UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

INTEGRACIÓN DE UN SISTEMA DE RECONOCIMIENTO FACIAL EN DISPOSITIVOS MÓVILES BASADO EN OPENCV

GRADO EN INGENIERÍA DE SISTEMAS AUDIOVISUALES

Autor: Mónica García Sevilla

Tutor: Raúl Sánchez Reíllo

Leganés, 1 de septiembre de 2014

A mi familia

Agradecimientos

Quiero dedicar este trabajo a mi familia, especialmente a mis padres Paloma y Fernando y a mi hermano Antonio por el apoyo y el cariño que me han dado siempre.

También se lo dedico a mis amigos de la universidad y a mis amigas del colegio. A Álvaro por hacerme críticas constructivas, y a Jorge por compartir este proceso conmigo, por el apoyo constante y por todo lo que hemos vivido juntos.

Resumen

Gracias a los grandes avances tecnológicos llevados a cabo en las últimas décadas, el uso de dispositivos electrónicos, y especialmente de teléfonos móviles, se ha convertido en una parte casi indispensable de nuestro día a día. Esto es en parte debido a la multitud de funciones que nos ofrecen. Por ello, actualmente el desarrollo de aplicaciones es uno de los negocios más activos. Además estos avances tecnológicos han impulsado algunos campos de investigación, como el de la biometría, con el cual es posible llevar a cabo la identificación de una persona a través de la captura de datos físicos o de comportamiento. La incorporación de los datos capturados al ordenador permite procesar la información más rápidamente, automatizando el proceso de reconocimiento.

En este proyecto se unen ambos campos, el de la telefonía y el de la biometría, incorporando las técnicas desarrolladas para el reconocimiento facial en una aplicación para móviles en entornos Android. Los algoritmos aplicados para el reconocimiento se obtienen a partir de un conjunto de librerías implementadas por OpenCV, una entidad dedicada al desarrollo de código libre para aplicaciones de visión artificial. Además, la aplicación está encapsulada bajo el estándar BioAPI, creado con el fin de unificar los sistemas biométricos, permitiendo el uso de software con distintos dispositivos, independientemente del proveedor de los mismos.

La función principal de la aplicación diseñada consiste en organizar, de forma casi automática, las imágenes almacenadas en un teléfono móvil o tableta, en función de los contactos añadidos por el usuario. Para ello emplea las técnicas de reconocimiento facial, analizando las imágenes almacenadas y creando modelos de decisión.

Abstract

The technological progress experienced in the last decades has completely changed our way of living. Nowadays, electronic devices such as mobile phones or computers are essential in our daily life. This is caused mainly because of the variety of functionalities they provide, being the development of applications one of the most profitable trades right now. Also, these technological advances have affected many areas of investigation, including biometrics, where some data extracted from physiological characteristics or behavior can be enough to identify a person. The introduction of this information in a computer allows quick processing, providing an identification automatically.

In this project, both topics are combined, introducing a biometric system of facial recognition into a mobile phone application in Android. The algorithms used to apply the facial recognition are obtained from OpenCV, an open source library developed for computer vision applications. In addition, the code is encapsulated under BioAPI standard, designed mainly to unify biometric processes allowing software to work with different devices, independently of their vendors.

The main functionality of this application is to organize automatically the images stored in a mobile phone, according to some contacts added by the user. This process is done by applying the algorithms, creating a model of decision.

Índice

Índice de Figuras

Índice de Tablas

Listado de Acrónimos

API	Application Programming Interface (Interfaz de programación de aplicaciones)
BDIR	Biometric Data Interchange Record (Registro de intercambio de datos biométricos)
BioAPI	Biometric Application Programming Interface (Interfaz de programación para aplicaciones biométricas)
BIR	Biometric Information Record (Registro de identificación biométrica)
BSP	Biometric Service Provider (Proveedor de servicios biométricos)
GUTI	Grupo Universitario de Tecnologías de Identificación
IDE	Integrated Development Environment (Entorno de desarrollo integrado)
IEC	International Electrotechnical Commission (Comisión Electrotécnica Internacional)
ISO	International Organization for Standardization (Organización Internacional de Normalización)
NDK	Native Development Kit (Kit de desarrollo nativo)
OpenCV	Open Source Computer Vision (Código abierto de visión artificial)
OS	Operating System (Sistema operativo)
SDK	Software Development Kit (Kit de desarrollo de software)
TFG	Trabajo Fin de Grado
UC3M	Universidad Carlos III de Madrid

1 Introducción

A lo largo de este documento se detalla el desarrollo de la realización del Trabajo Fin de Grado (TFG), presentando los requisitos planteados, el diseño inicial de la solución y las variaciones y cambios introducidos durante el proceso. Además, se introducirá la situación actual de las tecnologías involucradas en el proyecto, así como el de las plataformas empeladas para su desarrollo. Finalmente, se comentarán los resultados obtenidos tras la realización de pruebas y sus limitaciones, y se analizará el impacto del mismo en su entorno.

1.1 Motivación y Objetivos

El principal propósito de este documento es exponer el trabajo llevado a cabo por el alumno para la realización de este proyecto, mostrando la capacidad de resolución de problemas adquirida así como de toma de decisiones, partiendo de los requisitos planteados.

El objetivo de este TFG es el de introducir un sistema de reconocimiento facial en plataformas móviles, concretamente en entornos Android. Para ello, debían emplearse las librerías creadas por la biblioteca OpenCV, de código libre, y destinadas al desarrollo de aplicaciones de visión artificial. La aplicación creada ha sido diseñada con el objetivo de aprovechar este sistema. Otro requisito del trabajo es la encapsulación del proyecto bajo el estándar descrito por BioAPI, cuya finalidad es la de estandarizar el diseño de sistemas biométricos con el fin de permitir la utilización del mismo programa por parte de dispositivos procedentes de diferentes propietarios.

Gracias a los grandes avances tecnológicos realizados en los últimos años, la introducción del uso de sistemas biométricos es cada vez mayor, ya no sólo en el ámbito forense o policial, sino también en nuestro día a día a través de nuestros dispositivos móviles u ordenadores. Hoy en día existen sistemas de desbloqueo para el ordenador a partir de la captura de la huella dactilar, o de desbloqueo del teléfono o tableta aplicando reconocimiento facial. Sin embargo, aún queda un largo recorrido en la evolución e introducción de estos sistemas, siendo todavía reducido su uso. Por ello, una de las principales motivaciones en el desarrollo de este proyecto es aprovechar estos sistemas introduciéndolos en una nueva aplicación para dispositivos móviles, ya que su incorporación en estos entornos da lugar a multitud de nuevas funcionalidades.

Uno de los principales motivos por los que el uso de estas aplicaciones no está tan extendido, es la falta de estandarización de los sistemas, lo cual provoca cierta inseguridad por parte de

los usuarios en cuanto a su privacidad, y dificulta su evolución al desarrollarse programas compatibles únicamente con los dispositivos seleccionados. Por ello, varias entidades buscan establecer una norma o estándar que unifique todos estos sistemas, entre las cuales se encuentra la plataforma BioAPI. Esta se desarrolló con el fin de estandarizar el desarrollo de los procesos biométricos. En este proyecto se utiliza la API definida por esta entidad, encapsulando el proceso de reconocimiento según el estándar, con el fin de facilitar la interoperabilidad del sistema.

1.2 Marco Regulador

Debido a la reciente extensión en el uso de tecnologías biométricas, ya no solo en ámbitos policiales sino también en dispositivos móviles u ordenadores como métodos de identificación del usuario, han surgido una serie de entidades u organizaciones cuyo fin es crear un estándar en el uso de estos sistemas. Sin embargo aún no se ha conseguido establecer un procedimiento fijo, por lo que cada empresa diseña sus aplicaciones empleando una metodología e interfaces propias. Esto dificulta notablemente su uso por otras empresas o con otros dispositivos.

Existe un organismo encargado de la estandarización de estos sistemas biométricos a nivel mundial, el Sub-Comité 17 (SC17) del Comité Técnico Mixto de Tecnologías de la Información (ISO/IEC JTC 1), perteneciente a la Organización Internacional de Estandarización (ISO) y a la Comisión Electrotécnica Internacional (IEC). En EE.UU. cuentan con organismos propios con la misma finalidad, como son el Comité Técnico M1 del *International Committee for Information Technology Standards* (INCITS), el *American National Standards Institute* (ANSI) o el *National Institute of Standards and Technology* (NIST). Sin embargo, algunas entidades no gubernamentales están intentando desarrollar estándares con este mismo propósito, como son Biometrics Consortium, International Biometrics Groups o BioAPI. Esta última, formada por un conjunto de empresas interesadas en el desarrollo de estos sistemas, se encargó de elaborar un estándar que solventase los problemas de interoperabilidad entre dispositivos de distintos propietarios. Para el desarrollo de este TFG, se ha empleado el estándar definido por BioAPI, encapsulando todo el proceso biométrico de reconocimiento facial diseñado para la aplicación móvil bajo esta metodología, adaptándola a un entorno Android.

En 2005, el comité técnico ISO/IEC JTC 1 desarrolló un estándar definiendo un formato para los datos biométricos a intercambiar, el ISO/IEC 19794. A partir del mismo, fueron describiéndose diferentes formatos o métodos de codificación de estos archivos, dependiendo del tipo de datos contenidos. De esta manera se generaron 10 apartados dentro de la norma:

- 19794-1: Framework (en 2011).
- 19794-2: Minucias de los dedos (en 2005).
- 19794-3: Patrones de los dedos (en 2006).
- 19794-4: Imágenes de dedos (en 2005).
- 19794-5: Imágenes de caras (en 2005).
- 19794-6: Imágenes de iris (en 2005).

- 19794-7: Firmas (en 2007).
- 19794-8: Patrón esquemático de los dedos (en 2011).
- 19794-9: Imagen vascular (en 2007).
- 19794-10: Geometría de la mano (en 2007).

El estándar definido en la norma ISO/IEC 19794-5, para el intercambio de imágenes y datos de caras humanas, se ha empleado en este proyecto con el fin de almacenar estos ficheros dentro de la arquitectura de BioAPI. La aplicación e introducción de la norma se detalla en un apartado de la sección de desarrollo, dedicada al proceso de encapsulación.

Es importante mencionar también las leyes implicadas, referentes a la captura de datos de los usuarios, lo cual constituye uno de los aspectos más polémicos que conciernen al uso de los sistemas biométricos. Actualmente se almacenan una gran cantidad de datos personales en sistemas computacionales, por lo que existe un alto riesgo de sufrir ataques informáticos si estos no son protegidos adecuadamente. La obtención de esta información puede tener graves consecuencias, como la falsificación de identidad o el acceso a cuentas bancarias u otros contenidos personales. Por ello es necesaria una regulación en el tratamiento de esta información.

El origen de una ley a favor de la protección de los datos personales se remonta a 1948, con la elaboración a manos de las Naciones Unidas de la Declaración Universal de Derechos Humanos, en la cual se señala: "*Nadie será objeto de injerencias arbitrarias en su vida privada, su familia, su domicilio o su correspondencia, ni de ataques a su honra o a su reputación. Toda persona tiene derecho a la protección de la ley contra tales injerencias o ataques.*"

A lo largo de los años, los países han ido desarrollando sus propias leyes al respecto, buscando establecer los límites, los permisos y las penalizaciones en cuanto al manejo de esta información. Existen dos vertientes principales: la seguida en Europa, en la cual se protege este tipo de información aun cuando el individuo ha fallecido, y el modelo seguido en EE.UU., en el cual estas leyes de protección cesan en el momento en el que la persona muere debido principalmente a fines comerciales. En 1981 la Unión Europea firmó el primer convenio internacional de protección de datos. Este es conocido como "Convenio 108" o "Convenio de Estrasburgo". Más tarde en los 90 se estableció una norma común, la Directiva 95/46/CE, en la cual se detalla la protección del individuo en el tratamiento de sus datos personales y la libre circulación de los mismos. En España, la Ley Orgánica 15 de 1999 establece la Protección de Datos de Carácter Personal.

Estas leyes pretenden proteger al usuario y el uso de su información, y definen cuatro derechos fundamentales, denominados derechos ARCO: Acceso, Rectificación, Corrección y Oposición. Además obligan a las empresas y particulares a avisar sobre el uso de esta información, para lo cual el sujeto debe dar su consentimiento.

La protección de los individuos y la estandarización de estos procesos son de vital importancia para asegurar el progreso en el uso de estos sistemas, protegiendo al usuario y favoreciendo la evolución de estas técnicas.

1.3 Entorno Socio-económico

La aplicación diseñada para la realización de este TFG está orientada principalmente a un uso personal, ya que es una herramienta destinada a organizar las imágenes almacenadas por el usuario en el teléfono. Por ello, su utilización se limitaría al uso de particulares o de alguna empresa con fines puramente comerciales.

Sin embargo, el sistema de reconocimiento elaborado podría encontrar numerosas aplicaciones en otros ámbitos, como puede ser el policial, añadiendo una serie de modificaciones en la aplicación. Eliminando el sistema de importación automática de imágenes y aprovechando la utilización del algoritmo de reconocimiento facial basado en imágenes del teléfono, podría llevarse a cabo una identificación de un criminal reincidente a partir de una imagen tomada con la cámara. En el teléfono podrían almacenarse carpetas con imágenes de los mismos, y utilizarse en un proceso de identificación o verificación. La principal ventaja que supondría sería el rápido acceso a una aplicación de reconocimiento a partir de un dispositivo portátil y pequeño como puede ser un teléfono móvil o una tableta, los cual pueden transportarse fácil y cómodamente.

También sería posible aumentar el rendimiento de la aplicación utilizando más datos que los obtenidos con el análisis de las imágenes. Estos datos, como puede ser el color del cabello, facilitarían el proceso de reconocimiento al aportar más información a la imagen, y se podrían introducir manualmente, guardándolos en los archivos BIR de BioAPI, en los campos establecidos para ello según la norma ISO/IEC 19794-5.

1.4 Estructura del Documento

Al inicio de este documento, se busca introducir al lector en las tecnologías implicadas en el desarrollo de este proyecto así como el de las plataformas empleadas, comentando la finalidad de las mismas y su proceso de evolución, desde su origen hasta su estado actual. A continuación, se procede a describir el diseño inicial elaborado para la realización del trabajo, explicando las modificaciones llevadas a cabo en el proceso y las decisiones tomadas, así como su justificación. El proceso de elaboración del trabajo se detalla en el apartado de desarrollo, el cual se divide en las tres fases principales de su realización: la instalación de las librerías de OpenCV, el desarrollo en Android de la aplicación y la adaptación de la misma para seguir el estándar definido por BioAPI. Finalmente se describen las pruebas realizadas y los resultados obtenidos, y se analizan las posibles mejoras o modificaciones como líneas futuras, comentando las conclusiones finales extraídas del TFG. Al final de este documento se incluye un anexo con la planificación y el presupuesto del trabajo, incluyendo un desglose de los cálculos realizados para los costes.

2 Estado del Arte

2.1 Historia de la Telefonía Móvil

El origen de la telefonía móvil y su rápida expansión ha supuesto un cambio drástico en nuestra forma de vida, especialmente a lo largo de los últimos 30 años. Hoy en día es una de las tecnologías más comúnmente utilizadas a diario, llegando a considerarse casi indispensable en el día a día. El avance que se ha producido en los medios de comunicación, ya no únicamente por voz sino también a través de mensajes de texto y contenidos multimedia, nos ha permitido un contacto permanente y de forma casi instantánea independientemente de la distancia, y ha dado forma en gran parte a lo que es hoy en día nuestra sociedad y la forma en la que interactuamos.

El primer equipo de telefonía móvil, llamado "Handie-Talkie H12-16", apareció a comienzos de la Segunda Guerra Mundial con el fin de facilitar la comunicación con las tropas. Este se basaba en un sistema de comunicación a través de ondas de radio que utilizaba frecuencias inferiores a 600kHz. En años posteriores este sistema empezó a perfeccionarse y a finales de la década de los 40 comenzaron a utilizarse sistemas de radio analógicos con modulación en amplitud (AM) y posteriormente en frecuencia (FM) logrando así cubrir mayores distancias con más precisión. Estos sistemas estaban al alcance de la población, eran de uso civil, lo cual suponía una gran ventaja para grandes empresarios, cuyo interés por estar continuamente comunicados era mayor. Sin embargo no causó tanto interés entre el resto de consumidores ya que los aparatos eran de gran tamaño y alto coste. Durante años se comercializaron también, aunque no con mucho éxito, teléfonos para automóviles. El equipo se instalaba en el maletero y el auricular, situado en la cabina del conductor, se conectaba al mismo a través de un cable. No fue hasta principios de los 80 cuando empezaron a surgir teléfonos de uso personal más asequibles al gran público contando con un tamaño y peso más adecuados. El surgimiento de estos primeros teléfonos es lo que se conoce como la primera generación (1G).

En 1973 sucedió un acontecimiento clave que marcaría un antes y un después en la historia de la telefonía móvil. El directivo de Motorola Martin Cooper presentaba al mundo su nuevo proyecto, el teléfono móvil "DynaTAC 8000X" (*Dynamic Adaptive Total Area Coverage*), realizando la primera llamada telefónica de la historia desde un teléfono móvil (Figura 1[8]). La llamada, realizada desde una calle de Nueva York, fue dirigida a Joel Engel, compañero y mayor rival en aquel momento, cuyo cargo era el de jefe de desarrollo de los Laboratorios Bell de AT&T. Este es considerado el primer teléfono móvil de la historia, ya que fue el

primero en comercializarse. Sin embargo, esto no sucedió hasta once años más tarde, en 1984, año en el que fue presentado oficialmente. En los años posteriores fueron lanzados nuevos modelos diseñados por la misma empresa Motorola, así como procedentes de otras grandes compañías del momento. En 1981 Ericsson lanzó el sistema NMT 450 (*Nordic Mobile Telephony*), el cual funcionaba a 450MHz. Cinco años más tarde lo sustituyeron por un modelo muy similar, el NMT 900, cuya banda de frecuencias era superior, en torno a 900MHz, lo cual permitía un mayor alcance y por tanto un mayor número de usuarios.

Fig. 1 – Martin Cooper realizando la primera llamada desde un teléfono móvil [8]

La segunda generación (2G) surgió en la década de los 90, trayendo consigo nuevas tecnologías como el GSM, IS-136, iDEN e IS-95. Surgieron numerosos estándares de comunicaciones móviles que buscaban una unificación a nivel internacional, entre los cuales finalmente se impuso el sistema GSM. La nueva generación usaba frecuencias superiores a las de la primera, alcanzando los 1800MHz, pero sin lugar a dudas el mayor avance que introdujo fue el paso de un sistema de comunicación analógico a uno digital, lo cual mejoraba considerablemente la calidad de voz. Además la seguridad en las comunicaciones aumentó, proporcionando una mayor privacidad a los usuarios.

La producción y fabricación de terminales se simplificó, y gracias al uso de materiales más económicos, comenzaron a ser más asequibles para el público general, lo cual aumentó notablemente las ventas. Se fabricaban teléfonos celulares mucho más cómodos en cuanto a tamaño y portabilidad y con el aumento del número de consumidores surgieron numerosas compañías operadoras y fabricantes de terminales, aumentando de esta manera la competitividad entre fabricantes y acelerando así el proceso de evolución en el campo de la telefonía móvil.

Con el tiempo, el sistema GSM se quedó atrás, ya que se buscaba ofrecer mejores prestaciones a los usuarios y una mayor velocidad en la transmisión. De esta forma empezó a desarrollarse lo que más tarde se convertiría en la tercera generación, 3G. No obstante, y debido a que las nuevas tecnologías que se querían incorporar aún estaban en vías de desarrollo, surgió una generación intermedia o de transición con fines puramente comerciales, la llamada generación 2.5, la cual mejoraba ciertos aspectos y servicios de su

predecesora. Algunos de estos servicios incluían la posibilidad de incluir contenidos multimedia en los mensajes de texto tales como melodías o iconos. Esto se conoce como el sistema EMS (más tarde SMS). También apareció el sistema MMS, o Sistema de Mensajería Multimedia, que permitía incluir imágenes, sonidos y vídeos además de texto. Para ello hacía uso de la tecnología GPRS (*General Packet Radio Service*) que permitía una velocidad de transmisión de datos entre 56kbps y 114kbps.

Con la llegada en 2001 de la tercera generación, cuyos sistemas fueron establecidos mediante el proyecto de la UIT sobre Telecomunicaciones Móviles Internaciones 2000 (IMT-2000), las redes pasaron a tener una mayor capacidad, se generaron servicios de red más avanzados y las velocidades de transmisión de datos eran tales que permitían a los usuarios conectarse a Internet. Esto trajo consigo numerosas posibilidades, ya que permitía la descarga de programas y contenidos, el acceso al correo electrónico, la mensajería instantánea, video

llamadas y otros muchos nuevos servicios.

La expansión del 3G fue en sus inicios algo lenta, pero su gran avance y la posibilidad de ser utilizada en otros dispositivos portátiles supuso una gran aceptación y en la actualidad está ampliamente extendida. Con ella surgió el nuevo sistema UMTS (*Universal Mobile Telecommunications System*), sucesor directo de GSM, que al admitir velocidades de transmisión más altas permitía el uso de los nuevos servicios.

Para la definición de la nueva generación, 4G, y al igual que sucedió con generaciones anteriores, la Unión Internacional de Comunicaciones (UIT) desarrolló unos requisitos a cumplir. Uno de estos requisitos era el estar basado en una red que funcionase en su totalidad a través de protocolos IP, usando tanto redes de cables como inalámbricas. Otro requerimiento de gran relevancia era el de proveer tasas de datos mayores de 100Mbit/s para alta movilidad, como en el acceso móvil, y en reposo hasta 1Gb/s. Todo esto además manteniendo una calidad de servicio (QoS) de un extremo a otro, permitiendo así el uso de cualquier servicio con total garantía y seguridad.

Hoy en día los teléfonos con 4G nos ofrecen la posibilidad de usar servicios tales como Google Music, Netflix o Spotify, que hacen uso de lo que actualmente se denomina "La Nube" y que requieren velocidades de conexión muy altas. Son servicios que, hasta hace unos años, esperaríamos poder usar únicamente desde un ordenador pero que, debido a los rápidos avances en el mundo de la telefonía móvil, ya son posibles.

2.2 Aplicaciones Móviles

En el mundo de las telecomunicaciones, el desarrollo de aplicaciones móviles es hoy en día uno de los negocios más activos. Actualmente existe una gran variedad de aplicaciones para dispositivos móviles, desde juegos, noticias, música o vídeos hasta aplicaciones de diseño, fotografía, medicina, moda o enseñanza. Uno de los factores más influyentes y que más impulsó este desarrollo fue sin duda la aparición de los Smartphone (teléfonos inteligentes).

Aunque las primeras aplicaciones móviles no se popularizaron hasta finales de los 90, hay que destacar que ya a principios de la década comenzaron a aparecer teléfonos móviles con más funciones que las de llamar y mandar mensajes de texto. Es el caso de "Simon", un teléfono lanzado por IBM en 1993 que, con la incorporación de servicios de voz y datos, además de hacer las funciones básicas de un teléfono de la época, servía como asistente

digital, ya que contaba con otras funciones como un calendario, acceso al correo electrónico, una libreta de direcciones, un bloc de notas, una calculadora o un reloj mundial, además de juegos. Incluso funcionaba como una máquina de fax. Además, el teléfono contaba con una pantalla táctil que incluía un teclado QWERTY como los que se ven actualmente. Sin embargo debido al diseño, tamaño y peso, y sobre todo a su elevado coste, no se vendieron muchas unidades. Más destacado fue el "Palm Pilot", lanzado en 1996 y su sucesor, el "Pilot 1000", muy utilizados por ejecutivos y hombres de negocios, ya que les permitía transportar sus datos en un pequeño dispositivo portátil. Estos últimos son de interés ya que, a pesar de no ser teléfonos móviles, eran pequeños dispositivos que incluían ya algunas funciones más avanzadas. Con ellos se popularizó el uso del término PDA, que significa Asistente Digital Personal en inglés.

Ya a finales de los 90 y principios del 2000 comenzaron a ser más frecuentes en los teléfonos algunas aplicaciones sencillas. Estas eran principalmente juegos, agendas o editores de melodías y tonos de llamada, aplicaciones muy simples en cuanto a funcionalidad y diseño, y cuyo objetivo era únicamente el de ofrecer algún tipo de entretenimiento o funcionalidad extra a los dispositivos. En 1997 la compañía Nokia lanzó para sus teléfonos el conocido juego de la serpiente, el Snake. El éxito que tuvo entre los consumidores fue tal, que el resto de fabricantes se vio obligado a incorporar en sus teléfonos móviles aplicaciones y juegos similares para complacer a sus clientes y mantener su interés. Algunos juegos como el "Tetris", el "Pong" o el "Tic-Tac-Toe" surgieron en los teléfonos a partir de esto.

Sin embargo, las posibilidades que ofrecían los teléfonos de la época no permitían ir mucho más allá. Esto se debía en gran parte a que estaban programados a bajo nivel, directamente en código máquina, e integrados en la memoria ROM del teléfono. Esto generaba algunas limitaciones y por ello comenzaron a surgir en Japón móviles programables que incorporaban una zona de memoria reservada a datos para poder descargar al teléfono programas descargados de Internet. Además, tenían la capacidad de utilizar programas con lenguajes de más alto nivel similares a Java, lo cual daba más opciones a la hora de desarrollar las aplicaciones.

Con el objetivo de ofrecer a los usuarios un mayor número de funcionalidades se intentó incluir acceso a Internet en los teléfonos móviles. Sin embargo, debido a que por aquella época las páginas web ya contaban con multitud de color y contenidos multimedia, resultaba imposible cargarlas desde los dispositivos, que aún eran en blanco y negro, y sus capacidades de procesamiento y memoria eran muy limitadas. Además las velocidades de descarga de datos no eran lo suficientemente rápidas y el coste era elevado. Por ello se creó un estándar, WAP (*Wireless Application Protocol*) que tenía el objetivo de suplir este problema. Era una versión del protocolo de las páginas web HTTP adaptada a plataformas móviles que mostraba un diseño más sencillo. En un principio pareció una buena solución para las operadoras y los fabricantes, pero la comercialización de aplicaciones a través de este servicio no resultaba sencilla y la descarga de contenidos era lenta y frustrante para los clientes, a los cuales se les cobraba cada segundo que hacían uso del servicio. Por ello la respuesta que tuvo en general, con la excepción de algunos países como Japón, no fue la esperada, e incluso recibió fuertes críticas por parte de los consumidores quienes en forma de sátira cambiaron el significado de las siglas WAP a "Wait And Pay" (espera y paga).

Con el paso del tiempo mejoraron los procesadores y las memorias de los teléfonos y proliferaron las pantallas a color. Comenzaron a incorporarse en los teléfonos pequeñas

cámaras digitales, lo cual se ha convertido hoy en día es un requisito básico. Así empezaron a surgir los primeros Smartphone, que contaban con múltiples aplicaciones que dejaban ya en un segundo plano las funciones básicas del teléfono.

En 2002 la compañía canadiense *Research In Motion* (RIM), hasta entonces popular por sus pagers o bipers, lanzó la primera Blackberry 5810, que permitía al usuario navegar por Internet y consultar su correo electrónico, además de hacer llamadas telefónicas. De esta manera, fusionó las opciones que ofrecían las PDA con las funciones propias del teléfono. Otros sistemas operativos para teléfonos comenzaron a surgir, convirtiendo ya a los teléfonos en un dispositivo con funciones cada vez más parecidas a las que podía proporcionar un ordenador, aunque con ciertas limitaciones.

Fig. 2 – Steve Jobs presentando el primer iPhone [12]

Sin embargo fue Apple, con su primer iPhone lanzado en 2007 (Figura 2[12]), quien realmente revolucionó el mercado de la telefonía móvil y las aplicaciones. Esto provocó la rápida aparición de numerosos competidores. Pocos meses después del lanzamiento del iPhone, Google presentó el sistema operativo Android y dos años más tarde Motorola lanzó con gran éxito el "Droid", un Smartphone que contaba con este OS. Rápidamente se extendió el uso de los Smartphone y con la aparición de las tiendas para aplicaciones, App Store de Apple y Android Market de Android (a la cual se le cambió el nombre años más tarde al actual Play Store), comenzaron a aparecer multitud de nuevas aplicaciones que ofrecían al usuario personalizar cada vez más sus teléfonos añadiéndoles multitud de funcionalidades. Además, a diferencia de lo que ocurría con teléfonos anteriores en los cuales se podía programar casi únicamente en la memoria del teléfono, los nuevos sistemas operativos Mac iOS, RIM, Android, Symbian o Windows Mobile estaban abiertos al desarrollo por parte de terceros, lo que generó rápidamente un mercado enorme de aplicaciones, en el cual se competía por llamar la atención del consumidor buscando la originalidad y utilizando diseños atractivos.

Desde entonces se ha generado un mercado de aplicaciones enorme y el uso del teléfono móvil o las tabletas y de algunas aplicaciones como WhatsApp, Twitter o Instagram se ha convertido en una parte importante, casi imprescindible de nuestro día a día y una nueva forma de relacionarse, compartir y aprender. Actualmente este mercado sigue evolucionando a un ritmo vertiginoso, y con las grandes y rápidas mejoras de hardware en los dispositivos,

que cada vez cuentan con mejores procesadores y memorias y con pantallas de mayor calidad y precisión, las opciones que surgen para las aplicaciones son incalculables.

2.3 Historia de la Biometría

El término Biometría, del griego *bios* (vida) y *metron* (medida), define a la ciencia que se encarga de estudiar las características físicas o de comportamiento que definen de forma única a un ser humano y por tanto permiten su identificación. Las tecnologías biométricas se basan en la extracción de estas características a través de dispositivos de captación para su posterior procesamiento y almacenamiento en forma de secuencias numéricas.

Los sistemas biométricos pueden utilizarse principalmente con dos fines, el de identificación o para llevar a cabo una autentificación o verificación. En el proceso de autentificación, las características o rasgos biométricos extraídos de un individuo se comparan únicamente con un patrón existente. De esta manera se comprueba si la persona es o no la misma que la que representa el patrón. Esto se utiliza por ejemplo en las aduanas de los aeropuertos o en otros ámbitos policiales. En cambio en los procesos de identificación, las comprobaciones no son de 1:1 como en el caso anterior sino de 1:N, es decir, las características extraídas de un usuario se comparan con una base de datos que contiene los patrones de varias personas. Así se identifica cuál de todas esas personas es el individuo en cuestión. Este proceso es mucho más lento que el anterior, y aumenta su duración cuanto mayor es el número de usuarios (N) almacenados en la base de datos.

A pesar de que todos estos sistemas parecen de uso muy reciente, el uso de técnicas biométricas para la identificación de personas tuvo su origen hace varios siglos. Durante el siglo XIV en China ya se usaba entre los comerciantes un sistema de impresión de la palma de la mano y los pies de los niños con papel y tinta para poder así identificarlos. Este procedimiento fue documentado por João de Barros, un historiador y lingüista de origen portugués durante sus años de exploración en Asia. También en occidente se hicieron importantes descubrimientos que impulsarían el uso de estas técnicas. En 1686 Marcello Malpighi, un anatomista y biólogo italiano considerado el fundador de la histología (ciencia que se dedica al estudio de los tejidos orgánicos), descubrió que los patrones de la piel en los dedos diferían entre individuos. Casi un siglo y medio más tarde, en 1823, el médico y científico de origen checo Jan Evangelista Purkine, estudió esta naturaleza única de las huellas digitales de los individuos e identificó las elipses, espirales y triángulos presentes en las mismas. El 28 de octubre de 1880, Henry Faulds publicó en la revista Nature el artículo "On the Skin-Furrows of the Hand", en el cual describía unos métodos para identificar a criminales a partir de sus huellas digitales.

Uno de los avances más relevantes en la historia de la biometría fue llevado a cabo por Alphonse Bertillon, jefe del departamento fotográfico de la policía de Paris además de hijo y hermano de expertos en estadística y demografía, quien en 1882 propuso una nueva disciplina que tenía como fin identificar a los criminales reincidentes: la antropometría (también conocida como "Bertillonaje"). Consistía en tomar medidas de diferentes partes del cuerpo y la cabeza así como identificar marcas individuales de los sospechosos como podían ser tatuajes o cicatrices y características personales del individuo. Elaboró una metodología a seguir para llevar a cabo el registro y hacer las comprobaciones haciendo uso de todos los datos procesados y la publicó en su libro "Identification Anthrométrique" incluyendo

ilustraciones explicativas como la que se muestra en la Figura 3[15]. Su procedimiento ganó mucho prestigio, usándose para la identificación de hasta 241 delincuentes múltiples, y rápidamente se extendió por el resto de Europa y EE.UU. Sin embargo su éxito y prestigio disminuyó enormemente cuando, al llevar a cabo su procedimiento de medidas para un individuo, estas coincidieron con las de otro ya registrado. Más tarde se comprobaron sus identidades y se descubrió que eran gemelos idénticos. Además surgieron dificultades a la hora de adaptar el mismo sistema en otros lugares, donde los métodos y las unidades de medida eran diferentes. Por ello, desde entonces es considerada una pseudociencia.

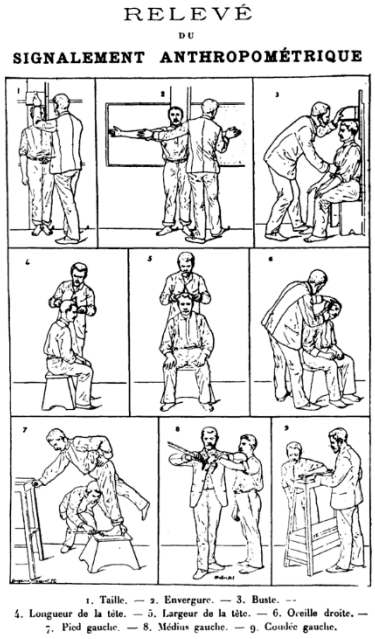

Fig. 3 – Ilustración del libro "Identification Anthrométrique" de Alphonse Bertillon [15]

Durante esos años se desarrollaron también otros métodos de identificación basados en las huellas dactilares. Juan Vucetich Kovacevich, de origen croata y nacionalizado en Argentina, desarrolló y puso en práctica el primer sistema eficaz de identificación de personas a través de sus huellas. Con él definía los elementos a utilizar para una captación lo más precisa posible y sistematizó el método. Su sistema se basaba en las ideas de sir Francis Galton, aunque para la caracterización hizo una simplificación empleando únicamente 4 rasgos de las huellas: los arcos, las presillas internas, las presillas externas y los verticilos. El 1 de septiembre de 1891 realizó las primeras fichas dactilares del mundo empleando las huellas de 23 procesados. Tres años más tarde la policía de Buenos Aires adoptó su sistema oficialmente tras comprobar su correcto funcionamiento y posteriormente fue incorporada también a la Policía Federal Argentina (por aquel entonces la Policía de la Capital Federal). En 1892 sir Francis Galton publicó el libro "Finger Prints", con el cual presentaba un nuevo sistema de clasificación que empleaba las huellas de todos los dedos de las manos. Este método es conocido como "Galtoneano" o "Icnofalangometría".

En 1896 un subalterno de Edward Henry, Inspector General de la Policía de Bengala, desarrolló un método de almacenamiento y clasificación de todas las fichas e información de las huellas dactilares, muy eficaz y que facilitaba su procesamiento. Más tarde, sir Henry estableció en Londres el primer archivo de huellas digitales usando este sistema, lo cual

impulsó la aparición y el uso de sistemas similares entre organizaciones de justicia criminal como Scotland Yard, donde se adoptó el mismo sistema usando por Edward Henry. También comenzaron a utilizarse estas técnicas en las Fuerzas Aéreas, el Ejército y la Armada de EE.UU. En 1902 se resolvió por primera vez un caso empleando el sistema de huellas dactilares y en 1915 en California se creó una organización destinada a estos procesos de identificación formada por veintidós hombres y denominada la Asociación Internacional para Identificación Criminal (IAI).

Los sistemas de reconocimiento mediante huella dactilar evolucionaron y se expandieron en gran medida durante años posteriores. La primera máquina que se patentó para este uso fue desarrollada por William T. Cirone en 1950 y consistía en una cámara de identificación. En 1968 se patentó un sistema de seguridad que consistía en el uso de tarjetas personales portables con una identificación de huella digital, el cual se le asignó a la empresa IBM. En 1969 el FBI contactó con el *National Institute of Standards and Technology* (NIST) para automatizar el proceso definiendo dos pasos fundamentales: el escaneo de las tarjetas de huellas digitales para la identificación de las minucias y el sistema de identificación y comparación de las mismas. Más tarde se consolidó el desarrollo de escáneres y otras tecnologías para la extracción de las minucias a través de técnicas capacitivas.

Debido a la expansión de estos sistemas, los criminales comenzaron a descubrir trucos como el de quemarse las yemas de los dedos o hacerse pequeñas modificaciones en la cara mediante cirugía para dar negativo en las pruebas de identificación. Sin embargo en 1935 se descubrió un nuevo sistema de identificación única con el cual ninguna modificación sería posible. Isidore Goldstein y Carleton Simon, dos oftalmólogos neoyorkinos, publicaron en el "New York State Journal of Medicine" un artículo titulado "A New Scientific Method of Identification" en el cual presentaban la retina como un nuevo método de identificación basándose en los patrones vasculares, únicos en cada individuo. Posteriormente, Paul Tower publicó un artículo en "Archives of Ophthalmology" titulado "The fundus oculi in monozygotic twins: report of six pairs of identical twins" en el cual afirmaba, basándose en varias pruebas, que el mejor método de distinción entre gemelos idénticos era a través de los patrones vasculares de la retina. También se propuso el iris como método de identificación. La idea fue desarrollada por el oftalmólogo Frank Burch en 1936. Sin embargo no fue hasta muchos años más tarde, a mediados de la década de los 80, cuando los doctores Leonard Flom y Aran Safir retomaron la idea y a finales de la década desarrollaron con la colaboración de John Daughman un proceso de reconocimiento del iris basado en unos algoritmos específicos. Este método, base de todos los posteriores productos en este campo, fue patentado en 1994 y pasó a ser propiedad de "Iridian Technologies". La Agencia Nuclear de Defensa de EE.UU. desarrolló también en estos años un prototipo, el cual una vez finalizado y probado pasó a estar disponible comercialmente.

El estudio de la biometría se expandió, provocando cada vez un mayor interés y comenzaron a aparecer nuevos métodos y sistemas de identificación. En 1960 Gunnar Fant, profesor del Instituto Real de Tecnología (KTH) de Estocolmo y especialista en las características acústicas de la voz humana, publicó un modelo describiendo los componentes fisiológicos de la misma y avanzó mucho en el campo del análisis y la síntesis del habla. Nueve años más tarde John Pierce, trabajador de "Bell Telephone Laboratories", publicó un artículo titulado "Whither Speech Recognition?" en "The Journal of Acoustical Society of America" en el que trataba el campo de la investigación en el reconocimiento del habla. Ya en 1970 el Dr. Joseph

Perkell analizó las conductas en el discurso empleando máquinas de rayos X para visualizar los movimientos realizados por la lengua y la mandíbula durante el proceso. Los sistemas de reconocimiento facial también evolucionaron durante estos años. Entre 1964 y 1965 Woodrow Wilson Bledsoe, Helen Chan Wolf Y Charles Bisson trabajaron en un sistema de reconocimiento facial a través de computadora y desarrollaron así el primer sistema semi-automático de reconocimiento. Años más tarde un grupo de investigadores desarrollaron un proceso de automatización basándose en 22 marcas específicas y subjetivas de la cara y la cabeza como el color del pelo o el grosor de los labios. Estas mediciones y localizaciones se hacían manualmente. También surgieron sistemas de reconocimiento a partir de la firma, siendo el primero el desarrollado por la Aviación Norteamericana en 1965. El 25 de noviembre de 1969 Salvatore R. Danna patentó un instrumento capaz de llevar a cabo esta identificación, y años más tarde, el 25 de mayo de 1976, Jacob Sternberg y Robert W. Freund patentaron en Estados Unidos un dispositivo y una metodología para grabar la firma.

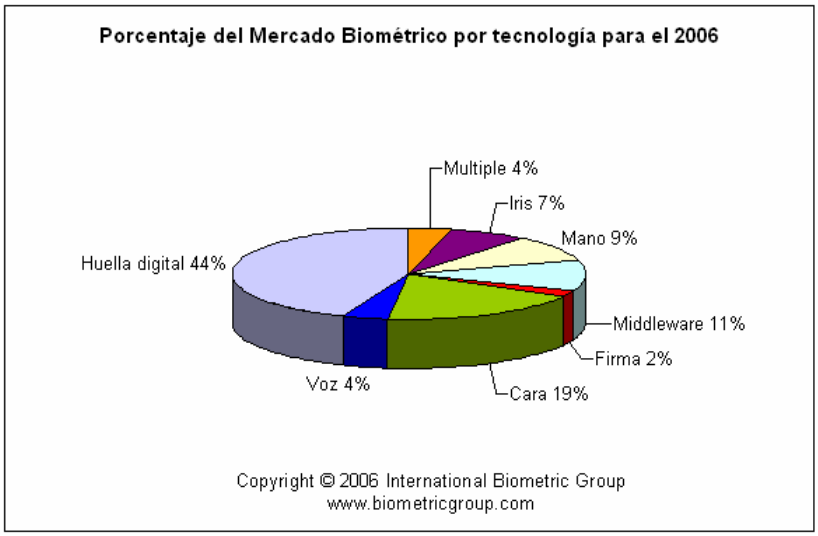

Fig. 4 – Uso de las tecnologías biométricas en 2006 [13]

En la actualidad todos estos sistemas siguen desarrollándose convirtiéndose cada vez más en procedimientos muy precisos y elaborados. Su aplicación se ha extendido en gran medida, usándose ya no únicamente en entornos policiales (dónde son ampliamente utilizados) sino también en los nuevos dispositivos móviles, y especialmente en sistemas de seguridad como en transacciones bancarias, servicios sociales y de salud, o en las infraestructuras de grandes redes empresariales. Los métodos más utilizados son el reconocimiento facial, las huellas dactilares, el iris, la retina, las venas de la mano o la geometría de la misma. En la Figura 4[13] se muestra el uso de estas tecnologías según un estudio llevado a cabo en 2006. También se emplean sistemas dinámicos, donde las características ya nos son físicas sino que se basan en comportamientos, como el registro de la firma, la forma de andar o la forma de teclear en un ordenador. Los reconocimientos por voz se consideran en un ámbito intermedio, ya que se basan tanto en características físicas como de comportamiento. A continuación se muestra una tabla con las características propias de algunos de los métodos comentados:

Tabla 1 – Características de los sistemas biométricos más comunes

	Iris	Retina	Huellas dactilares	Escritura y firma	Voz	Cara 2D	Cara 3D
Fiabilidad	Muy alta	Muy alta	Muy alta	Media	Alta	Media	Alta
Facilidad de uso	Media	Baja	Alta	Alta	Alta	Alta	Alta
Prevención de ataques	Muy alta	Muy alta	Alta	Media	Media	Media	Alta
Aceptación	Media	Baja	Alta	Muy alta	Alta	Muy alta	Muy alta
Estabilidad	Alta	Alta	Alta	Baja	Media	Media	Alta

Con la extensión de las técnicas biométricas han surgido numerosos estándares que buscan una regulación de los procedimientos. El principal organismo encargado de desarrollar estas normas es el Sub-Comité 17 (SC17) del *Joint Technical Committee on Information Technology* de la Organización Internacional de Estandarización y la Comisión Electrotécnica Internacional (ISO/IEC JTC1). También existen otros organismos no gubernamentales que tienen el mismo propósito, como son Biometrics Consortium, International Groups y BioAPI (*Biometric Application Programming Interface*). Este último desarrolló conjuntamente con otros consorcios y asociaciones una plataforma para integrar el software de diferentes propietarios y favorecer así la interoperabilidad entre plataformas. Sin embargo aún no se ha conseguido extender el uso de estos estándares y los propietarios siguen empleando interfaces de diseño propias, dificultando su extensión a otros dispositivos o programas.

Uno de los mayores problemas o preocupaciones que plantean las técnicas biométricas en la actualidad es la privacidad del usuario, ya que estos sistemas emplean para su funcionamiento grandes bases de datos que contiene información física y de comportamiento de un gran número de individuos. El miedo surge al pensar que estos datos pueden usarse para disminuir las libertades de la población. Además muchas de estas técnicas se han desarrollado sin llevar a cabo un sistema de seguridad adecuado que proteja la información, lo cual genera una preocupación de robo de identidad que podría significar un total acceso a cuentas, datos, infraestructuras u otros sistemas propiedad del usuario cuya autentificación se base en estas técnicas. Por ello, aunque cada vez se utilizan más estos mecanismos, parece que mientras estas cuestiones no sean solucionadas, el uso de las técnicas biométricas seguirá siendo reducido en comparación con las múltiples aplicaciones que podrían llegar a tener teniendo en cuenta los grandes avances tecnológicos realizados en los últimos años.

2.4 Evolución de los Sistemas de Reconocimiento Facial

Los sistemas de reconocimiento facial consisten en aplicaciones capaces de identificar a un individuo a través de la obtención de determinadas características faciales extraídas a partir de una imagen digital. Estas técnicas se basan en algoritmos capaces de extraer los rasgos más relevantes y llevar a cabo una comparación con una base de datos. El objetivo de estos sistemas es el de alcanzar las capacidades de reconocimiento facial humanas, pero usando extensas bases de datos para una identificación casi automática.

En la actualidad es un campo de investigación muy activo, y aunque los resultados no son tan fiables como los de otros sistemas biométricos tales como los basados en huella dactilar o iris, es más aceptado y por ello se utiliza en una gran cantidad de ámbitos.

Los primeros sistemas de reconocimiento facial se basaban en la metodología desarrollada por Alphonse Bertillon a finales del siglo XIX para llevar a cabo la identificación de individuos en el ámbito de la criminología. Este procedimiento consistía principalmente en registrar marcas y características físicas propias del individuo y en tomar determinadas medidas del cuerpo. Algunas de estas medidas tenían en cuenta los rasgos faciales como por ejemplo la distancia entre los ojos o el ancho del óvalo facial a diferentes alturas o posiciones. Sin embargo estos sistemas eran poco fiables debido a la falta de un criterio unificado y el uso de diferentes instrumentos, siendo muy comunes los errores en el proceso de toma de medidas.

Muchas décadas pasaron hasta que se hicieron mayores avances en estas técnicas. A finales de los 60, el matemático e informático teórico Woodrow Wilson Bledsoe desarrolló, junto con un grupo de investigadores, un sistema de reconocimiento facial mediante ordenador que consistía en tomar 20 distancias entre puntos característicos del rostro de una persona en una imagen digital. Estos puntos debían seleccionarse manualmente, por lo que era un sistema semi-automático. Algunos de estos eran las pupilas, los ojos, las cejas y las esquinas de la boca. Años más tarde, en la década de los 70, surgieron nuevos métodos siguiendo procedimientos similares, como el diseñado por Goldstein, Harmon y Lesk, que se basaba en 21 marcadores subjetivos incluyendo el color del cabello o el grosor de los labios. Sin embargo, estos procedimientos eran demasiado subjetivos y de poca utilidad ya que la mayoría de estos rasgos podían variar con el paso del tiempo. Además no se tenían en cuenta cambios producidos por el entorno como la iluminación, o el ángulo e inclinación de la cara, por lo que no resultaban de gran utilidad más que en el ámbito de la investigación. En noviembre de 1973 el informático teórico Takeo Kanade presento en su disertación post-doctoral “Picture Processing System by Computer Complex and Recognition of Human Faces” el desarrollo del primer sistema completamente automático. En ella explicaba la extracción de características de la cara a partir de imágenes en escala de grises. La identificación se realizaba mediante una comparación de los histogramas locales. A pesar del gran avance que supuso al no necesitar seleccionar los puntos manualmente, los resultados no eran tan precisos, teniendo un acierto del 75% usando 20 imágenes.

En la década de los 80, con la incorporación de algoritmos matemáticos para el procesamiento de imágenes, se consiguió dar un gran salto en las técnicas biométricas. En 1901 Karl Pearson diseñó el procedimiento estadístico conocido como PCA (Análisis de Componentes Principales), muy empleado en la actualidad y cuyo objetivo es principalmente el de disminuir las dimensiones de un conjunto de datos dando mayor relevancia a los que

causan una mayor varianza. Este procedimiento fue aplicado sobre imágenes de caras en 1987 por Sirovich, consiguiendo así una compresión de la información. Más tarde Turk y Pentland lo emplearon para un algoritmo de identificación muy conocido, el "Eigenfaces". Con el PCA las imágenes se normalizan a partir de la alineación de los ojos y la boca. Otro método muy extendido y de mejores prestaciones para el procesamiento de imágenes en reconocimiento facial es el LDA (Análisis Discriminante Lineal), cuyo objetivo principal es, como en el PCA, reducir la dimensionalidad de los datos a partir de la creación de combinaciones lineales con las características significativas de un conjunto. A partir de una aproximación estadística, realiza una diferenciación de las muestras conocidas y las desconocidas. En él se basa el análisis lineal de Fisher, otro algoritmo para este uso, cuyos resultados lo convierten en el más óptimo. A diferencia de lo que ocurre con el PCA, realiza la clasificación en base a diferencias y no similitudes.

En la actualidad, además de emplear imágenes fijas, se utilizan imágenes en 3D para el reconocimiento. Estas permiten llevara a cabo una reconstrucción más o menos fiel de la cara que proporciona medidas mucho más exactas del individuo sin que afecten factores como la iluminación o el ángulo de captura. A continuación se muestra en la Figura 5[18] un ejemplo de un método de reconstrucción facial en 3D. Para estas imágenes se emplean algoritmos diferentes, como el P2CA (Análisis de componentes parcial/principal), una evolución del PCA. Sus resultados son 10 veces más eficaces que los anteriores, una mejora considerable.

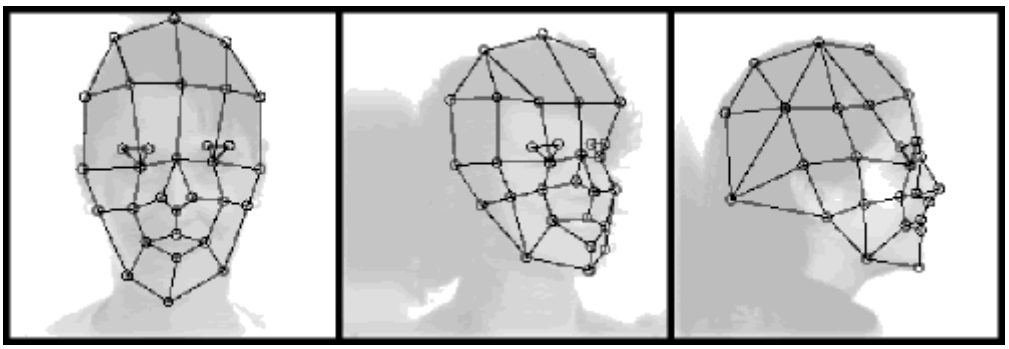

Fig. 5 – Reconstrucción facial en 3D [18]

El procedimiento seguido en los sistemas de reconocimiento facial, tal y como se muestra en la Figura 6[18] se basa en cuatro fases principales:

1. Adquisición de la imagen: obtención de la imagen o vídeo a través de un dispositivo de captura.
2. Detección de la cara: la imagen o vídeo se analiza para comprobar si aparecen o no una o más caras. Existen determinados algoritmos para su localización.
3. Acondicionamiento y Normalización: la cara obtenida se normaliza respecto a propiedades geométricas y fotométricas a partir de un punto, como la posición de la nariz, o simplemente escalando la imagen.
4. Extracción de características: la imagen se procesa extrayendo las características principales, las cuales serán entregadas en forma de vector de dimensiones reducidas en la siguiente fase, utilizándose para comparar y realizar el reconocimiento. Las principales técnicas utilizadas (algunas de ellas han sido comentadas ya previamente) son:
    a. PCA, análisis de componentes principales.

b. FLD, análisis de discriminantes lineales de Fisher.
c. LPP, conservación de proyecciones locales.
d. Proyecciones aleatorias.
e. Redes neuronales artificiales.
f. IAC, análisis de componentes independientes.
g. Características locales o análisis de subregiones.
h. Correlación.
i. DCT, transformada discreta del coseno.

Estas técnicas también se utilizan en ocasiones de forma combinada.

5. Algoritmo de reconocimiento: El vector de características proporcionado se compara con la base de datos y, en el caso de ser un proceso de identificación, se obtienen diferentes porcentajes definiendo el grado de similitud con los diferentes usuarios registrados, siendo el más alto el resultante. Si se trata de un proceso de verificación, se obtiene un único porcentaje, el cual define, según si pasa o no un umbral establecido, si el resultado es positivo o negativo.

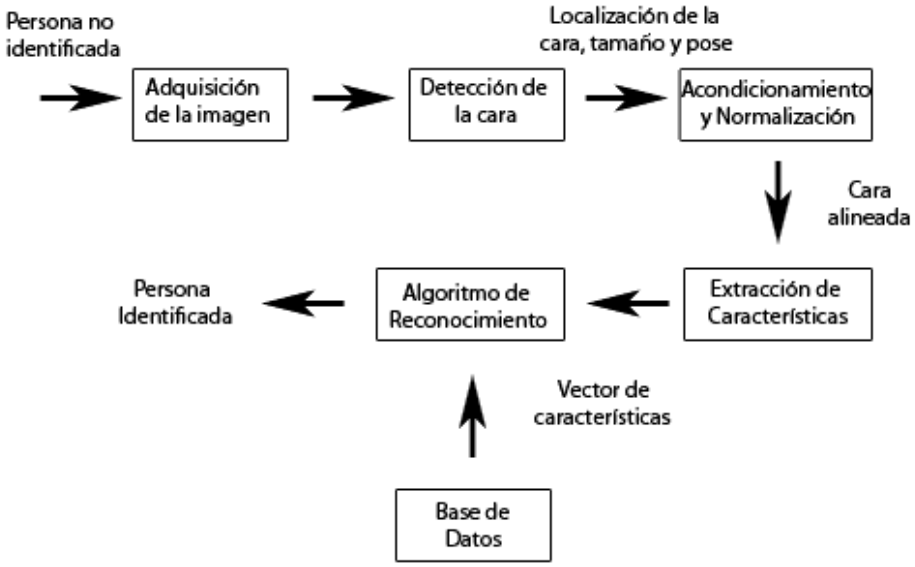

Fig. 6 – Diagrama del proceso de reconocimiento facial [18]

El uso de estas técnicas biométricas es considerado en algunos círculos una pérdida de la privacidad y una intromisión de la misma así como una violación de los derechos y libertades de los individuos. Por ello se está luchando por regularizar este campo y conseguir así una mayor extensión del uso de estas técnicas, cuyas posibilidades de aplicación son muy amplias gracias a los grandes avances tecnológicos y en investigación que se han realizado en los últimos años.

3 Plataformas de Desarrollo

3.1 Android

Android es un sistema operativo propiedad de Google y diseñado para plataformas móviles con pantallas táctiles, como son los Smartphones, las Tablets, la versión intermedia llamada Phablet o los relojes inteligentes, así como para televisores e incluso coches (ordenador de a bordo). Actualmente su uso está muy extendido, superando en ventas a su mayor competidor, Apple, con su propio sistema operativo iOS.

La plataforma hardware se basa en la arquitectura ARM para 64 bits (conjunto de instrucciones x86) y su núcleo está basado en el Kernel de Linux, destinado a la gestión de la memoria, de los procesos, la seguridad, la pila de red y los modelos de controladores. Este hace de intermediario entre el hardware y el software. Su arquitectura se basa en una serie de componentes principales, por encima del núcleo en Linux. La capa superior consiste en una serie de aplicaciones, escritas en lenguaje Java. Estas pueden ser por ejemplo el correo, los SMS, la calculadora, la agenda, el navegador, los mapas o el calendario, aplicaciones que normalmente vienen por defecto instaladas, u otras aplicaciones adquiridas a través de la tienda (Google Play, anteriormente Android Market), como juegos o servicios de mensajería instantánea. A continuación se encuentra el marco de aplicaciones o Framework, el cual consiste en las clases y APIs utilizadas por las aplicaciones. Estas están disponibles también para los desarrolladores, quienes cuentan con un kit de desarrollo software, el Android SDK. En una capa más interna, justo anterior al Kernel, se encuentran unas bibliotecas, escritas en C y C++, y usadas por componentes del sistema. Estas incluyen bases de datos en SQLite, bibliotecas de medios y de gráficos, OpenGL y otras. En esta misma capa se encuentra el entorno de ejecución o "Runtime Android", que contiene las bibliotecas base para las aplicaciones en Java. Cada aplicación obtiene una instancia de la máquina virtual Dalvik, de manera que al mismo tiempo pueden funcionar múltiples máquinas virtuales. A continuación se muestra en la Figura 7[24] un esquema de esta arquitectura comentada:

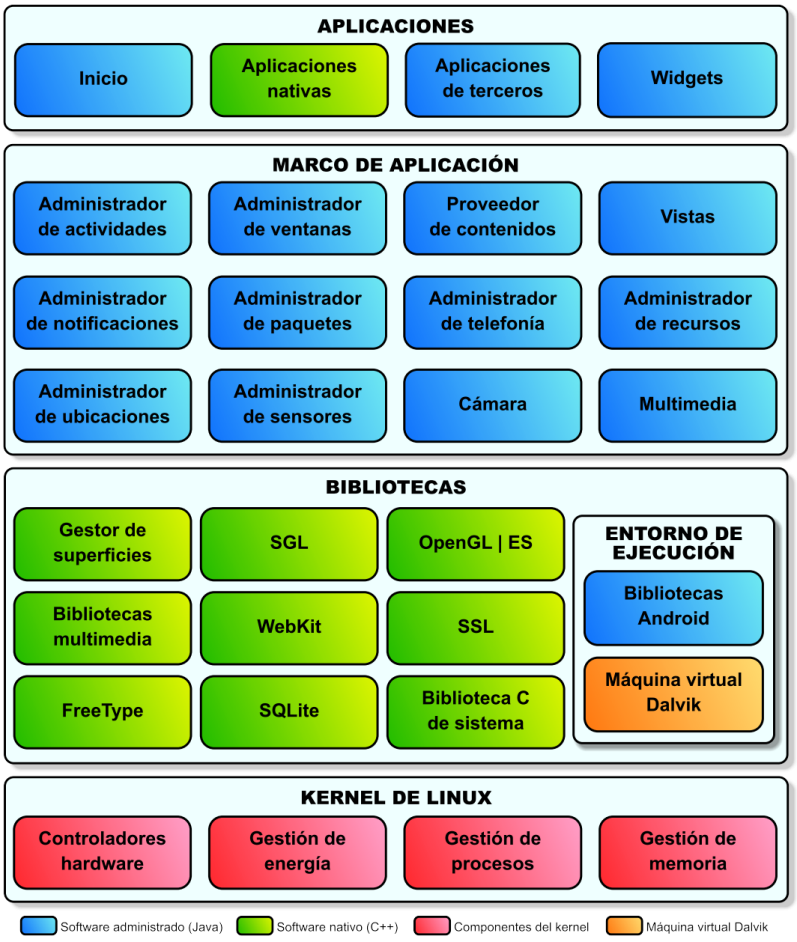

Fig. 7 – Arquitectura Android [24]

En cuanto a su historia, el sistema operativo inicial fue desarrollado por Android Inc., una pequeña compañía fundada en California en 2003. Sin embargo pasó rápidamente a ser propiedad de Google, quien compró en 2005 la firma debido a su gran interés por incorporar sus servicios de búsqueda y aplicaciones en teléfonos móviles. Tras adquirir diversas patentes, el 5 de noviembre de 2007 se presentó el nuevo sistema operativo, a la vez que se creó *Open Handset Alliance* (OHA), un consorcio formado por 78 compañías dedicadas al hardware, software y al mundo de las telecomunicaciones, entre las cuales estaban presentes Motorola, T-Mobile, LG, Intel y Samsung Electronics. Este consorcio se formó con el propósito de elaborar estándares abiertos para los dispositivos móviles. Presentaron así su primer producto, Android, construido sobre la versión 2.6 de Linux. Google liberó la mayoría del código bajo la licencia Apache, libre y de código abierto. El sistema operativo inicialmente se lanzó para funcionar mediante un teclado y un cursor, similar al funcionamiento de las Blackberry, pero tras la aparición del primer iPhone, se adaptó su uso a dispositivos con pantallas táctiles.

El 22 de octubre de 2008 nació el primer teléfono con Android disponible en el mercado, el HTC Dream. Poco después, el 9 de diciembre de 2008 se unieron 14 nuevos miembros al proyecto Android, entre los cuales estaban Toshiba, Sony Ericsson y Vodafone. A comienzos del 2010, Google colaboró con HTC para sacar un nuevo teléfono, el Nexus One, cuya gama

ha continuado posteriormente con otras compañías, como Samsung, con la cual se ha lanzado el Nexus S, el Galaxy Nexus y la tableta Nexus 10. También ha colaborado con otras empresas como LG y Asus para sacar otros productos Nexus. Toda esta línea de teléfonos se emplea para el desarrollo e implementación de Android, y sus teléfonos son los encargados de estrenar las nuevas versiones.

Desde sus inicios, y debido a que este era uno de los grandes propósitos o motivos por los que se creó este nuevo sistema operativo, han ido desarrollándose nuevas versiones con actualizaciones, optimizando cada vez más sus características, corrigiendo fallos e incluyendo nuevas funciones gracias a los avances que han ido surgiendo en el hardware. Los nombres asignados a las diferentes versiones corresponden a postres en inglés, cuyas letras iniciales siguen un orden alfabético. La primera versión beta fue lanzada en noviembre de 2007, y en septiembre de 2008 se presentó la primera versión comercial, Android 1.0, conocida desde 2009 como Apple Pie. A continuación en la Figura 8[25] se pueden ver las diferentes versiones con sus nombres y fechas de lanzamiento, hasta la actual:

Fig. 8 – Evolución de las versiones Android [25]

La utilización de estas versiones por parte de los usuarios va cambiando a medida que pasa el tiempo, disminuyendo el uso de las versiones antiguas, y aumentado el de las nuevas. A continuación, en la Figura 9[26] se muestra un gráfico con esta evolución:

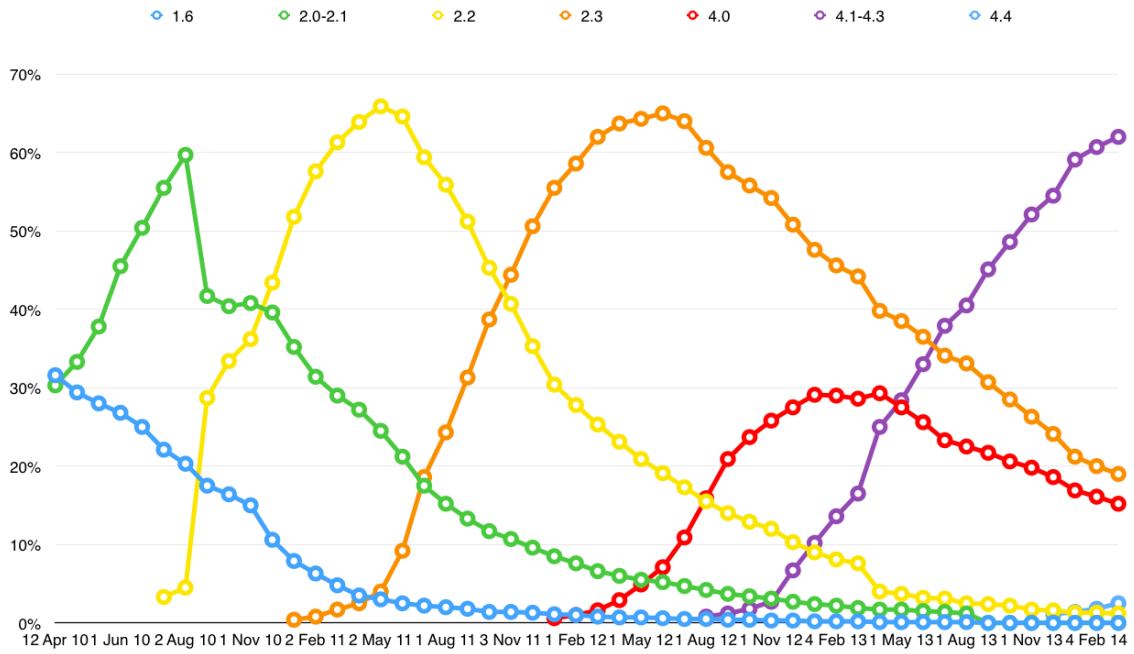

Fig. 9 – Evolución temporal en el uso de las versiones Android [26]

A lo largo de los años, Android ha ido ganándole terreno a otros sistemas operativos para Smartphones, alcanzando según un estudio realizado en España entre diciembre de 2012 y febrero de 2013, un 92% de las ventas de estos dispositivos, en comparación con el 4.4% de iOS. Además ofrece un mayor número de aplicaciones que el sistema operativo de Apple y a un precio más bajo, existiendo en la actualidad aproximadamente 1.000.000.

3.2 OpenCV y JavaCV

3.2.1 OpenCV

OpenCV, cuyo nombre viene de "Open Source Computer Vision", consiste en una librería de código abierto bajo la licencia BSD que contiene multitud de clases y funciones destinadas a su uso en sistemas de visión artificial. Contiene más de 500 funciones diferentes incluyendo reconocimiento de objetos, calibración de cámaras, visión robótica y otras muchas aplicaciones. Fue desarrollada por el centro de investigación de Intel, y actualmente pertenece a una fundación sin ánimo de lucro, OpenCV.org, por lo que no es necesario realizar ningún tipo de pago, independientemente de su utilización con propósitos comerciales o de investigación.

La plataforma fue lanzada oficialmente en 1999, como iniciativa de Intel para aplicaciones computacionales avanzadas en proyectos de imágenes tridimensionales y de trazados de rayos. En su desarrollo trabajaron expertos en optimización de Intel Russia junto con el equipo de Intel dedicado al rendimiento de las librerías. Un año más tarde se lanzó la primera versión alfa, accesible públicamente y presentada en la conferencia "IEEE Conference on Computer Vision and Pattern Recognition", la cual tuvo cinco actualizaciones diferentes, lanzadas como versiones beta a lo largo de los siguientes cinco años.

La versión 1.0 se lanzó finalmente en 2006, y a mediados de 2008 la iniciativa obtuvo soporte corporativo por parte de Willow Garage, lanzando a finales de ese mismo año la versión 1.1. En 2009 surgió OpenCV 2, que incluía grandes cambios en la interfaz para el lenguaje C++, con nuevas funciones, más seguras y sencillas, y mejorando el rendimiento de las implementadas hasta el momento, especialmente para sistemas y procesadores multinúcleo. A partir de entonces, nuevas versiones aparecían cada seis meses, desarrollándose a manos de un equipo independiente ruso financiado por corporaciones de ámbito comercial.

En agosto de 2012, OpenCV pasó a manos de una fundación creada sin fines lucrativos encargada de sustentarla y mantenerla, OpenCV.org. Esta creó además dos páginas web destinadas a desarrolladores y a usuarios, las cuales son ampliamente utilizadas hoy en día.

Entre las principales aplicaciones de OpenCV destacan:

- Herramientas para funciones de 2D y 3D
- Sistemas de reconocimiento facial
- Interacción entre usuario y máquina
- Reconocimiento de gestos
- Detectores de movimiento
- Identificación de objetos
- Realidad aumentada
- Herramientas para robótica

Para el desarrollo de estas áreas emplea librerías de aprendizaje automático aplicadas a cálculos estadísticos, con algoritmos como los árboles de decisión, k-nn (k nearest neighbors), boosting, redes neuronales o SVM (Support Vector Machine).

OpenCV es multiplataforma, por lo que puede utilizarse tanto en el sistema operativo Windows como en Linux, OS X o incluso Android, Maemo, FreeBSD, OpenBSD, iOS y Blackberry 10. Todas las descargas están disponibles en la página de SourceForge.

La principal interfaz de programación empleada para OpenCV es C++, aunque contiene algunas librerías en lenguaje C. También se han desarrollado recientemente interfaces de programación para otros lenguajes como Python, Java o MATLAB/Octave, cuya documentación esta accesible en Internet.

3.2.2 JavaCV

Debido a que OpenCV está desarrollado para lenguajes C y C++, y que para su utilización en otros lenguajes únicamente aporta soluciones de integración en código nativo de sus librerías (lo cual conlleva cierta dificultad), el trabajo en entornos Java se convierte en un proceso arduo. Por ello, y con el objetivo de facilitar la conversión a Java de librerías de visión artificial desarrolladas por OpenCV, además de por otros grupos de investigación en este campo como FFmpeg, libdc1394, PGR FlyCapture, OpenKinect, videoInput y ARToolKitPlus, surgió la iniciativa de desarrollar una biblioteca de código libre con adaptaciones de algunas de estas librerías a Java, denominada JavaCV. Esta herramienta además permite su utilización en entornos Android, y está en constante evolución y desarrollo. Toda la documentación y librerías se pueden localizar en GitHub en el enlace https://github.com/bytedeco/javacv.

3.3 BioAPI

BioAPI (*Biometric Application Programming Interface*) es el nombre de un consorcio fundado con el fin de desarrollar una API que permitiese el funcionamiento de las aplicaciones biométricas en cualquier dispositivo y plataforma, independientemente del propietario. Esto fomenta la competitividad entre vendedores y disminuye los riesgos para el consumidor. Está formado por un conjunto de 120 compañías y organizaciones interesadas en el desarrollo de un estándar, con el fin de extender el uso de las tecnologías biométricas y favorecer su crecimiento y evolución. Algunas de estas compañías son Compaq, BioScrypt, NIST, Iridiam, Infineon, Saflink y Unisis, así como otras informáticas como IBM o Hewlett-Packard.

3.3.1 Historia

La formación de este consorcio fue anunciada en abril de 1998, y a finales de año ya se había definido la arquitectura de la API así como algunos otros componentes asociados. En marzo de 1999, el NIST (*National Institute of Standards and Technology*) junto con el US Biometric Consortium promovieron la integración en BioAPI del grupo HA-API (*Human Authentication API*), el cual había desarrollado ya en 1997 una API con la misma finalidad. Ambos accedieron, incorporando al consorcio este nuevo grupo de trabajo.

La primera versión de la especificación, 1.0, salió en marzo del 2000, y el modelo o referencia de implementación en septiembre del mismo año. En marzo del año siguiente se lanzó la siguiente versión, 1.1, también denominada ANSI/INCITS 358-2002 al constituir un estándar nacional americano (*American National Standard*). En 2003 dicha especificación fue ofrecida al recientemente creado comité internacional de estandarización ISO/IEC JTC1/SC37, el cual terminó publicando en 2005 una nueva versión 2.0, más avanzada, bajo el número de estándar ISO/IEC 19784.

A pesar de que BioAPI forma un estándar propio, es compatible con otros desarrollados también para aplicaciones biométricas. Muchos de estos los emplea en su plataforma. Algunos de estos estándares pertenecen a las organizaciones ISO (*International Organization for Standardization*), ANSI (*American National Standards Institute*), NIST o INCITS (*International Committee for Information Technology Standards*).

La especificación realizada, tanto previamente, como bajo la norma ISO/IEC 19784, está realizada en lenguaje ANSI C, por lo que ha tenido muchos problemas de adaptación a la industria, ya que en la época en la que fue publicada, casi todas las aplicaciones se desarrollaban en lenguajes con orientación a objetos. Esto llevó a que se iniciasen en 2009 los trabajos para crear una nueva versión BioAPI ajustada a este tipo de lenguajes, es decir, lo que se conoce como Object Oriented BioAPI, y que será publicado en 2015 con el código ISO/IEC 30106. En los desarrollos de dicha especificación trabaja muy intensamente el Grupo Universitario de Tecnologías de Identificación (GUTI) de la Universidad Carlos III de Madrid, grupo en el que se ha realizado el presente TFG. Este trabajo es un paso más en la depuración de dicho estándar, así como en la depuración de la documentación, permitiendo que desarrolladores sin conocimientos previos del estándar, puedan desarrollar nuevas aplicaciones basadas en ISO/IEC 30106 en un breve espacio de tiempo.

3.3.2 Arquitectura

El Framework de BioAPI funciona como una interfaz entre la aplicación y los dispositivos pertenecientes a distintos propietarios. Estos dispositivos o módulos integrados pueden ser lectores de huella dactilar, escáneres de iris, cámaras, sistemas de imágenes vasculares o cualquier dispositivo de captura. También pueden tratarse de dispositivos que ayudan en el procesamiento de imágenes o en la extracción de características. Todo esto forma parte de la arquitectura de BioAPI, la cual sigue un modelo API/SPI por capas. A continuación en la Figura 10 se muestra un esquema básico de la misma:

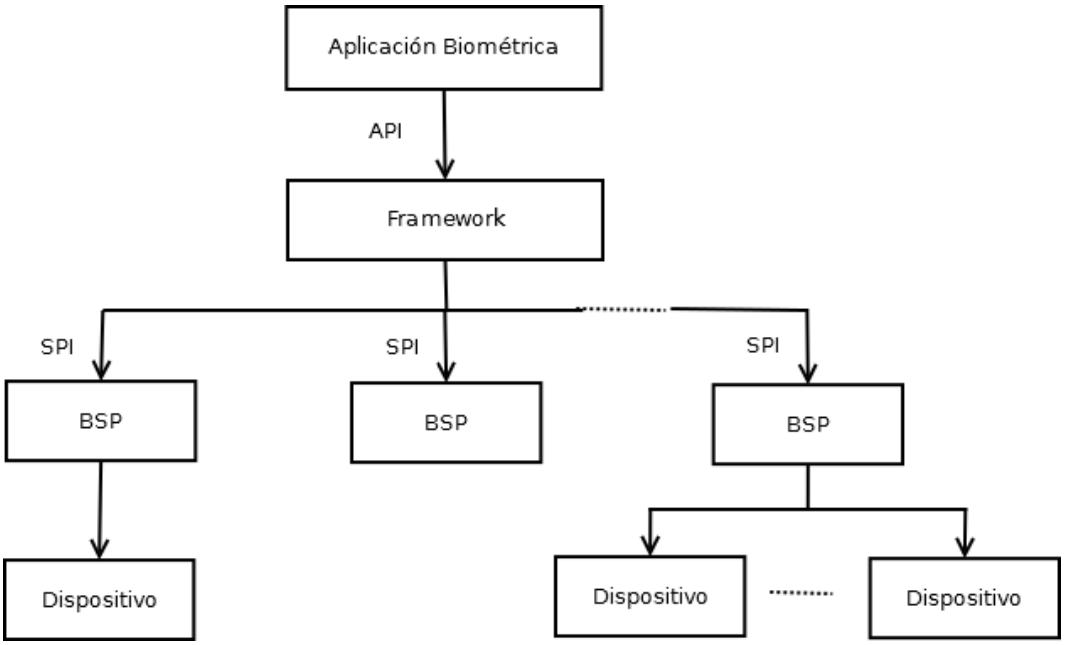

Fig. 10 – Arquitectura BioAPI

Como se puede observar, el nivel inferior lo constituyen los dispositivos, a los cuales acceden los BSP (*Biometric Service Providers*). Estos proveen de servicios a la aplicación a través de la interfaz SPI. Como ya hemos mencionado, el *Framework* hace de intermediario, comunicando la aplicación con las unidades BSP. La aplicación emplea las funciones de la API, compatibles con el Framework, para enviar y recibir la información.

3.3.3 BioAPI en Java

Actualmente, la única versión comercializada de BioAPI está desarrollada para lenguaje en C. Para trabajar en entornos orientados a objetos como Java, o en Android, el consorcio propone como solución el uso de empaquetadores o "wrappers" del código, usando JNI. Este procedimiento conlleva una gran complejidad. Por este motivo, además de la adaptación a lenguaje C# orientado a objetos llevada a cabo por la Universidad Carlos III de Madrid, la Universidad de Purdue llevó a cabo una adaptación de las especificaciones en C, pasándolas a lenguaje Java. Además puso a disposición las clases creadas organizadas en tres paquetes:

- org.bioapi: Contiene todos los componentes responsables de conectar, controlar y procesar las unidades que interactúan con el Framework. Incluye el registro de los componentes, el manejo de la sesión, las operaciones del BSP y el "match processing".
- org.bioapi.data: Define todas las estructuras de datos y modelos que describen las interfaces de "org.bioapi". Incluye la estructura del formato BIR, y los esquemas del BSP y BFP.
- org.bioapi.net: Almacena las interfaces empleadas en operaciones de la red. Incluye el Recurso Internacional de Identificación.

Estas carpetas se han empleado para el desarrollo de este proyecto, debido a la necesidad de adaptar a un entorno Android las especificaciones de BioAPI.

3.4 NetBeans

NetBeans es un entorno de desarrollo integrado (IDE) desarrollado principalmente para la realización de proyectos en lenguaje Java. Fue fundado en el mes de julio del año 2000 por Sun MicroSystems. La plataforma permite diseñar interfaces gráficas de forma visual, desarrollar aplicaciones web o crear aplicaciones para teléfonos móviles con Android, es decir, permite el desarrollo de todas las posibles aplicaciones Java (J2SE, web, EJB y aplicaciones móviles) aportando multitud de herramientas que facilitan el proceso. Además permite llevar un control de versiones, trabajando de forma colaborativa con otras personas, así como utilizar ANT. Contiene también la posibilidad de instalar packs de extensión que incorporan nuevas funcionalidades al entorno de desarrollo. Es un entorno multiplataforma, por lo que funciona indistintamente en diferentes sistemas operativos, ya sea Windows, Mac OS X o Linux. Existen otros entornos de desarrollo similares a NetBeans, como Eclipse, pero las herramientas que proporcionan son tan similares, que la utilización de uno u otro depende casi únicamente de las preferencias personales del desarrollador.

Como editor de texto incorpora muchas funcionalidades que facilitan la escritura en código. Es capaz de analizar el código y, además de resaltar en diferentes colores palabras según su función en el lenguaje, empareja o relaciona corchetes y palabras, puede actualizar un nombre al cambiarlo en todas sus apariciones, hace sugerencias y provee de plantillas. Todas estas opciones están disponibles para otros lenguajes además de Java, como C y C++, XML y HTML, PHP, Groovy, Javadoc, JavaScript y JSP, y mediante extensiones pueden aplicarse a otros lenguajes.

La plataforma está formada por un conjunto de módulos o componentes de software desarrollados de forma independiente, cada uno con un grupo de clases Java y un archivo *Manifest*, que interactúan con las APIs de NetBeans. Estos cuentan con una función específica, que puede consistir en dar soporte al sistema de control de versiones, soporte para Java, para edición o para otras funcionalidades, cubriendo así todas las posibilidades ofrecidas por la plataforma.

Desde el lanzamiento en el 2000 de la primera versión, la 3.1, se han ido generando nuevas versiones, introduciendo poco a poco mejoras. El 20 de noviembre de 2008 salió la versión 6.5, en la cual se extendían las características del Java EE existentes hasta el momento. Se

crearon nuevos packs, como el “NetBeans Enterprise Pack” que permitía el desarrollo de aplicaciones empresariales con Java EE 5, el “NetBeans C/C++” para proyectos en estos dos lenguajes, o el “PHP Pack” para PHP 5. La última versión lanzada actualmente es la 8.0, aunque ya está programado el lanzamiento de la nueva 8.0.1.

4 Diseño de la Aplicación

En este apartado se plantean los principales objetivos a cubrir propuestos para la realización del TFG y las diferentes opciones encontradas para su resolución, así como las decisiones tomadas a lo largo del proceso y los cambios realizados a lo largo de su desarrollo.

4.1 Planteamiento

El objetivo de este trabajo era el de realizar una aplicación para dispositivos móviles que aplicase técnicas de reconocimiento facial haciendo uso de las librerías y algoritmos desarrollados en OpenCV con esta finalidad. Además, el proceso biométrico de reclutamiento y verificación debería ser encapsulado en la plataforma para Java de BioAPI, favoreciendo mediante la incorporación de este estándar la interoperabilidad de la aplicación entre dispositivos.

4.2 Diseño de la Solución

Como punto de partida se pensó en una aplicación para Android en la que se pudiese aplicar un sistema de reconocimiento facial y resultase de cierta utilidad. Tras varias ideas, se tomó la decisión de desarrollar una aplicación destinada a organizar de forma automática las imágenes de un teléfono basándose en los contactos o amigos añadidos de forma manual por el usuario, utilizando para clasificarlas las librerías y algoritmos desarrollados en OpenCV. Una vez hecho esto, se llevó a cabo una planificación del trabajo. Inicialmente se instalarían y probarían las librerías de OpenCV en Android, para a continuación desarrollar la aplicación, su esquema y funcionamiento. Finalmente, y una vez terminada la aplicación, se encapsularía todo el proceso de reconocimiento en la plataforma de BioAPI.

4.2.1 Integración de las Librerías de OpenCV en Android

El desarrollo de este apartado supuso la toma de numerosas decisiones y la constante búsqueda de información con el objetivo de analizar las posibilidades que ofrecía cada opción así como su grado de complejidad.

Instalación

La primera dificultad se encontraba a la hora de adaptar las librerías al lenguaje de programación Android, ya que la biblioteca de OpenCV está escrita únicamente en C y C++. Para ello existían principalmente dos soluciones:

- Utilización del código nativo: Para la integración del código en el proyecto, una primera opción consistía en emplear directamente las librerías en C/C++ haciendo uso del JNI (*Java Native Interface*), un *Framework* de programación que permite la interacción de un programa en Java ejecutado en la máquina virtual (JVM) con otro u otros escritos en otros lenguajes como C, C++ o Ensamblador, embebiéndolos en el proyecto. De esta manera los métodos escritos en Java tendrían la opción de llamar a métodos implementados en código nativo, pudiendo acceder así a estas librerías. Para la compilación de estos lenguajes, el entorno de desarrollo de Android proporciona un paquete o extensión del SDK con diversas herramientas para este uso, el NDK (*Native Development Kit*).
- Adaptación a Java: La otra opción consistía en realizar una adaptación de estas librerías a lenguaje Java implementándolas directamente en Android. Sin embargo, y debido al creciente uso de las librerías de visión artificial, existe ya una adaptación de parte de la biblioteca de OpenCV escrita en Java y de código libre muy extendida llamada JavaCV. Esta librería incluye, entre otras, clases de reconocimiento facial.

La principal ventaja de la utilización del código nativo es el completo acceso a las librerías originales desarrolladas por OpenCV. Sin embargo existen también numerosos inconvenientes. A pesar de que el proceso de instalación del NDK no es extremadamente complicado, el uso de un programa en C o C++ embebido en un proyecto en Android supone un alto grado de dificultad, especialmente a la hora de su depuración. La localización de errores es muy complicada por lo que el desarrollo de la aplicación se convierte en un trabajo arduo. Además los programas resultan menos compatibles.

Por otro lado la utilización de la biblioteca desarrollada en JavaCV, aunque evita las dificultades mencionadas en el uso del código nativo, sólo es una opción en caso de que exista una adaptación a este lenguaje de las librerías de interés. El proceso de instalación de JavaCV puede tener cierta complejidad tanto para el desarrollo de programas en Java como para proyectos en Android, ya que es necesario tener en cuenta varios detalles, como la compatibilidad de versiones entre ambas plataformas a descargar (OpenCV y JavaCV), la correcta selección de las librerías de interés para cada caso, y su localización en el directorio adecuado de manera que esté accesible para el IDE de desarrollo empleado. La incorrecta instalación en alguno de estos aspectos genera un error difícil de localizar, por lo que este proceso puede resultar bastante lento. Sin embargo, y a diferencia de lo que sucede con código nativo, una vez superada la instalación, el desarrollo de la aplicación es sencillo al estar ya todo en lenguaje compatible con Android.

Inicialmente para el desarrollo de la aplicación se optó por usar el NDK, al ser aún desconocidas las librerías en JavaCV. Sin embargo, y debido a las múltiples dificultades encontradas, se profundizó en la búsqueda encontrando esta otra alternativa. Una vez comprobada la existencia de unas librerías y clases adaptadas para el reconocimiento facial se consultó con el tutor el uso de esta opción y se intentó llevar a cabo su instalación siguiendo

el tutorial proporcionado por los desarrolladores de la biblioteca así como la entrada en Drndos's Blog "How to run JavaCV (with sample face recognition) on Android ARM device – Netbeans and nbandroid" [34], que incluía un tutorial de como incorporar JavaCV en un proyecto Android. Fue necesario un traspaso del proyecto a un nuevo entorno de desarrollo, sustituyendo el uso de Eclipse como IDE por NetBeans debido a las facilidades proporcionadas en este entorno para la construcción del proyecto así como a preferencias personales. Tras varias dificultades encontradas en el proceso, debido principalmente a las versiones y los directorios, finalmente se consiguió con éxito su instalación.

Cabe mencionar que, debido a las dificultades encontradas para la correcta lectura de las librerías, se intentó aunque sin éxito, el uso de la herramienta de software Maven, empleada en el desarrollo de proyectos Java y que permite definir dependencias de componentes externos y descargarlas dinámicamente de los repositorios. De esta manera se pretendía añadir como dependencias las librerías en JavaCV.

Librerías

En la carpeta de descarga de JavaCV, concretamente la versión 0.7, están disponibles una serie de "samples" o clases de prueba entre las cuales se encuentran dos destinadas a aplicaciones de reconocimiento facial. Estas son:

- *FaceRecognition*: Se trata de una adaptación de un ejemplo desarrollado por OpenCV con el mismo nombre. La clase consta de una serie de métodos que se encargan de procesar las imágenes para realizar el aprendizaje y el posterior reconocimiento a partir de unos directorios fijos con imágenes previamente seleccionadas, especificadas en el código y localizadas en unos directorios determinados.
- *FacePreview*: Tal y como se especifica en los comentarios iniciales de la clase, se trata de un fusión entre la clase *facedetect* desarrollada por OpenCV y el ejemplo de Android *CameraPreview*. Hace uso de las librerías localizadas en las carpetas "javacv" y "javacpp" para combinar ambas. La función principal de esta clase es abrir la cámara del teléfono (en posición horizontal) y detectar caras de forma automática haciendo uso del clasificador *haarcascade_frontalface_alt.xml* disponible en el paquete de descarga de JavaCV.

El uso de ambas clases se puede combinar de manera que al usar la clase *FacePreview* no solo detecte caras a través de la cámara sino que además sea capaz de reconocer a la persona llamando a la clase *FaceRecognition* con la imagen de la cara encontrada. Con el objetivo de probar los resultados del algoritmo, se llevó a cabo este procedimiento. Sin embargo, tras varias modificaciones de las clases y una vez desarrollado gran parte del sistema de gestión de imágenes de la aplicación, se comprobó que los resultados obtenidos no eran del todo satisfactorios, por lo que se buscaron otras clases que pudiesen proporcionar mejores resultados. Finalmente se encontró otra clase desarrollada por Petter Christian Bjelland, y compartida en la entrada de blog "Doing face recognition with JavaCV" [38], desarrollada para entornos Java y que hace uso de librerías de JavaCV permitiendo así el acceso a contenidos y funciones adaptadas de OpenCV. Esta nueva clase, *OpenCVFaceRecognizer*, contiene la llamada a tres métodos que permiten la utilización de tres algoritmos diferentes para el reconocimiento: *createFisherFaceRecognizer()* para el algoritmo de Fisher,

createEigenFaceRecognizer() para el Eigenfaces (usado en el ejemplo anterior) y *createLBPHFaceRecognizer()* para el algoritmo Local Binary Patterns Histograms.

Esta clase es la utilizada en la versión final, aunque con diversas modificaciones añadidas para el uso en la aplicación y su funcionamiento en Android. En ella se desarrolla tanto el proceso de aprendizaje o entrenamiento, extrayendo las imágenes de un directorio específico cuyo nombre contiene su identificador o "label" (etiqueta), creando de esta manera un modelo a partir del algoritmo elegido, así como el reconocimiento o identificación de una imagen perteneciente a otro directorio.

Debido a que esta clase realiza el aprendizaje directamente con las imágenes proporcionadas, aplicándoles únicamente el preprocesado necesario para el reconocimiento, es necesario un paso previo en el cual se lleve a cabo la detección de caras y el recorte de las mismas para su almacenamiento en los directorios, empleándose posteriormente en *OpenCVFaceRecognizer*. Existen varios ejemplos de clases con este propósito, uno de ellos localizable en GitHub [35], el cual detecta rostros haciendo uso de librerías de JavaCV y dibuja un rectángulo alrededor. Esta clase fue empleada en este proyecto, añadiendo algunas líneas de código para llevar a cabo el recorte de la cara y adaptando su tamaño a uno fijo, procesos previos necesarios para la aplicación del algoritmo.

4.2.2 Desarrollo de la Aplicación

Una vez añadidas las librerías para el proceso de reconocimiento facial, se comenzó el desarrollo de la aplicación en Android. A medida que se fue avanzando en el trabajo, la estructura y funciones de la aplicación fueron variando, adaptándolas a un diseño más adecuado y sencillo para el usuario e incluyendo nuevas funcionalidades. En este apartado se comentarán únicamente la idea inicial y el resultado final, pudiendo así comprender la evolución en su desarrollo hasta llegar a su estado actual. Además se incluirá un diagrama de flujo para facilitar la comprensión de su funcionamiento.

Inicialmente, la aplicación contaba con una pantalla principal en la cual aparecían, además del título o nombre de la app, cuatro botones:

- Añadir contacto: Al pulsar esta opción, se abría una nueva *Activity* en la cual aparecía un cuadro de texto a rellenar para introducir el nombre del nuevo contacto. Una vez introducido se pulsaba OK. En caso de tratarse de un nombre válido (con contenido) se añadía como contacto, creando a su vez una carpeta en memoria externa para almacenar las imágenes.
- Álbum: Permitía acceder a la lista de contactos añadidos por el usuario. Estos aparecían como botones ordenados en forma de lista vertical, cada uno con su nombre. Al pulsar sobre un contacto, se abría una nueva pantalla con el nombre como título y las imágenes, mostradas una a continuación de otra en horizontal. Pulsando sobre una imagen se podía obtener un previsualización.
- Eliminar contacto: Abría el álbum con los contactos, indicando mediante un mensaje en forma de *Toast* que para eliminar alguno se debía mantener presionado. Al hacerlo mostraba un mensaje o diálogo de confirmación.

- Cámara: Mediante esta opción se podía abrir la cámara para tomar una foto y añadirla al álbum. Para ello, una vez hecha la fotografía, mostraba una previsualización de la misma y permitía seleccionar, mediante una lista de tipo *Spinner*, el álbum en el cual guardarla.

Algunas de estas opciones se han mantenido en la versión actual de la aplicación. Otras, tal y como comentaremos a continuación, han sido cambiadas por motivos de aspecto y simplificación o por modificaciones en el funcionamiento general de la misma. Además, nuevas pantallas y mensajes se añadieron con la introducción de los algoritmos de reconocimiento y detección de caras.

En la versión actual, la página inicial es directamente el álbum con los contactos, aunque mostrados a través de una lista dinámica. Cada entrada de la lista pertenece a un contacto y contiene su imagen, el nombre y dos botones en forma de icono que permiten editar el nombre o borrar el contacto.

Para añadir un nuevo contacto e incluirlo en la lista, se selecciona la opción correspondiente accesible a partir del menú. De esta manera se abre un mensaje o diálogo en el que se pide al usuario que introduzca el nuevo nombre. Como en el caso anterior, se comprueba la validez del mismo. La opción de editarlo esta accesible, además de a partir del icono, manteniendo presionado el elemento de la lista.

Al seleccionar un contacto se abre una *Activity* con el título y sus imágenes, tal y como se diseñó en la versión inicial de la aplicación, pero mostrando las imágenes en forma de tabla. En ella aparecen además dos nuevas opciones accesibles desde el menú que permiten importar imágenes o bien a partir de la galería del teléfono o bien utilizando la cámara para hacer una nueva foto. Al seleccionar esta última opción, el procedimiento a seguir es el mismo que el comentado para la versión inicial sólo que, tras la introducción de las librerías de reconocimiento, se incorporó una nueva *Activity* que, tras procesar la imagen a importar en busca de caras, muestra las caras encontradas para que el usuario indique cual pertenece al contacto. Esto último sucede también tras seleccionar la imagen desde galería. En caso de no encontrar ninguna cara, se informa al usuario y la imagen se guarda en el álbum directamente saltándose este último paso.

Dentro de cada álbum se muestra el botón “Buscar en Teléfono”, encargado de realizar la función principal de la aplicación, el reconocimiento. Al seleccionarla, permite elegir el directorio en el cual realizar la búsqueda de imágenes del contacto en cuestión. Una vez seleccionado el directorio, se muestra el progreso de búsqueda e información del proceso y al finalizar importa las imágenes de resultado positivo al álbum. Las imágenes importadas incorrectamente por la aplicación pueden ser fácilmente eliminadas por el usuario manteniéndolas seleccionadas. Aparece un mensaje de confirmación y se eliminan del directorio. Los detalles de la búsqueda pueden consultarse mediante una opción habilitada para ello. También hay una opción para elegir el número de imágenes comparadas en cada búsqueda, pudiendo ser 1, 5 o 10.

Con el objetivo de facilitar el aprendizaje y entendimiento del usuario a la hora de usar la aplicación, se añadió un tutorial. Este puede visualizarse al instalar la aplicación por primera vez, ya que aparece un diálogo ofreciendo al usuario esta posibilidad. También esta accesible en cualquier momento a través del menú de la pantalla principal.

Por último, y con el objetivo de asemejar el aspecto de la aplicación a las ya existentes, se introdujo una pantalla inicial de presentación con el título y el icono cuya duración es únicamente de 5 segundos. Una vez transcurridos estos segundos, se muestra la pantalla inicial comentada anteriormente con la lista de contactos.

En cuanto a la versión de Android mínima utilizada para el desarrollo de la aplicación, se seleccionó la 3.2, correspondiente a la API 13. Esta incluye algunas tabletas y los teléfonos más recientes. Inicialmente se intentó desarrollar una app compatible con versiones anteriores a esta, pero debido a la utilidad de las nuevas funciones incorporadas en versiones más recientes, las mejoras en aspecto y la creciente desaparición del uso de dispositivos empleando versiones antiguas se eligió esta versión. La Figura 11[36] contiene un gráfico extraído de la página "Android Developers" que muestra, según los datos obtenidos del 1 al 7 de julio de 2014 a partir del Google Play Store, el porcentaje de dispositivos en uso corriendo cada versión de Android:

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	11.4%
4.1.x	Jelly Bean	16	27.8%
4.2.x		17	19.7%
4.3		18	9.0%
4.4	KitKat	19	17.9%

Data collected during a 7-day period ending on July 7, 2014.
Any versions with less than 0.1% distribution are not shown.

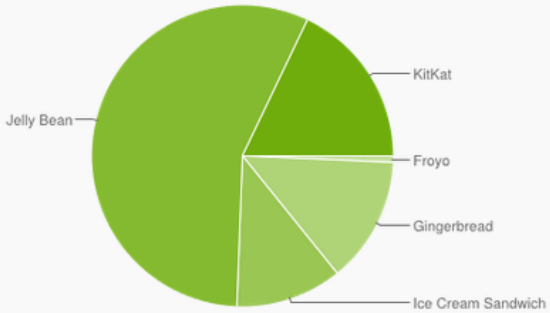

Fig. 11 – Gráfico sobre el uso de las versiones de Android en julio de 2014 [36]

A continuación se muestran una serie de diagramas de flujo representando el funcionamiento final de la aplicación:

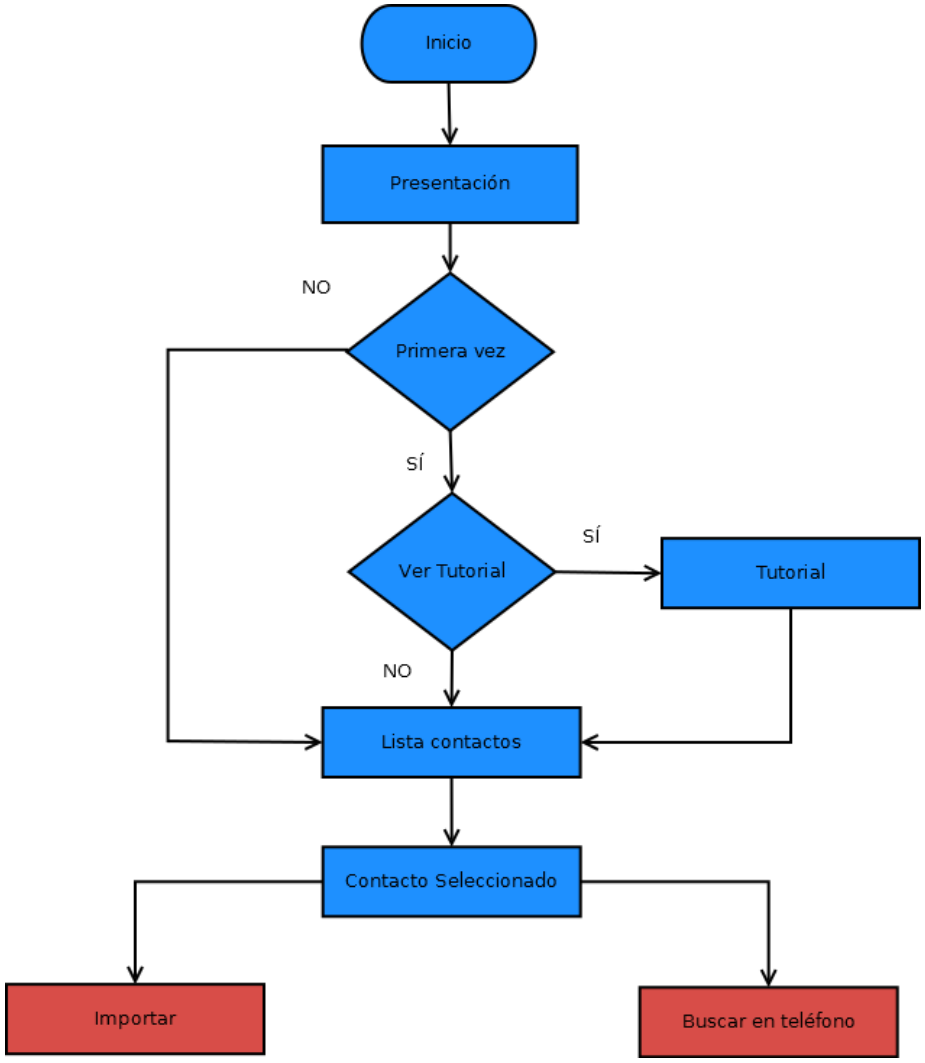

Fig. 12 – Diagrama representativo del inicio de la aplicación

En el diagrama mostrado en la Figura 12 se muestra el funcionamiento de la aplicación al ser ejecutada. En el caso de ser la primera ejecución, ofrece al usuario la opción de seguir el tutorial con el fin de entender el funcionamiento de la aplicación y aprender a utilizarla. Si no es el caso, tras mostrar una presentación de la aplicación con el icono de la misma y el título, se abre directamente la lista de personas añadida por el usuario mostrando sus nombres e imágenes de contacto así como las opciones de borrar y editar. Al seleccionar uno de los contactos se abre una nueva pantalla en la cual aparecen las diferentes imágenes contenidas en el álbum. En ella, aparece en forma de botón la opción de "Buscar en Teléfono", así como las opciones del menú para importar imágenes mediante cámara o seleccionándolas de la galería.

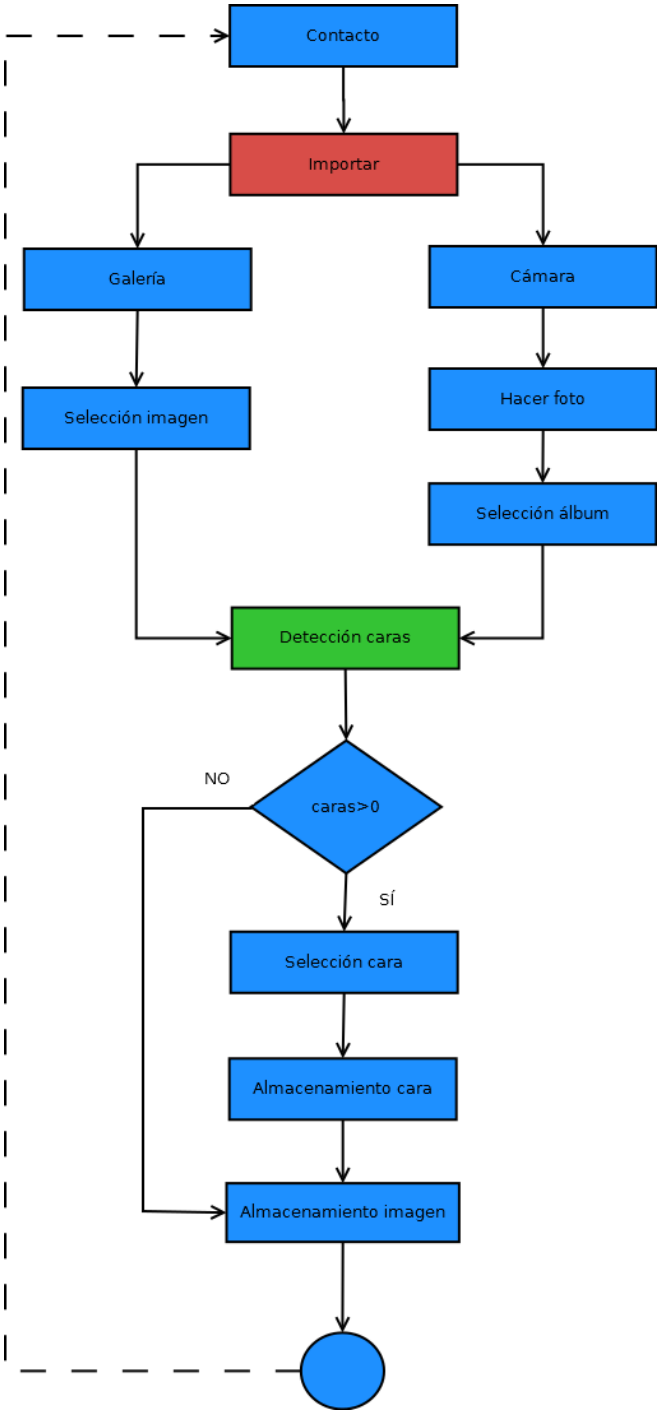

Fig. 13 – Diagrama para el proceso de importación

En el caso de querer importar imágenes al álbum de contacto, existen dos opciones: abrir la cámara para hacer una foto o abrir la galería para seleccionar una de las fotos almacenadas en el teléfono. Tal y como se muestra en la Figura 13, estos procedimientos aplican las librerías de OpenCV (y JavaCV) incluidas en el proyecto para llevar a cabo la detección de caras sobre la imagen importada. Tras llevar a cabo este proceso, se muestran al usuario las caras encontradas con el fin de que seleccione la perteneciente al contacto y de esta manera poder

aplicar a posteriori los algoritmos de reconocimiento facial para la búsqueda de imágenes del contacto en el teléfono. Tanto la cara como la imagen seleccionada se almacenan en el teléfono. En caso de no encontrar caras, se almacena únicamente la imagen, por lo que no se utiliza para el reconocimiento. Al finalizar, vuelve a mostrar el álbum del contacto con la nueva imagen añadida.

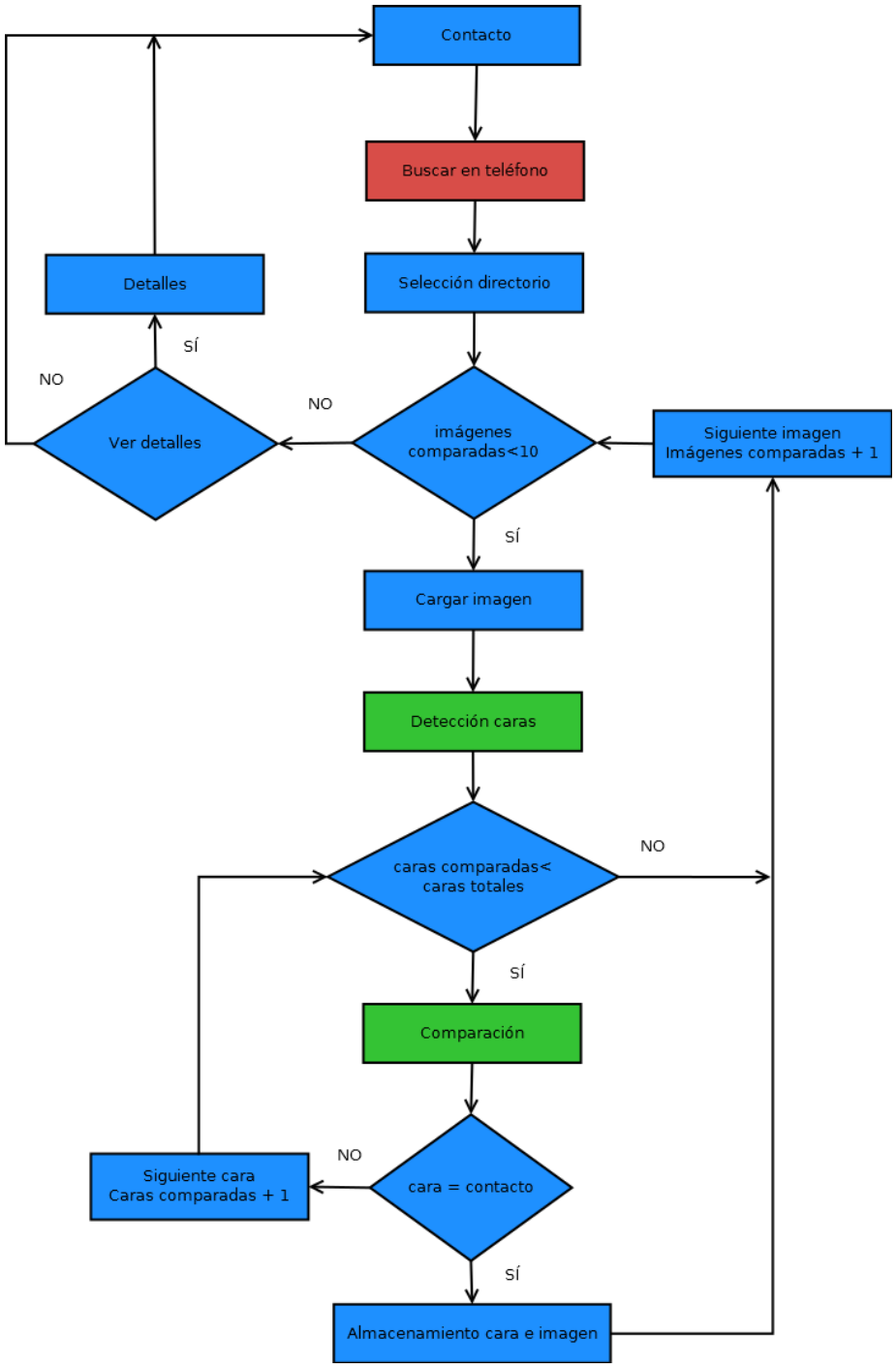

Fig. 14 – Diagrama del proceso de búsqueda de imágenes en el teléfono

Una vez añadidos varios contactos e imágenes, se puede seleccionar la opción de "Buscar en teléfono" para encontrar e importar automáticamente fotos del contacto siguiendo el procedimiento mostrado en la Figura 14. Para ello se selecciona previamente un directorio de búsqueda, del cual la aplicación obtiene únicamente las 10 (o 1 o 5, según las seleccionadas) primeras imágenes. Por cada imagen aplica la detección de caras, y por cada cara encontrada realiza la comparación. Si en el proceso de reconocimiento, el algoritmo considera que la cara pertenece al contacto, se almacenan tanto la imagen como la cara y se pasa a la siguiente imagen para realizar el mismo procedimiento. Al finalizar, se cierra el mensaje de progreso y se muestra el álbum con las imágenes importadas. Al volver a seleccionar esta opción, se vuelve a elegir el directorio de búsqueda, y en caso de tratarse de un directorio en el cual ya se ha realizado una búsqueda previamente, la aplicación selecciona las siguientes 10 imágenes encontradas.

4.2.3 Encapsulación en la plataforma de BioAPI

Finalmente, una vez terminada (o muy avanzada) la aplicación en Android y con las librerías de OpenCV introducidas en el código, debía emplearse la adaptación a Java de la plataforma desarrollada por BioAPI para encapsular los algoritmos de reconocimiento bajo este estándar. El principal objetivo sería la creación de una unidad BSP con los módulos necesarios para el desarrollo de los algoritmos.

Para el funcionamiento de la plataforma era necesaria la introducción de un Framework de BioAPI que permitiese a la aplicación comunicarse con el BSP creado. Este fue proporcionado por el GUTI, además de un proyecto de adaptación a entornos en Android.

Para la creación del BSP debían generarse las unidades necesarias para llevar a cabo el proceso de reconocimiento. Inicialmente se escogieron las siguientes:

- Sensor: Destinada a la captura de la imagen.
- Archivo: Unidad para la gestión del almacenamiento de imágenes haciendo uso de una base de datos.
- Procesado: Encargada de aplicar el procesado a la imagen.
- Comparación: Dedicada al proceso de comparación e identificación.

Sin embargo, con el avance en el proceso de encapsulación se tomó la decisión de eliminar la unidad de sensor debido a que el dispositivo de captura en este caso en vez de tratarse de una unidad externa consistía en la cámara del teléfono, por lo que su gestión era más adecuada directamente desde el proyecto Android, es decir, de forma externa al BSP. Además para la obtención de imágenes se utilizaba también la importación desde galería, lo cual hubiera supuesto para su funcionamiento la supresión de uno de los dos métodos (procedimiento que se siguió inicialmente utilizando únicamente la galería) o la creación de otra unidad de sensor, complicando mucho el proceso.

El resto de unidades se mantuvieron, introduciendo en la unidad de comparación la llamada a la clase *OpenCVFaceRecognizer* y en la de procesado la conversión de la imagen a formato BIR (definido por BioAPI) siguiendo además el estándar ISO/IEC 19794-5 para el almacenamiento de imágenes de caras humanas en sistemas de reconocimiento facial. Para la generación de imágenes según este estándar, fue necesaria la creación de una librería que

generase los campos definidos (fecha, información facial, propiedades de la imagen, etc.), los rellenase con su valor correspondiente, e introdujese en forma de *array* de bytes la imagen procesada.

Una vez desarrollada la librería e implementados los métodos de reclutamiento o *enrol* y de identificación, así como sus correspondientes llamadas a métodos de procesamiento, almacenamiento y comparación (métodos también implementados), debía incorporarse su funcionamiento a la aplicación principal. Para ello se generaron unas clases y funciones encargadas de inicializar el *Framework*, extraer los BSP (en este caso uno), cargar sus unidades, asociarlas y guardar un objeto *session*, encargado de la comunicación entre ambos proyectos.

4.2.4 Evaluación de los Resultados

Además de las pruebas realizadas durante el desarrollo de la aplicación para la comprobación de un correcto funcionamiento de las clases y librerías seleccionadas, una vez finalizada la aplicación se llevarían a cabo unas pruebas más elaboradas destinadas a probar el rendimiento de los algoritmos. Debido a la disposición de tres algoritmos diferentes para el proceso de reconocimiento facial (Eigenfaces, Fisher y LBPH) se evaluarían los tres empleando un set de imágenes determinado y variando el número de imágenes empleadas para su entrenamiento. Una vez realizadas las pruebas se comprobarían los resultados y se llevarían a cabo los cambios necesarios para optimizar el proceso y mejorar así el rendimiento general de la aplicación.

En cuanto a la comprobación del correcto funcionamiento de la aplicación, se destinarían unas semanas de pruebas en las cuales el desarrollador podría probar diferentes procedimientos y buscar errores o bugs. Además se preguntaría a diferentes personas por su colaboración para la búsqueda de mejoras tanto visuales como de comprensión y facilidad en el uso de la aplicación y se entregaría al tutor para su corrección y evaluación.

5 Desarrollo

5.1 Introducción

En este apartado se comentará en detalle la evolución en el desarrollo de la aplicación, incluyendo las dificultades encontradas a lo largo del proceso, así como las soluciones y decisiones tomadas. Basándose en la planificación y el diseño del trabajo comentados anteriormente, el texto se ha dividido en cuatro bloques principales ordenados cronológicamente según su realización que definen las tres fases principales del trabajo.

5.2 Integración de OpenCV en Android

Como ya hemos visto en el apartado de diseño, la integración de las librerías de OpenCV en un entorno Android supuso grandes dificultades y tomas de decisiones, así como una búsqueda exhaustiva de los diferentes procedimientos y posibilidades existentes a la hora de llevar a cabo su instalación. Este proceso supuso además una importante dedicación de tiempo.

Previamente al desarrollo del proyecto y con el propósito de comprender en mayor medida las funcionalidades de la herramienta OpenCV, se investigó acerca de la biblioteca y su uso. Esta consiste en una serie de librerías de código libre destinadas a aplicaciones de visión artificial que incluyen funciones como la detección de movimiento o el reconocimiento de objetos. Entre ellas se encuentran una serie de librerías desarrolladas para llevar a cabo sistemas de reconocimiento facial, de interés para el desarrollo de este proyecto en cuestión.

OpenCV es multiplataforma, por lo que su uso se extiende a diferentes sistemas operativos como GNU/Linux, Windows o Mac OS X. Además incluye enlaces a descargas disponibles en su página web con algunas clases y ejemplos para los entornos móviles Android e iOS. Para su funcionamiento en Android, es necesaria la integración de sus librerías en el proyecto. La opción recomendada para ello por los desarrolladores es el uso del JNI (*Java Native Interface*) que permite la interacción y comunicación entre código en Java y código en otros lenguajes. De esta manera permite la incorporación del código nativo de OpenCV (lenguaje en C o C++) en el proyecto Android. Además, para el uso específico de aplicaciones Android interactuando con código nativo, existe una herramienta de desarrollo llamada NDK (*Native Development Kit*).

Para la implementación de la aplicación, inicialmente se intentó emplear estas herramientas siguiendo diversos tutoriales, entre ellos los proporcionados por OpenCV. Sin embargo, el proceso de instalación consta de cierta complejidad, y el desarrollo del proyecto utilizando código nativo se convierte en un proceso lento y complejo debido a la dificultad en su depuración. Por ello, tras conocer los inconvenientes que suponía y al encontrar dificultades en su instalación, se buscó una alternativa que permitiese el uso de las librerías sin la necesidad de usar estas herramientas.

5.2.1 Instalación de JavaCV

Con el objetivo de facilitar el proceso de conversión a Java de librerías de visión artificial desarrolladas por grupos de investigación en este campo como son OpenCV, FFmpeg, libdc1394, PGR FlyCapture, OpenKinect, videoInput y ARToolKitPlus, existe una biblioteca de código libre y en constante evolución con adaptaciones de algunas de estas librerías creadas para su uso en Java. Esta herramienta, denominada JavaCV, es por tanto una biblioteca cuya finalidad es la de generar una interfaz para el uso de librerías de visión artificial en entornos en este lenguaje. Además es compatible con plataformas en Android.

Debido a las dificultades encontradas al inicio del desarrollo del proyecto para la integración de las librerías de OpenCV en Android empleando código nativo, fue necesaria la búsqueda de una alternativa, descubriendo finalmente la existencia de esta biblioteca. Aunque su proceso de instalación resultó algo complejo, las ventajas que suponían el trabajar únicamente con código Java y sin las dificultades introducidas al incluir código en otro lenguaje, supusieron una gran ventaja para el desarrollo de la aplicación.

En la página principal del proyecto de JavaCV están disponibles para descargar las diferentes versiones publicadas hasta el momento. Estas están actualizadas según las versiones presentadas por OpenCV. Para su utilización, es necesario el uso de versiones compatibles entre ambas bibliotecas.

La instalación se llevó a cabo siguiendo el tutorial disponible en Drndos's Blog [34]. En él se utiliza como entorno de desarrollo el programa NetBeans, por lo que para poder seguir el procedimiento descrito se llevó a cabo un traspaso del proyecto a partir del IDE usado hasta el momento, Eclipse. Además fue necesaria la configuración del programa, instalando el *plugin* correspondiente para Android, "NBAndroid", tal y como se indica en la página del mismo [36]. Este permite desarrollar aplicaciones en este lenguaje usando NetBeans. Para el correcto funcionamiento de las librerías, el ordenador debía usar la versión 6 o la 7 del Java SE. En cuanto al SDK de Android, este ya estaba instalado en el ordenador debido a que ya se habían desarrollado aplicaciones previamente, por lo que únicamente fue necesaria su localización desde NetBeans así como algunas actualizaciones.

Una vez preparado el entorno de desarrollo, se descargaron los archivos necesarios para el uso de JavaCV en el proyecto, empleando las versiones más recientes de ambas bibliotecas, siendo en aquel momento la 0.7 para JavaCV y 2.4.8 en OpenCV. Para JavaCV se descargaron las carpetas en .zip "javacv-bin" y "javacv-cppjars", precompiladas y compatibles con los sistemas operativos más comunes, Windows, Mac OS X, Linux y Android. Una vez extraídas, se copiaron las librerías *javacpp.jar* y *javacv.jar* a la carpeta "libs" del proyecto. Dentro de la misma se creó otra carpeta, "armeabi", en la cual se incluyeron los archivos descargados con extensión .so.

Aunque el proceso de instalación parece sencillo, surgieron dificultades debido a problemas con la localización de las librerías por parte de NetBeans. Sin embargo finalmente, y tras varios días realizando pruebas e intentando solucionarlo, empleando incluso alternativas como el uso de Maven para la descarga de las librerías a partir de repositorios, se consiguió alcanzar su correcto funcionamiento.

5.2.2 Adaptación de las Librerías

Para la introducción de un sistema de reconocimiento facial en la aplicación, estaban disponibles en la carpeta "bin" de JavaCV dos clases de prueba (o "samples") escritas en Java. Estas clases eran:

- *FaceRecognition*: Esta clase contiene una serie de métodos que llevan a cabo tanto el aprendizaje como el reconocimiento de un conjunto de imágenes. En su método *main* se realizan dos llamadas, una al método *learn* para llevar a cabo el aprendizaje, y a continuación otra a *recognizeFileList*, para el reconocimiento. En la llamada de ambos métodos se pasa como parámetro un fichero .txt en el cual se indican los nombres de las imágenes a utilizar en el proceso y, en el caso de ser el fichero utilizado para el aprendizaje, un identificador delante de cada nombre para clasificarlas en función de las personas. Estos ficheros deben generarse manualmente. Para ambos procesos, la imagen pasa por una serie de transformaciones (paso a escala de grises y cambio de tamaño) realizadas en diferentes métodos de la clase. El algoritmo aplicado es el Eigenfaces, utilizando por ello el análisis de componentes principales (PCA) y creando así para cada persona una cara representativa como media de las utilizadas, cuyo resultado se denomina "eigenface".
- *FacePreview*: La función de esta clase es detectar caras automáticamente mediante la imagen proporcionada por la cámara. Al ejecutarla abre la cámara y comienza a procesar las imágenes capturadas en el preview cambiándolas de tamaño y aplicando una serie de transformaciones y algoritmos como el "canny pruning". Para la detección emplea un clasificador, el *haarcascade_frontalface_alt.xml*, el cual debe incluirse manualmente en el proyecto. Al encontrar una cara obtiene su posición y dibuja un rectángulo a su alrededor.

En el tutorial seguido para la instalación de JavaCV se utilizan ambas clases conjuntamente realizando pequeñas adaptaciones en la clase *FacePreview* para su interacción con la clase de reconocimiento. De esta manera, además de detectar caras, estas pasan por un proceso de reconocimiento en la clase *FaceRecognition* dando un resultado de identificación a mostrar por pantalla.

Para la comprobación del correcto funcionamiento de estas clases y sus resultados, se instalaron en la aplicación. Por motivos de comodidad a la hora de llevar a cabo las pruebas y gestionar la información, se cambió en sistema de ficheros por una base de datos SQLite en la cual almacenar los directorios de las imágenes y su identificador o "label". Además hubo un proceso de comprensión de las librerías con el fin de entender el procedimiento seguido.

Sin embargo, tras varias pruebas se comprobó que los resultados no eran los deseados, por lo que se buscó una alternativa para el proceso de reconocimiento, encontrando finalmente una clase más reciente y optimizada desarrollada para JavaCV y llamada *OpenCVFaceRecognizer*

[38] que, empleando las nuevas librerías de OpenCV para *FaceRecognizer*, llevaba a cabo un proceso similar al de la clase *FaceRecognition* de aprendizaje y reconocimiento. Debido al amplio uso de esta clase por sus mejores resultados y la posibilidad que ofrecía de usar tres algoritmos diferentes, se eligió para la realización de este trabajo. Su funcionamiento es el siguiente:

- *OpenCVFaceRecognizer*: Esta clase recibe en el *main* como parámetros dos argumentos, el primero con el directorio de las imágenes a utilizar como entrenamiento del modelo, y el segundo con la dirección de la imagen a reconocer. Las imágenes de entrenamiento se guardan en un vector de tipo *MatVector* aplicando un filtro para utilizar únicamente las de extensión .png, y se crea un *array* de enteros con el mismo tamaño para almacenar los identificadores correspondientes de cada imagen. Los identificadores o "labels" se obtiene a partir del nombre con el mismo procedimiento que se utilizaba en la clase *FaceRecognition*, es decir, siguiendo el formato *<identificador>-nombre_imagen.jpg*, por lo que las imágenes deben estar previamente etiquetadas siguiendo este sistema. A continuación se convierten las imágenes a escala de grises y se genera el modelo empleando uno de los tres algoritmos disponibles (Eigenfaces, Fisherfaces o LBPH). Por último se llama al método *predict* del algoritmo seleccionado con la imagen a reconocer pasada a escala de grises y se obtiene como resultado un entero indicando el identificador de la persona a la que más se asemeja, y que por tanto es considerada su identidad.

Para la aplicación se usó el procedimiento seguido en esta clase para el reconocimiento y el aprendizaje, introduciendo un par de cambios, como la eliminación del filtro para el uso de imágenes únicamente de formato .png, ya que las imágenes encontradas en el teléfono serían en su mayoría .jpg, y posteriormente estableciendo unos valores de confianza para la predicción, realizando algunas modificaciones. Inicialmente se estableció el algoritmo de Fisher, pero tras las pruebas finales se sustituyó por el LBPH al aportar mejores resultados.

Con la introducción de esta librería en el proyecto, fue necesaria la creación de una clase que llevase a cabo previamente la detección de caras, ya que para el entrenamiento y el reconocimiento, las imágenes pasadas como parámetro a la clase *OpenCVFaceRecognizer* debían contener únicamente las caras y además estar normalizadas, teniendo todas el mismo tamaño. Existen varias clases con este propósito que realizan la detección siguiendo diferentes procedimientos. Para este trabajo se empleó la clase *FaceDetection* encontrada en GitHub [35].

- *FaceDetection*: Esta clase recibe como parámetro en el *main* el directorio de la imagen, cargándola como IplImage, formato estándar empleado en las librerías de OpenCV. A continuación llama al método *detect* que carga el clasificador seleccionado, en este caso *haarcascade_frontalface_default.xml* y lo emplea para buscar las caras en la imagen. Por cada cara encontrada dibuja a su alrededor un rectángulo rojo, mostrando finalmente la imagen resultante.

Para la utilización de esta clase fue necesaria la introducción de varios cambios, los cuales se comentan a continuación:

1. El clasificador se sustituyó por *haarcascade_frontalface_alt.xml* y se cambió su ubicación debido a problemas en su localización por parte de la aplicación.

2. El método *main* se sustituyó por *start*, y se añadieron nuevos parámetros incluyendo además del directorio de la imagen a procesar, la dirección en la que se guardan las imágenes de las caras de entrenamiento, el identificador del contacto y el nombre original de la imagen. Estos dos últimos parámetros son necesarios para el almacenamiento de la imagen siguiendo el formato *<identificador>-nombre_imagen.jpg* utilizado posteriormente para el reconocimiento.
3. En la llamada al método *detect* se añadieron los mismos parámetros empleados en *start* para realizar el correcto almacenamiento de las imágenes y se modificó para que devolviese el número total de caras encontradas, el cual sería devuelto también por el método *start*. Este cambio fue necesario debido a motivos relacionados con el funcionamiento general de la aplicación.
4. La principal modificación realizada en esta clase fue la introducción de un sistema de recorte de las caras encontradas. La clase original únicamente dibujaba rectángulos enmarcando las caras, por lo que se copió la imagen original (sin rectángulos rojos) y, empleando las posiciones de las caras encontradas, tomando para ello como referencia los rectángulos dibujados, se llevó a cabo el procedimiento de recorte en la imagen copiada. Las caras recortadas se guardarían en el directorio temporal pasado como parámetro. De esta manera, en caso de estar realizando una importación de imágenes, podrían mostrarse a continuación al usuario para la selección de la cara del contacto. Si el proceso llevándose a cabo fuese el de reconocimiento, estas caras serían empleadas una a una para la comparación y posteriormente borradas del directorio temporal.
5. Previo al almacenamiento de las caras, se introdujo el método cvResize con el objetivo de guardar todas las imágenes con el mismo tamaño, concretamente a 100x100 píxeles.

Todos estos cambios comentados fueron realizándose a medida que se avanzaba en el desarrollo de la aplicación y se generaba su modo de funcionamiento. Inicialmente se utilizó la clase *FaceRecognition* y no fue hasta más adelante al descubrir sus fallos cuando se introdujo la clase *OpenCVFaceRecognizer*, habiendo creado ya parte de la aplicación, incluyendo el almacenamiento de imágenes en memoria externa por carpetas según los contactos, y la utilización de una base de datos para su gestión. La clase *FaceDetection*, aunque necesaria para el correcto funcionamiento de la anterior, no se introdujo hasta más tarde, utilizando inicialmente imágenes preseleccionadas en las cuales aparecía únicamente una cara recortada, siendo además todas del mismo tamaño.

5.3 Desarrollo de la Aplicación Móvil

Previamente a la creación de la aplicación, y teniendo en cuenta los objetivos a cubrir en la realización de este proyecto, se decidieron las funcionalidades de la aplicación a desarrollar, creando así una aplicación móvil que aplicase los sistemas de reconocimiento facial con un objetivo más que el de una simple prueba del rendimiento de los algoritmos.

La aplicación ideada consistiría en un sistema de clasificación de fotografías del teléfono en función de contactos o amigos. En la propia aplicación se podrían añadir y borrar estos contactos, así como importar imágenes a cada uno de sus álbumes utilizando para ello la

cámara del teléfono. Las librerías y algoritmos de OpenCV se utilizarían para importar automáticamente imágenes encontradas en la memoria del teléfono en las cuales apareciese la cara del contacto.

Una vez resuelta la integración de las librerías de OpenCV en Android empleando para ello las librerías adaptadas de JavaCV, comenzó a crearse la aplicación, generando las diferentes clases y *Activities* por las cuales el usuario podría navegar.

5.3.1 Arquitectura y Aspecto Visual

El esquema de la aplicación y su aspecto fueron variando en gran medida a lo largo del proceso. En este apartado se comentará esta evolución, explicando los motivos de estos cambios. Además se verá el papel desempeñado por las librerías de OpenCV (JavaCV) dentro del proyecto así como su funcionamiento e interacción con la aplicación en el desarrollo de los procesos de detección y reconocimiento.

En sus inicios, la aplicación contaba con una página principal en la cual aparecían cuatro botones:

1. Añadir Contacto: Abría una nueva pantalla en la cual solicitaba la introducción de un nombre para el nuevo contacto. Al pulsar OK el contacto se guardaba y se creaba una carpeta en memoria externa con este nombre.
2. Álbumes: Al pulsar este botón se abría una nueva pantalla en la cual parecían todos los contactos añadidos hasta el momento. Estos aparecían en forma de lista, siendo cada uno un botón con el nombre del contacto. Al seleccionar uno de ellos se abría una nueva *Activity* en la cual aparecían las imágenes del álbum.
3. Cámara: Mediante este botón se accedía a la cámara del teléfono. Al tomar una fotografía, se abría una nueva pantalla en la cual se mostraba una vista previa de la imagen y una lista desplegable con los contactos, para seleccionar el álbum en el que guardar la imagen. Al pulsar OK la imagen se guardaba en la carpeta del contacto y se mostraba en su álbum dentro de la aplicación.
4. Borrar Contacto: Esta opción únicamente abría la lista de álbumes y mostraba un mensaje indicando que para borrar un contacto debía presionarse sobre su botón durante unos segundos. Al hacerlo se mostraba un mensaje de confirmación y al dar a OK borraba el contacto de la lista así como su carpeta en memoria externa incluyendo las imágenes almacenadas en la misma.

Este esquema se utilizó como punto de partida, añadiendo progresivamente cambios. Para mejorar su aspecto se diseñó por cada botón de la página principal un icono provisional. Se sustituyó la *Activity* de añadir un nuevo contacto por un *Dialog* y se mejoró la presentación de las imágenes de cada álbum, mostrándolas ordenadas en forma de tabla en lugar de la lista horizontal inicial. También se añadió una comprobación del nombre introducido en la opción "Añadir Contacto" de manera que no admitiese nombres vacíos, es decir, sin ninguna letra.

Sin embargo, el cambio introducido más importante fue la incorporación de un nuevo sistema de importación de imágenes, la selección desde galería. Esta función de "Importar Imagen" se añadió a las opciones del menú dentro de la pantalla de álbum. Al seleccionar la opción, se

abría la galería mostrando todas las imágenes del teléfono, pudiendo seleccionar una para importarla al álbum y guardarla así en la carpeta de memoria externa correspondiente.

Una vez diseñada y creada la aplicación con las funciones básicas, se introdujo el uso de las librerías de OpenCV. Inicialmente, y como ya hemos comentado previamente, se empleó la clase *FaceRecognition*, para la cual se creó una carpeta en memoria externa llamada "all" en la cual se almacenarían todas las imágenes incluidas en los diferentes álbumes de los contactos siguiendo el formato *<identificador>-nombre_imagen.jpg*. Esta carpeta formaría el directorio con las imágenes de entrenamiento empleadas por el algoritmo para llevar a cabo el proceso de reconocimiento. Para gestionar los datos de las imágenes y contactos se creó una base de datos en SQLite con una tabla en la cual cada entrada contenía el nombre de la imagen añadida, el del contacto en el cual se había guardado, y un identificador propio del contacto.

Tras varias pruebas, empleando para el reconocimiento un directorio específico con imágenes previamente seleccionadas, se comprobó que el funcionamiento del algoritmo no era el adecuado, por lo que se sustituyó la clase utilizada hasta el momento por una nueva, *OpenCVFaceRecognizer*. Esta permitía usar tres algoritmos diferentes e incluía librerías de OpenCV más recientes y optimizadas. Inicialmente, y debido a que esta nueva clase no aplicaba la detección de caras previa, se utilizaron imágenes preseleccionadas y recortadas de caras, añadiendo algunas en la carpeta "all" para el entrenamiento y el resto en el directorio de prueba. Más tarde, y una vez realizadas algunas pruebas, se incorporó la clase *FaceDetection* con este propósito. Se modificó el código de manera que además de detectar caras en una imagen las recortase, para a continuación guardarlas todas con el mismo tamaño (100x100) en un directorio temporal llamado "caras". Debido a que en una imagen pueden aparecer varias personas y por tanto encontrarse varios rostros, se introdujo una nueva *Activity* a la aplicación para el proceso de importación, en la cual se mostrarían al usuario las diversas caras encontradas para que indicase la perteneciente al contacto. Al seleccionar una, esta se guardaría en la carpeta "all" para utilizarla en el reconocimiento, borrándose a continuación el resto de caras encontradas, así como la carpeta temporal.

Aprovechando el almacenamiento de las caras de los contactos, se mejoró el aspecto de la pantalla de álbumes sustituyendo los botones, que consistían en un simple recuadro gris con el nombre del contacto, por una imagen del mismo con su nombre debajo. Además, en vez de aparecer uno debajo de otro, se colocaron en forma de tabla, mejorando así su presentación. Se diseñó también una imagen por defecto para los contactos que no tuviesen aún ninguna cara guardada.

Para el proceso de búsqueda de imágenes en el teléfono utilizando el sistema de reconocimiento facial, se empleó inicialmente un directorio específico con unas pocas imágenes preseleccionadas para comprobar su correcto funcionamiento. Debido a que la búsqueda podía alargarse, se introdujo el uso de un *AsyncTask*, ya que permite llevar a cabo operaciones en segundo plano, lo cual resultaba apropiado para este proceso. Además, se mantiene comunicada con el hilo principal, por lo que permite publicar los resultados o el progreso de la operación. Esto último resultaba de gran utilidad ya que, al tratarse de un proceso largo, permitía mostrar de esta manera al usuario el progreso según las imágenes procesadas y las imágenes restantes. Se creó para ello un cuadro de diálogo con el estilo *STYLE_HORIZONTAL*, mostrando una barra de progreso con el porcentaje de su evolución y el número de imágenes procesadas y por procesar, tal y como se muestra a continuación en la

Figura 15. Al finalizar, se cerraría el cuadro de diálogo y se mostraría un mensaje de "Importación finalizada". Además las imágenes procesadas consideradas del contacto según el algoritmo se añadirían al álbum automáticamente.

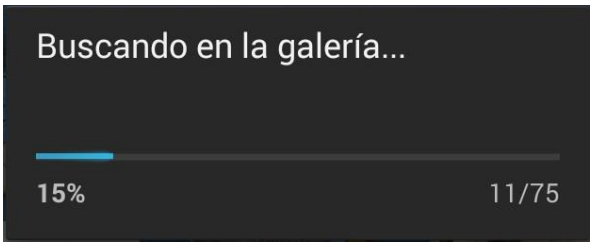

Fig. 15 – ProgressDialog horizontal

Cuando más tarde dejó de utilizarse el directorio de prueba y se probó la aplicación con los directorios reales de almacenamiento de imágenes (<*memoria_externa*>*/DCIM/100ANDRO* en teléfonos de Sony o <*memoria_externa*>*/DCIM/Camera* en otros) se descubrió que el proceso era parado automáticamente por Android pasado un tiempo. Por ello se intentó solucionar creando un *Service* (servicio) para realizar este proceso. Sin embargo, esto no resulto ser una solución ya que aun usándolo, el proceso terminaba trascurrido un tiempo al igual que pasaba con el *AsyncTask*. Tras consultar los mensajes generados por Android en el *log* se descubrió que, debido a que el proceso requería de bastante memoria RAM, la aplicación sobrepasaba el límite adjudicado por Android, por lo que el proceso se paraba sin generar errores. Debido a la dificultad que suponía intentar gestionar el uso de esta memoria liberándola durante el proceso y el desconocimiento del procedimiento y sus limitaciones, se decidió solucionarlo reduciendo a 10 el número de imágenes procesadas en cada llamada a la operación. Además se incorporó un contador de manera que al volver a realizar una búsqueda comenzase en la imagen siguiente a la última procesada. Este contador permitía además a la aplicación adaptarse a diferentes teléfonos en los cuales la cantidad de memoria RAM podía variar, siendo en algunos tan limitada que ni siquiera llegase a procesar 10 imágenes y se parase antes de su finalización.

Otro problema surgió al descubrir que las imágenes capturadas con la cámara, si se tomaban giradas, no aparecían corregidas en el álbum y al procesarlas en busca de caras no encontraba ninguna. Al buscar información acerca del suceso, se descubrió que en Android no está estandarizado el método en el cual se guarda la información acerca de la posición de la imagen. Esto provoca que cada marca emplee un método diferente, habiendo marcas que guardan esta información en el EXIF de la imagen y otras que siguen un procedimiento diferente. Debido a que no se podían aplicar todos los métodos, se eligió uno, el encontrado como más extendido. Por ello, este aspecto de la aplicación no funciona bien en determinados teléfonos que aplican métodos diferentes.

Tras la encapsulación de los procesos de importación y reconocimiento en la plataforma de BioAPI, lo cual se comentará más adelante en el apartado 5.4 de este documento, se realizaron las últimas modificaciones en el aspecto y arquitectura de la aplicación, dando lugar finalmente a la aplicación actual. Algunos de estos cambios fueron sugerencias realizadas por parte de personas a las que se les dio a probar la aplicación. Otros surgieron al realizar pruebas de errores. Los principales cambios realizados se comentan a continuación:

1. Se eliminó la pantalla de inicio en la que aparecían los cuatro botones, integrando sus funciones dentro de otras secciones de la aplicación. Concretamente, la opción de cámara se añadió en las opciones del menú de la pantalla de álbum de un contacto, añadiendo además un icono de cámara para mostrar en la *Action Bar* de la *Activity* (barra superior de la aplicación donde aparecen las opciones del menú). También se añadió un icono para la opción de "Importar Imagen". Las opciones de añadir o borrar contacto se añadieron a la pantalla de álbumes. Esta pasaría ahora a ser la nueva pantalla de inicio, mostrando directamente los contactos.
2. La pantalla de contactos también se modificó, mostrando los contactos en vez de formando una tabla, organizados en una lista dinámica (con la herramienta *ListView* de Android). Cada entrada de la lista corresponde a un contacto añadido, con su imagen de contacto, el nombre, y dos nuevos botones para borrar el contacto o editarlo. La opción de editar introduce la opción de cambiar el nombre de un contacto, lo cual no cambia únicamente el mostrado en la lista, sino que además se actualiza en la base de datos y en la carpeta de memoria externa. Entre las opciones del menú de esta pantalla se incorporó el "Añadir Contacto", introduciendo además una comprobación del nombre para ver si el contacto ya había sido añadido, y en tal caso notificarlo.
3. La opción de "Buscar en Galería" pasó a llamarse "Buscar en Teléfono", al definir así mejor su función. Además se creó para ella un botón dentro de la pantalla de álbum de manera que estuviese más accesible, ya que anteriormente estaba entre las opciones del menú y, teniendo en cuenta que es la función principal de la aplicación, no estaba lo suficientemente visible.
4. Para aportar a la aplicación un aspecto más profesional se añadió una pantalla de presentación que se mostraría al iniciarla. Su duración es de 5 segundos y contiene el logo y el título (FaceSnapp) de la aplicación.
5. También se añadió un "Tutorial" para mostrar las funciones de la aplicación y explicar cómo usarla. En la primera ejecución de la aplicación, tras la presentación, se muestra un mensaje en el que se ofrece seguir o no el tutorial. También está accesible para consultarlo en cualquier momento en las opciones del menú de la pantalla principal.
6. En cada pantalla se añadió también una opción de "Ayuda", en la cual se muestran las funciones específicas de la misma en más detalle.
7. Se adaptó la aplicación para mostrarse en inglés o en español según el idioma del teléfono. El lenguaje por defecto es inglés.

Tras la corrección del tutor, se añadieron unas últimas modificaciones:

1. Para la opción de "Buscar en Teléfono" se añadió una nueva función con la cual se permitía al usuario seleccionar el directorio en el cual realizar la búsqueda. Anterior a la introducción de esta opción, la búsqueda se realizaba en dos directorios definidos por ser los más habituales en los teléfonos. Sin embargo de esta manera el usuario no sabía en donde se realizaba la búsqueda, y en el caso de querer buscar en alguna carpeta fuera de estos directorios, no existía la opción. Por ello se introdujo la opción

de seleccionarlo a partir de una lista de carpetas mostrada, pertenecientes a la memoria externa del teléfono.

2. Se añadieron variables que guardasen por separado el contador del número de imágenes buscadas según el directorio. Además, al completar las búsquedas de un directorio, se incorporó un mensaje informativo notificándolo y avisando del reinicio de la búsqueda en la siguiente selección.
3. También se añadió la posibilidad de visualizar en detalle información del proceso de reconocimiento, para permitir al usuario consultar las imágenes procesadas, las caras encontradas, y los resultados obtenidos. Para ello se incorporó una opción en el menú, y además se incluyó un mensaje mostrado al finalizar el reconocimiento, con el cual se pregunta al usuario si desea consultar estos datos. Todo el registro de búsquedas aparece, cada uno con su fecha y hora de inicio, e incluye el nombre de la imagen, las caras encontradas, y por cada cara el contacto devuelto como resultado e información de si se ha importado o no (lo cual depende de si coincide o no con el contacto para el cual se realizó la búsqueda). En la pantalla se muestran todas las búsquedas realizadas, pudiendo borrarse mediante un botón incluido en la propia pantalla.
4. Se incorporó al menú una nueva opción con la cual seleccionar el número de imágenes a buscar en el directorio seleccionado. Pueden ser una, cinco o diez.
5. Se hizo un nuevo tutorial incluyendo las nuevas funciones y añadiendo una primera pantalla explicativa del propósito de la aplicación.

A continuación se muestran una serie de capturas de pantalla presentando el resultado final de la aplicación:

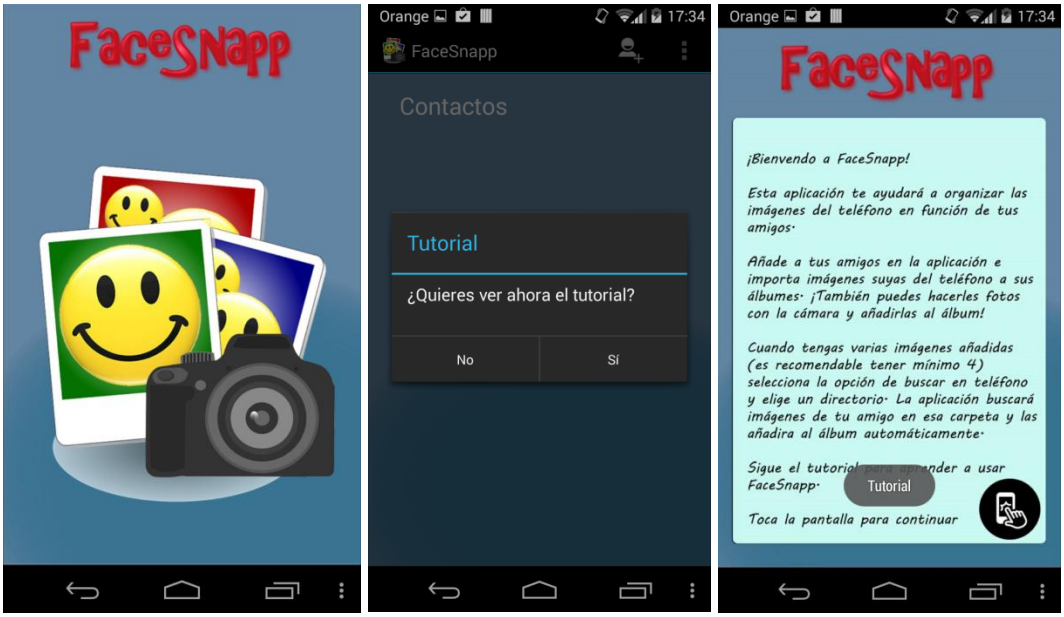

Fig. 16 – Presentación y tutorial

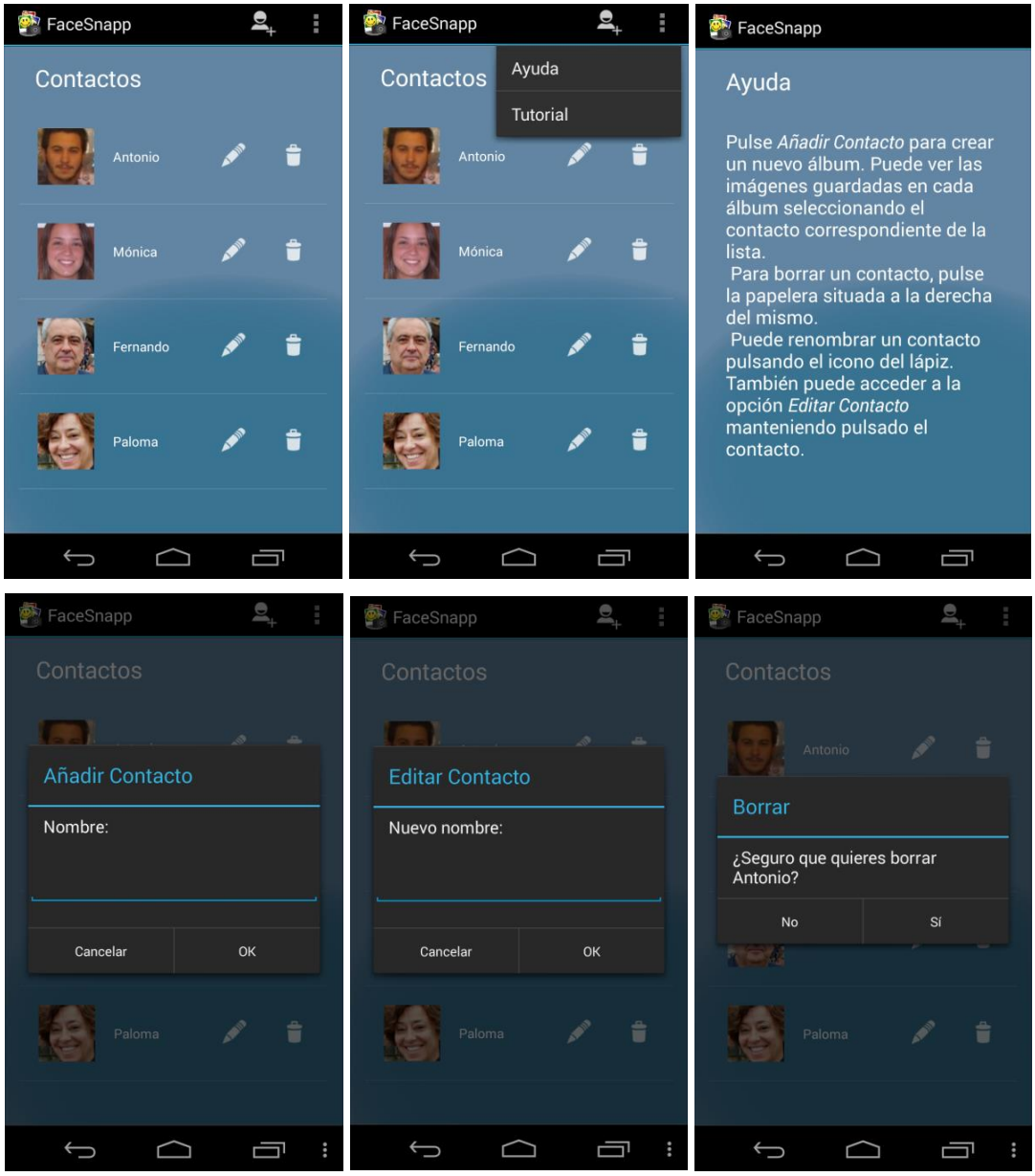

Fig. 17 – Lista de contactos, opciones del menú y botones

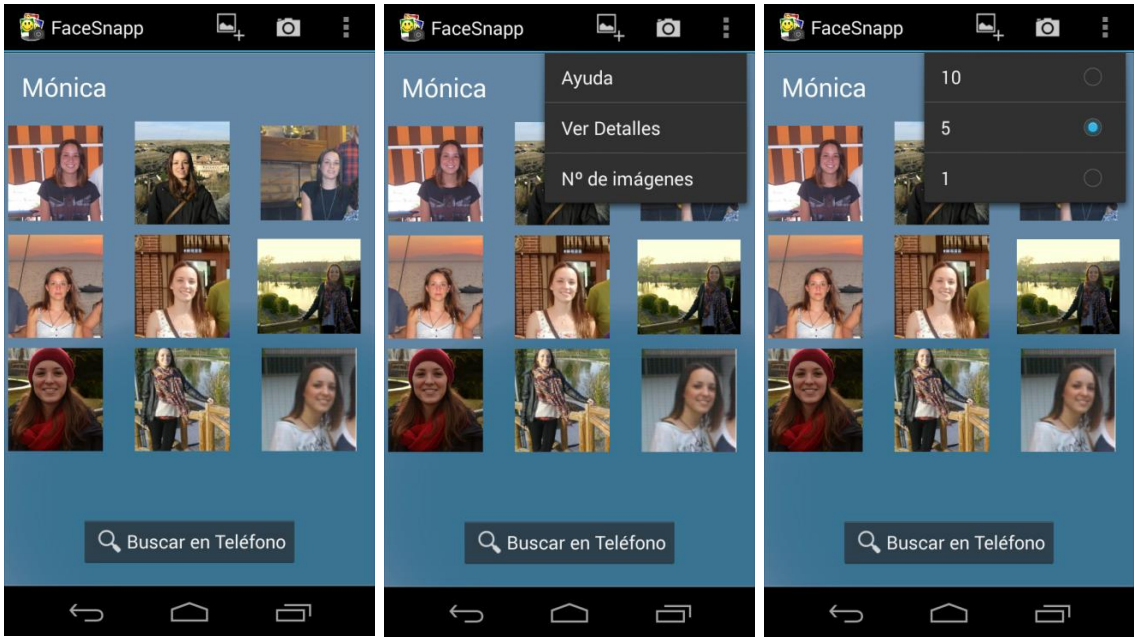

Fig. 18 – Álbum y opciones

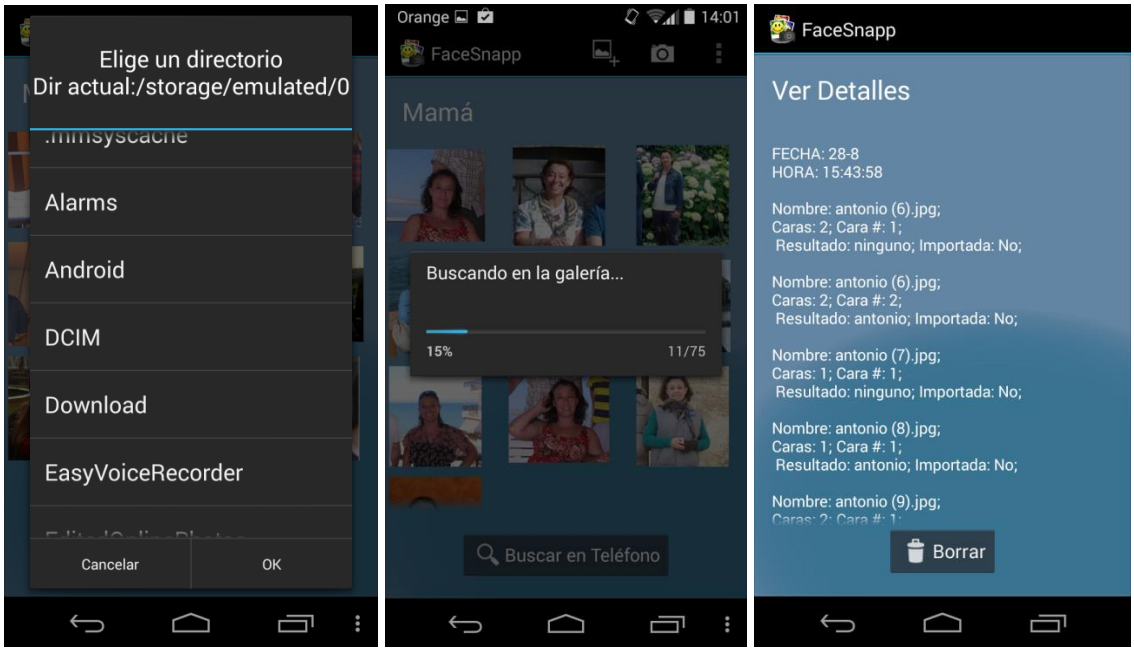

Fig. 19 – Búsqueda y detalles

5.3.2 Generación de una Base de Datos

Con el objetivo de gestionar los datos acerca de las imágenes importadas, el álbum del contacto en el que se añadían, y crear un "label" diferente por contacto para identificarlos en el proceso de reconocimiento, se creó una base de datos en SQLite y se creó en ella una tabla en la cual guardar toda esta información. Los campos de la tabla son los siguientes:

- *id*: En este campo se guarda el identificador, necesario para el proceso de reconocimiento. Cada id es propio del contacto y se genera automáticamente a partir de los métodos implementados en la base de datos.
- *name*: En él se indica el nombre del contacto en cuyo álbum se está guardando la imagen.
- *image*: Contiene el nombre con el que se guarda la imagen en el álbum. Si la imagen esta importada desde la galería, el nombre coincide con el original.

Con la encapsulación de los procesos de reconocimiento en BioAPI, y debido a que en esta plataforma las imágenes se guardan en otra base de datos con un formato propio, era necesario establecer una relación entre las imágenes guardadas en su tabla y las de la aplicación. Por ello se añadió un nuevo campo:

- *uuid*: Representa un identificador único para la imagen, formado por un total de 36 caracteres (16 bytes). El objetivo de este campo es establecer una relación con la tabla de imágenes en BioAPI para poder gestionar ambas conjuntamente. De esta manera al borrar una imagen, además de eliminarla en la tabla de la aplicación, se borraría en la de BioAPI accediendo a su posición en la misma a partir del *uuid* guardado (el cual coincide en ambas tablas).

Para administrar los datos de la tabla, se implementaron en la base de datos un conjunto de métodos con los cuales llevar a cabo la generación de los id, la eliminación de imágenes, la obtención de datos y otras acciones. A continuación se comentan los métodos creados y sus aplicaciones:

1. *getID(String nombre)*: Busca en la tabla algún contacto con el nombre pasado como parámetro y devuelve su identificador. Este método se utiliza principalmente para guardar en la carpeta "all" la imagen con el nombre precedido de su identificador. Como ya se ha explicado previamente, esto es necesario para el proceso de reconocimiento.
2. *getIDMax()*: Lee todos los id almacenados en la tabla y devuelve el de valor más alto. Esto se utiliza para adjudicar a un nuevo contacto un identificador, asegurándose de que no coincide con el de ningún otro ya creado. De esta manera, se obtiene el valor más alto, se le suma 1 y se le asigna al nuevo contacto.
3. *getNamesAlbums()*: Devuelve todos los nombres de los contactos o álbumes añadidos.
4. *getPicturesFromAlbum(String nombre)*: Devuelve los nombres de las imágenes almacenadas en el álbum del contacto pasado como parámetro.
5. *getUUID(String imagen)*: Devuelve el *uuid* de una imagen determinada. Esto se emplea para acceder a dicha imagen de la tabla en BioAPI.

6. *getUUIDreference(String nombre)*: Devuelve el *uuid* de la primera imagen encontrada en la tabla perteneciente al contacto, cuyo nombre es el indicado mediante el parámetro. Se emplea en el reconocimiento.
7. *getAlbum(int id)*: Sirve para obtener el nombre de un contacto o álbum a partir de su identificador. Esto se utiliza en la opción de "Ver Detalles" para mostrar el contacto devuelto como identificado en cada búsqueda, ya que este valor es el id del mismo.
8. *getUUIDsFromAlbum(String nombre)*: Devuelve los *uuid* de todas las imágenes almacenadas en el álbum indicado como parámetro. Esto se utiliza para borrar las entradas de las imágenes BIR guardadas en la base de datos de BioAPI, al borrar un álbum o contacto de la lista.
9. *deleteAlbum(String nombre)*: Borra todas las entradas de la tabla cuyo nombre de álbum (o contacto) sea el indicado en el parámetro.
10. *deletePicture(String imagen)*: Borra de la tabla la entrada correspondiente a la imagen indicada en el parámetro.
11. *update(String nombre_inicial, String nombre_final)*: Sustituye todas las entradas con el nombre de contacto *nombre_inicial* por el nuevo nombre *nombre_final*. Este método es llamado cuando se utiliza la opción de editar contacto, de manera que se actualizan todos los campos con el nuevo nombre.
12. *checkIfContactExists(String nombre)*: Comprueba si el nombre introducido pertenece a algún contacto existente. Este método se utiliza en la opción de añadir contacto, para notificar de este suceso.
13. *checkIfExisits(String imagen)*: Comprueba si la imagen esta almacenada ya en el álbum del contacto. Esto se utiliza en el proceso de búsqueda, ya que en tal caso la imagen no se compara.

5.3.3 Gestión y Almacenamiento de Imágenes

Una de las principales funciones de la aplicación es la de organizar por carpetas en memoria externa las imágenes. Para ello se estableció que, al añadir un nuevo contacto, se crease una carpeta con su nombre en memoria externa, la cual se guardaría dentro de un directorio creado específicamente para la aplicación y con el nombre FaceSnapp. Todas las imágenes importadas a la carpeta o añadidas desde la cámara se guardarían en su directorio correspondiente. Además los nombre en las imágenes importadas se mantendrían respecto a los nombres originales, y en las de las fotografías hechas a partir de la cámara, se generaría uno aleatorio (concretamente se decidió usar el tiempo y la fecha de la captura empleando la función *currentTimeMillis()* que devuelve el tiempo pasado en milisegundos desde el 1 de enero de 1970 00:00:00.0 UTC). Las imágenes incorporadas al álbum al usar la opción "Buscar en Teléfono" también se añadirían en la carpeta automáticamente.

Debido a que era necesario contar con un directorio específico para las imágenes empleadas en el reconocimiento facial, se creó una carpeta llamada "all" en la cual se guardarían todas las caras. Al importar una imagen, esta se añade a la carpeta del contacto y posteriormente se procesa en busca de rostros. Las caras encontradas se recortan y se guardan en una carpeta creada de forma temporal dentro del directorio de la aplicación llamada "caras". En el caso de

encontrar alguna, se muestran al usuario para que determine la que pertenece al contacto, y finalmente se guarda la seleccionada en la carpeta "all" con el identificador delante del nombre, y el resto de imágenes y el directorio temporal se eliminan.

El directorio "all" es empleado también para establecer la imagen de contacto mostrada en la pantalla principal. El procedimiento es el siguiente:

1. Por cada contacto de la lista se extrae su identificador propio o id.
2. Si el identificador no es 0 (id establecido para contactos que no aparecen en la base de datos al no tener ninguna imagen en su álbum) se obtienen los nombres de todas las imágenes guardadas en "all".
3. Se recorren todos los nombres obtenidos, extrayendo únicamente el id del mismo, ya que siguen el formato *<identificador>-nombre_imagen.jpg*.
4. Si el identificador del nombre de la imagen coincide con el del contacto, se obtiene la dirección de la imagen y se guarda. En ese momento deja de buscarse una imagen en "all" para ese contacto y se pasa al siguiente.
5. Finalmente, al crear la lista para mostrarla usando un *ArrayAdapter<String>* se consultan las direcciones guardadas y se establecen las imágenes encontradas. En caso de no haber encontrado ninguna imagen se utiliza una imagen por defecto proporcionada por la aplicación.

Al borrar una imagen desde la aplicación, la imagen se borra también en la carpeta en memoria externa y se comprueba si se guardó alguna cara en "all" extraída de la misma. Se comprueban todos los nombres de las imágenes en "all" y si, comparando el nombre sin el identificador con el de la imagen a borrar coinciden, se elimina la cara también. Lo mismo sucede al borrar un contacto. Al eliminarlo, además de borrarse la carpeta en memoria externa, se obtienen los nombres de todas las imágenes en "all" y se extraen sus identificadores. Si estos coinciden con el del contacto borrado, sus imágenes se eliminan de la carpeta para dejar de usarse en el algoritmo.

Por último, si se selecciona la opción de editar un contacto, además de cambiar el nombre en la lista y en la base de datos, se actualiza el de la carpeta, manteniendo todas las imágenes guardadas hasta el momento. El identificador se mantiene, por lo que no es necesario realizar ningún tipo de modificación en la carpeta "all".

5.4 Encapsulación en BioAPI

Una vez finalizada la aplicación, a falta únicamente de hacer pequeños cambios en su diseño y aspecto, se llevó a cabo el proceso de encapsulación en BioAPI, comprendiendo previamente a su implantación el funcionamiento del mismo y su estructuración. Los procesos de reclutamiento y verificación, es decir, la incorporación de contactos e imágenes y su posterior comparación en el reconocimiento, debían integrarse bajo esta Interfaz de Programación de Aplicaciones o API. El procedimiento seguido para la completa inclusión del estándar se puede dividir en varias fases, las cuales se detallaran en los siguientes apartados. Estas son: la agregación de las librerías y clases de BioAPI necesarias para el uso en Android y para la creación del *Framework* y de la unidad BSP; la implementación de las funciones de cada módulo del BSP así como las funciones principales de enrolamiento y

verificación, y la creación de una librería para guardar imágenes destinadas a sistemas de reconocimiento facial según la norma ISO/IEC 19794-5.

Finalmente fue necesario añadir y modificar algunas funciones de las clases y *Activities* de la aplicación original para adaptarlas al uso de este estándar, estableciendo una comunicación entre ambos sistemas.

5.4.1 Incorporación de las Librerías de BioAPI Java

Para poder trabajar con el estándar, el primer paso fue instalar las librerías y clases necesarias para la correcta comunicación entre la aplicación y las funciones de BioAPI, para la creación de las unidades BSP y para la implementación de sus funciones. Por ello se incorporaron como librerías de la aplicación los siguientes proyectos:

- BioAPIAndroid: Contiene un conjunto de clases que permiten el desarrollo de un sistema BioAPI en entornos de lenguaje Android. Algunas de estas clases son: *BSP.java* y *BSPManager.java*, para la creación y el uso de las unidades BSP; *Comparison.java*, *Archive.java*, *Processing.java* y *Sensor.java*, para las unidades de comparación, almacenamiento, procesamiento y captura; *BioAPIException.java*, para gestionar las excepciones lanzadas en el sistema; *Framework.java*, para la generación y utilización del Framework; *GUIProgressEventListener.java*, *GUISelectEventListener.java*, *GUIStateEventListener.java*, para comunicar a la aplicación eventos ocurridos dentro del sistema BioAPI; *AttachedSession.java*, para establecer una sesión entre la aplicación y el sistema.
- BioAPIFramework: Contiene la clase *Framwork_GUTI.java*, desarrollada por el GUTI con las funciones propias del Framework incluyendo la gestión de las unidades BSP.
- BSP_GUTI_Face: Es una adaptación de la clase *BSP_GUTI_Fingerprint.java*, desarrollada por el GUTI, a sistemas de reconocimiento facial, y concretamente al uso en esta aplicación. En ella hay dos clases, una del *BSPManager*, sin modificar, y otra en la cual están todas las funciones que se han implementado en el desarrollo de este proyecto para realizar el enrolamiento y verificación del proceso de reconocimiento facial.

Debido a que estas librerías necesitan acceder a las clases de OpenCV, fue necesario extraer las clases *FaceDetection.java* y *OpenCVFaceRecognizer.java* de la carpeta de la aplicación y crear un nuevo proyecto, al cual se le puso el nombre de “OpenCVClasses”, y en el cual se añadieron como librerías, estando de esta manera accesibles a BioAPI.

También se creó una nueva librería para el almacenamiento de imágenes faciales, la cual se comentará más adelante en el apartado 5.4.4.

5.4.2 Unidades del BSP y Funciones

Para la construcción de la arquitectura de BioAPI, se crearon una serie de módulos del BSP que llevarían a cabo las diferentes funciones en los procesos de enrolamiento y verificación. Estas unidades se eligieron según las necesidades de la aplicación y los métodos empleados en la misma. Las unidades son:

- GalerySensorUnit: Se encarga del proceso de captura de las imágenes a partir del sensor. Contiene, entre otros, el método *capture* e implementa la clase *Sensor*.
- SQLiteArchiveUnit: Es la unidad de archivo o almacenamiento. Implementa la clase *Archive* y contiene métodos encargados de gestionar las diferentes acciones a realizar en la base de datos, como *storeBIR* para añadir un nuevo archivo BIR o *deleteBIR* para eliminarlo.
- FaceProcessingUnit: En ella se realiza todo el procesamiento de imágenes, incluyendo la conversión a formato BIR. Implementa la clase *Processing* y contiene el método *process*.
- FaceComparisonUnit: Lleva a cabo la comparación para la identificación o verificación en el proceso de reconocimiento, empleando para ello la clase creada a partir de las librerías de OpenCV con este propósito. Implementa la clase *Comparison* y contiene los métodos *identify* y *verify*, para usar uno u otro según si el objetivo del proceso es realizar una identificación o una verificación. En el caso de esta aplicación, aunque el algoritmo lleva a cabo una identificación, el proceso general es una verificación, ya que únicamente se comprueba si la imagen procesada se trata o no del contacto.

A continuación se muestra un esquema de la arquitectura diseñada:

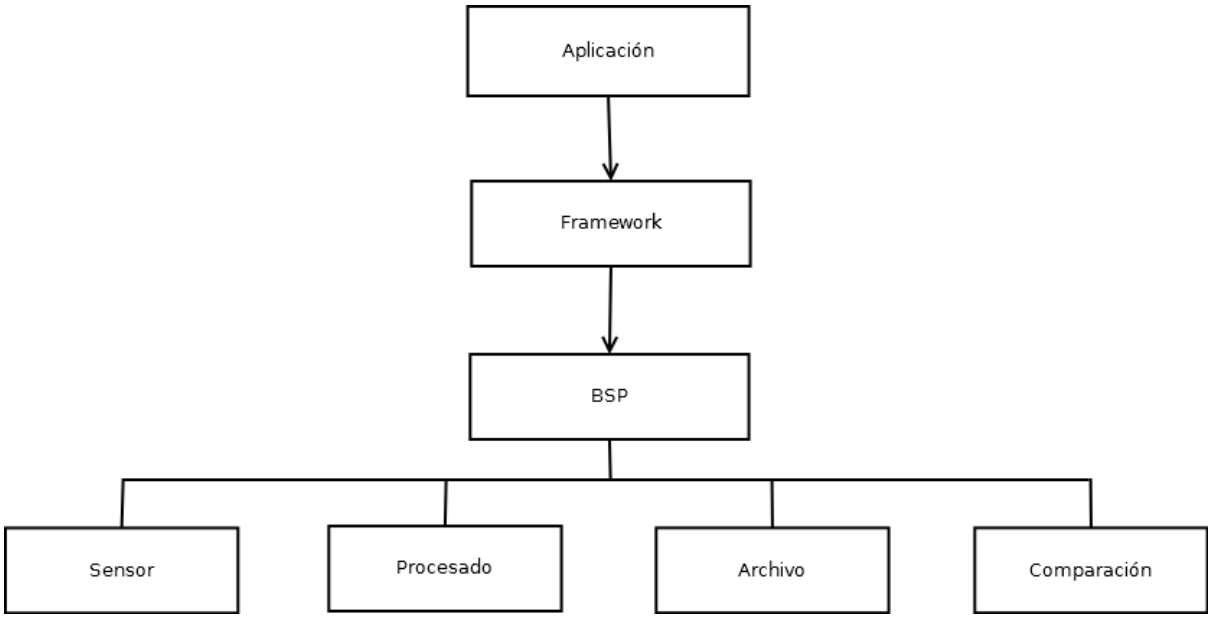

Fig. 20 – Diagrama de la arquitectura BioAPI diseñada

Debido a que para la captura de imágenes se emplean dos métodos, la importación desde galería y la captura con la cámara del teléfono, era necesaria la creación de dos unidades de sensor, lo cual complicaba enormemente el proceso. Por ello se decidió inicialmente seleccionar un único método, el más útil en la aplicación, que resultó ser la importación desde galería. Sin embargo, al avanzar en la implementación de los métodos, se descubrió la

complejidad de realizar esta operación (o la alternativa de la cámara) desde BioAPI, ya que son procesos íntimamente relacionados con Android y necesitan utilizar funciones propias del lenguaje, por lo que su incorporación en estas clases supondría un constante traspaso de un entorno a otro, perdiendo el sentido de la encapsulación. Por ello, y debido a que realmente la aplicación no empleaba ningún dispositivo externo para el proceso, se eliminó el uso de esta unidad, manteniendo fuera de la plataforma el proceso de captura.

En la librería "BSP_GUTI_Face" se encuentra la clase que contiene estas unidades y la implementación de sus funciones. Además contiene los métodos *enrol* y *verifyAggregated*, encargados del enrolamiento y verificación. Estos emplean los módulos y sus funciones a lo largo de todo proceso. En la aplicación, al añadir una imagen al álbum, esta pasa por un proceso de detección de caras y guardado. Esto es el proceso de enrolamiento en BioAPI, con el cual se añaden contactos e imágenes a la base de datos y se emplean posteriormente para llevar a cabo el reconocimiento facial. En la opción de "Buscar en Teléfono", la aplicación extrae las imágenes de la galería y las compara con el modelo de reconocimiento, creado a partir de estas imágenes incorporadas en el enrolamiento. Todo este procedimiento forma parte del proceso de verificación.

A continuación se explican los procedimientos seguidos para ambos procesos y las llamadas realizadas a las diferentes funciones de las unidades en cada una de estas acciones:

- *Enrol*: Ya que, como hemos comentado, el proceso de captura se realiza de forma externa, la función de *enrol* recibe directamente la cara recortada de la imagen a añadir. Por ello, solo realiza dos acciones, la de procesamiento, en la cual llama a *process* de la unidad *FaceProcessingUnit* y convierte la imagen al formato BIR, y la de guardado, en la cual recibe el BIR y lo introduce en la base de datos mediante el método *storeBIR(BIR biometricReference)* de la unidad *SQLiteArchiveUnit*.
- *VerifyAggregated*: Es el método encargado de la comparación. La imagen recibida, con la cara ya recortada, se procesa con el método *process* y se convierte a BIR para pasarla a continuación al método *verify*, el cual devuelve un resultado positivo o negativo según si se trata o no del contacto. Para realizar esta comprobación, se necesita un identificador de referencia, ya que el algoritmo de reconocimiento devuelve como resultado el id del contacto al que más se asemeja la cara procesada. Este identificador de referencia se extrae de la base de datos de la aplicación a partir de un uuid perteneciente a una imagen del contacto e introducido como parámetro al llamar a *verifyAggregated* desde fuera de BioAPI. De esta manera ambos valores se comparan para decidir el resultado positivo o negativo de la verificación. En caso de coincidir, la imagen se guarda como BIR en la base de datos mediante el método *storeBIR*. Además se guarda la nueva entrada en la otra base de datos creada para la aplicación y se añade la imagen al directorio correspondiente del contacto así como a la carpeta "all". Si no coincide, el método devuelve *false*.

5.4.3 Integración de BioAPI en la Aplicación

Para la incorporación de BioAPI en la aplicación, fue necesario introducir una serie de cambios, así como crear nuevas clases para gestionar la comunicación entre ambas.

Al iniciar la aplicación, se inicializa e instancia el *Framework* de BioAPI, para a continuación extraer los BSPs instalados (en este caso uno) mediante *enumBSP* y cargarlos. Una vez hecho esto, se obtiene un BSP y se consultan las unidades contenidas en el mismo mediante *queryUnits*. Las unidades de interés se asocian mediante *bspAttach*. Finalmente se inicializa la sesión con BioAPI y se guarda el objeto *session*, con el cual se comunicará con la aplicación. Todo este procedimiento se lleva a cabo con la llamada a una función, *bspInit*, añadida a la *Activity* inicial. También se introdujo la función *bspFinish* para cortar la comunicación, aunque no se utiliza en ningún momento en la aplicación ya que se desconecta automáticamente pasado un tiempo. Cada vez que se inicia (o reinicia) la aplicación se vuelve a realizar el proceso.

Se creó una nueva clase, *BioAPISessionSingleton*, para poder gestionar el uso de la sesión. Esta clase contiene los métodos *init*, utilizado en *bspInit* para inicializarla, y *getSession*, creado para poder acceder a este objeto *session*, de tipo *AttachedSession*, desde cualquier punto de la aplicación. Esto último fue necesario para posibilitar el acceso a las funciones de *enrol* y *verifyAggregated* desde la *Activity* del álbum, ya que ambas llamadas se realizan a partir de este objeto.

En los procesos de importar imágenes desde galería o a partir de la cámara, se incorporó una llamada a *enrol*. Tras importar la imagen se realiza el proceso de detección de caras, recortándolas y mostrándolas al usuario tal y como se realizaba antes de la introducción de BioAPI. Una vez seleccionada la cara del contacto, se guarda la imagen en la carpeta correspondiente y la cara en "all", y su directorio se almacena en las preferencias compartidas del sistema o *SharedPreferences*, indicando además mediante otra variable de tipo booleano que se va a llevar a cabo el enrolamiento. Esta variable es consultada al reiniciar la pantalla de álbum tras realizar estos pasos, llamando en caso de tener el valor positivo *true*, mediante el objeto *session* al método *enrol*. Aquí se extrae la imagen con la dirección guardada en *SharedPreferences*, y se procesa y guarda en la base de datos de BioAPI. A continuación, y otra vez en el entorno de la aplicación, es decir, fuera de BioAPI, se añade la nueva entrada en la tabla de la base de datos creada, rellenando el campo *uuid* con el valor correspondiente devuelto por el método *enrol*. Al finalizar, se muestra la pantalla del álbum con la nueva imagen añadida.

Para la opción de "Buscar en Teléfono", ya que el proceso de reconocimiento se integró en BioAPI, se incluyó en su función la llamada a *verifyAggregated* para llevarlo a cabo. Al seleccionar esta acción, se elige el directorio de búsqueda y se cargan 10, 5 o 1 imagen del mismo. Por cada imagen, se lleva a cabo la detección de caras, recortándolas y almacenándolas en el directorio temporal "caras". Entonces, por cada una de estas caras se llama al método en BioAPI, guardando previamente su dirección en *SharedPreferences*, y siguiendo así el mismo procedimiento mencionado para la importación. Además, a partir del método *getUUIDreference* de la base de datos, se extrae un *uuid* de referencia para la comparación y se pasa como parámetro. En *verifyAggregated* se extrae la imagen de la cara mediante el *path*, se procesa, y se guarda en un directorio temporal "tmp". Una vez procesada, en el método *verify* se utiliza la clase *OpenCVFaceRecognizer* para llevar a cabo

el reconocimiento, pasando la dirección de la cara guardada en "tmp" y el de las imágenes para el entrenamiento contenidas en "all", como parámetros. El proceso devuelve el *id* del contacto al que más se asemeja, y a partir del *uuid* de referencia se extrae el identificador del contacto con el cual se realizó la búsqueda para comparar. Si coinciden, se guarda la imagen en el álbum y la cara en "all". La imagen de la cara se guarda también en la base de datos de BioAPI, y con el *uuid* creado para almacenarla, se añade una nueva entrada en la tabla de la base de datos de la aplicación. Si por el contrario los identificadores no coinciden, el método devuelve un resultado negativo y se descarta la cara, pasando a comparar la siguiente de las encontradas en la imagen. Si se encuentra una cara que coincide o si ninguna de las extraídas lo hace, se carga la siguiente imagen y se sigue el mismo procedimiento hasta comparar todas las imágenes. Finalmente, al acabar el proceso se refresca la pantalla del álbum añadiendo las imágenes importadas.

Como ya se ha comentado en apartados anteriores, con la integración de BioAPI en la aplicación, fue necesario ampliar la tabla de la base de datos añadiendo un nuevo campo, el *uuid*. Esto fue necesario para relacionar las imágenes almacenadas en ambas tablas. Para ello, además de introducir el uso de este nuevo campo en todos los procesos de gestión de la tabla implementados hasta el momento, fue necesario introducir llamadas a la tabla de BioAPI al realizar operaciones de borrado, de manera que al borrar una imagen se borrase también su entrada en esta tabla. Al añadir una nueva entrada también debía añadirse este nuevo campo. Además se implementaron dos nuevas funciones en la base de datos, *getUUID*, con la cual se obtiene el *uuid* de una imagen determinada, y *getUUIDreference*, que devuelve el *uuid* de una imagen cualquiera perteneciente a un contacto determinado. Esta última se usa como ya hemos visto en el proceso de verificación.

5.4.4 Desarrollo de una Librería Siguiendo la Norma ISO/IEC 19794-5

El estándar de BioAPI establece un formato común para la información y los archivos intercambiados con la aplicación. Este formato se denomina BIR, *Biometric Information Record*, y puede contener datos en bruto (*raw* data), muestras procesadas y preparadas para utilizarse en un proceso de verificación o identificación, o datos del enrolamiento. Las muestras suelen almacenarse en la base de datos generada con este propósito. Este formato contiene tres campos principales, el SBH, el BDB y el SB. El primero consiste en un *Header* o encabezado con información acerca del contenido del BIR, lo cual permite a los dispositivos BioAPI interpretarlo. Su estructura según sus campos se muestra a continuación en la Figura 21[30]:

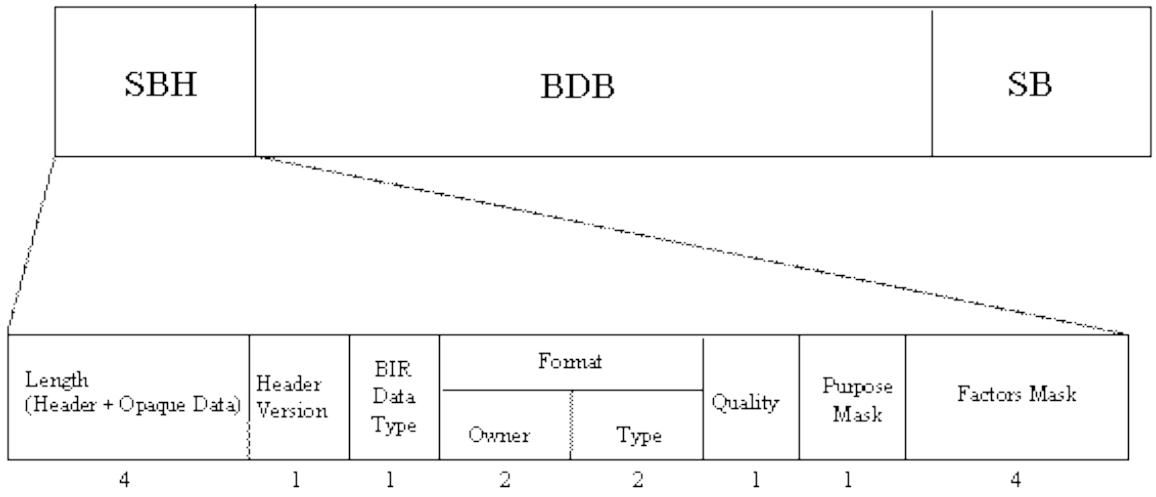

Fig. 21 – Estructura de un BIR [30]

Sin embargo, es en el campo BDB (*Biometric Data Block*) en el cual se almacenan los datos de mayor interés. Existen una serie de estándares que definen la estructura en la cual guardar los datos y la información dentro del BDB, como es el caso de BDIR (*Biometric Data Interchange Record*). Estos estándares se distinguen según el tipo de información que almacenan, siendo por ejemplo la ISO/IEC 19794-4 para imágenes de huellas o la ISO/IEC 19794-7 para firmas. En nuestro caso se aplicó la norma ISO/IEC 19794-5, destinada al almacenamiento de caras e información biométrica en sistemas de reconocimiento facial. Debido a que no se contaba con ninguna librería encargada de almacenar la información tal y como dicta la norma, fue necesario el desarrollo de una. A continuación en la Figura 22 se muestra un esquema con la estructura descrita en el estándar y seguida para el desarrollo de esta librería:

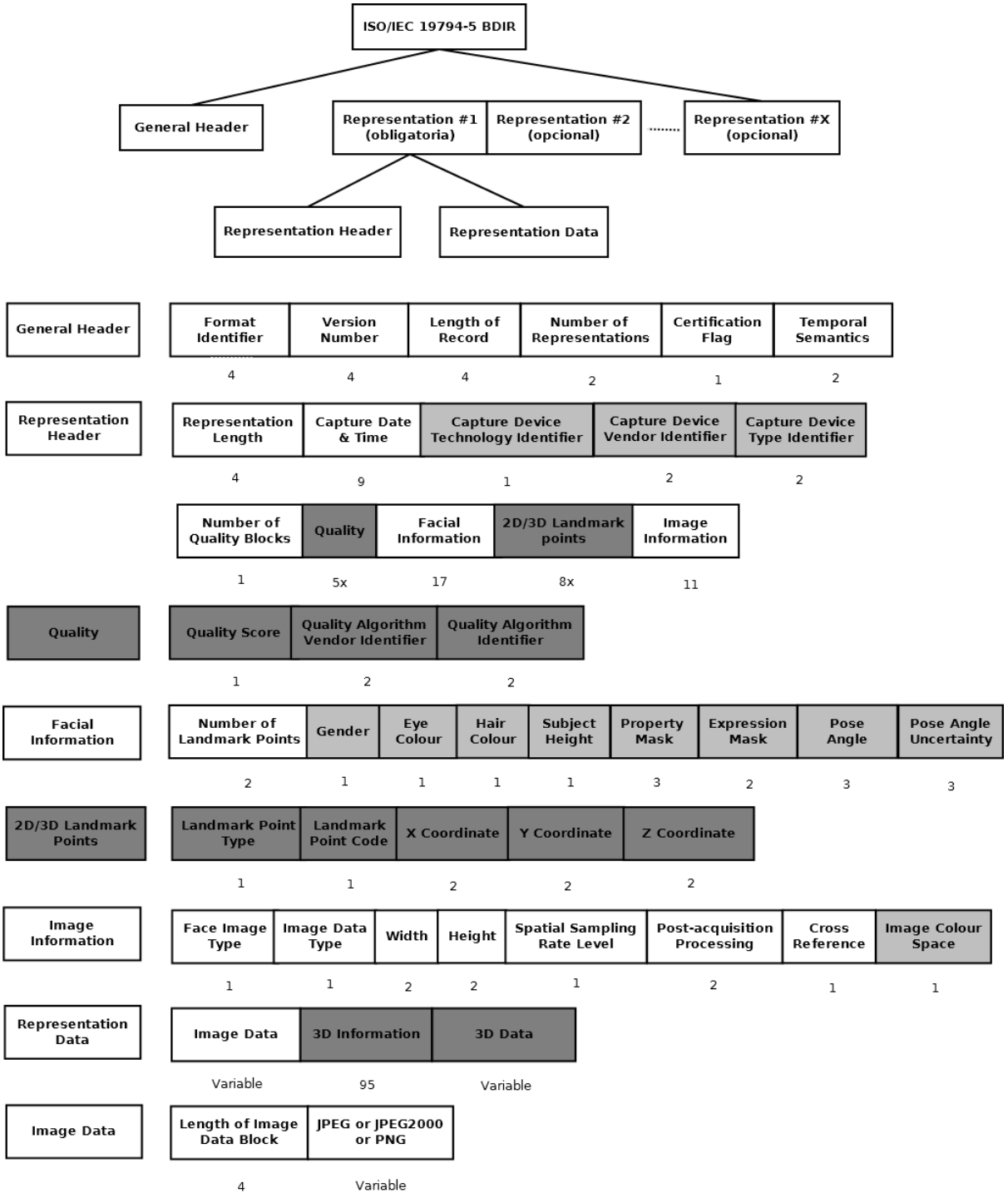

Fig. 22 – Esquema del BDIR según la norma ISO/IEC 19794-5

En la figura, los campos representados en blanco son los obligatorios, los gris claro son campos que deben estar aunque su valor puede aparecer como no especificado, y los gris oscuro son opcionales. Además se muestra debajo de cada bloque el tamaño en bytes de cada uno.

Para la aplicación, se emplearon únicamente los campos obligatorios, dejando los desconocidos con el valor de no especificado. Los campos se rellenaron con los valores adecuados para el tipo de imagen a guardar, siguiendo el procedimiento descrito en el estándar.

Para la creación de la librería, se creó un nuevo proyecto y se incluyeron dos clases, *IsoImageBdir.java* e *ImageRepresentation.java*. La última genera los campos de *Representation#1*, incluyendo el *Representation Header* y el *Representation Data*, mientras que la primera genera el *General Header* y llama a la clase *ImageRepresentation* para unir los dos bloques y crear el BDIR resultante.

Por cada clase se generaron variables representando cada campo, con el tamaño y formato especificado. Se crearon además métodos para la obtención de cada uno de estos valores, así como para rellenarlos. También se implementaron métodos para convertir estos valores de su formato en número entero o en forma de cadena de caracteres, a bytes en binario y viceversa. Los valores de las variables en algunos campos, como son el tipo de imagen o el número de representaciones, en los cuales se mantiene fijo, se añadieron manualmente, mientras que para otros, como es el caso del campo con la fecha o el tamaño del bloque, se generaron unos métodos específicos para calcularlos y rellenarlos. Los valores para el tamaño de las imágenes se obtienen en la llamada a la clase a partir de sus parámetros, por lo que sus campos correspondientes se rellenan dinámicamente.

Una vez implementada la norma, se incorporó al proyecto como librería de la aplicación y se introdujo su uso en las funciones de procesado de BioAPI para la generación de los BIR, rellenando a su vez los campos de la cabecera (SBH). También se emplearon sus funciones de extracción de la imagen para el proceso de verificación, en el cual la cara procesada y guardada como BIR debe obtenerse, para guardarla como *Bitmap* en un directorio temporal y emplearla en el reconocimiento.

6 Pruebas y Resultados

A lo largo del desarrollo de la aplicación se realizaron diversas pruebas para comprobar el correcto funcionamiento de la misma en cuanto a su diseño en Android, así como la correcta encapsulación en BioAPI. También se hicieron pequeñas pruebas del proceso de reconocimiento y detección. Sin embargo esto último fue evaluado de forma más exhaustiva al finalizar el trabajo, realizando una serie de pruebas comentadas a continuación.

6.1 Introducción

En este primer apartado se explican los tres algoritmos disponibles para el proceso de reconocimiento, comentando brevemente su funcionamiento y aplicación, e incluyendo una evaluación de los mismos llevada a cabo por OpenCV. Posteriormente se muestran las pruebas realizadas en la aplicación desarrollada con el fin de evaluar su rendimiento en la misma y compararlo, y se presentan los resultados obtenidos, comentándolos y explicando las limitaciones encontradas.

Las librerías de OpenCV cuentan con tres algoritmos diferentes para llevar a cabo los procesos de reconocimiento facial. A continuación se explica brevemente el funcionamiento de cada uno de ellos:

1. Eigenfaces: Este método se basa en el uso del Análisis de Componentes Principales (PCA) para reducir la dimensionalidad de los datos, empleando únicamente los considerados más relevantes o significativos de la imagen. Para ello, selecciona las direcciones que representan una mayor varianza (las componentes principales) y de esta manera reduce el set de datos correlacionados a uno más pequeño formado por variables incorrelacionadas. Para llevar a cabo el reconocimiento facial, proyecta las imágenes de entrenamiento en el subespacio PCA y después la imagen a identificar, la cual se compara con las anteriores para identificar el vecino más cercano ("nearest neighbor"), es decir, el más parecido. Por cada individuo se elabora una cara media a partir de sus imágenes, llamada *eigenface*.

   A continuación en la Figura 23[37] se muestra una imagen con las *eigenfaces* obtenidas de varios sujetos, originalmente a escala de grises pero pasada a colores con el fin de facilitar la interpretación y visualización de las diferencias en la distribución y en las variaciones:

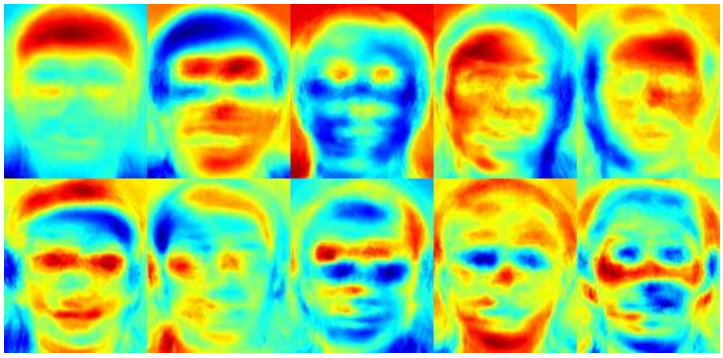

Fig. 23 – Eigenfaces de 10 sujetos diferentes [37]

2. Fisherfaces: El método del PCA encuentra una combinación lineal de variables que maximiza la varianza total en los datos, pero no considera clases, lo aplica sobre todos los datos sin clasificarlos, por lo que información discriminante puede perderse en el proceso. Esto puede provocar que, si la variación proviene de un factor externo, los resultados se alteren y no sean representativos, y las proyecciones se mezclen, dificultando la correcta identificación. El Análisis Discriminante Lineal (LDA), aplicado en Fisherfaces, sí que tiene en cuenta las clases y busca diferenciar bien unas de otras juntando las proyecciones pertenecientes a la misma y separándolas entre ellas todo lo posible. El algoritmo por tanto encuentra características faciales que sirven para discriminar entre individuos.

3. Local Binary Patterns Histograms (LBPH): Los algoritmos descritos anteriormente funcionan bien cuando se cuenta con varias imágenes de entrenamiento, y cuando factores como la posición o rotación de la cara no varía demasiado entre muestras. Sin embargo, hay casos en los que estas condiciones no se cumplen y por ejemplo se dispone de una única imagen. El LBPH, aunque suele tener peor rendimiento que los anteriores en sets con al menos 6 imágenes de entrenamiento, es más recomendable para estos casos. Su procedimiento se basa en definir dentro de una imagen su representación local mediante comparación entre píxeles vecinos. Se comparan las intensidades en un bloque de por ejemplo 8 píxeles, tomando como referencia el central y asignando valores binarios (1 o 0) según si sobrepasan o no el umbral. De esta manera surgen 2^8 posibles combinaciones, llamadas *Local Binary Patterns*. El procedimiento se explica en la Figura 24[37] empleando un bloque de 3x3:

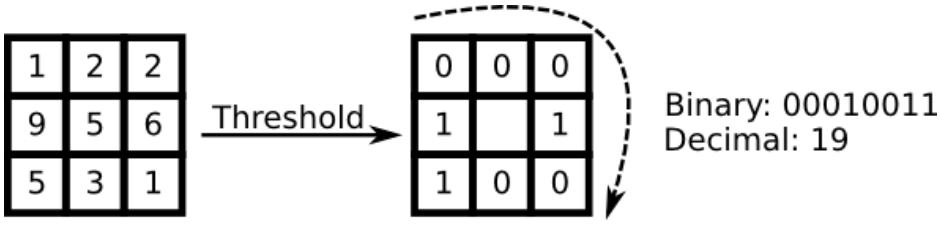

Fig. 24 – Procedimiento del LBPH en un bloque de 3x3 [37]

El resultado tras aplicar este método en imágenes con diferentes condiciones de iluminación se muestra en la Figura 25[37]:

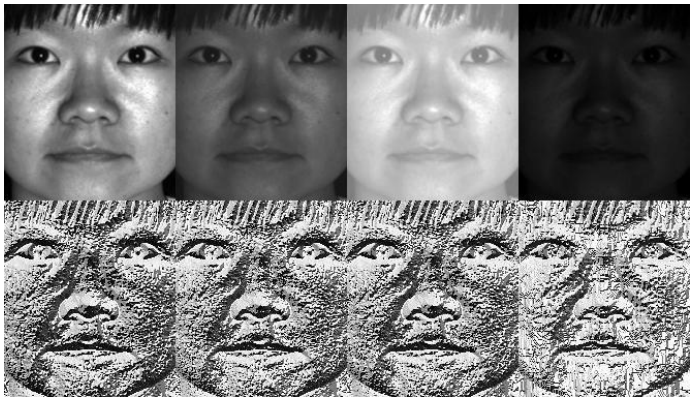

Fig. 25 – LBPH sobre cuatro imágenes con diferente iluminación [37]

A partir de su página web, en el apartado dedicado al reconocimiento facial, OpenCV muestra un estudio realizado con el fin de evaluar el rendimiento de estos algoritmos mencionados. Para su evaluación emplea tres bases de datos diferentes con imágenes de rostros. Estas son:

- AT&T Facedatabase: También llamada ORL Database of Faces, está formada por 400 imágenes pertenecientes a 40 sujetos diferentes. Cada uno cuenta con 10 imágenes en las cuales las condiciones de iluminación, las expresiones, el tiempo y ciertos detalles como si llevan o no gafas son los causantes de las variaciones entre las mismas. Todas las imágenes fueron tomadas sobre un fondo negro homogéneo, y de frente, pudiendo variar ligeramente la posición. Esta base de datos es muy sencilla, por lo que es adecuada para pruebas iniciales pero no sirve para evaluar las diferencias de rendimiento entre algoritmos.
- Yale Facedatabase A: Se conoce también como Yalefaces, y su set de imágenes es más elaborado por lo que es más adecuado que el anterior para llevar a cabo experimentos. Contiene imágenes pertenecientes a 15 sujetos, 14 hombres y una mujer, con diferentes condiciones de iluminación (variando la dirección), diferentes expresiones faciales, y con o sin gafas. Cada individuo aporta 11 imágenes en escala de grises de tamaño 320x243 píxeles.
- Extended Yale Facedatabase B: Contiene 2414 imágenes que pertenecen a 38 personas diferentes. Las imágenes únicamente cuentan con diferencias muy acusadas en cuanto a la iluminación. Debido al amplio set de imágenes que proporciona, es adecuado si se quiere llevar a cabo una comparación entre los rendimientos de los algoritmos en función de estos cambios.

Empleando la primera base de datos AT&T Facedatabase, llevan a cabo una comparación entre los algoritmos Eigenfaces y Fisherfaces, aplicando el método de validación cruzada de 10 iteraciones, y evaluando su evolución en función del número de imágenes empleadas. Los resultados obtenidos se muestran en la Figura 26[37]:

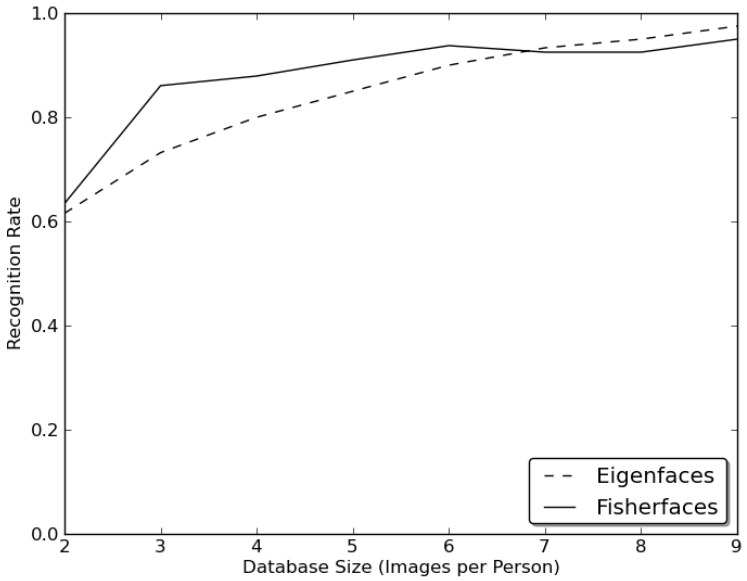

Fig. 26 – Comparación de rendimiento entre Eigenfaces y Fisherfaces [37]

Como podemos observar, el rendimiento de los algoritmos aumenta a medida que crece el set de entrenamiento seleccionado para cada individuo, estabilizándose a partir de 6 o 7 imágenes con un rendimiento muy alto de cerca del 100%. Para un número de imágenes inferior a 7 parece ser más eficiente el Fisherfaces.

6.2 Pruebas

Con el fin de evaluar el rendimiento de todo el proceso de reconocimiento realizado por la aplicación, se generó un set de imágenes y se guardó en un directorio específico para así poder seleccionarlo a través de la aplicación y realizar las búsquedas en el mismo. Los resultados podían consultarse a través de la opción "Ver detalles" o desde el *log*.

Para poder llevar a cabo una evaluación representativa, el set de imágenes seleccionado no consistía únicamente en imágenes de caras con ligeros cambios de expresión e iluminación (como en las bases de datos comentadas anteriormente), sino que se utilizaron imágenes reales de teléfonos, en las cuales pueden aparecer varias personas o ninguna, cambiando mucho de expresión, con caras ladeadas, separadas en el tiempo, y con condiciones de iluminación muy diferentes.

Un factor muy importante a tener en cuenta es que, al realizar una búsqueda con la aplicación, es muy probable el caso de que no todas las personas encontradas en el teléfono estén añadidas como contactos, por lo que es necesario establecer un valor de confianza en la predicción con el cual se decida si el parecido encontrado es o no suficiente como para devolver una identificación. El no incluirlo supondría una importación de todas estas imágenes identificadas erróneamente a los álbumes. Este caso no se contempló en las evaluaciones realizadas por OpenCV, ya que trabajaban con un set de imágenes de personas conocidas y a todas se les asignaba un identificador. Por ello en la evaluación simplemente se

comprobaba si la identificación había sido o no correcta. En nuestro caso, para evaluar el rendimiento se tuvieron en cuenta todas las posibilidades y se utilizó F1 Score como medida de los resultados, calculando por cada prueba la precisión y la exhaustividad (o "Recall"), aplicando el mismo peso a ambos valores:

$$Precisión = \frac{Verdadero\ Positivo}{\#\ Predicciones\ Positivas} = \frac{Verdadero\ Positivo}{Verdadero\ Positivo + Falso\ Positivo}$$

$$Exhaustividad = \frac{Verdadero\ Positivo}{\#\ Positivos\ Reales} = \frac{Verdadero\ Positivo}{Verdadero\ Positivo + Falso\ Negativo}$$

$$F1\ Score = 2 \times \frac{Precisión\ \times Exhaustividad}{Precisión + Exhaustividad}$$

Cada valor empleado en el cálculo representa la ocurrencia de un caso:

- Verdadero Positivo: Son las imágenes de las cuales se obtiene una identificación correcta.
- Falso Positivo: Es el caso en el cual a una imagen se le asigna un identificador erróneo, perteneciente a otro contacto. Incluye los casos en los que el individuo no pertenece a la lista de contactos y sin embargo en el proceso se obtiene una identificación.
- Falso Negativo: Incluye todos aquellos casos en los que la imagen no se identifica como uno de los contactos añadidos y sin embargo si pertenece a uno de ellos.
- Verdadero Negativo: Se aplica para las imágenes en las cuales no aparece ningún contacto conocido y el proceso de identificación no devuelve ningún identificador, lo cual es correcto.

Para cada prueba realizada se extrajeron los valores de los diferentes casos y se calculó el resultado final.

Es importante también tener en cuenta que en el proceso no se evalúa únicamente el rendimiento de los algoritmos de reconocimiento disponibles, sino también el proceso de detección de rostros, ya que aunque en general aporta resultados muy positivos, hay ocasiones en las que las caras no son detectadas y por tanto no se analizan.

Todos estos factores comentados repercuten enormemente en el rendimiento, convirtiendo el proceso en algo mucho más complejo y por tanto con resultados bastante más bajos que los obtenidos en los estudios realizados por OpenCV.

6.3 Resultados

Las pruebas se realizaron inicialmente para los tres algoritmos disponibles con el fin de elegir el más adecuado, descartando finalmente para la comparación los resultados obtenidos por el Eigenfaces al ser muy deficientes. El valor de confianza se estableció tras comprobar los resultados obtenidos para cada caso, fijándolo de manera que no se importasen demasiadas imágenes erróneas, pero que tampoco se importasen únicamente las de confianza muy alta.

Únicamente se hicieron pruebas para 2, 4, 6, 8, 10 y 12 imágenes de entrenamiento por contacto, haciendo por cada valor 3 pruebas y aplicando una media para el cálculo final.

A continuación se muestra la evolución del F1 Score obtenido para los algoritmos LBPH y Fisherfaces en función del número de imágenes empleadas para el entrenamiento:

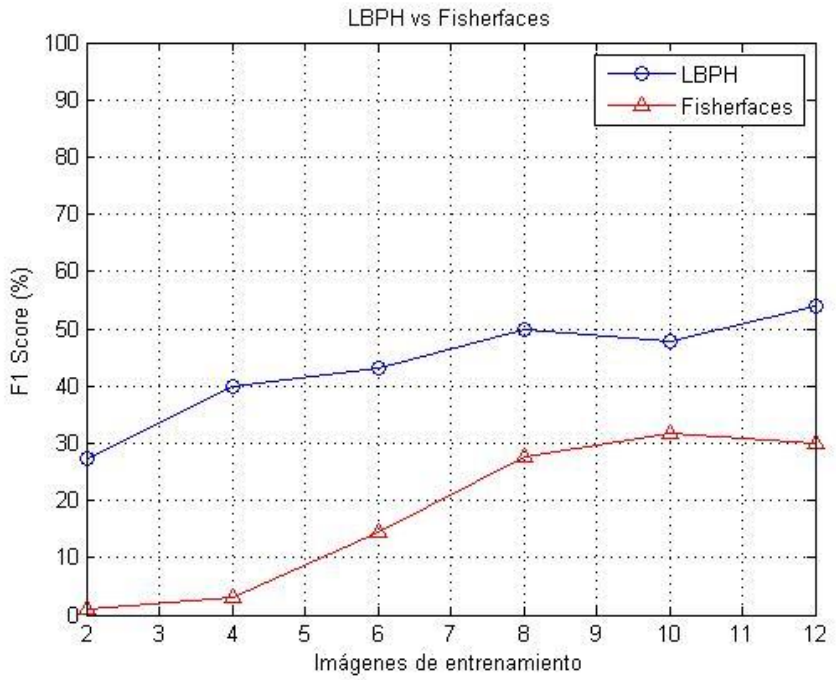

Fig. 27 – Evolución de los algoritmos en función de las imágenes de entrenamiento

Como se puede observar los valores obtenidos son muy bajos, aumentando a medida que se incrementa el número de imágenes de entrenamiento. Los resultados proporcionados por el LBPH son claramente superiores a los obtenidos con Fisherfaces, lo cual se debe probablemente a las grandes variaciones entre imágenes, haciendo más adecuado el primero. Por ello se ha elegido el mismo como algoritmo para la aplicación.

El siguiente gráfico y la tabla añadida a continuación representan los valores de Falso Positivo, Verdadero Positivo, Falso Negativo y Verdadero Positivo promediados de las pruebas realizadas con el LBPH. Debido a que cada vez se utilizan más imágenes de entrenamiento dentro del set, el número total de imágenes comparadas va reduciéndose:

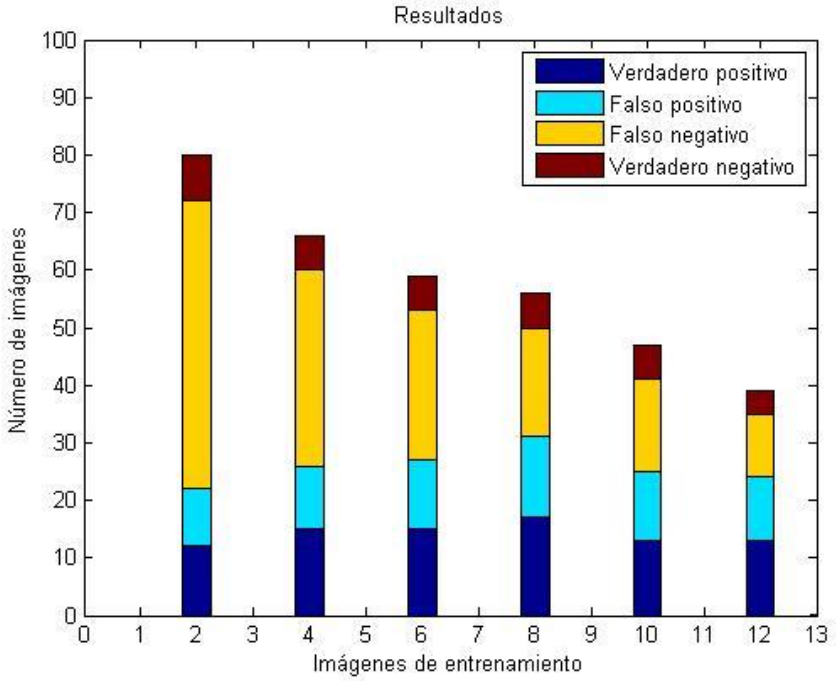

Fig. 28 – Valores promedio obtenidos en función del número de imágenes de entrenamiento

Tabla 2 – Valores promedio obtenidos para el LBPH

	2	4	6	8	10	12
VP	12	15	15	17	13	13
FP	10	11	12	14	14	11
FN	50	34	26	19	16	11
VN	8	6	6	6	7	4
Precisión	0.54	0.58	0.55	0.54	0.53	0.54
Exhaustividad	0.20	0.30	0.36	0.47	0.44	0.54
F1 Score (%)	**29.19**	**39.55**	**43.52**	**50.26**	**48.08**	**54.00**

Aunque en algunos campos los números varían poco, hay que tener en cuenta que la repercusión de los mismos a medida que aumenta el número de imágenes de entrenamiento es mayor, debido a que el número de imágenes de comparación disminuye. Por ejemplo, los VP oscilan entre 12 y 17, pero cada vez aciertan más imágenes en porcentaje, ya que en el primer caso aciertan 12 de aproximadamente 60 caras de contactos, y en el último 13 de 24. Además el número de FN sí va reduciéndose en gran medida. Estos valores los representa la medida de Exhaustividad, que como puede apreciarse va creciendo. El valor de la precisión sin embargo se mantiene.

En cuanto a la detección de rostros, de los 100 añadidos 88 se detectaron correctamente y los no reconocidos se debían en su mayoría a la posición de la cabeza (muy ladeada o inclinada).

6.4 Limitaciones

Como se ha podido observar, las principales limitaciones encontradas tienen su causa en el entorno de aplicación, ya que las imágenes con las que se trata no son adecuadas para llevar a cabo un buen proceso de reconocimiento facial. Esto se debe principalmente a las características muy variables en cuanto a iluminación y posición encontradas en las imágenes almacenadas en un teléfono. Además los resultados pueden variar en gran medida en función de las mismas, de manera que si en un teléfono las imágenes encontradas son más homogéneas, el funcionamiento puede ser más óptimo, mientras que en caso contrario puede ser muy deficiente.

7 Conclusiones y Líneas Futuras

7.1 Conclusiones

Con la realización de este trabajo se ha conseguido integrar en un entorno Android un conjunto de librerías de reconocimiento facial desarrolladas por OpenCV. Además se ha diseñado una aplicación que aprovecha estas librerías para llevar a cabo una clasificación casi automática de las imágenes de un teléfono en función de contactos añadidos. Todo este procedimiento se ha encapsulado bajo una adaptación a Java del estándar elaborado por el consorcio BioAPI, fomentando así la interoperabilidad.

Aunque los resultados obtenidos en las pruebas por parte de los algoritmos no son los esperados, el motivo se debe al set de imágenes empleado, por lo que las funciones de reconocimiento facial pueden ser aprovechadas en una aplicación con un propósito distinto y empleando imágenes más adecuadas para una correcta identificación. Todo el proceso de reclutamiento y la selección del directorio de búsqueda son manuales, por lo que las imágenes se pueden seleccionar de cualquier directorio existente o creado para este uso.

Las principales conflictos encontrados a lo largo del desarrollo del trabajo han surgido a la hora de incorporar las librerías, ya que estas están escritas en C++, un lenguaje diferente al empleado en entornos Android, por lo que era necesario realizar una adaptación. También se han encontrado obstáculos a la hora de adaptar el funcionamiento de la aplicación a la API de BioAPI. En cuanto al desarrollo de la aplicación en Android, han aparecido algunos impedimentos debido a las limitaciones del propio entorno, como el sistema de gestión de la memoria en el SO o la falta de estandarización de algunos procedimientos, lo cual impide la extensión del uso de la aplicación a todos los dispositivos. Sin embargo todos estos obstáculos han sido superados finalmente, tomando decisiones y encontrando una solución adecuada para cada uno de ellos.

La realización de este trabajo, aparte de facilitar la incorporación y el uso de estas librerías en otros proyectos similares para Android, ha supuesto un aprendizaje para el desarrollador, debido a que ha constituido la elaboración de un proyecto individual extenso, en el cual era imprescindible buscar multitud de información y tomar importantes decisiones en cuanto a la implementación y el diseño. Además ha permitido un mayor dominio del desarrollo de aplicaciones en Android, obteniendo un mayor conocimiento de las áreas y campos tratados, como la biometría o los sistemas de reconocimiento facial, y en especial de las librerías destinadas a este uso, desarrolladas por OpenCV.

7.2 Líneas Futuras

La aplicación desarrollada y el trabajo realizado en este TFG facilitan el proceso de incorporación de las librerías de OpenCV en entornos Android, pudiendo además emplearse la aplicación diseñada para el desarrollo de otras similares de reconocimiento facial en este sistema operativo. El procedimiento de reclutamiento a partir de la cámara o de imágenes del teléfono, el almacenamiento de la información empleando una base de datos y el proceso de selección del directorio de búsqueda pueden aplicarse a diferentes aplicaciones que hagan uso de estas librerías. Además puede emplearse cualquiera de los tres algoritmos disponibles y varias sus parámetros en función de los resultados deseados.

Una posible aplicación sería el desarrollo de un sistema de identificación de criminales reincidentes empleando una base de datos de imágenes almacenada en el teléfono. A partir de una fotografía realizada podría llevarse a cabo una identificación inmediata del mismo, lo cual podría resultar útil para un policía situado lejos de la comisaria, evitando la necesidad de transportar un ordenador o dispositivo más pesado para ello.

También sería posible mejorar el rendimiento de la aplicación diseñada introduciendo más información al proceso de reconocimiento, como puede ser el color del cabello o la utilización de gafas. Esta información podría almacenarse empleando el sistema descrito por BioAPI, introduciendo los datos en los campos habilitados para ello (ya generados para su uso al desarrollar la librería ISO/IEC 19794-5 en la realización de este TFG). Además podría combinarse el proceso con otros algoritmos de reconocimiento diseñados por OpenCV, pudiendo aportar de esta manera más información relevante.

Bibliografía

[1] Wikipedia, «List of International Organizations for Standardization standards» 20 agosto 2014. [En línea]. [Consulta: 20 agosto 2014]. Disponible en: http://en.wikipedia.org/wiki/List_of_International_Organization_for_Standardization_standards

[2] L.C.S Jazmín López Sánchez, «Privacidad y Seguridad» Mayo-junio 2012. .*Seguridad, Cultura de prevención para TI* [Revista]. [Consulta: 20 agosto 2014]. Disponible en: http://revista.seguridad.unam.mx/sites/revista.seguridad.unam.mx/files/revistas/pdf/Num13_Web.pdf

[3] L.C.S Jazmín López Sánchez, «Gestión de Seguridad y Riesgos» julio-agosto 2012. .*Seguridad, Cultura de prevención para TI* [Revista]. [Consulta: 20 agosto 2014]. Disponible en: http://revista.seguridad.unam.mx/sites/revista.seguridad.unam.mx/files/revistas/pdf/SeguridadNum14.pdf

[4] Wikipedia, «Historia del teléfono móvil» 13 julio 2014. [En línea]. [Consulta: 18 julio 2014]. Disponible en: http://es.wikipedia.org/wiki/Historia_del_tel%C3%A9fono_m%C3%B3vil

[5] Informatica-Hoy, «La historia del Teléfono Celular» [Blog]. [Consulta: 18 julio 2014]. Disponible en: http://www.informatica-hoy.com.ar/telefonos-celulares/La-historia-del-Telefono-Celular.php

[6] Herranz, A., «Promesas (y futuras realidades) de 4G» julio 2007, *PCWorld* [Blog]. [Consulta: 19 julio 2014]. Disponible en: http://www.pcworld.es/archive/promesas-y-futuras-realidades-de-4g

[7] 4G Americas, «Comprendiendo las diferencias entre 1G, 2G, 3G y 4G» [Blog]. [Consulta: 19 julio 2014]. Disponible en: http://www.4gamericas.org/index.cfm?fuseaction=page&sectionid=406

[8] OBlog, «Los teléfonos móviles que cambiaron la historia». *Oblog* [Blog]. [Consulta: 19 julio 2014]. Disponible en: http://blog.ono.es/2013/08/los-telefonos-moviles-que-cambiaron-la-historia/

[9] Juanguis, «Breve historia de los Smartphones». *Punto Geek* [Blog]. [Consulta: 21 julio 2014]. Disponible en: http://www.puntogeek.com/2011/01/14/breve-historia-de-los-smartphones/

[10] Jonadep, «Smartphones». *Historia de la Informática* [Blog]. [Consulta: 21 julio 2014]. Disponible en: http://histinf.blogs.upv.es/2012/12/03/smartphones

[11] Clark, J. F., «History of Mobile Applications» [PDF]. [Consulta: 21 julio 2014]. Disponible en: http://www.uky.edu/~jclark/mas490apps/History%20of%20Mobile%20Apps.pdf

[12] Seguridad Apple, «Fue Noticia en Seguridad Apple: Del 30 Septiembre al 13 de Octubre» 13 octubre 2013. Seguridad Apple. [Blog] http://www.seguridadapple.com/2013/10/fue-noticia-en-seguridad-apple-del-30.html

[13] Wikipedia, «Biometría» 3 junio 2014 [En línea]. [Consulta: 22 julio 2014]. Disponible en: http://es.wikipedia.org/wiki/Biometr%C3%ADa

[14] Galvis, Carlos Mauricio, «Introducción a la Biometría». *Monografías.com* [En línea]. [Consulta: 23 julio 2014]. Disponible en: http://www.monografias.com/trabajos43/biometria/biometria.shtml

[15] Wikipedia, «Marcello Malpighi» 20 julio 2014 [En línea]. [Consulta: 23 julio 2014]. Disponible en: http://es.wikipedia.org/wiki/Marcello_Malpighi

[16] Wikipedia, «Alphonse Bertillon» 24 enero 2014 [En línea]. [Consulta: 23 julio 2014]. Disponible en: http://es.wikipedia.org/wiki/Alphonse_Bertillon

[17] Sosa, Carlos «Principio de Identidad – Criminalística Libre 3.5» 10 enero 2008 *Principio de Identidad* [Blog]. [Consulta: 23 julio 2014]. Disponible en: http://principiodeidentidad.blogspot.com.es/2008/01/biografa-de-juan-vucetich.html

[18] Guerrero, Diego. «Reconocimiento Facial. Pasado, presente y futuro» [Blog]. [Consulta: 2 agosto 2014]. Disponible en: http://www.diegoguerrero.info/tag/reconocimiento-facial/

[19] Mark Williams, «Better Face-Recognition Software» 30 mayo 2007, *MIT Technology Review* [Blog]. [Consulta: 2 agosto 2014]. Disponible en: http://www.technologyreview.com/news/407976/better-face-recognition-software/

[20] Binetskaya, Maya. «Reconocimiento facial en el ámbito forense» [PDF]. [Consulta: 2 agosto 2014]. Disponible en: http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20130916MayaBinetskaya.pdf

[21] T. Kanade. Ph.D. dissertation, Kyoto University, 1973. «Picture Processing System by Computer Complex and Recognition of Human Faces» [Actas de conferencia]. [Consulta: 3 agosto 2014]. Disponible en: http://www.ri.cmu.edu/publication_view.html?pub_id=2530

[22] Wikipedia, «Principal component analysis» 10 agosto 2014 [En línea]. [Consulta: 3 agosto 2014]. Disponible en: http://en.wikipedia.org/wiki/Principal_component_analysis

[23] Fernández, Valeria, «Biometría facial» 1 enero 2014. *dolthink* [Blog]. [Consulta: 2 agosto 2014]. Disponible en: http://www.dolthink.com/biometria-facial.html

[24] marineru, «0013-01-pila-software-android» 11 julio 2011. *El androide libre* [Documento de sitio web]. [Consulta: 20 agosto 2014]. Disponible en:

http://www.elandroidelibre.com/2011/07/android-y-linux-la-delgada-linea-verde.html/0013-01-pila-software-android

[25] Bedoya Franco, Luis Jose, «Mis primeros tres años con Android» 1 diciembre 2013. *El blog de luisjoseb* [Blog]. [Consulta: 20 agosto 2014]. Disponible en: http://luisjoseb.blogspot.com.es/2013/12/mis-primeros-tres-anos-con-android.html

[26] Jan Dawson, «Updated Android version charts» 5 marzo 2014. *Beyond Devices* [Blog]. [Consulta: 20 agosto 2014]. Disponible en: http://www.beyonddevic.es/2014/03/05/updated-android-version-charts/

[27] Wikipedia, «Android» agosto 2014 [En línea]. [Consulta: 20 agosto 2014]. Disponible en: http://es.wikipedia.org/wiki/Android

[28] Wikipedia, «OpenCV» agosto 2014 [En línea]. [Consulta: 19 agosto 2014]. Disponible en: http://en.wikipedia.org/wiki/OpenCV

[29] OpenCV DevZone, «OpenCV Wiki» [En línea]. [Consulta: 19 agosto 2014]. [Disponible en: http://code.opencv.org/projects/opencv/wiki

[30] Young, Matthew, Purdue University con SAFLINK Corporation, junio 2005. «History of the API and Relationship to Other Standards». *BioAPI Consortium* [En línea]. [Consulta: 18 agosto 2014]. Disponible en: http://www.bioapi.org/history.asp

[31] K. Modi, Shimon, Ph.D. y Watson, Keith, «BioAPI Standards: Java Specification and Reference Implementation – Impact on API Adoption» [PDF]. [Consulta: 18 agosto 2014]. Disponible en: http://biometrics.org/bc2009/presentations/tuesday/Modi%20MR14%20Tue%20%201120-1140.pdf

[32] M. Domínguez Dorado, «NetBeans IDE 4.1. La alternativa a Eclipse» de *Todo Programación Nº 13*, Madrid, Editorial Iberprensa, noviembre 2005, pp 32-34.

[33] Drndos, «How to run JavaCV (with sample face recognition) on Android ARM device – Netbeans and nbandroid» 11 abril 2013. *Drndos's Blog* [Blog]. [Consulta: 31 marzo 2014]. Disponible en: http://blog.drndos.sk/2013/04/how-to-run-javacv-with-sample-face-recognition-on-android-arm-device-netbeans-and-nbandroid/

[34] Nikhil9, «Code for Face Detection in Javacv using haar classifier» 16 noviembre 2012. *GitHub Gist* [En línea]. [Consulta: mayo 2014]. Disponible en: https://gist.github.com/nikhil9/4087860

[35] NBAndroid, «Installation» 28 mayo 2014. [En línea] [Consulta: 12 agosto 2014]. Disponible en: http://nbandroid.org/wiki/index.php/Installation

[36] OpenCV, «Face Recognition with OpenCV» 25 junio 2014. [En línea] [Consulta: 26 agosto 2014]. Disponible en: http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_tutorial.html

[37] Christian Bjelland, Petter, «Doing face recognition with JavaCV» 1 diciembre 2012. [En línea] [Consulta: 15 abril 2014]. Disponible en: http://pcbje.com/2012/12/doing-face-recognition-with-javacv/

Anexo A: Planificación y Presupuesto

En este apartado se indican las fases de desarrollo de este TFG y las horas dedicadas, desglosándolas según las tareas realizadas. También se muestra el cálculo aproximado de los costes, teniendo en cuenta para ello las horas de dedicación.

A.1 Planificación

El desarrollo de este TFG se puede dividir en 3 fases bien diferenciadas, las cuales se han empleado también para la estructuración de los apartados en este documento. Cada una de estas fases ha supuesto una cantidad determinada de horas, cuya estimación se muestra a continuación:

Fase 1: Integración de las librerías de OpenCV

I. Documentación previa (10 horas)
II. Preparación de las herramientas de trabajo (9 horas)
III. Búsqueda y realización de tutoriales (18 horas)
IV. Comprensión de las librerías (5 horas)
V. Integración en la aplicación (11 horas)
VI. Pruebas de funcionamiento (7 horas)

Fase 2: Desarrollo de la aplicación

I. Arquitectura y aspecto visual (68 horas)
II. Creación Base de Datos (7 horas)
III. Almacenamiento en memoria externa (10 horas)
IV. Adaptaciones para el uso de BioAPI (15 horas)

Fase 3: Encapsulación en BioAPI

I. Documentación previa (6 horas)
II. Instalación librerías (6 horas)
III. Comprensión librerías (7 horas)

IV. Desarrollo funciones (30 horas)

V. Creación librería según la norma ISO/IEC 19794-5 (11 horas)

Fase 4: Pruebas

I. Pruebas del rendimiento de los algoritmos (19 horas)

II. Adaptaciones según los resultados (1 hora)

Fase 5: Elaboración de la memoria

I. Redacción de la memoria (65 horas)

II. Corrección (15 horas)

A continuación se representan mediante un diagrama de Gantt las fases y su duración:

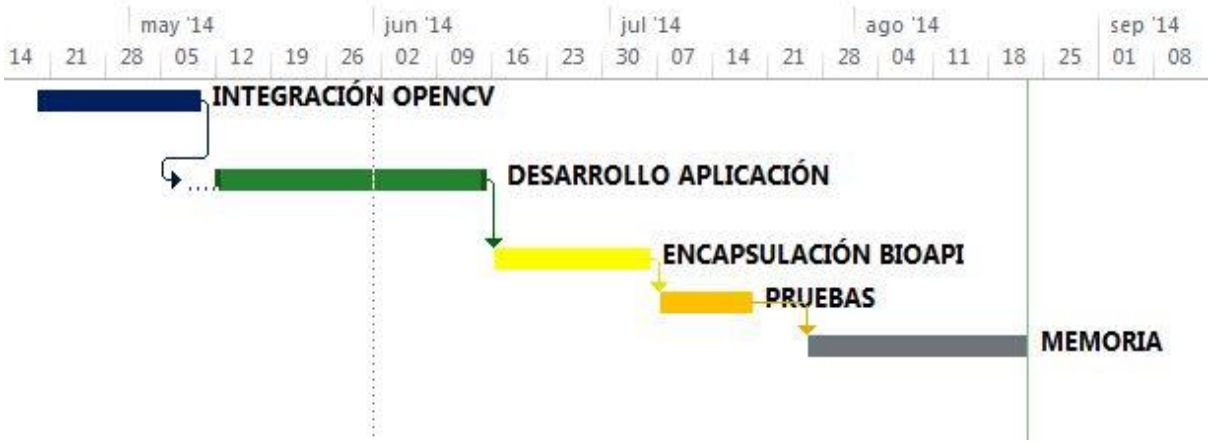

Fig. 29 – Diagrama de Gantt de las fases del TFG

Tabla 3 – Desglose de tareas

FASES	HORAS EMPLEADAS
Integración de las librerías de OpenCV	60
Desarrollo de la aplicación	100
Encapsulación en BioAPI	60
Pruebas	20
Elaboración de la memoria	80
TOTAL	**320**

A.2 Presupuesto del Trabajo Fin de Grado

A.2.1 Costes materiales

El coste destinado a los materiales empleados para la realización del proyecto se basa principalmente en el uso de un ordenador de prestaciones medio-altas, con el cual se ha desarrollado el código de la aplicación así como la memoria, y un dispositivo móvil con la versión de Android más actual (4.4 KitKat), concretamente un Moto G de reciente adquisición. Para ambos se ha calculado el coste resultante de los cuatro meses de realización del proyecto aplicando un plazo de amortización de tres años. Además se han incluido los costes derivados de la conexión a Internet. Debido a que todas las herramientas de software empleadas son de uso gratuito, y las librerías de código abierto, no se incluye el coste de ninguna de estas en el cálculo.

Tabla 4 – Costes Materiales

CONCEPTO	PRECIO	COSTE PROYECTO (€)
Ordenador Toshiba de prestaciones medio-altas	700 €	77,78
Teléfono móvil Moto G	200 €	22,22
Acceso a Internet	24 €/mes	96,00
TOTAL		**196,00**

A.2.2 Costes de personal

La realización de este trabajo ha sido llevada a cabo por un ingeniero, bajo la supervisión de un jefe de proyecto. Los costes calculados se muestran a continuación en la Tabla 5.

Tabla 5 – Costes de Personal

OCUPACIÓN	HORAS	PRECIO/HORA	IMPORTE (€)
Jefe de proyecto	25	90,00	2250,00
Ingeniero	295	40,00	11800,00
TOTAL	**320**	**-**	**14050,00**

A.2.3 Costes totales

Tabla 6 – Costes Totales

CONCEPTO	PRECIO (€)
Costes materiales	196,00
Costes de personal	14050,00
Costes indirectos (20%)	2849,20
Subtotal	17095,20
IVA (21%)	3589,99
TOTAL	**20685,19**

El coste total del proyecto es de VEINTE MIL SEISCIENTOS OCHENTA Y CINCO CON DIECINUEVE CÉNTIMOS.

Leganés, 1 de septiembre de 2014

El ingeniero