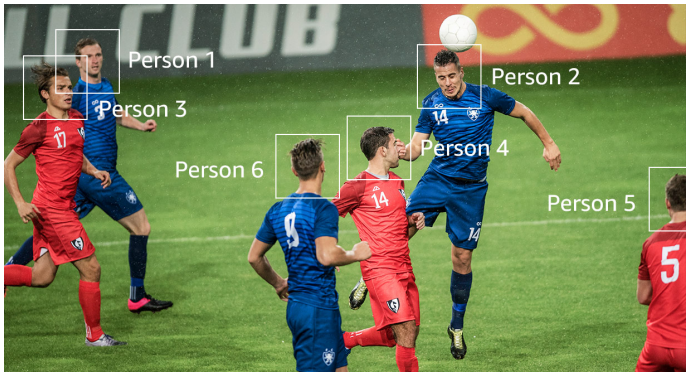

Figura 2.29: Seguimiento de movimientos de Amazon Rekognition. Recuperada de [5].

NFL, CBS Corporation, Influential, Marinus Analytics y Aella Credit, entre otros, son clientes de Amazon Rekognition.

2.4.4. Face++

Face++ [17] es una plataforma abierta de inteligencia artificial de la empresa china Megvii[2] que ofrece tecnologías de visión por computador que permite agregar fácilmente tecnologías líderes de reconocimiento de imágenes basadas en aprendizaje profundo y de análisis en sus aplicaciones, con interfaces de programación de aplicaciones (API) y kits de desarrollo de software (SDK) simples y potentes.

[2] https://megvii.com/en

Esta plataforma utiliza inteligencia artificial y visión por computador en una gran variedad de formas para detectar rostros, analizar 106 puntos de información en el rostro y confirmar la identidad de una persona con un alto grado de precisión. También permite a cualquier desarrollador crear aplicaciones utilizando sus algoritmos, lo que ha ayudado a convertirla en la plataforma de reconocimiento facial más extensa del mundo con 300.000 desarrolladores de 150 países que la utilizan.

Face++ está integrado en la plataforma City Brain de Alibaba[3] que analiza la red de CCTV (*Closed Circuit Television*) en las ciudades para optimizar los flujos de tráfico y observar incidentes que requieren atención policial o médica.

2.4.5. Kairos

Kairos [29] proporciona reconocimiento facial ético y de vanguardia a desarrolladores y empresas de todo el mundo. Se puede integrar el reconocimiento facial a través de la API en la nube de Kairos o alojar a Kairos en servidores para obtener el máximo control de los datos, la seguridad y la privacidad.

Kairos está desarrollando nuevos mercados utilizando su arquitectura ultraescalable, lo que significa que las personas pueden buscar 10 millones de rostros aproximadamente al mismo tiempo que un rostro.

2.4.6. Google Cloud Vision

Google ha optado por diferenciar sus servicios de reconocimiento facial en imágenes de los de reconocimiento facial en vídeo [24]. La API Vision de Google Cloud ofrece modelos de aprendizaje automático preparados previamente y muy potentes a través de las API REST y RPC. Permite asignar etiquetas a imágenes y clasificarlas rápidamente en millones de categorías predefinidas. También detectar objetos y caras, leer texto impreso y manuscrito, y conseguir metadatos de gran valor. Mientras que la API de Video Intelligence cuenta con modelos de aprendizaje automático entrenados previamente que reconocen de forma automática una gran cantidad de objetos, lugares y acciones en vídeos almacenados y en *streaming*. Resulta muy eficiente para los usos más habituales y mejora con el tiempo a medida que se introducen nuevos conceptos.

2.4.7. OpenCV

OpenCV (*Open Source Computer Vision Library*) [41] es una biblioteca de de software código libre de visión por computador y de aprendizaje automático (*machine learning*). OpenCV se creó para proporcionar una infraestructura común para aplicaciones de visión por computador y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia BSD (*Berkeley Software Distribution*), OpenCV facilita que las empresas utilicen y modifiquen el código.

La biblioteca tiene más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de aprendizaje automático y visión por computador clásicos y de última generación. Estos algoritmos se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estéreo, buscar imágenes similares de una base de datos de imágenes, eliminar ojos rojos de imágenes tomadas con flash, seguir los movimientos oculares, reconocer paisajes y establecer marcadores para superponerlos con realidad

[3] https://www.alibabacloud.com/solutions/intelligence-brain/city

aumentada, etc. OpenCV tiene más de 47 mil usuarios y un número estimado de descargas superior a 18 millones. La biblioteca se utiliza ampliamente en empresas, grupos de investigación y organismos gubernamentales.

Junto con empresas bien establecidas como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota que emplean la biblioteca, hay muchas nuevas empresas como Applied Minds, VideoSurf y Zeitera, que hacen un uso extensivo de OpenCV. Los usos desplegados de OpenCV abarcan desde unir imágenes de *streetview*, detectar intrusiones en vídeos de vigilancia en Israel, monitorear equipos de minas en China, ayudar a los robots a navegar y recoger objetos en Willow Garage, detectar accidentes por ahogamiento en piscinas en Europa, gestionar arte interactivo en España y Nueva York, revisar las pistas de aterrizaje en busca de escombros en Turquía, inspeccionar las etiquetas de los productos en fábricas de todo el mundo hasta la detección rápida de rostros en Japón.

Tiene interfaces C ++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS. OpenCV se inclina principalmente hacia aplicaciones de visión en tiempo real y aprovecha las instrucciones MMX y SSE cuando están disponibles. Actualmente se están desarrollando activamente interfaces CUDA y OpenCL con todas las funciones. Hay más de 500 algoritmos y aproximadamente 10 veces más funciones que componen o admiten esos algoritmos. OpenCV está escrito de forma nativa en C ++ y tiene una interfaz con plantilla que funciona a la perfección con los contenedores STL.

CAPÍTULO 3
Análisis del problema

En este capítulo se realiza el análisis de las posibles soluciones de diseño y desarrollo del sistema de reconocimiento facial. En primer lugar, se presenta la especificación de requisitos del sistema de reconocimiento facial, funcionales y no funcionales. A continuación, se indica el diagrama de casos de uso, para mostrar de manera gráfica los requisitos funcionales del sistema. Seguidamente, se seleccionan las dos mejores posibles soluciones y se citan las ventajas de cada una frente a la otra. Para finalizar, se expica brevemente la solución propuesta y el plan de desarrollo, implantación y validación que se va a seguir.

3.1 Especificación de requisitos

Los requisitos [72] se pueden definir como una especificación de lo que se debe implementar, son descripciones de cómo debería comportarse el sistema. Estos pueden clasificarse en requisitos funcionales y no funcionales. Los requisitos funcionales son una descripción de un comportamiento que exhibirá un sistema bajo condiciones específicas mientras que los requisitos no funcionales son una descripción de una propiedad o característica que un sistema debe exhibir o una restricción que debe respetar.

Esta sección contiene los principales requisitos del sistema agrupados por tipo (requisitos funcionales y requisitos no funcionales) y ámbito de actuación. Cabe indicar que los requisitos no se han escrito con excesivo detalle ya que el sistema no se ha hecho para un cliente externo y tanto el desarrollo como las pruebas no van a ser externalizados.

3.1.1. Requisitos funcionales

Los requisitos funcionales son descripciones de los comportamientos que un sistema deberá exhibir bajo unas condiciones específicas [72]. A continuación se detallan el identificador y la descripción de los principales requisitos funcionales del sistema de cada ámbito de actuación (administración de usuarios y reconocimiento facial).

Administración de usuarios

Identificador	Descripción
RF-A-01	El sistema debe permitir, tanto al desarrollador como al administrador, añadir perfiles de usuario a la base de datos mediante un formulario y una captura instantánea de la cara del usuario.
RF-A-02	El sistema debe permitir, tanto al desarrollador como al administrador, añadir perfiles de usuario a la base de datos mediante un formulario y la selección de una fotografía guardada en el equipo de la cara del usuario.
RF-A-03	El sistema debe permitir, tanto al desarrollador como al administrador, eliminar perfiles de usuario de la base de datos.
RF-A-04	El sistema debe permitir, tanto al desarrollador como al administrador, asignar un rol al perfil de usuario: usuario, administrador o desarrollador.
RF-A-05	El sistema (con tres modos: usuario, administrador y desarrollador) debe permitir, tanto al administrador como al desarrollador, pasar del modo usuario al de administrador y viceversa y, al desarrollador, del modo administrador al de desarrollador y viceversa.

3.1.2. Requisitos no funcionales

Seguidamente se presentan los requisitos no funcionales del sistema. Los requisitos no funcionales son descripciones de propiedades o características que un sistema debe exhibir o restricciones que debe respetar, no especifican qué hace el sistema pero sí cómo de bien hace las cosas [72].

Identificador	Descripción
RNF-A-01	El sistema debe poder detectar y reconocer las caras de los usuarios en lugares con una iluminancia igual o superior a 300 lux.
RNF-A-02	El sistema debe ser capaz de detectar y reconocer las caras de los usuarios en una resolucion de vídeo igual o superior a 480p.
RNF-A-03	El sistema solamente debe permitir añadir a la base de datos fotografías de una única cara descubierta y vista de frente (cara detectable).
RNF-A-04	El sistema ha de ser compatible con sistemas macOS, Linux y Windows.
RNF-A-05	El sistema debe poder reconocer rostros en menos de medio segundo.

3.2 Diagrama de casos de uso

En la Figura 3.1 se muestra el diagrama de casos de uso, que captura los requisitos funcionales del sistema a desarrollar. Los actores (quienes intercambian información con el sistema) son el usuario, el administrador y el desarrollador. Los casos de uso (representados en la Figura 3.1 como elipses) contienen una secuencia de transacciones que intercambian los actores y el sistema cuando se desea ejecutar cierta funcionalidad del mismo, como por ejemplo, añadir usuario, asignar rol o cambiar a modo desarrollador.

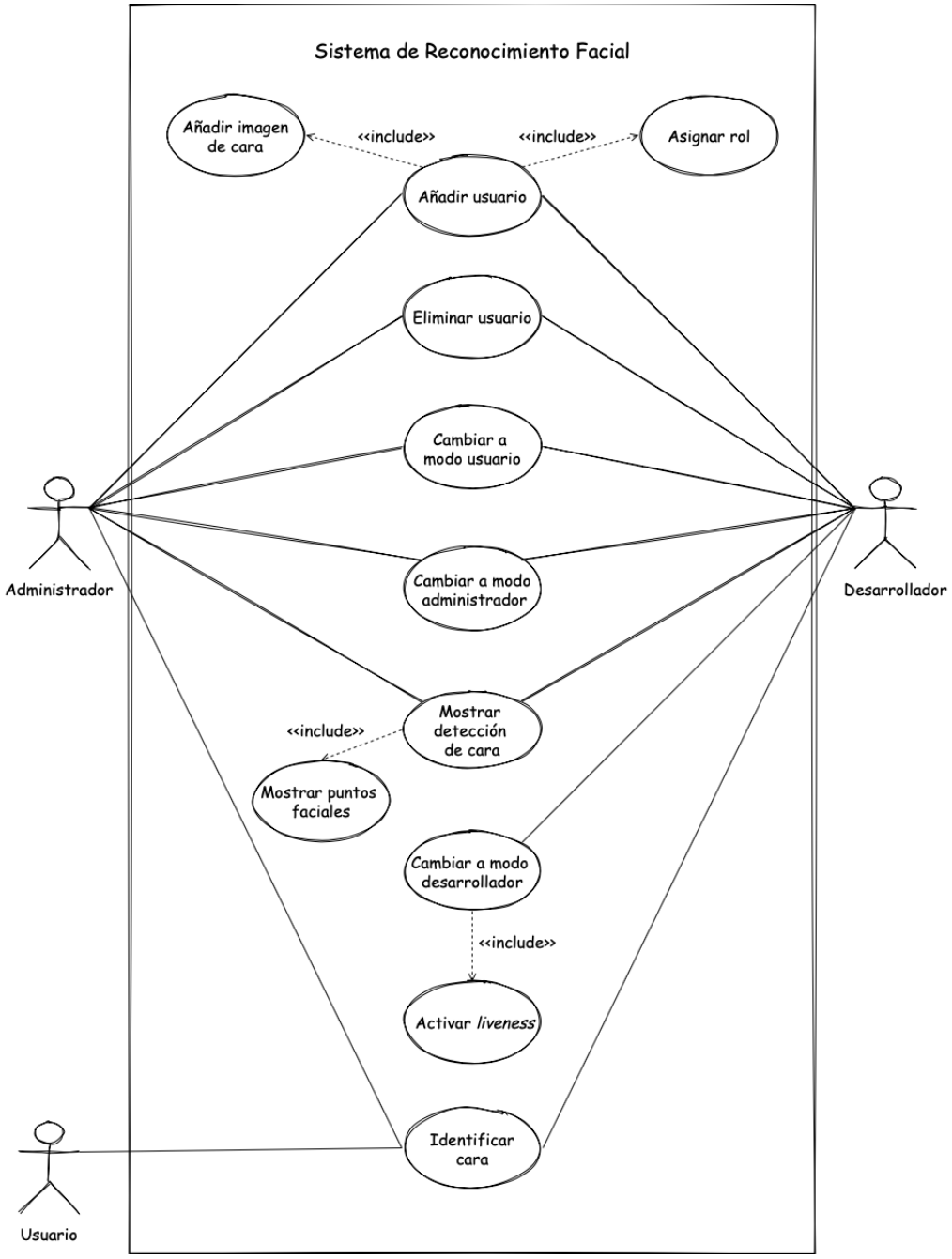

Figura 3.1: Diagrama de casos de uso del sistema de reconocimiento facial a desarrollar. Elaboración propia.

3.3 Identificación y análisis de posibles soluciones

La biblioteca libre de visión por computador OpenCV admite una multitud de algoritmos relacionados con la visión artificial y el aprendizaje automático (*machine learning*) y una amplia variedad de lenguajes de programación como C ++, Python, Java, etc.

Se ha optado por utilizar OpenCV-Python (interfaz del API de OpenCV para el lenguaje Python) para desarrollar el sistema de reconocimiento facial dado que hace uso de Numpy, que es una biblioteca altamente optimizada para operaciones numéricas y que Python es un lenguaje de programación sencillo de aprender y de utilizar [36, 39]. Ade-

más, por esta última razón, Python es muy usado por desarrolladores de *machine learning* [64].

Una vez escogido el lenguaje de programación del sistema, se ha realizado una búsqueda en internet para tratar de encontrar ejemplos de programas básicos de detección e identificación facial que utilicen OpenCV-Python. Entre la gran cantidad de ejemplos encontrados se destacan [9, 46, 50, 49, 51, 70], pero los que mejor se ajustan a los requisitos especificados en la sección 3.1 de esta memoria son Face Recognition [21, 22, 23, 60] y Deepface [54]. Además, estas dos bibliotecas se caracterizan por su sencillez y facilidad de instalación, configuración y uso, y por su gran portabilidad.

Para determinar los aspectos favorables de cada posible solución se han tomado como códigos base, el código de [60] de la biblioteca Face Recognition [21] y la función stream de la biblioteca Deepface [54]. En lo sucesivo, cuando se mencione Face Recognition se estará haciendo referencia tanto a la biblioteca Face Recogntion como a su código base y cuando se mencione Deepface se estará haciendo referencia tanto a su biblioteca completa como a la función stream perteneciente a dicha biblioteca. Las ventajas que se han identificado de Face Recognition respecto con Deepface son la completitud de la documentación de la biblioteca, la claridad del código de las funciones y su función face_landmarks devuelve un diccionario Python con las coordenadas de la imagen de los puntos característicos faciales, que puede permitir mostrarlos sobre la imagen de vídeo usando OpenCV-Python y también implementar la funcionalidad que detecta si la persona es real (*liveness*). En cambio, los pros de Deepface con respecto a Face Recognition son la gran variedad de modelos de reconocimiento facial de los que dispone (VGG-Face, Facenet, OpenFace, DeepFace, DeepID y Dlib), la variedad de funciones de pérdida (similitud del coseno, distancia euclídea y la forma L2), la cantidad de detectores faciales (cascada Haar de OpenCV, SSD, Dlib y MTCNN) y la facilidad de uso e intercambio de de las opciones anteriores. Además, su función analysis permite hacer un análisis de edad, género, expresión facial y raza.

La siguiente tabla muestra a grandes rasgos cómo se ha tomado la decisión. Sus ítems no están definidos con precisión y pueden tener un componente subjetivo. El peso será uno (bajo), dos (medio) o tres (alto), tomando como referencia la especificación de requisitos de la sección 3.1 y también según criterio personal. Los valores de la tercera y cuarta columna de la tabla de decisión se utilizan para determinar qué posible solución (alternativa) gana a la otra para cada ítem, tomando valores binarios, cero (perdedora) o uno (ganadora). Para asignar estos valores binarios se ha seguido un criterio subjetivo en algunos casos.

Ítem	**Peso**	**Face Recognition**	**Deepface**
1. Completitud de la documentación	3	1	0
2. Claridad de las funciones	2	1	0
3. Sencillez del programa	3	1	0
4. Puntos de características faciales	3	1	0
5. Detección facial	3	1	1
6. Identificación facial	3	1	1
7. *Liveness*	3	1	0
8. Análisis facial	1	0	1
9. Variedad de modelos de reconocimiento facial	2	0	1
10. Variedad de funciones de pérdida	2	0	1
11. Variedad de detectores faciales	2	0	1
12. Tiempo de reconocimiento facial	3	1	0

Tabla 3.1: Tabla de decisión (Face Recognition, Deepface). Elaboración propia.

Utilizando los datos de la tabla de decisión (v. Tabla 3.1), y para cada alternativa (Face Recognition y Deepface) se realiza el sumatorio de los productos de cada peso por el valor binario correspondiente y se obtiene una puntuación de 23 para la alternativa de Face Recognition y de 13 para la de Deepface y por tanto, se elegirá la primera alternativa como mejor solución.

3.4 Solución propuesta

La solución propuesta, que incluirá todos los casos de uso de la Figura 3.1, se elaborará a partir del código de [60] elegido en la sección 3.3 por los motivos presentados en dicha sección. Este código o programa base, detecta y compara las caras capturadas por la cámara con las caras de las imágenes con extensión «.jpg» que se encuentran en una carpeta que se debe llamar «known_people» y que debe estar situada en el mismo directorio que el archivo con el código. Dicho programa muestra los resultados de la detección y la identificación en tiempo real, colocando un marco rojo, sobre los fotogramas que se muestran en una ventana y alrededor de las caras detectadas, y situando debajo de este marco, una etiqueta con el nombre de la imagen de la cara de la persona que está capturando la cámara de vídeo (que debería coincidir con el nombre de la persona) seguido de «.jpg». Si no se puede identificar la cara, la etiqueta muestra *Unknown* . La solución extenderá el código escogido, añadiendo a los componentes básicos de detección e identificación nuevos componentes o bloques: uno de administración de usuarios (gestión), otro de almacenamiento de datos (persistencia), otro que se encarga de mostrar los puntos faciales, otro que muestra si la persona que detecta es «real» (*liveness*) y otro de salida (fichero de registro de personas detectadas). Estos componentes o bloques se muestran gráficamente en el esquema de la Figura 3.2. En la figura, en color azul aparecen los componentes básicos, es decir, los componentes que ya incluía el código base y en color naranja, se muestran los que se van a añadir. Los componentes de detección e identificación facial, puntos faciales y *liveness* aparecen dentro de un bloque azul más grande llamado Face Recogntion debido a que estos bloques están construidos o desarrollados usando la biblioteca Face Recognition. Las flechas, unidireccional y bidireccionales, indican que se llevan a cabo intercambios de información (unidireccionales o bidireccionales) entre los componentes a los que apuntan.

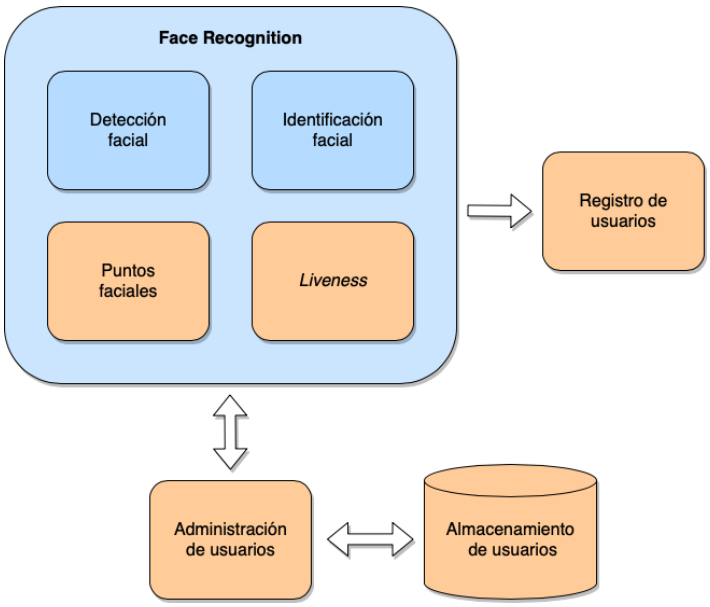

Figura 3.2: Esquema del sistema de reconocimiento facial a desarrollar. Elaboración propia.

Tal y como se ha mencionado en la especificación de requisitos, el bloque de administración de usuarios permitirá añadir y eliminar usuarios del sistema, el de persistencia deberá guardar la información de cada usuario añadido al sistema, como por ejemplo, el nombre, los apellidos, el número de documento de identificación, etc. y el de salida tendrá que escribir el nombre de cada usuario detectado junto con la fecha y la hora de detección en un archivo (*log*).

Al sistema de reconocimiento facial se le realizarán pruebas parciales de caja negra conforme se vayan añadiendo funcionalidades y al terminar se le realizarán pruebas globales para verificar que todos los componentes funcionan de la manera esperada.

CAPÍTULO 4
Diseño de la solución

En este capítulo se realiza el diseño de la solución, es decir, se detallan las tecnologías que se van a emplear para desarrollar la solución así como las funcionalidades del sistema de reconocimiento facial que se llevarán a cabo con cada una de estas tecnologías. Así mismo se presenta la estructura de directorios del sistema.

4.1 Tecnologías utilizadas

Esta sección presenta las principales tecnologías utilizadas en el desarrollo del sistema de reconocimiento facial: **Python**, **OpenCV-Python**, **Face Recognition**, **Tkinter**, **JSON**, **Numpy** y **Playsound**.

4.1.1. Python

Python [47] es un lenguaje de programación, creado por Guido van Rossum, interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipificación dinámica, tipos de datos dinámicos de muy alto nivel y clases. Admite múltiples paradigmas de programación más allá de la programación orientada a objetos, como la programación funcional y de procedimientos. Python combina una potencia notable con una sintaxis muy clara. Tiene interfaces para muchas llamadas del sistema y bibliotecas, así como para varios sistemas de ventanas, y es extensible en C o C ++. También se puede utilizar como lenguaje de extensión para aplicaciones que necesitan una interfaz programable. Finalmente, Python es portable: se ejecuta en muchas variantes de Unix, incluidos Linux y macOS, y en Windows.

4.1.2. OpenCV-Python

OpenCV-Python [39] es una adaptación de la implementación original en C ++ de la biblioteca OpenCV (biblioteca de software de visión artificial y aprendizaje automático de código abierto, *Open Source Computer Vision Library*) para resolver problemas de visión por computador usando el lenguaje de programación Python.

Aunque comparado con lenguajes como C / C ++, Python es más lento, OpenCV-Python tiene dos ventajas: primero, el código es tan rápido como el código C / C ++ original (ya que es el código C ++ real que funciona en segundo plano) y segundo, es más fácil de codificar en Python que en C / C ++, el código es más simple y legible y permite al programador expresar ideas en menos líneas de código sin reducir la legibilidad. Además, OpenCV-Python hace uso de Numpy, que es una biblioteca altamente

optimizada para operaciones numéricas. Todas las estructuras de matrices de OpenCV se convierten a matrices Numpy. Esto también facilita la integración con otras bibliotecas que usan Numpy.

4.1.3. Face Recognition

Face Recognition es una biblioteca de reconocimiento facial, la más sencilla del mundo según [21, 22, 23], para reconocer y manipular caras desde Python o desde línea de comandos. Ha sido creada con el reconocimiento facial de última generación de Dlib[1], desarrollado con aprendizaje profundo y su modelo tiene una preción del 99,38 % en el banco de pruebas *Labeled Faces in the Wild* (LFW).

Detección facial

Face Recognition [20, 23] permite la detección de caras mediante su función «face_locations» que devuelve una lista de cuatro tuplas (x, y) por cada cara detectada con las coordenadas de las esquinas de los marcos que enmarcan las caras detectadas. El primer parámetro de esta función es la imagen como matriz numpy; el segundo, un número para indicar el número de veces que se va a aumentar la imagen para tratar de encontrar caras (valor uno por defecto) y el tercero es el modelo que se va a usar para detectar caras: «hog» (por defecto) o «cnn». El modelo de detección basado en el Histrograma de Gradientes Orientados (*HOG*, *Histogram of Oriented Gradients*) [13] (considera la dirección de los cambios de brillo de la imagen), es menos preciso pero más rápido en CPUs (*Central Processing Units*). El modelo CNN es un modelo de aprendizaje profundo más preciso, acelerado por GPU (*Graphics Processing Unit*) o CUDA (*Compute Unified Device Architecture*). Ambos modelos o detectores son importados de la biblioteca Dlib.

Posicionamiento y proyección de caras

Esta biblioteca también tiene una función llamada «face_landmarks» que devuelve una lista de diccionarios Python con las localizaciones (puntos de referecnia faciales) de las características faciales (nariz, ojos, etc.) para cada cara de la imagen. El tercer parámetro de esta función («model») puede ser «large» (por defecto) o «small». La opción «large» devuelve las coordenadas de los 68 puntos de referencia faciales (*landmarks*) [31] (v. Figura 4.1) calculados por un predictor de Dlib mientras que la opción «small» solamente devuelve cinco pero es más rápido. Esta función puede usarse para cambiar de posición y proyectar caras haciendo transformaciones como por ejemplo, rotar o escalar la cara, para así poder centrar (alinear, normalizar) e identificar una cara.

[1] http://dlib.net

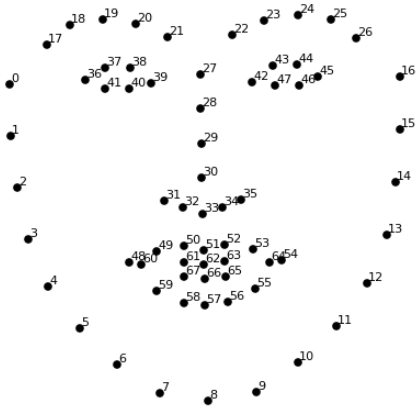

Figura 4.1: Los 68 puntos de referencia faciales (*face landmarks*). Recuperada de [20].

Codificación facial

Antes de identificar una cara es necesario obtener una medida o medidas de esa cara y de otras caras para compararlas o extraer características. Face Recognition tiene una función con nombre «face_encodings» que utiliza un modelo previamente entrenado (la red neuronal ya ha aprendido a generar de manera fiable las medidas o dimensiones para cada cara de cada persona) junto con los puntos de referencia faciales (*face landmarks*) para devolver una lista de codificaciones de caras de 128 dimensiones o medidas (una lista para cada cara de la imagen).

Identificación facial

Para la identificación y verificación facial en Face Recognition se usa la función «compare_faces», que compara una lista de caras codificadas con una cara codificada para ver si hay coincidencias (devuelve una lista de valores *True* o *False* según haya coincidencias o no, respectivamente) y también se usa la función «face_distance» dada una lista de codificaciones de caras, las compara con una codificación de una cara conocida y obtiene la distancia euclidiana para cada cara de comparación. La distancia indica cuan similares son las caras.

4.1.4. Tkinter

El paquete tkinter («interfaz Tk») [48] es la interfaz estándar de Python para el kit de herramientas Tk GUI. Tanto Tk como tkinter están disponibles en la mayoría de las plataformas Unix, así como en los sistemas Windows. (Tk en sí no es parte de Python; se mantiene en ActiveState).

4.1.5. JSON

JSON (*JavaScript Object Notation*) [28] es un formato ligero de intercambio de datos. Es fácil de leer y de escribir para los humanos, y de analizar y de generar para las máquinas. Se basa en un subconjunto del estándar de lenguaje de programación JavaScript ECMA-262 3ª edición - diciembre de 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son familiares para los programadores de la familia de lenguajes C, incluido C, C ++, C #, Java, JavaScript, Perl, Python

y muchos otros. Estas propiedades hacen de JSON un lenguaje ideal para el intercambio de datos.

JSON se basa en dos estructuras:

- una colección de pares de nombre / valor (v. Figura 4.2). En varios lenguajes, se conoce como un objeto, registro, estructura, diccionario, tabla hash, lista con clave o matriz asociativa;

- una lista ordenada de valores. En la mayoría de los lenguajes, esto se realiza como una matriz, vector, lista o secuencia.

Estas son estructuras de datos universales. Prácticamente todos los lenguajes de programación modernos los admiten de una forma u otra. Tiene sentido que un formato de datos que sea intercambiable con lenguajes de programación también se base en estas estructuras.

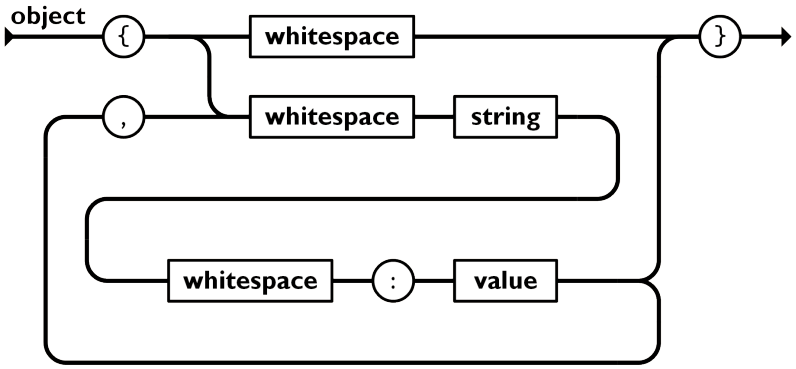

Figura 4.2: Diagrama de la estructura de objeto JSON. Recuperada de [28].

4.1.6. NumPy

NumPy [40] es un proyecto (biblioteca) de código abierto que tiene como objetivo permitir la computación numérica con Python. Fue creado en 2005, basándose en el trabajo inicial de las bibliotecas Numerical y Numarray. NumPy es un software 100 % de código abierto, de uso gratuito para todos y publicado bajo los términos liberales de la licencia BSD modificada y se desarrolla abiertamente en GitHub, a través del consenso de NumPy y de la comunidad científica de Python en general.

4.1.7. Playsound

Playsound [37] es un módulo de función única, multiplataforma, todo implementado en Python y sin dependencias para reproducir sonidos.

El módulo Playsound contiene solo una cosa: la función (también llamada) playsound. Requiere un argumento: la ruta al archivo con el sonido que le gustaría reproducir. Puede ser un archivo local o una URL. Hay un segundo argumento opcional, block, que se establece en *True* de forma predeterminada. Establecerlo en *False* hace que la función se ejecute de forma asincrónica.

4.2 Diseño del sistema de reconocimiento facial

El lenguaje de programación que se utilizará para desarrollar el sistema será Python, más concretamente, Python 3.8.3. Las funciones de la biblioteca OpenCV-Python ya se usan en el código base [60] para abrir la cámara de captura de vídeo, cambiar el tamaño de los fotogramas de vídeo (y así procesarlos más rápido), mostrar los fotogramas de vídeo por pantalla en una ventana, dibujar un rectángulo que enmarca la cara detectada, «dibujar» el nombre de la persona identificada sobre el fotograma de vídeo, etc. La funciones de OpenCV-Python se usarán también en la fase de desarrollo para añadir nuevas funcionalidades como manejar los eventos de ratón y poder controlar los botones de la interfaz, desarrollar la interfaz de usuario (barra de progreso de carga inicial, modo usuario, modo administrador y desarrollador, formulario para añadir usuario...), dibujar los puntos faciales sobre la imagen de la cara, gestionar las ventanas del sistema, cargar los iconos de los botones, guardar imágenes en el sistema... Además, se emplearán funciones de OpenCV-Python para hacer pruebas de detección de rostros rotando los fotogramas capturados, cambiando su brillo, su contraste y su resolución, así como para calcular el tiempo de procesamiento de las imágenes de caras. Ciertas funcionalidades se han desarrollado con OpenCV-Python, y no con otra herramienta, para explorar las posibilidades de la biblioteca y para que el sistema sea portable al mayor número de plataformas.

La biblioteca de Face Recognition [21, 22] se usa para cargar imágenes de caras, detectar caras, calcular medidas faciales, calcular la distancia euclidiana entre caras y comparar una cara con otras etiquetadas (para poder identificar la primera). En el desarrollo se usará para evitar que una imagen que no sea de una cara se introduzca en el sistema, conseguir las coordenadas de los puntos faciales y detectar el guiño de un ojo de la persona capturada por la cámara (e indicar que no es un artefacto inanimado). En resumen, básicamente, se utilizará Face Recognition para detectar e identificar caras y para detectar *liveness*. Tkinter servirá tanto para obtener la resolución de la pantalla que está ejecutando el sistema de reconocimiento facial (y así situar los elementos de la interfaz: botones, texto... en relación al tamaño de la pantalla) como para crear cajas de diálogo (información, advertencia, seleccionar imagen de un rostro del equipo...).

En la etapa de desarrollo se creará un archivo con extensión JSON para almacenar la información de los usuarios añadidos al sistema con la estructura matriz (*array*) de objetos (conjunto desordenado de pares de nombre / valor) (v. Figura 4.2). Cada objeto representará a un usuario distinto, y en cada par del objeto, el nombre será la etiqueta de cada dato o valor a almacenar del usuario como por ejemplo «nombre», «apellidos» y «número de documento de identidad» (identificador). Se ha decidido utilizar JSON porque es flexible, sencillo y ligero.

Numpy se utiliza en el código base para encontrar el índice de menor valor en la matriz de distancias faciales mediante su función argmin y, en el desarrollo, se va a usar para «dibujar» el fondo de pantalla de la interfaz del modo usuario y de los formularios utilizando matrices de intensidades de color. Se recurrrirá a Playsound para reproducir sonidos tales como el de disparo de una fotografía. El archivo de registro de usuarios detectados (*log*) tendrá extensión CSV y en él se escribirá, cada cierto tiempo, el nombre del usuario detectado (*Unknown* en su defecto) junto con la fecha y la hora de detección mediante el módulo Datetime de Python.

La estructura de directorios del sistema (v. Figura 4.3) estará formada por un directorio padre, llamado «face_recognition», y dentro de él, un archivo, «app.py», con el código del sistema de reconocimiento facial, un directorio con nombre «Images», donde se colocarán todos los iconos, un directorio «known_people», en el cual se guardarán las imágenes de las caras de los usuarios registrados en el sistema, un archivo con extensión JSON,

«known_people_database.json», para guardar la información de cada usuario añadido al sistema, un archivo de *log*, «log.csv», un archivo de texto, «README.txt», con información acerca del uso del sistema y otro, llamado «Sounds», para almacenar los archivos de sonido.

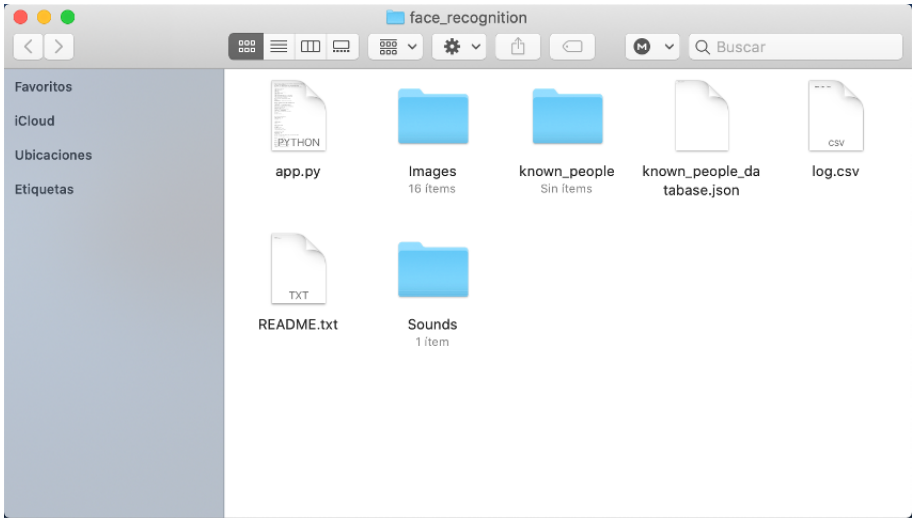

Figura 4.3: Estructura de directorios del sistema de reconocimiento facial. Elaboración propia.

CAPÍTULO 5
Desarrollo del sistema de reconocimiento facial

Antes que nada, y a modo de repaso, cabe explicar más detalladamente qué hace el código base de Face Recognition [60] una vez que se ejecuta desde la consola de sistema (o interfaz de línea de comandos) utilizando la orden «python3 recognise_face.py».

Este programa requiere que hayan imágenes de caras con extensión «.jpg» en la carpeta «known_people», la cual se debe encontrar en el mismo directorio que el archivo «recognise_face.py». Al ejecutarse, el programa carga estas imágenes (como matrices numpy) y calcula las medidas faciales de las caras que aparecen en ellas (mediante la función «face_encodings»). Esta información se guarda en el diccionario de variables globales («globals») al inicio del código, fuera del bucle «while» (bucle principal) que se encarga de hacer la detección e identificación, de mostrar los fotogramas y (dentro de un bucle «for») de mostrar los resultados de la detección e identificación en tiempo real. Fuera de este bucle «while», las medidas faciales también se añaden a una matriz llamada «known_face_encodings», que es la que se va a utilizar para comparar las medidas de las las caras de las imágenes de «known_people» con las medidas de las caras capturadas en tiempo real. Si en alguna imagen de la carpeta «known_people» no se puede detectar una única cara, el programa falla.

Seguidamente, haciendo uso de la cámara que esté conectada al equipo desde el cual se lanza el programa, (dentro del bucle principal) se detectan las caras de los fotogramas capturados y se comparan, utilizando la distancia euclidiana, con las caras de la carpeta «known_people». El nombre de cada una de las imágenes de caras de esta carpeta debería ser el mismo que el nombre de la persona cuya cara aparece en la imagen y la extensión del archivo de imagen debe ser «.jpg».

Finalmente, el programa muestra los fotogramas capturados en una ventana y los resultados de la detección y de la identificación en tiempo real. Es decir, coloca un marco de color rojo sobre las caras detectadas en el fotograma. Si identifica la cara (porque se encuentra en una imagen de la carpeta «known_people» y porque se ha podido detectar) muestra el nombre de la imagen junto con la extensión del archivo debajo del marco y si no identifica la cara, muestra, *Unknown*.

Para salir del bucle principal y cerrar la ventana del programa se ha de pulsar la tecla «Q». En la Figura 5.1 se muestra la ventana del programa base, tras ejecutar el código base de [60], detectando e identificando la cara de Francisco Morcillo Vizuete, alias Paco, autor de esta memoria. En la carpeta «known_people», hay una fotografía suya llamada «Francisco.jpg». Para desarrollar el sistema de reconocimiento facial (SRF), se añadirán a este programa base los componentes que se explican en las siguientes secciones de este capítulo.

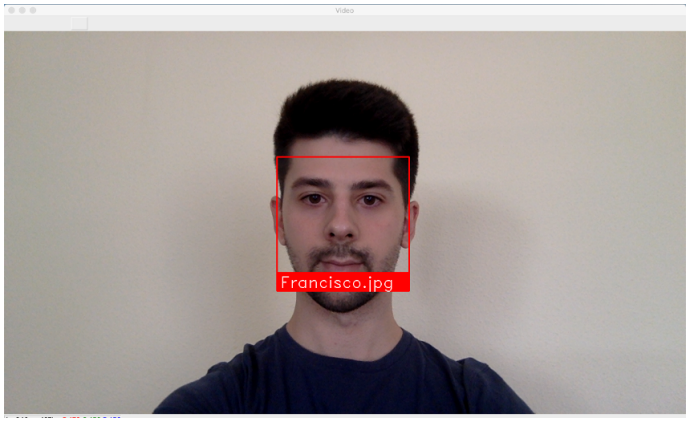

Figura 5.1: Programa base de reconocimiento facial [60]. Elaboración propia.

5.1 Carga inicial de imágenes de rostros

Lo primero que se ha añadido al código base es una ventana con una barra de progreso de carga para mostrar al usuario, cuando inicializa el sistema de reconocimiento facial (SRF), la evolución de la carga de las imágenes y del cálculo de las medidas faciales de las imágenes que se encuentran en la carpeta llamada «known_people», la cual se utiliza para almacenar las imágenes de las caras registradas en el sistema.

Se han desarrollado dos modelos de barras de progreso, ambos utilizando el lenguaje Python y las funciones de dibujo de OpenCV-Python («rectangle», «circle» y «putText») para así no depender de terceros y poder mantener la portabilidad. La función encargada de mostrar el progreso de la barra se llama dentro del bucle que carga y codifica cada una de las imágenes de caras que se encuentran en la carpeta «known_people». Uno de los modelos está inspirado en las barras del sistema operativo Windows (v. Figura 5.2) y otro en macOS (v. Figura 5.3). Finalmente, se ha implementado la segundo opción (la inspirada en las barras de progreso de macOS o iOS) en la versión final del SRF.

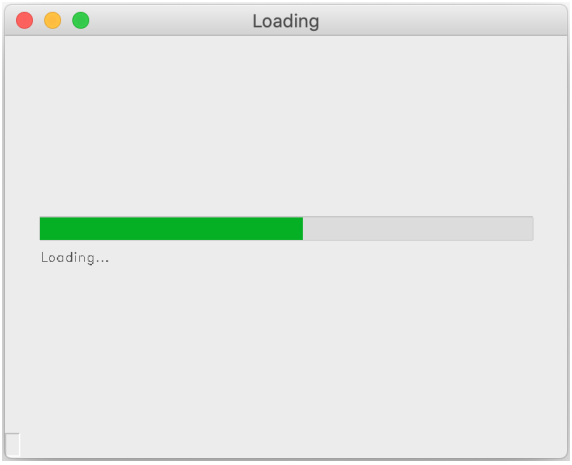

Figura 5.2: Ventana de barra de progreso inspirada en Windows. Elaboración propia.

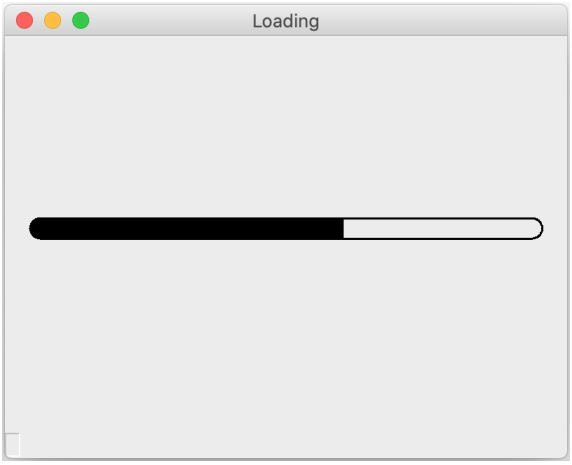

Figura 5.3: Ventana de barra de progreso inspirada en macOS. Elaboración propia.

5.2 Registro de usuarios detectados

La siguiente funcionalidad que se ha implemantado es el registro de usuarios detectados por el SRF en un fichero de *log*. Dicha funcionalidad se consigue mediante una sencilla función que escribe el nombre de la persona cuya cara se ha detectado (*Unknown* si la persona es desconocida para el SRF), junto con la hora y la fecha de detección (que se consiguen con las funciones del módulo «datetime») en un archivo con extensión CSV. Esta función se ejecutará cada determinado número de fotogramas.

En este momento se implementan también algunos pequeños detalles como pueden ser: centrar, mediante la función de OpenCV-Python «getTextSize», los nombres de las etiquetas de las caras detectadas; permitir más extensiones de archivos de imagen (JPG, JPEG y PNG); poder salir del SRF clicando sobre el botón de cerrar ventana y, por último, tomar el nombre que se coloca sobre la etiqueta (para identificar a la persona detectada) de un archivo JSON en vez de tomarlo del nombre del archivo de imagen en el que aparece la cara de la persona detectada.

5.3 Administración de usuarios

La administración de usuarios se puede realizar tanto desde el modo administrador como desde el modo desarrollador. El funcionamiento y las particularidades de cada uno de estos dos modos se explican en la sección 5.3.1. En ambos modos, se ha añadido un botón de menú (centrado) en la parte inferior de la pantalla (v. Figura 5.4). Clicando sobre este botón aparecen otros dos botones que sirven para añadir y eliminar usuarios, respectivamente (v. Figura 5.9). Los tres botones (iconos, imágenes PNG) se han colocado sobre la pantalla utilizando las funciones de OpenCV-Python. Para detectar los clics sobre los botones se utilizan las funciones controladoras de eventos de ratón de OpenCV-Python y las coordenadas (x, y) de la ventana obtenidas mediante Tkinter.

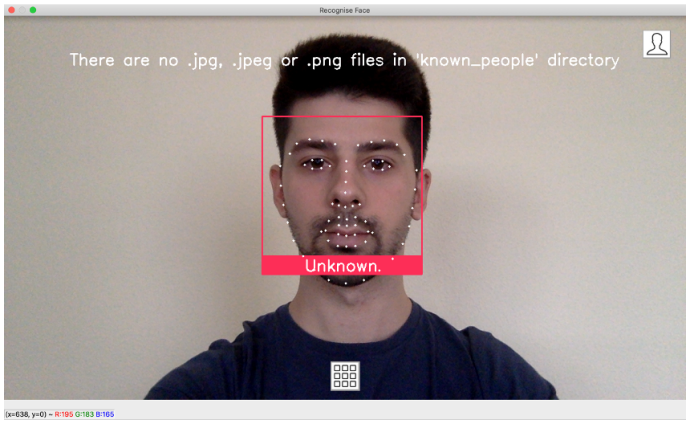

Figura 5.4: Ventana del SRF, al inicio, sin usuarios añadidos (modo administrador). Elaboración propia.

Al clicar sobre el icono de añadir usuario, se abre un formulario (v. Figura 5.5) en una nueva ventana para introducir la información del usuario que se quiera añadir (nombre, apellidos, identificador, rol, ...). El funcionamiento de este formulario se ha programado usando Python y algunas de las funciones de dibujo de OpenCV-Python («line», «rectangle» y «putText»). Una vez rellenado el formulario, para acabar el registro del usuario, se debe añadir una imagen de una cara al SRF. Para ello se han implementado dos opciones. La primera consiste en la captura de una imagen instantánea durante la ejecución del SRF (fotografía) y la segunda consiste en la selección de una imagen guardada en el equipo que está ejecutando el SRF. Para la captura de la imagen instantánea se abre una nueva ventana (v. Figura 5.6), y cuando el usuario pulsa la barra espaciadora, si se detecta una única cara, la captura o foto se guarda en la carpeta «known_people» en formato JPEG, utilizando la función «imwrite» de OpenCV. Para seleccionar la imagen del equipo se ha utilizado la función «askopenfilename» del módulo «filedialog» de Tkinter. Si en la imagen capturada o en la imagen seleccionada no se detecta una única cara, el sistema muestra una ventana de aviso, permitiendo al usuario tomar una nueva foto o seleccionar una nueva imagen, respectivamente, o cancelar la operación.

Figura 5.5: Formulario de «añadir usuario». Elaboración propia.

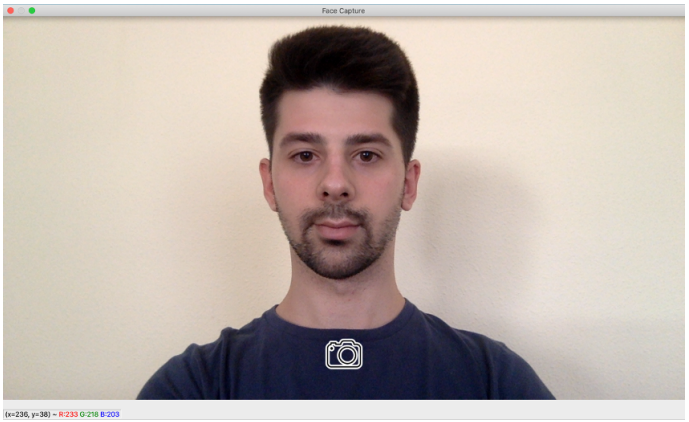

Figura 5.6: Ventana del SRF para captura de imagen de cara. Elaboración propia.

Clicando sobre el icono de eliminar usuario, se muestra en una nueva ventana un formulario con un único campo de texto para introducir el identificador del usuario a eliminar. Una vez introducido el identificador, al seleccionar el botón «Delete» y pulsar la tecla «Entrar» del teclado, se muestra un cuadro de diálogo implementado con Tkinter como el de la Figura 5.7. La manera en que se guarda y se elimina la información y la imagen de los usuarios en la base de datos del SRF (archivo JSON y carpeta «known_people») se explica en la sección 5.4.

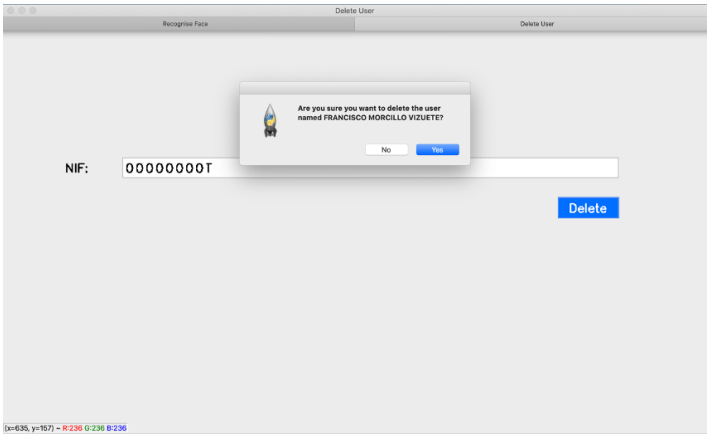

Figura 5.7: Formulario de «eliminar usuario». Elaboración propia.

5.3.1. Modos de usuario

Según se indica en la especificación de requisitos, sección 3.1 de esta memoria, se van a implementar tres modos de uso de acuerdo a los roles o perfiles que puede haber: usuario, administrador y desarrollador. Las flechas bidireccionales de la Figura 5.8 indican cómo se puede cambiar el modo de usuario. Para cambiar al modo desarrollador es necesario tener asignado el rol de desarrollador.

Figura 5.8: Modos del SRF. Elaboración propia.

Modo administrador

El modo administrador es el que se muestra por defecto al inicializar el SRF. Si al arrancar el SRF no hay ningún usuario registrado, se muestra una caja de diálogo de Tkinter, mediante la función «showwarning» del módulo «messagebox», avisando de esto. También aparece en la parte superior de la ventana principal un aviso que se retira al añadir la primera imagen. Sobre los fotogramas de todas las caras detectadas se muestran los 68 puntos de referencia faciales, utilizando la función «face_landmarks» de Face Recognition para obtener las coordenadas y la función «circle» de OpenCV-Python para «dibujar» los puntos sobre el fotograma. Las caras detectadas que no puedan ser identificadas se enmarcan con un cuadro rojo y se etiquetan como *Unknown* (v. Figura 5.9), usando las funciones de dibujo de OpenCV. Cada cara identificada (también detectada, por supuesto) se enmarca con un cuadro de color verde y se etiqueta con la cadena de caracteres que se haya introducido en el campo «Name» del formulario de ese usuario (v. Figura 5.5 y 5.10). Este modo permite tanto añadir y eliminar usuarios como cambiar el SRF al modo desarrollador (solamente si el rol del usuario detectado es «Developer») clicando en el icono de la esquina superior izquierda y, al modo usuario, clicando sobre el de la esquina superior derecha. Tanto el modo desarrollador como el modo usuario son como «capas» sobre el modo administrador.

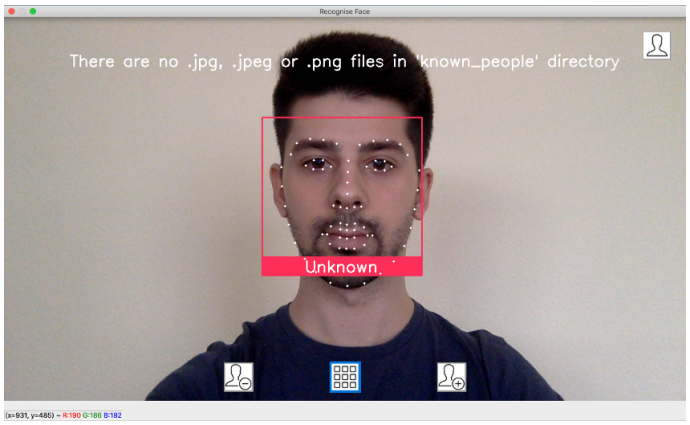

Figura 5.9: Ventana del SRF tras clicar sobre el botón de menú, sin usuarios registrados (modo administrador). Elaboración propia.

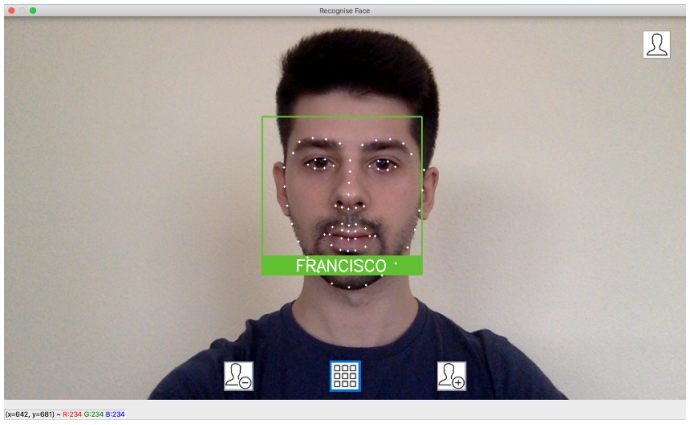

Figura 5.10: Ventana del SRF identificando a un usuario registrado (modo administrador). Elaboración propia.

Modo usuario

El modo usuario se ha creado mostrando una imagen de color blanco (fondo) utilizando la función «ones» de numpy, en vez de los fotogramas capturados por la cámara, con un icono en el centro para mostrar el progreso del reconocimiento facial al usuario. La animación del icono se crea usando seis imágenes diferentes y las funciones «imread» y «imshow» de OpenCV-Python (v. Figura 5.11). Mientras se detecta la cara de una persona, el icono de progreso avanza, «pintándose» de color azul. Si no, vuelve al estado inicial (icono entero de color negro). Si se identifica la cara se autoriza el acceso al usuario y aparece una frase de bienvenida y si no se informa de que el acceso ha sido denegado (v. Figura 5.12). En la esquina superior derecha de la ventana del modo usuario se ha colocado un icono para poder pasar al modo administrador.

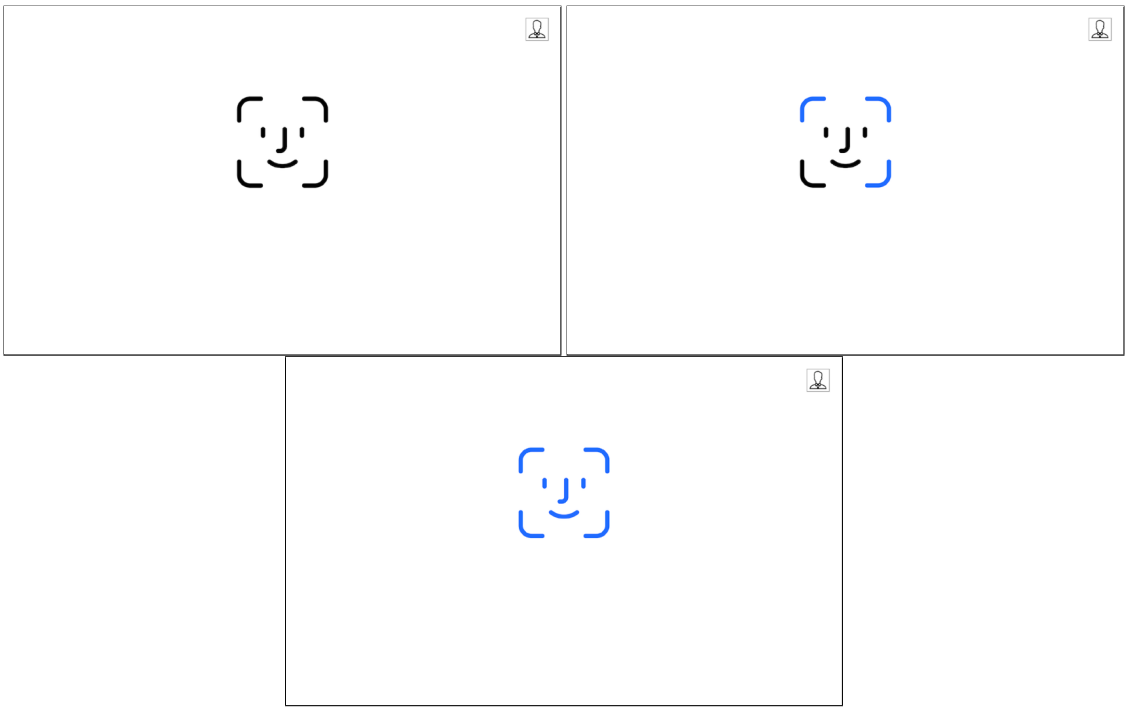

Figura 5.11: Ventanas del SRF reconociendo una cara (modo usuario). Elaboración propia.

Figura 5.12: Ventanas del SRF de acceso autorizado y acceso denegado, respectivamente (modo usuario). Elaboración propia.

Modo desarrollador (*Liveness*)

El modo desarrollador se ha creado a partir del modo administrador y se le ha añadido la funcionalidad de detección de *liveness* (detección de persona viva) mediante las líneas de código que aparecen a continuación.

```
# LIVENESS
if face_locations != [] and faces_landmarks != []:
    l_ey = faces_landmarks[0]['left_eye'][4][1]- faces_landmarks[0]['left_eye'][2][1]
    r_ey = faces_landmarks[0]['right_eye'][4][1]- faces_landmarks[0]['right_eye'][2][1]
    ch = faces_landmarks[0]['chin'][16][0]- faces_landmarks[0]['chin'][0][0]
    left_ratio = l_ey/ch
    right_ratio = r_ey/ch

    if left_ratio < 0.04 and right_ratio > 0.04:
        frame = overlay_transparent(frame, heart_icon, width_px, height_px)
    print("Person winked the right eye")
```

El objetivo de estas líneas de código es detectar el guiño de del ojo derecho usando los puntos de referencia faciales que se muestran en la Figura 4.1. Se obtiene la distancia entre párpados de cada uno de los ojos y la distancia entre dos puntos de la cara que mantengan constante su posición al guiñar un ojo, como en este caso, la anchura de la cara a la altura de los ojos. Después se calculan las ratios distancia entre párpados de cada ojo y anchura de la cara y se comparan con un índice umbral obtenido mediante pruebas. Las ratios se utilizan para relativizar las medidas y que el SRF no confunda un alejamiento de una cara a la cámara con un guiño de ojo. La función «overlay_transparent» coloca el icono del corazón sobre el fotograma de vídeo en las coordenadas (píxeles) «width_px» y «height_px».

Para entrar en el modo desarrollador del SRF es necesario que se detecte un usuario con rol «Developer». Una vez detectado un usuario con este rol, aparece un icono en la esquina superior izquierda de la ventana (v. Figura 5.13). Se pasa al modo desarrollador haciendo clic sobre este icono. El borde azul de este icono significa que se ha pasado al modo desarrollador. Cuando el usuario guiña el ojo derecho, se muestra un icono de un corazón rojo en la esquina superior derecha de la ventana indicando que el usuario no es un artefacto inanimado, es decir, que tiene vida (v. Figura 5.14). Se ha decidido poner como respuesta a la detección *liveness* el icono del corazón pero podría haber sido cualquier otra acción especificada por un cliente externo, como por ejemplo, desbloquear una puerta o un sistema informático.

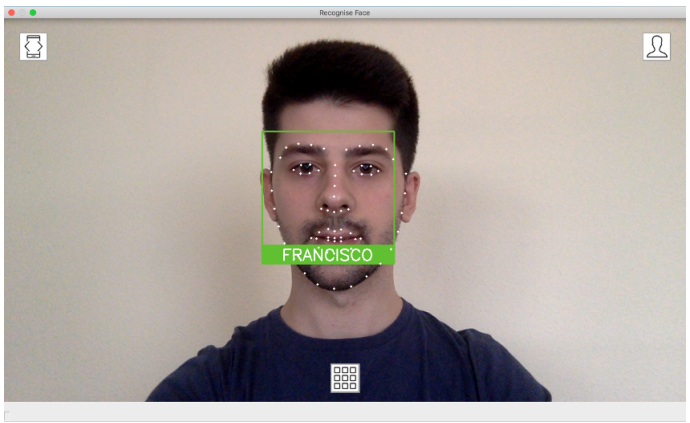

Figura 5.13: Ventana del SRF detectando un usuario con rol de desarrollador (Modo administrador). Elaboración propia.

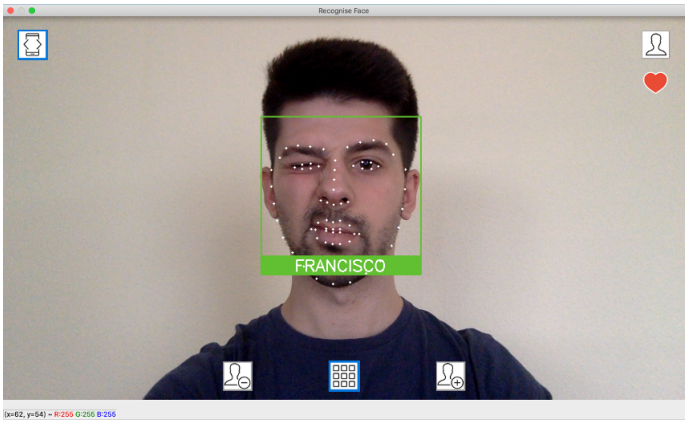

Figura 5.14: Ventana del SRF detectando *liveness* (Modo desarrollador). Elaboración propia.

5.4 Almacenamiento de usuarios

En las siguientes subsecciones de esta sección se explica cómo se añaden y se eliminan usuarios de la base de datos del SRF, compuesta por el directorio «known_people» y por el archivo JSON, llamado «known_people_database.json»

5.4.1. Añadir usuarios

Los caracteres que se introducen en el enésimo campo de texto del formulario de «añadir usuario», se agregan a la enésima fila de una matriz llamada «txt_field» utilizando la función «append» y los caracteres que se borran del formulario también se eliminan de la matriz con la función «pop». Cada carácter se guarda en una posición de la matriz. En una fila, los caracteres se guardan en posiciones de la matriz consecutivas conforme se teclean. Es decir, cada carácter del formulario se localiza en la matriz mediante su fila «i» y su columna «j».

Cuando se selecciona una imagen del equipo, la función «askopenfilename» de «filedialog» de Tkinter devuelve la ruta de la imagen. Si esta imagen es válida (imagen de una única cara que se pueda detectar), se copia en el directorio «known_people» en formato JPEG mediante la función «imwrite» de OpenCV con un nombre que se construye llamando a la siguiente función («get_image_name»), que dado el número de imágenes que hay en la carpeta «known_people» («n_known_people»), devuelve un nombre para la nueva imagen, concatenando el actual número de imágenes con la fecha, hora y la extensión «.JPEG»:

```
1 def get_image_name(n_known_people):
2     now = datetime.now()
3     dtstr = now.strftime('%m%d%Y%H%M%S')
4     image_name =  str(n_known_people) + dtstr + ".JPEG"
5     return image_name
```

Una vez guardada la imagen, esta se carga como array numpy, se codifica (se calculan las medidas faciales) y se guardan como variables globales en el diccionario de variables globales. La variable con las medidas faciales se agrega a la matriz de caras conocidas codificadas, «known_face_encodings». Con esto se logra que no sea necesario reiniciar el SRF para que pueda identificar la cara recientemente añadida.

A continución, mediante una función, se concatenan los caracteres de las filas de la matriz «txt_field» mencionada anteriormente en esta sección, para formar las palabras

(usando la función «join»). Estas palabras, que coinciden con la información introducida por el usuario en el formulario, se guardan como valores en un diccionario Python para posteriormente guardarlo dentro de una matriz, llamada también «known_people», en un objeto JSON. En la Figura 5.15 se muestra el archivo JSON tras haber añadido a Francisco Morcillo Vizuete como desarrollador, a Jeff Bezos como administrador y a Elon Musk como usuario. El valor de la clave «image_face» del objeto que representa a cada usuario es el nombre de la imagen de cara obtenido de la función «get_image_name».

known_people_database.json

```
{
    "known_people": [
        {
            "first_name": "FRANCISCO",
            "nickname": "PACO",
            "last_name": "MORCILLO VIZUETE",
            "nif": "00000000T",
            "sex": "M",
            "date_of_birth": "12/03/1997",
            "country": "SPAIN",
            "city": "VALENCIA",
            "rol": "DEVELOPER",
            "image_face": "009112020031849.JPEG"
        },
        {
            "first_name": "JEFFREY",
            "nickname": "JEFF",
            "last_name": "BEZOS",
            "nif": "11111111H",
            "sex": "M",
            "date_of_birth": "12/01/1964",
            "country": "USA",
            "city": "MEDINA",
            "rol": "ADMINISTRATOR",
            "image_face": "111012020020518.JPEG"
        },
        {
            "first_name": "ELON",
            "nickname": "ELON",
            "last_name": "MUSK",
            "nif": "22222222J",
            "sex": "M",
            "date_of_birth": "28/06/1971",
            "country": "USA",
            "city": "BEL-AIR",
            "rol": "USER",
            "image_face": "211012020021016.JPEG"
        }
    ]
}
```

Figura 5.15: Archivo JSON. Objeto JSON formado por una matriz («known_people») de objetos JSON (usuarios). Elaboración propia.

El procedimiento para guardar una imagen que provenga de una captura instantánea (fotografía) es prácticamente el mismo que el mencionado para guardar una imagen seleccionada del equipo. La única diferencia es que la imagen que se guarda mediante la función «imwrite» en la carpeta «known_people» es un fotograma del vídeo que se muestra en la ventana que se ha abierto para tomar la fotografía (v. Figura 5.6).

5.4.2. Eliminar usuarios

El formulario de «eliminar usuario» tiene el mismo funcionamiento que el formulario de «añadir usuario» (v. sección 5.4.1) pero, en vez de tener varios campos, tiene uno solo, que sirve para introducir el identificador del usuario a eliminar. Una vez se consigue el identificador del usuario a eliminar, se busca el usuario que tiene dicho identificador en el archivo JSON y se elimina la imagen de la cara de ese usuario de la carpeta «known_people» usando la función «remove» del módulo «os». Después, se ha de eliminar la información del usuario del archivo JSON (objeto JSON) y de las variables globales. Para poder eliminar la información de las variables globales, cada vez que se ha introducido un nuevo usuario o se ha inicializado el SRF, se ha guardado en un diccionario global de Python llamado «del_dict» el identificador del usuario, como clave, y un número entero (índice) como valor. Este índice se hace coincidir con otros tres valores: el que se encuentra en el nombre de la variable global que guarda la imagen de la cara como matriz numpy; el que se encuentra en el nombre de la variable global que guarda las medidas faciales y el de la posición donde se encuentran las medidas faciales de los usuarios en la matriz «face_encodings», utilizada para comparar e identificar caras. A continuación, para clarificar lo anterior, se muestra el código que guarda la información de las imágenes de cada usuario registrado al inicializarse el SRF. La función «read_data_from_json» devuelve el identificador del usuario.

```
1  for i in range(number_files):
2      globals()['image_{}'.format(i)] = face_recognition.load_image_file(
           list_of_files[i])
3      globals()['image_encoding_{}'.format(i)] = face_recognition.face_encodings(
           globals()['image_{}'.format(i)])[0]
4      known_face_encodings.append(globals()['image_encoding_{}'.format(i)])
5
6      # Create array of known names
7      names[i] = names[i].replace("known_people/", "")
8      known_face_names.append(names[i])
9
10     # Create dict to delete image variables
11      del_dict[read_data_from_json('known_people_database.json', names[i], 'nif'
            )] = del_val_dict
```

A partir del identificador del usuario a eliminar se puede conseguir el índice del diccionario Python. Con este índice ya se pueden eliminar las variables globales. Para borrar un objeto JSON (que representa a un usuario) se recorren los objetos comparando los identificadores hasta encontrar el objeto buscado.

CAPÍTULO 6
Pruebas de rendimiento

Este capítulo describe las pruebas de rendimiento realizadas al sistema de reconocimiento facial con el fin de conocer en cuánto tiempo es capaz de reconocer caras y en qué medida influye la variación de diferentes factores o parámetros del código en su precisión y velocidad de detección e identificación de caras. Estas pruebas se han realizado únicamente con el mismo ordenador con el que se ha desarrollado el SRF, un MacBook Pro, cuya especificación se detalla en el Apéndice A.

6.1 Pruebas de tiempos de ejecución

Estas pruebas se han realizado utilizando las funciones de OpenCV-Python [42] diseñadas para ello: «getTickCount» y «getTickFrecuency». La primera de ellas, «getTickCount», devuelve el número de ciclos de reloj hasta el momento en que se llama a esta función. Entonces, si se llama antes y después de la ejecución de una función o de una parte de código, se puede obtener el número de ciclos de reloj utilizados para ejecutar esta función o esta parte del código. La segunda, «getTickFrecuency», devuelve la frecuencia de los ciclos de reloj o el número de ciclos de reloj por segundo. Por tanto, el tiempo de ejecución en segundos se puede calcular de la siguiente manera:

```
e1 = cv.getTickCount()
# your code execution
e2 = cv.getTickCount()
time = (e2 - e1)/ cv.getTickFrequency()
```

Entonces, utilizando el código anterior, se van a tomar tiempos de la ejecución de ciertas líneas de código de la inicialización del SRF y del reconocimiento facial con el fin de conocer tiempos de espera del usuario y de averiguar qué dimensiones de fotografía o qué parámetros de funciones pueden ser más convenientes para lograr un mejor rendimiento.

6.1.1. Inicialización

El tiempo que se quiere calcular es el de la ejecución de la parte del código de la inicialización que se encarga de guardar en variables globales tanto las imágenes de caras del directorio «known_people» cargadas en matrices numpy como las medidas faciales (bucle «for» que aparece en la sección 5.4.2 de esta memoria), ya que supone el mayor tiempo de espera para el usuario del SRF y es el que depende del número de fotos de la carpeta «known_people» y de sus dimensiones, como se verá a continuación. Es durante este periodo de tiempo cuando se muestra la ventana con el avance de la barra de progre-

so de la Figura 5.3. Cabe decir que antes de ejecutarse el código a medir, transcurre casi un segundo y que los parámetros opcionales, «num_jitters» y «model», de la función que calcula las medidas faciales, «face_encodings», tienen asignados los valores por defecto.

La imagen que se va a añadir a la carpeta «known_people» para hacer las pruebas es una fotografía de una cara realizada (a unos 30 cm de distancia) con la cámara del ordenador (cuyas especificaciones aparecen en el Apéndice A), y utilizando la funcionalidad de «añadir usuario mediante captura instantánea» del SRF.

En la Tabla 6.1, se muestran los resultados de los cálculos de los tiempos de ejecución de esa parte del código tras haber añadido 1, 3, 12 y 48 veces la misma imagen de la cara a la carpeta «known_people». En la columna «Nº imágenes» se indica el número de veces que se ha copiado la imagen en el directorio «known_people». Las diferencias entre Imagen 1 e Imagen 2 son únicamente el tamaño (bytes) y las dimensiones de las imágenes (píxeles). La Imagen 1 se ha conseguido reduciendo las dimensiones de la Imagen 2 a la mitad.

- **Imagen 1.** Imagen JPEG, 75.822 bytes, 640x360;
- **Imagen 2.** Imagen JPEG, 257.800 bytes, 1280x720.

Nº imágenes	Tiempo con Imagen 1 (s)	Tiempo con Imagen 2 (s)
1	1,7752	2,1713
3	2,2760	3,9440
12	4,0843	8,6700
48	11,4477	30,0122

Tabla 6.1: Tiempo de ejecución de la inicialización en segundos. Elaboración propia.

Los resultados de la Tabla 6.2 se han obtenido haciendo estas mismas pruebas con otra imagen con las siguientes características:

- **Imagen 3.** Imagen JPEG, 48.720 bytes, 1220x1751.

Nº imágenes	Tiempo con Imagen 3 (s)
1	3.0596
3	5.5068
12	16.8732
48	63.2215

Tabla 6.2: Tiempo de ejecución de la inicialización en segundos. Elaboración propia.

Como se observa en las tablas anteriores, el tiempo de ejecución de la inicialización usando la Imagen 3 (v. Tabla 6.2) es mucho mayor que el de la Imagen 1 e Imagen 2 (v. Tabla 6.1), debido a las mayores dimensiones de la Imagen 3. Los tiempos de ejecución obtenidos tras hacer estas mismas pruebas con estas imágenes pero en formato PNG han sido muy similares a los obtenidos con las de formato JPEG. Entonces, se puede decir que el tiempo de ejecución, cuando se usan imágenes de dimensiones no excesivamente grandes, por ejemplo de 640x360, es relativamente bueno. Una vez inicializado el SRF, la detección (e identificación) es prácticamente instantánea.

6.1.2. Reconocimiento facial

En esta sección se muestran los resultados de tomar los tiempos de ejecución de las funciones «face_locations» y «face_encodings», de la biblioteca Face Recognition [23], encargadas de detectar caras en los fotogramas de vídeo y calcular las medidas faciales. Estas dos funciones se ejecutan en el bucle «while» principal y su tiempo de ejecución depende del tamaño del fotograma que ambas toman como parámetro.

Para la realización de las siguientes pruebas independientes se han mantenido constantes los parámetros no representados en las tablas y en todas ellas se ha capturado la imagen de una cara (en las mismas condiciones, por tanto, se podría decir, imagen constante) a unos 30 cm del objetivo de la cámara del ordenador. En todos los casos el SRF ha sido capaz de reconocer la cara capturada al instante de la inicialización.

En la Tabla 6.3, se observan los resultados del tiempo de ejecución de la detección de caras y codificación de los fotogramas de vídeo al realizar variaciones en los valores de los parámetros «fx» y «fy» de la función «resize» de OpenCV, encargada de reducir el tamaño de los fotogramas con el fin de procesarlos y hacer la identificación más rápidamente.

fx	fy	Tiempo (s)
0.1	0.1	0,0390
0.25	0.25	0,0692
1	1	0,5581

Tabla 6.3: Tiempo de ejecución del reconocimiento facial en segundos. Elaboración propia.

De la Tabla 6.3 se extrae, que cuanto más se reduce el fotograma más rápido se procesa. Pero reducir demasiado el tamaño del fotograma, dependiendo del tamaño de la cara (más lejos o más cerca de la cámara), puede provocar que el SRF sea menos preciso.

En la Tabla 6.4, se muestran los resultados del tiempo de ejecución de la localización y codificación de los fotogramas de vídeo, como en el caso anterior, pero al realizar variaciones en los valores de los parámetros «number_of_times_to_upsample» (que indica cuantes veces se amplia la imagen para buscar caras) y «model» de la función «face_locations».

number_of_times_to_upsample	model	Tiempo (s)
0	“hog”	0,0819
1	“hog”	0,0702
2	“hog”	0.1726
0	“cnn”	0.1859
1	“cnn”	0.6713
2	“cnn”	2.7690

Tabla 6.4: Tiempo de ejecución del reconocimiento facial en segundos. Elaboración propia.

Observando los resultados, se puede decir que para reconocer caras a unos 30 cm de una cámara similar a la del ordenador con el que se realizan las pruebas, 0 o 1 son los valores ideales para el parámetro «number_of_times_to_upsample». Como se ve en la Tabla 6.4 y según se explica en la interfaz de programación de aplicaciones (API) de Face Recognition [23], el modelo “hog” es más rápido en CPU (*Central Processing Unit*) que el “cnn”. “cnn” es un modelo de aprendizaje profundo más preciso, el cual puede ser acelerado por GPU / CUDA. Pero Dlib no ha sido compilado con extensiones de CUDA.

En la Tabla 6.5, se muestran los resultados del tiempo de ejecución de la localización de caras y codificación de los fotogramas de vídeo al realizar variaciones en los valores de los parámetros «num_jitters» (cuantas veces se muestrea la cara para codificarla) y «model» (con "small" devuelve 5 puntos de referencia faciales y con "large", 68) de la función «face_encodings».

num_jitters	model	Tiempo de ejecución (s)
0	"small"	0,0719
1	"small"	0,0671
2	"small"	0,0945
0	"large"	0,0717
1	"large"	0,0688
2	"large"	0,0999

Tabla 6.5: Tiempo de ejecución del reconocimiento facial en segundos. Elaboración propia.

Analizando la Tabla 6.5, no se aprecian grandes diferencias entre los distintos casos de prueba. Según la API de Face Recognition y según se intuye en la tabla, cuanto mayor es el número de muestreos, el tiempo de ejecución es mayor pero la codificación es más precisa. Si se usa el modelo pequeño la codificación es más rápida también.

Finalmente, teniendo estos resultados en cuenta, se pueden elegir los valores de los parámetros de estas funciones evaluadas del SRF según si se requiere una mayor precisión o una mayor velocidad de reconocimiento facial. Esto aporta determinida flexibilidad.

6.2 Pruebas de reconocimiento facial

En esta sección se realizan diferentes pruebas para ver cómo influyen diferentes factores en la detección e identificación de caras del SRF.

6.2.1. Rotación de cara sobre el eje horizontal

Las pruebas de rotación de las imágenes de caras sobre el eje horizontal se han hecho usando las funciones de OpenCV-Python: «getRotationMatrix2D», que calcula una matriz afín de rotación 2D y «warpAffine», que aplica una transformación afín a una imagen. A la imagen de una cara se le han aplicado rotaciones, de 5° en 5°, con eje horizontal en el centro de la imagen y perpendicular a ella y los resultados han sido que la función «face_locations» de Face Recognition no puede detectar caras (y por tanto, tampoco se pueden identificar) desde los 110° de rotación hasta los 270°. Se ha visto también que la función «face_encodings», usando esta misma imagen, no puede codificar caras desde los 49° (v. Figura 6.1) hasta los 311°. Sería esta última función la que limitaría el reconocimiento. Así pues, el SRF es capaz de detectar caras con una rotación desde los 0° hasta los 105° y desde los 275° hasta los 360° e identifica caras desde los 0° hasta los 48° (v. Figura 6.1) y desde los 312° hasta los 360°. Esto quiere decir que en el directorio «known_people» solamente se podrán añadir imagénes de caras con estas rotaciones.

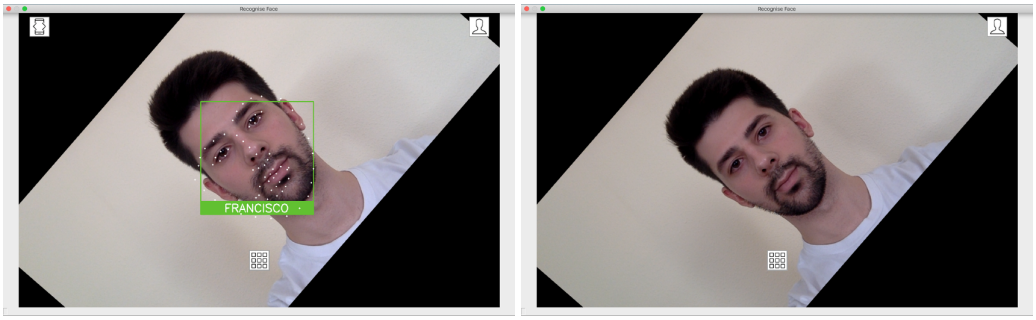

Figura 6.1: Ventanas del SRF con imágenes de cara a 48° y 49° de inclinación, de izquierda a derecha, respectivamente. Elaboración propia.

6.2.2. Cabeceo

Tras realizar númerosas pruebas, se observa que el SRF no es capaz de detectar ni reconocer caras con un cabeceo hacia arriba, hacia abajo, hacia la izquierda o hacia la derecha de con más de unos 15° de inclinación.

6.2.3. Oclusión

El SRF, cuando se oculta parte de la cara, puede ser capaz de reconocer la cara dependiendo de muy pequeñas variaciones de factores como la luz, el ángulo de la cara, el grado de oclusión, etc. Durante las pruebas se ha pasado de reconocer a no reconocer una cara en cuestión de segundos sin cambiar aparentemente las condiciones, como se puede observar en la Figura 6.2.

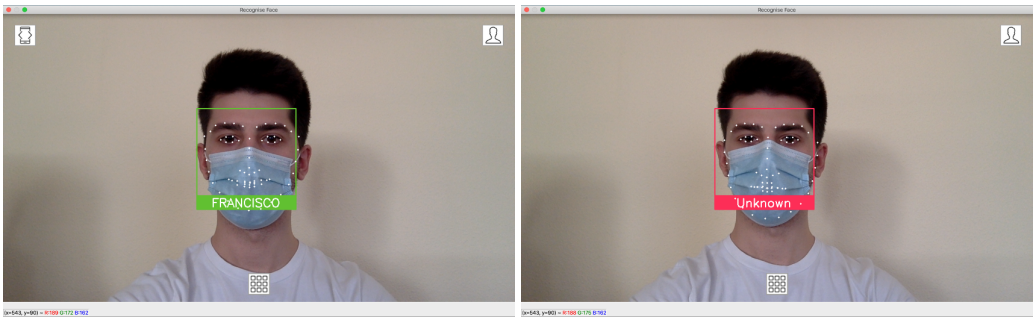

Figura 6.2: Ventanas del SRF con oclusión de cara. Elaboración propia.

6.2.4. Luz

En la Figura 6.3 se observa que el SRF puede reconocer caras en condiciones de baja luminosidad. Se pueden realizar pruebas de luz alterando el brillo y el contraste de la imagen capturada mediante la función «addWeighted» de OpenCV-Python, pero es complicado extraer conclusiones porque intervienen otros factores, como por ejemplo la luz ambiente.

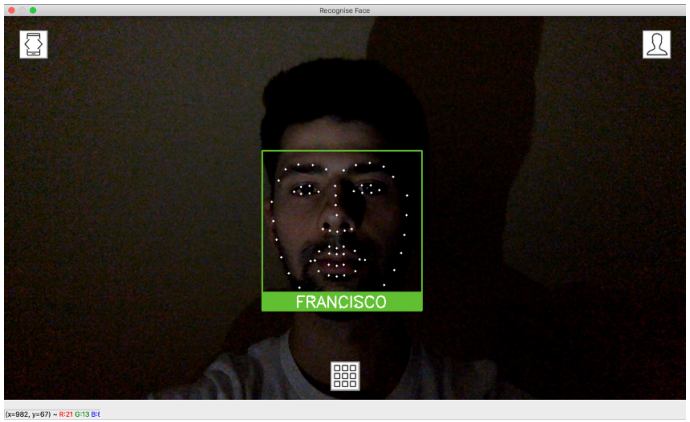

Figura 6.3: Ventana del SRF con imágen de cara en condiciones de baja luminosidad. Elaboración propia.

CAPÍTULO 7
Conclusiones

Todos los objetivos propuestos para la realización de este trabajo se han llevado a cabo de forma satisfactoria y es un orgullo haberlo conseguido. Gracias a este trabajo se ha podido aprender el funcionamiento de un sistema de reconocimiento facial y conocer la potencia de la biblioteca OpenCV en tareas de visión por computador y aprendizaje automático.

En primer lugar, la detección y la identificación de caras se ha conseguido realizar a partir de un programa elemental de reconocimiento facial que utiliza OpenCV y una sencilla biblioteca (Face Recognition) que usa modelos previamente entrenados, pero la selección de este programa fue el resultado de una laboriosa búsqueda y descarte de posibles opciones.

En la administración o gestión de usuarios, se decidió utilizar la biblioteca OpenCV para explorar las posibilidades de esta biblioteca. Aunque OpenCV no está específicamente diseñado para esta función, se ha conseguido implementar los formularios de «añadir usuario» y «eliminar usuario» exitosamente. Debido a las limitaciones que presenta OpenCV para realizar esta tarea se llega a la conclusión que habría sido más eficiente utilizar otra herramienta. Para implementar los cuadros de diálogo se ha utilizado la biblioteca Tkinter, una herramienta que no se conocía antes de realizar el trabajo y que ha resultado muy útil.

El problema de implementar los distintos modos de uso del sistema de reconocimiento facial se ha resuelto desarrollando primero uno de ellos (principalmente con OpenCV y Face Recognition) y luego añadiendo funcionalidades nuevas al mismo como si de una capa se tratara para conseguir los otros modos.

Para acometer el problema de la persistencia de la información de los usuarios se ha utilizado un archivo JSON. Esto ha permitido conocer con más detalle la estructura de un objeto JSON y la lectura y escritura de información usando el lenguaje Python.

La detección *liveness* o detección de personas con vida se resolvió utilizando una función de la biblioteca Face Recognition que detecta los puntos de referencia faciales. Se tomaron medidas faciales relativas para saber si la persona que está intereactuando con el sistema de reconocimiento facial guiña el ojo.

Los principales errores cometidos han sido la planificación del tiempo y la falta de orden del código desarrollado, seguramente debido a la poca experiencia en el desarrollo de aplicaciones pero esto servirá para no cometer los mismos errores en el futuro.

Para trabajos futuros, se puede pensar en una implementación derivada del presente sistema desarrollado, que permita pasar de un despliegue solamente en local a uno en un entorno distribuido, dada su baja dependencia de otras bibliotecas.

7.1 Relación del trabajo desarrollado con los estudios cursados

Aunque en el grado de Ingeniería Informática no he estudiado el lenguaje Python ni la biblioteca OpenCV, gracias a los conocimientos adquiridos sobre paradigmas de la programación, sobre el lenguaje Java, sobre sistemas inteligentes y en general, a todo lo estudiado en el grado, he podido aumentar mis conocimientos a medida que iba realizando el trabajo y desarrollarlo con éxito, cumpliendo todos los objetivos propuestos.

Bibliografía

[1] Agrawal, R. (13 de mayo de 2019). Shallow Neural Networks. Towards data science. Recuperado el 25 de julio de 2020 de https://towardsdatascience.com/shallow-neural-networks-23594aa97a5.

[2] Ahonen, T., Hadid, A. y Pietikäinen, M. (2006). Face description with local binary patterns: application to face recognition. *IEEE transactions on pattern analysis and machine intelligence, 28*(12), 2037–2041. doi:10.1109/TPAMI.2006.244.

[3] Ahonen, T., Pietikäinen, M., Hadid, A. y Mäenpää, T. (2004). Face recognition based on the appearance of local regions. *Proceedings of the 17th International Conference on Pattern Recognition, 2004, 3*, 153-156. doi: 10.1109/ICPR.2004.1334491.

[4] All about Facial Recognition for Businesses. (s.f.). Recuperado el 22 de agosto de 2020 de https://geekflare.com/facial-recognition-for-business/.

[5] AWS. (s.f.). Amazon Rekognition. Recuperado el 18 de octubre de 2020 de https://aws.amazon.com/es/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate&blog-cards.sort-order=desc.

[6] Belhumeur, P. N., Hespanha, J. P. y Kriegman, D. J. (1997). *Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*(7), 711-720. doi: 10.1109/34.598228.

[7] Best facial recognition software. (29 de noviembre de 2019). Recuperado el 22 de agosto de 2020 de https://www.analyticsinsight.net/best-facial-recognition-software/.

[8] Bledsoe, W. W. (1966). The model method in facial recognition.

[9] Brownlee, J. (24 de agosto de 2020). How to Develop a Face Recognition System Using FaceNet in Keras. Machine Learning Mastery. Recuperado el 14 de octubre de 2020 de https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier/.

[10] Cao, Z., et al. (2010). Face recognition with learning-based descriptor. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[11] Chan, T., Jia, K., Gao, S., Lu, J., Zeng, Z. y Ma, Y. (2015). PCANet: A Simple Deep Learning Baseline for Image Classification?. *IEEE Transactions on Image Processing, 24*(12), 5017-5032. doi: 10.1109/TIP.2015.2475625.

[12] Chen, D., Cao, X., Wen, F., y Sun, J. (2013). Blessing of Dimensionality: High-Dimensional Feature and Its Efficient Compression for Face Verification. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 3025-3032.

[13] Dalal, N. y Triggs, B. (2005). Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 1*, 886-893. doi: 10.1109/CVPR.2005.177.

[14] Deep Vision AI. (s.f). Recuperado el 18 de octubre de 2020 de https://www.deepvisionai.com.

[15] Deng, J., Guo, J. y Zafeiriou, S. (2018) Arcface: Additive angular margin loss for deep face recognition.

[16] Dian, Y., Xiao-hong, S., y Hao, X. (2018). The Dropout Method of Face Recognition Using a Deep Convolution Neural Network. *2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*, 1-5.

[17] Face++. (s.f). Recuperado el 18 de octubre de 2020 de https://www.faceplusplus.com.

[18] Facial recognition system. (s.f.). En *Wikipedia*. Recuperado el 15 de julio de 2020 de https://en.wikipedia.org/wiki/Facial_recognition_system.

[19] Freund, Y. y Schapire, R. E. (1999). A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence, 14*(5), 771-780.

[20] Geitgey, A. (24 de julio de 2016). Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning. Medium. Recuperado el 27 de octubre de 2020 de https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78.

[21] Geitgey, A. (2017). Face Recognition. Github. Recuperado el 17 de octubre de 2020 de https://github.com/ageitgey/face_recognition.

[22] Geitgey, A. (2017). Face Recognition. Pypi. Recuperado el 17 de octubre de 2020 de https://pypi.org/project/face-recognition/.

[23] Geitgey, A. (2017). Face Recognition. Welcome to Face Recognition's documentation!. Recuperado el 27 de octubre de 2020 de https://face-recognition.readthedocs.io/en/latest/index.html.

[24] Google Cloud. (s.f.). Productos de IA y aprendizaje automático. Recuperado el 18 de octubre de 2020 de https://cloud.google.com/products/ai.

[25] Gupta, R. (7 de agosto de 2019). Breaking Down Facial Recognition: The Viola-Jones Algorithm. Towards data science. Recuperado el 15 de julio de 2020 de https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999.

[26] He, K., Zhang, X., Ren, S., y Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[27] He, X., Yan, S., Hu, Y., Niyogi, P. y Zhang, H. (2005). Face Recognition Using Laplacian Faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(3), 328-40. doi: 10.1109/TPAMI.2005.55.

[28] JSON. (s.f). Introducing JSON. Recuperado el 18 de octubre de 2020 de https://www.json.org/json-en.html.

[29] Kairos. (s.f.). Recuperado el 18 de octubre de 2020 de https://www.kairos.com.

[30] Kanade, T. (1973). Picture processing system by computer complex and recognition of human faces.

[31] Kazemi, V. y Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus*. doi: 10.1109/CVPR.2014.241.

[32] Krizhevsky, A., Sutskever, I. y Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12), 1*, 1097–1105.

[33] Liu, C. y Wechsler, H. (2002). Gabor feature based classification using the enhanced Fisher linear discriminant model for face recognition. *IEEE Transactions on Image Processing, 2002. 11*(4), 467-476. doi: 10.1109/TIP.2002.999679.

[34] Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., y Song, L. (2017). SphereFace: Deep Hypersphere Embedding for Face Recognition. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6738-6746.

[35] Low, C. (2015). Learning compact discriminant local face descriptor with VLAD. *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 825-833. doi: 10.1109/APSIPA.2015.7415388.

[36] Mallick, S. (30 de octubre de 2015). OpenCV (C++ vs Python) vs MATLAB for Computer Vision. Learn OpenCV. Recuperado el 3 de octubre de https://www.learnopencv.com.

[37] Marks, T. (2016). Playsound. Pypi. Recuperado el 18 de octubre de 2020 de https://pypi.org/project/playsound/.

[38] Moghaddam, B., Wahid, W., y Pentland, A. (1998). Beyond eigenfaces: probabilistic matching for face recognition. *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, 30-35, doi: 10.1109/AFGR.1998.670921.

[39] Mordvintsev, A. y Rahman K., A. (s.f.). Introduction to OpenCV-Python Tutorials. OpenCV. Recuperado el 3 de octubre de 2020 de https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html.

[40] NumPy. (s.f.). About us. Recuperado el 18 de octubre de 2020 de https://numpy.org/about/

[41] OpenCV. (s.f.). About. Recuperado el 18 de octubre de 2020 de https://opencv.org/about/.

[42] OpenCV. (s.f.). Performance Measurement and Improvement Techniques. Recuperado el 7 de noviembre de 2020 de https://docs.opencv.org/master/dc/d71/tutorial_py_optimization.html.

[43] Pardeshi, S. A. y Talbar, S. N. (2011). Local Feature Based Face Recognition. doi: 10.5772/20469.

[44] Parmar, D. N. y Mehta, B. B. (2014). Face Recognition Methods & Applications. *International Journal of Computer Technology & Applications, 4*, 84-86.

[45] Parkhi, O.M., Vedaldi, A., y Zisserman, A. (2015). Deep Face Recognition. *BMVC*, 1, 6.

[46] Ponnusamy, A. (17 de abril de 2018). CNN based face detector from dlib. Towards data science. Recuperado el 14 de octubre de 2020 de https://towardsdatascience.com/cnn-based-face-detector-from-dlib-c3696195e01c.

[47] Python. (s.f). General Python FAQ. Recuperado el 18 de octubre de 2020 de https://docs.python.org/3/faq/general.html#why-is-it-called-python.

[48] Python. (s.f.). Tkinter — Python interface to Tcl/Tk. Recuperado el 18 de octubre de 2020 de https://docs.python.org/3/library/tkinter.html.

[49] Rosebrock, A. (26 de febrero de 2018). Face detection with OpenCV and deep learning. Recuperado el 14 de octubre de 2020 de https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/.

[50] Rosebrock, A. (24 de septiembre de 2018). OpenCV Face Recognition. Pymagesearch. Recuperado el 14 de octubre de 2020 de https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/.

[51] Sandberg, D. (2016). Facenet. Github. Recuperado el 17 de octubre de 2020 de https://github.com/davidsandberg/facenet.

[52] Schroff, F., Kalenichenko, D., y Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 815-823.

[53] SenseTime. (s.f.). Recuperado el 18 de octubre de 2020 de https://www.sensetime.com/me-en.

[54] Serengil, S. I. (2020). Deepface. Github. Recuperado el 17 de octubre de 2020 de https://github.com/serengil/deepface.

[55] Sirovich, L. y Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A, 4*, 519-524.

[56] Sharif, A., Sharif, M., Riaz, S., Shaheen, A. y Badini, M. (2014). FACE RECOGNITION: HOLISTIC APPROACHES AN ANALYTICAL SURVEY. *Science International, 26*(2) 639-642.

[57] Shen, L., Bai, L. y Fairhurst, M. (2007). Gabor wavelets and General Discriminant Analysis for face identification and verification. *Image and Vision Computing, 25*, 553–563.

[58] Shepley, A. J. (2019). Deep Learning For Face Recognition: A Critical Analysis.

[59] Simonyan, K., y Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*.

[60] Smith, R. (30 de septiembre de 2019). A Beginners guide to Building your own Face Recognition System to creep out your Friends. Towards data science. Recuperado el 14 de octubre de 2020 de https://towardsdatascience.com/a-beginners-guide-to-building-your-own-face-recognition-system-to-creep-out-your-friends-df3f4c471d55

[61] Štruc, V., Gajsek, R. y Pavesic, N. (2009). Principal Gabor filters for face recognition. *IEEE 3rd International Conference on Biometrics: Theory, Applications and Systems*, 1-6. doi: 10.1109/BTAS.2009.5339020.

[62] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., y Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1-9.

[63] Taigman, Y., Yang, M., Ranzato, M., y Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1701-1708. doi: 10.1109/CVPR.2014.220.

[64] Top 5 Programming Languages and their Libraries for Machine Learning in 2020. (2020). Recuperado el 3 de octubre de 2020 de https://www.geeksforgeeks.org.

[65] Turk, M. A. y Pentland, A. P. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience, 3*(1), 71-86.

[66] Turk, M. A. y Pentland, A. P. (1991). Face Recognition Using Eigenfaces. *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 586-591. doi: 10.1109/CVPR.1991.139758.

[67] Tussy, K. A., Wojewidka, J. y Rose, J. (s.f.). Biometric Liveness Detection Explained. Liveness.com. Recuperado el 18 de octubre de 2020 de https://www.liveness.com

[68] Uijlings, J., Sande, K., Gevers, T. y Smeulders, A. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision, 104*(2), 154-171. doi: 10.1007/s11263-013-0620-5.

[69] Viola, P. y Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1*, I-511-I-518. doi: 10.1109/CVPR.2001.990517.

[70] Wang, C. (23 de julio de 2018). I Implemented a Face Detection Model. Here's How I Did It. Towars data science. Recuperado el 14 de octubre de 2020 de https://towardsdatascience.com/mtcnn-face-detection-cdcb20448ce0.

[71] Wang, M. y Deng, W. (2018). Deep Face Recognition: A Survey.

[72] Wiegers, K. E. y Beatty, J. (2013). *Software requirements*.

[73] Wright, J. *et al.* (2009). Robust Face Recognition via Sparse Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 31*(2), 210-227.

[74] Yuille, A. L., Cohen, D. S. y Hallinan, P. W. (1989). Feature extraction from faces using deformable templates. *Proceedings CVPR '89: IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 104-109, doi: 10.1109/CVPR.1989.37836.

[75] Zhang, L., Yang, M. y Feng, X. (2011). Sparse representation or collaborative representation: Which helps face recognition? *2011 International Conference on Computer Vision*, 471-478, doi: 10.1109/ICCV.2011.6126277.

APÉNDICE A
Especificaciones de la máquina utilizada

El ordenador utilizado tanto como para desarrollar el sistema de reconocimiento facial como para hacer las pruebas es un **MacBook Pro (Retina 13 pulgadas, principios de 2015)** con las siguientes características.

- **Sistema operativo:** macOS Catalina Versión 10.15.7.
- **Procesador:** 2,7 GHz Intel Core i5 de doble núcleo.
- **Memoria:** 8 GB 1 867 MHz DDR3.
- **Disco de arranque:** Macintosh HD.
- **Gráficos:** Intel Iris Graphics 6 100 1 536 MB.