UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO FIN DE GRADO

**Grado en Ingeniería Electrónica Industrial y Automática**

# ANÁLISIS DE UN SISTEMA DE RECONOCIMIENTO FACIAL A PARTIR DE UNA BASE DE DATOS REALIZADO MEDIANTE PYTHON

## Memoria y Anexos


**Autor:** Daniel Costa Mari
**Director:** Pablo Buenestado Caballero

**Convocatoria:** Junio 2020

# Resum

Envoltats per una societat en continu canvi i amb un afany de desenvolupament tecnològic sense límits, la intel·ligència artificial es posiciona com una de les àrees que més repercussió està tenint actualment. Dins d'aquest context, es troba el reconeixement facial, una eina cada vegada més emprada en qualsevol àmbit o en qualsevol situació; des del desbloqueig de dispositius mòbils, passant per centenars d'aplicacions, fins al seu ús com a mesura de seguretat presa per alguns governs.

Seguint aquest context, aquest projecte es basa en l'anàlisi, tant a nivell teòric, com experimental, d'un sistema de reconeixement facial a partir d'una base de dades. La implementació del codi s'ha realitzat en el llenguatge de programació Python 3.6 juntament a la llibreria de machine learning i visió per computador OpenCV.

El procés d'identificació d'una cara es divideix en dos subprocessos consecutius. Pel que fa al primer subprocés, és a dir, a la detecció facial, el mètode emprat es centra en l'algoritme de Viola-Jones basat en Haar Cascades. Pel que fa al segon subprocés, el reconeixement facial, els mètodes utilitzats en aquest projecte són Fisherfaces, Eigenfaces i LBPH, dels quals s'ha implementat el seu codi i s'ha realitzat un estudi detallat.

D'aquesta manera, s'ha realitzat una comparació dels tres mètodes de reconeixement facial per determinar el seu comportament sota diferents condicions, estudiant la seva ràtio d'èxit, així com altres característiques o propietats.

# Resumen

Rodeados por una sociedad en continuo cambio y con un afán de desarrollo tecnológico sin límites, la inteligencia artificial se posiciona como una de las áreas que mayor repercusión está teniendo actualmente. Dentro de este contexto, se encuentra el reconocimiento facial, una herramienta cada vez más usada en cualquier ámbito o en cualquier situación; desde el desbloqueo de dispositivos móviles, pasando por cientos de aplicaciones, hasta su uso como medida de seguridad tomada por algunos gobiernos.

Siguiendo dicho contexto, este proyecto se basa en el análisis, tanto a nivel teórico como experimental, de un sistema de reconocimiento facial a partir de una base de datos. La implementación del código se ha realizado en el lenguaje de programación Python 3.6 junto a la librería de machine learning y visión por computador OpenCV.

El proceso de identificación de una cara se divide en dos subprocesos consecutivos. Por lo que respecta al primer subproceso, es decir, a la detección facial, el método empleado se centra en el algoritmo de Viola-Jones basado en Haar Cascades. En cuanto al segundo subproceso, el reconocimiento facial, los métodos utilizados en este proyecto son Fisherfaces, Eigenfaces y LBPH (Local Binary Pattern Histogram), de los cuales se ha implementado su código y se ha realizado un estudio detallado.

De esta manera, se ha realizado una comparación de los tres métodos de reconocimiento facial para determinar su comportamiento bajo diferentes condiciones, estudiando su ratio de éxito, así como otras características o propiedades.

# Abstract

Surrounded by a society that is constantly changing and which is filled with the urge to achieve an unlimited technological development, artificial intelligence has positioned itself as one of the areas with the biggest influence nowadays. Inside this context emerges facial recognition, a tool used in a lot of areas and situations, from unlocking mobile devices, going through many applications, up to its use as a security measure in some governments' correct functioning.

Following said context, this work is based on the analysis, both theoretical and practical, of a facial recognition system built from a database. The implementation of the code has been done in Python 3.6 programming language altogether with the machine learning library and vision for computer OpenCV.

The facial identification process is divided into two consecutive subprocesses. For the first subprocess, which is facial detection, the method used focuses on the Viola-Jones algorithm based in Haar Cascades. Regarding the second subprocess, facial recognition, the methods used in this project are Fisherfaces, Eigenfaces and LBPH, from which their code has been implemented and a detailed study has been carried out.

In this way, a comparison of the three methods for facial recognition has been done in order to determine their behaviour under different conditions and studying their success ratio among other features and characteristics.

# Índice General

# Índice de Figuras

# Índice de Tablas

# Glosario de abreviaturas

**LBPH:** Local Binary Pattern Histogram

**SVM:** Support Vector Machine

**HOG:** Histograma de Gradientes Orientados

**FLD:** Discriminante Lineal de Fisher

**LDA:** Análisis Discriminante Lineal

**PCA:** Análisis de Componentes Principales

**ROI**: Region of Interest

**FRGC**: Face Recognition Grand Challenge

**ISO**: International Organization for Standardization

**IEC**: International Electrotechnical Commission

**IEEE**: Institute of Electrical and Electronics Engineers

**FRVT**: Face Recognition Vendor Tests

**EAR**: Eye Aspect Ratio

**pip**: Package Installer for Python

**OpenCV**: Open Source Computer Vision Library

**LOPDGDD**: Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales

**RGPD**: Reglamento General sobre Protección de Datos

# 1. Prefacio

## 1.1. Motivación

La motivación por la realización de este trabajo viene dada por el auge constante y progresivo de la inteligencia artificial, más concretamente, del machine learning. Actualmente, el machine learning se utiliza en cientos de aplicaciones; diagnóstico de enfermedades, industria 4.0, análisis y clasificación de datos, motores de búsqueda, anuncios en redes sociales, reconocimiento facial, reconocimiento de objetos, conducción automática, análisis de consumo y productividad, y un largo etcétera. Ahora mismo, casi cualquier cosa que nos podamos imaginar, tiene relación, de forma directa o indirecta, con la inteligencia artificial.

Otro detonante en la decisión de realizar este proyecto se basa en las ganas de aprender con un poco más de profundidad algún lenguaje de programación, que, en este caso, se trata de Python. La elección de este lenguaje de programación no es casual, sino que viene dada por dos motivos. El primero de ellos, es que en la universidad se dieron las primeras pinceladas en el aprendizaje de este lenguaje, por lo que ya se tenía un conocimiento, aunque escaso, de su uso. El segundo de los motivos es que Python se encuentra entre uno de los lenguajes de programación más utilizados a nivel global, además de ser uno de los que más demanda por parte de los empleadores presenta y que más potencial de crecimiento está experimentando.

El *índice TIOBE* (*index | TIOBE - The Software Quality Company*, n.d.), uno de los más reconocidos actualmente, se trata de un informe mensual realizado y publicado por parte de la empresa *TIOBE Software BV* en el que se detalla la popularidad de los lenguajes de programación existentes. Según el Índice TIOBE para Marzo de 2020, Python ocupa el tercer lugar, con un 10,11% obtenido en la clasificación, tal como se puede observar en la Figura 1.

| Mar 2020 | Mar 2019 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Java | 17.78% | +2.90% |
| 2 | 2 | | C | 16.33% | +3.03% |
| 3 | 3 | | Python | 10.11% | +1.85% |
| 4 | 4 | | C++ | 6.79% | -1.34% |
| 5 | 6 | ^ | C# | 5.32% | +2.05% |
| 6 | 5 | v | Visual Basic .NET | 5.26% | -1.17% |
| 7 | 7 | | JavaScript | 2.05% | -0.38% |
| 8 | 8 | | PHP | 2.02% | -0.40% |
| 9 | 9 | | SQL | 1.83% | -0.09% |
| 10 | 18 | ^^ | Go | 1.28% | +0.26% |

*Figura 1.- Índice TIOBE para Marzo de 2020 (10 primeros clasificados). Fuente: TIOBE*

Es por esta serie de motivos por los que se ha decidido realizar este Trabajo de Fin de Grado sobre machine learning, concretamente de reconocimiento facial, empleando el lenguaje de programación Python.

## 1.2. Requisitos previos

Entre los requisitos previos, principalmente se encuentra la capacidad de aprendizaje y de resolución de problemas de forma autónoma.

A parte, cabe destacar el conocimiento adquirido en las asignaturas de *Robótica Industrial y Visión por Computador*, de *Informática* y de *Estadística* del Grado en Ingeniería Electrónica Industrial y Automática. Gracias a ellas, se han adquirido tanto competencias específicas como transversales que han servido de apoyo y ayuda en la realización del presente proyecto.

# 2. Introducción

## 2.1. Objetivos del trabajo

El objetivo principal del presente proyecto se basa en realizar un análisis y posterior comparación de tres de los métodos principalmente utilizados en los sistemas de reconocimiento facial. De esta manera, se pretende realizar, en primer lugar, un estudio a nivel matemático de los principios que rigen cada uno de los tres métodos. En segundo lugar, con los métodos implementados en Python, se realizaran una serie de test para poder llevar a cabo una comparación de los resultados experimentales.

## 2.2. Alcance del trabajo

El alcance del proyecto se compone de los siguientes puntos:

- Realizar la instalación de todo el software y crear un entorno virtual de trabajo.
- Conseguir una base de datos suficientemente extensa y apropiada.
- Implementar el código en Python para detectar caras en una imagen.
- Implementar el código en Python para el reconocimiento facial con cada uno de los tres métodos diferentes.
- Realizar los test correspondientes para obtener los resultados experimentales y así poder compararlos posteriormente para sacar las conclusiones pertinentes.
- Realizar un análisis a nivel matemático de los principios estadísticos y algebraicos que rigen cada uno de los tres métodos utilizados.
- Demostrar y verificar experimentalmente los fenómenos de overfitting y underfitting.

## 2.3. Estructura de la memoria

La estructura de la memoria del TFG es la siguiente:

- **Prefacio:** Se explican brevemente los antecedentes del proyecto.
- **Introducción:** Se pone de manifiesto cuáles serán los objetivos y el alcance del trabajo.
- **Conceptos teóricos:** Se pretende definir de forma conceptual todo lo que involucra a la temática del presente proyecto. Es decir, los conceptos de biometría, inteligencia artificial y reconocimiento facial, además de un breve repaso a su estado del arte.

- **Análisis de los métodos de reconocimiento facial:** Se realiza un análisis a nivel matemático tanto del algoritmo usado para la detección facial, así como de los principios estadísticos y algebraicos que rigen cada uno de los tres métodos utilizados para el reconocimiento facial.
- **Desarrollo del proyecto:** Se detalla cómo ha sido el desarrollo de la parte experimental del proyecto además de mostrar y analizar los resultados obtenidos.
- **Pliego de condiciones:** Se establecen las condiciones generales (estándares y legislación) y técnicas que involucran a la temática del trabajo.
- **Impacto medioambiental:** Se explica cuál ha sido el impacto medioambiental en la realización del TFG.
- **Análisis económico:** Se ofrece un presupuesto aproximado del coste del desarrollo del proyecto.
- **Conclusiones:** Se evalúa el resultado final del proyecto teniendo en cuenta los objetivos iniciales.
- **Líneas futuras de trabajo:** Se describen una serie de posibles mejoras.
- **Bibliografía:** Incluye todo el material bibliográfico consultado y citado.
- **Anexos:** En el anexo A1 se describe el entorno de trabajo en PyCharm. En el resto de anexos se adjunta el código de cada método.

# 3. Conceptos teóricos

A continuación, se explican de forma teórica los conceptos que están directamente relacionados con la temática del presente proyecto.

## 3.1. Biometría

La biometría es la aplicación de técnicas estadísticas y matemáticas sobre las características físicas estáticas de los individuos para determinar o verificar su autenticidad.

Cuando se habla de una persona es importante distinguir entre características físicas estáticas o dinámicas. Estas últimas incluyen factores de comportamiento, como bien pueden ser una firma, la forma de teclear o la forma de caminar. Por su propia naturaleza, son fáciles de suplantar, ya que no se tratan de características intrínsecas al cuerpo humano, propias de su fisiología. Es por este motivo que cuando se habla de biometría se utilizan las características físicas estáticas, aquellas únicas, inamovibles y propias de cada ser vivo.

En función de la característica que se evalúa, se pueden distinguir diferentes métodos o procesos de autentificación biométricos:

- Reconocimiento de iris
- Reconocimiento facial
- Reconocimiento vascular (geometría de las venas del dedo)
- Reconocimiento por huella dactilar
- Reconocimiento de retina
- Reconocimiento de la geometría de la mano

Cabe destacar, pero, que no todos los métodos presentan la misma fiabilidad, estabilidad, ni aceptación, siendo los mejores el reconocimiento por huellas dactilares y el reconocimiento vascular.

Dicho esto, es importante marcar que todo sistema de reconocimiento biométrico debe presentar, al menos, cuatro propiedades referidas a la característica física propia de cada método:

- **Mensurable:** Se puede medir.
- **Única:** Debe ser una característica única en cada persona.
- **Universal:** Todos los individuos deben tenerla.
- **Estática:** Dicha característica no experimenta cambios notables a lo largo del tiempo.

También existen otras propiedades que no son comunes a todos los métodos, como por ejemplo puede ser el contacto físico para medir la característica; en el caso del reconocimiento facial o el reconocimiento a través del iris no hay contacto, mientras que en el reconocimiento vascular o en el reconocimiento por huella dactilar sí existe contacto.

Por lo que respecta a su uso, actualmente la biometría es ya utilizada a nivel global y presenta una gran cantidad de beneficios, como pueden ser: permite realizar trámites de forma remota, es fácil de usar y no precisa de recordar claves o patrones, es accesible para todos, se puede combinar con otras tecnologías, permite un control de presencia, aumenta la seguridad del control de acceso a lugares críticos, etc.

## 3.2. Inteligencia Artificial

El concepto de inteligencia artificial fue introducido en 1956 por John McCarthy, un reconocido informático que destacó en el campo de las Ciencias de la Computación y que recibió el premio Turing en 1971. Él mismo definió la inteligencia artificial como "la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cómputo inteligentes".

Actualmente, la inteligencia artificial se encuentra en constante desarrollo y evolución, y cada vez se dedican más recursos a ello, debido a que hoy en día su uso tiene aplicación casi en cualquier campo. La Figura 2 es una muestra de ello; se trata de una infografía realizada por Statista (*Las aplicaciones más rentables de la inteligencia artificial | Statista*, n.d.) en la que se pueden observar las aplicaciones de la inteligencia artificial clasificadas por los ingresos generados.

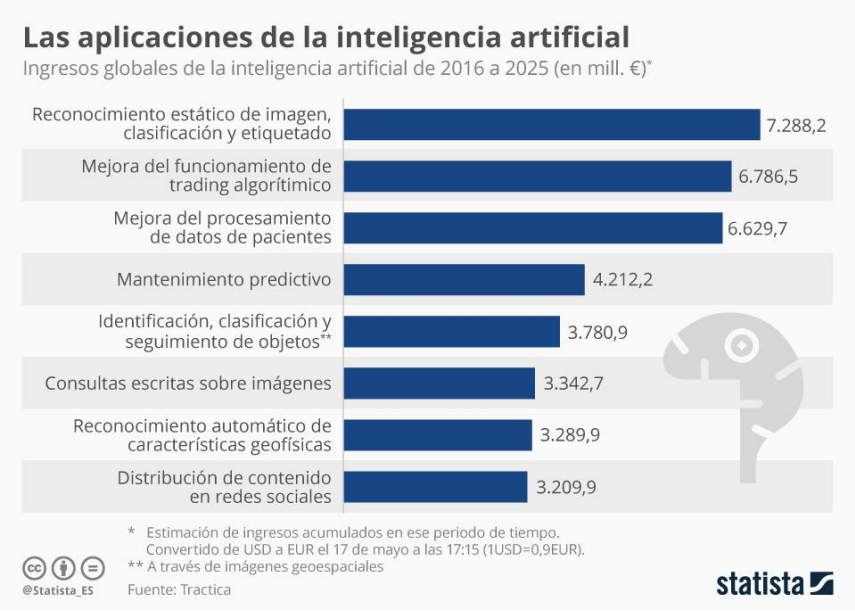


*Figura 2.- Aplicaciones de la inteligencia artificial en función de sus ingresos. Fuente: Statista.*

Dentro de la inteligencia artificial cabe destacar la existencia de varias ramas, como el machine learning, el deep learning, las redes neuronales y la visión por computador, entre muchas otras. A continuación se definirán estos conceptos, los cuales están estrechamente relacionados con la temática del proyecto y servirán para empezar a darle forma y a contextualizarlo.

### 3.2.1. Machine Learning

El machine learning o aprendizaje automático es una rama de la inteligencia artificial que se define como la capacidad que tiene una máquina o un software para aprender a partir de una base de datos, con el objetivo de tomar una decisión correcta sin la intervención de una persona. Este proceso se realiza principalmente a través de la programación de algoritmos y funciones matemáticas, aunque también toma una gran importancia el uso de la estadística.

Existen tres tipos de machine learning: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

#### 3.2.1.1. Aprendizaje supervisado

Este tipo de aprendizaje se basa en proporcionar a la máquina una base de datos con entradas y sus salidas correspondientes, o dicho de otra forma, se proporciona la relación entre las entradas y las salidas para que aprenda de ella.

Si se quiere entrenar al algoritmo para que reconozca, por ejemplo, a animales, entonces se proporciona una base de datos con imágenes de cada animal, y cada imagen estará etiquetada con el nombre de su respectivo animal. De esta manera, el objetivo será que la máquina sea capaz de determinar qué animal es a partir de su imagen.

#### 3.2.1.2. Aprendizaje no supervisado

En este caso todo el proceso se realiza sobre un conjunto de datos formado solamente por entradas al sistema. No se tiene información sobre las etiquetas de esos datos. El sistema se basa en el reconocimiento de patrones para poder agrupar los datos en función de sus características y así poder clasificarlos.

#### 3.2.1.3. Aprendizaje por refuerzo

Este tipo de machine learning se basa en el aprendizaje a partir del método prueba-error. El algoritmo se realimenta constantemente de los resultados obtenidos en las acciones que realiza, y trata de optimizar el resultado en cada iteración.

### Overfitting y Underfitting

Los conceptos de overfitting y underfitting hacen referencia a un fenómeno que ocurre cuando el software o la máquina "aprenden" o en exceso o demasiado poco. Estos fenómenos son unos de los mayores responsables al obtener malos resultados o bajas tasas de éxito en los test.

Más concretamente, el concepto de overfitting hace referencia al hecho de sobreentrenar un algoritmo, con el consiguiente de que queda ajustado con unas características demasiado específicas y dando como resultado una tasa de éxito menor a la esperada. En el lado opuesto se encuentra el concepto de underfitting, el cual se refiere al hecho de sub-entrenar un algoritmo, por lo que este no tiene suficiente información para realizar con éxito su predicción.

Generalmente, estos conceptos se pasan por alto, debido a que es natural pensar que cuanto más entrenado esté un algoritmo, más precisos serán sus resultados. En los artículos técnicos y de investigación sobre reconocimiento facial que se han utilizado como consulta se suele dar por hecho este suceso, que a mayor número de imágenes usadas para el training, mayor es la tasa de acierto, o ni siquiera se analiza este fenómeno y simplemente se dedican a exponer y explicar los resultados obtenidos. Con estos dos conceptos se pretende explicar por qué esto no siempre sucede así.

En el presente proyecto se intentará hacer énfasis en estos fenómenos, con el objetivo de comprobar si experimentalmente se verifican o no.

## 3.2.2. Deep Learning

El deep learning o aprendizaje profundo se constituye de estructuras lógicas que intentan asemejarse al sistema cognitivo de un ser humano, poseyendo múltiples capas o unidades de proceso, cada una de ellas formada por un algoritmo de machine learning.

Dichas capas se unen constituyendo una red neuronal artificial en la que todos sus nodos están interconectados y se comunican entre sí. De esta manera, los datos entran por la capa de entrada, se procesan en las capas escondidas o profundas y finalmente se obtiene la salida en la última capa, tal como se puede apreciar en el esquema de la Figura 3. En las capas escondidas se realiza un proceso de extracción de características mediante un método que pondera los datos que se transmiten entre cada neurona o nodo, de manera que se potencian o inhiben estos valores con el objetivo de minimizar una función de pérdida y de ajustarse al resultado que se pretende obtener. Entrenar una red neuronal se basa en ajustar estos pesos o ponderaciones de cada nodo para que el resultado se ajuste al máximo a los datos conocidos.

Actualmente, las mayores redes neuronales utilizadas en deep learning están constituidas por no más de 10 millones de neuronas. La aplicación de este tipo de inteligencia artificial en la vida real abarca

tareas complejas como pueden ser: aproximación de funciones y análisis de regresiones, reconocimiento de patrones, procesamiento de datos, robótica e ingeniería de control. El único inconveniente es que utilizar este tipo de IA requiere de grandes cantidades de recursos de almacenamiento y de procesamiento, limitados actualmente por hardware.

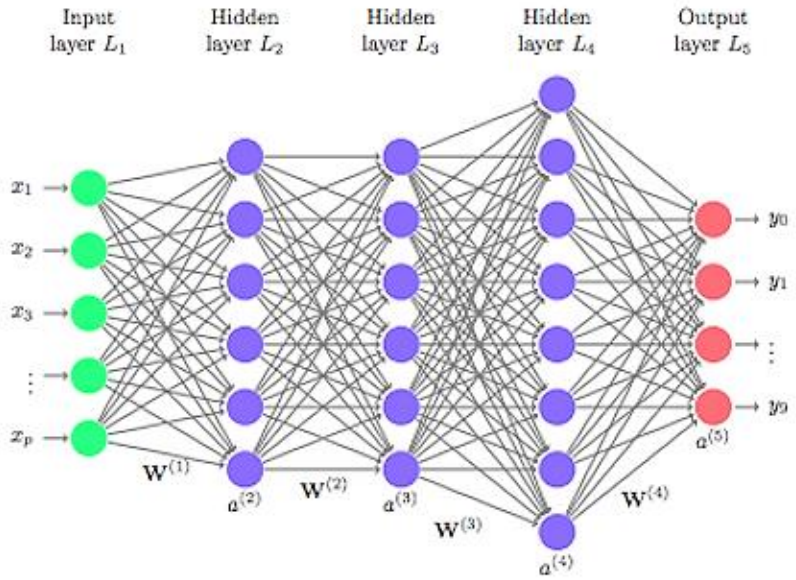


*Figura 3.- Red neuronal artificial. Fuente: UC Business Analytics R Programming Guide*

### 3.2.3. Visión por computador

La visión por computador o visión artificial es el conjunto de herramientas y procesos que se utilizan para adquirir, procesar, analizar y modificar imágenes del mundo real con el objetivo de poder ser tratadas por un ordenador.

Actualmente, una de las principales funciones para las que se utiliza la visión por computador es el reconocimiento y análisis de objetos en imágenes y videos, pero también tiene aplicación en campos como la industria, la medicina o la conducción autónoma.

## 3.3. Reconocimiento facial

Un sistema de reconocimiento facial permite obtener de forma automática la identidad de una persona a partir del análisis de sus características faciales extraídas de una imagen o fotograma digital, gracias a su comparación con una base de datos. Para llevar a cabo este proceso se utiliza la inteligencia artificial, y, en función de las técnicas empleadas, de los recursos disponibles y de los potenciales objetivos, el reconocimiento facial se puede implementar a través de diferentes ramas de la IA. A día

de hoy, el reconocimiento facial tiene múltiples aplicaciones, tal como se puede observar en la Tabla 1:

*Tabla 1.- Aplicaciones del reconocimiento facial*

| Áreas | Aplicaciones |
|---|---|
| **Seguridad** | Inicio de sesión, desbloqueo de dispositivos móviles y ordenadores, seguridad en aplicaciones, cifrado de información |
| **Cumplimiento de la ley y vigilancia** | Video vigilancia, control de circuitos cerrados, control de acceso a instalaciones o vehículos, hurtos, seguimiento de sospechosos |
| **Entretenimiento** | Videojuegos, realidad virtual, interacción humano-robot, interacción humano-computadora |
| **Identidad** | Carnet de identificación, carnet de conducir, pasaportes, inmigración, acceso a aeropuertos |

Por lo que respecta a los métodos empleados en este proyecto; Fisherfaces, Eigenfaces y LBPH, es importante marcar que se encuentran en el área del machine learning, concretamente en el aprendizaje supervisado, conjuntamente al uso de la tecnología de visión por computador. En este caso no se ha utilizado el deep learning porque los métodos actuales basados en machine learning son bastante eficientes y ofrecen resultados buenos sin consumir una ingente cantidad de recursos de memoria y de procesamiento, además de facilitar su implementación.

El funcionamiento de un sistema de reconocimiento facial se centra en cinco procesos:

- **Adquisición de la imagen:** Se captura la imagen, bien sea a partir de la toma de una fotografía o de la extracción de un fotograma de un vídeo.
- **Detección de la cara:** Se detectan los rostros existentes en la imagen.
- **Acondicionamiento de la imagen:** El objetivo es que todas las imágenes con las que se trabaja tengan características similares. Habitualmente se utiliza la rotación para que los rasgos faciales estén alineados, además de la ecualización del histograma para eliminar contrastes excesivos y defectos de iluminación. Además, según convenga, se pueden aplicar técnicas para eliminar ruido de la imagen, o para recortarla y obtener un tamaño igual en todas las imágenes. Todas las imágenes se transforman a escala de grises.
- **Extracción de características:** Se procesa la imagen para obtener las características que distinguen una cara de otra.
- **Algoritmo de reconocimiento:** Aplicando el algoritmo de reconocimiento, se comparan los resultados con la base de datos para obtener la identidad de la persona en cuestión.

A continuación, se explicarán más detalladamente los dos procesos principales involucrados en cualquier sistema de reconocimiento facial.

### 3.3.1. Detección facial

La detección facial se centra en determinar la presencia de caras en una imagen sin identificarlas, proporcionando su localización y su tamaño. Cuando se trata de un video, se habla del seguimiento de una cara.

Actualmente, el método de detección facial más conocido es el algoritmo de Viola-Jones basado en los descriptores Haar y descrito en su artículo técnico "*Rapid Object Detection using a Boosted Cascade of Simple Features*" (Viola & Jones, 2001). Dicho artículo posee más de 20.000 citas, debido a que se trata de uno de los primeros algoritmos de detección de objetos en conseguir buenos resultados en tiempo real y sin consumir grandes cantidades de recursos. Además, a pesar de ser un método creado a principios del siglo XXI, su uso sigue siendo muy extendido en aplicaciones de la vida cotidiana, tales como en móviles o cámaras fotográficas. Es por estos motivos que en el presente proyecto se ha utilizado el algoritmo de Viola-Jones para la detección de rostros, el cual será descrito con más detalle en el capítulo siguiente.

*Figura 4.- Detección facial con el algoritmo de Viola-Jones. Fuente: Propia*

Es importante hacer mención a otro algoritmo alternativo al anterior para la detección facial. Se trata de un algoritmo basado en los descriptores HOG (*Histogram of Oriented Gradients*) (Dalal & Triggs, 2005). Su funcionamiento se basa en dividir la imagen en varias celdas, dentro de las cuales, y analizando la intensidad de los píxeles, se obtiene su respectivo descriptor HOG (Figura 5). Esto se usa para obtener los contornos de un objeto, o en este caso, de una cara, y empleando el clasificador SVM (*Support Vector Machine*) se clasifica.

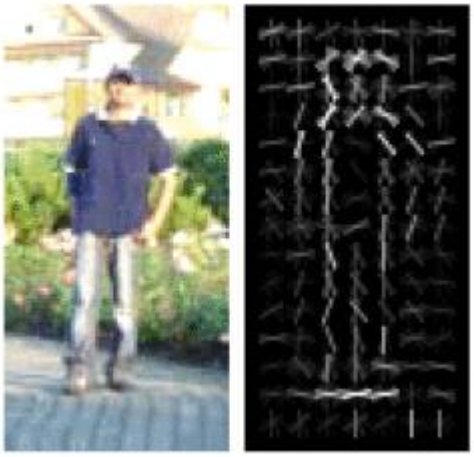

*Figura 5.- Aplicación de los descriptores HOG.* ***Izquierda:*** *Imagen bajo test dividida en varias celdas.* ***Derecha:*** *Visualización de los descriptores HOG sobre la imagen. Fuente:* (Dalal & Triggs, 2005)

### 3.3.2. Reconocimiento facial

Una vez se ha realizado la detección del rostro, se recorta la imagen y se trabaja solamente con la ROI (*Region of Interest*). Dentro de esta región (rectángulo azul de la Figura 6) es donde se procederá con el reconocimiento facial, el cual permitirá predecir la identidad del sujeto en cuestión a partir de la base de datos. Evidentemente si el sujeto que se intenta reconocer no está en la base de datos no lo reconocerá correctamente.

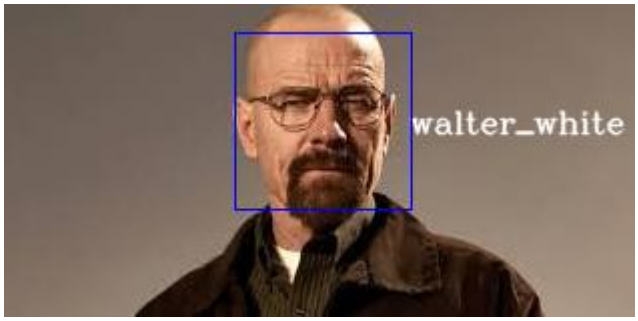


*Figura 6.- Reconocimiento facial. Método LBPH. Fuente: Propia*

Para llevar a cabo esta tarea de reconocimiento existen múltiples algoritmos que se pueden utilizar, de entre los cuales se han escogido los siguientes para realizar su estudio: Fisherfaces, Eigenfaces y LBPH (*Local Binnary Pattern Histogram*).

Por lo que respecta a los métodos o algoritmos de reconocimiento facial habituales, cabe destacar la existencia de dos tipos distintos de sistemas de reconocimiento:

- **Holísticos:** El reconocimiento se lleva a cabo a partir de toda la imagen, es decir, a partir de todas las características faciales en conjunto. Son métodos basados en correlación, donde las

imágenes se interpretan como vectores en un espacio de gran dimensionalidad. El sistema más primitivo de este tipo es el *template matching*, en el cual se utilizan modelos de comparación donde la unidad a comparar es el píxel, motivo por el cual es imposible implementarlo en tiempo real. Es por esto que normalmente se trabaja con métodos que se basan en la reducción de la dimensionalidad, como bien pueden ser PCA (*Principal Component Analysis*) o LDA (*Linear Discriminant Analysis*). Fisherfaces y Eigenfaces pertenecen a este tipo de sistemas de reconocimiento.

- **Locales o Geométricos:** El reconocimiento se lleva a cabo a partir de las diferentes regiones o características individuales de una cara. LBPH pertenece a este tipo de sistemas.

## 3.4. Estado del arte del reconocimiento facial

Tal y como se ha podido apreciar, hasta el momento solamente se ha hablado de sistemas de reconocimiento facial en 2D, desarrollados antes del 2004. A día de hoy, 15 años después, se han producido grandes avances en este campo. La organización del evento FRGC (*Face Recognition Grand Challenge*) se centra en el desarrollo de nuevas tecnologías y algoritmos para mejorar los sistemas de reconocimiento facial.

En las conferencias sobre *Visión por Computador y Reconocimiento de Patrones* realizadas en Junio de 2005 por parte de la sociedad IEEE, se expusieron los resultados del último FRGC (P. Jonathon Phillips et al., 2005), basados en el análisis de las nuevas técnicas aplicadas a seis experimentos diferentes y con una base de datos de 50.000 personas. Dichas técnicas de vanguardia incluyen; reconocimiento a partir de escáneres 3D, reconocimiento a partir de imágenes en alta resolución, reconocimiento a partir de múltiples imágenes, reconocimiento a partir del modelado 2D + 3D, reconocimiento a partir de varios algoritmos y el pre-procesado de imágenes. De esta manera, se obtuvieron y expusieron las siguientes conjeturas (P. Jonathon Phillips et al., 2005):

- **Conjetura I:** "La forma del canal de una imagen 3D es más eficaz para el reconocimiento facial que una imagen en 2D".
- **Conjetura II:** "Una imagen 2D en alta resolución es más eficaz para el reconocimiento facial que una imagen en 3D".
- **Conjetura III:** "Usar 4 o 5 imágenes 2D bien escogidas es más eficaz para el reconocimiento que una imagen 3D o que un modelo 2D+3D fusionado".
- **Conjetura IV:** "El aspecto más prometedor del 3D es abordar el caso en el que las imágenes conocidas de una persona son muestras biométricas en 3D y las muestras a reconocer son imágenes fijas no controladas".

- **Conjetura V:** "La solución al FRGC provocará un replanteamiento de cómo se implementa el reconocimiento facial". Esta conjetura hace una alusión al hecho de que un sistema de reconocimiento facial es un complejo sistema tecnológico formado por componentes clave como la obtención, el almacenamiento y la transmisión de las imágenes, además del algoritmo de reconocimiento; haciendo notar que todos estos componentes están obsoletos para poder tener sistemas de reconocimiento facial implementados en la vida real y al nivel que se requiere en la actualidad. Por eso es necesario hacer un replanteamiento general de cómo se implementa el reconocimiento facial a día de hoy.

Siguiendo la línea de la Conjetura V, y teniendo en cuenta de que se trata de una conferencia realizada en 2005, durante estos 15 años los gigantes tecnológicos han ido realizando y mejorando sus propios sistemas de reconocimiento facial. Neurotechnology, Google, IBM, Facebook y Microsoft, entre otros, se encuentran a la vanguardia de la tecnología de reconocimiento, ofreciendo tanto software de última generación, como hardware, incluyendo escáneres 3D y cámaras para tomar imágenes en alta resolución.

### 3.4.1. Breve historia del reconocimiento facial

A continuación se hace un breve repaso a la historia del reconocimiento facial (*Advancements in Computer based Facial recognition systems*, n.d.), remarcando los sucesos más importantes, y complementando el apartado anterior sobre estado del arte.

- **Marcadores faciales:** En el año 1971 se desarrolló un sistema (Goldstein et al., 1971) basado en 22 marcadores faciales computados de forma manual con el objetivo de reconocer la identidad de una persona en base a su rostro.
- **Eigenfaces, Fisherfaces y LBPH:** En los años 90 y 2000 se desarrollaron estos tres algoritmos de reconocimiento facial automatizados. Su funcionamiento se basa en el uso del algebra y de la estadística. Poseen un ratio de éxito bastante elevado si se tiene en cuenta que hace más de 20 años que fueron creados.
- **Programa FERET:** El programa FERET (*Facial Recognition Technology*) se inició en 1993 con el objetivo de potenciar el desarrollo de los sistemas de reconocimiento facial. Se creó una extensa base de datos para poder experimentar con ellos.
- **Face Recognition Vendor Tests (FRVT):** Se trata de un método de evaluación de sistemas de reconocimiento facial desarrollado en 2002 (P.J. Phillips et al., 2002). Se realizaron varios eventos en los que participaron universidades, equipos de investigación y empresas para poner a prueba sus sistemas de reconocimiento.

- **FRGC:** La organización del evento FRGC (*Face Recognition Grand Challenge*) se centra en el desarrollo de nuevas tecnologías y algoritmos para mejorar los sistemas de reconocimiento facial.
- **DeepFace (2014):** Se trata de un sistema de reconocimiento facial basado en deep learning y desarrollado por Facebook. Utiliza una red neuronal artificial de nueve capas con más de 120 millones de conexiones y ha sido entrenado por más de cuatro millones de imágenes. Tiene una tasa de aciertos del 97%.
- **FaceIt ARGUS:** La compañía Identix ha creado una tecnología de reconocimiento facial llamada FaceIt ARGUS, basada en el análisis de la textura de la piel (líneas, poros, cicatrices, etc.). Sus desarrolladores afirman que puede llegar a identificar a dos gemelos idénticos.
- **Actualidad**: Con las tecnologías existentes, se puede llegar a realizar el reconocimiento facial de una persona desde diferentes ángulos, con partes del rostro cubiertas y en menos de 1 segundo a partir de solamente una imagen como referencia. Además, es posible determinar el sexo y la edad de esa persona, tanto a partir de imágenes como a partir de un vídeo y con diferentes tipos de iluminación.

# 4. Análisis de los métodos de reconocimiento facial

Una vez se han explicado y entendido los conceptos que involucran al reconocimiento facial, se procederá a describir y analizar los métodos y algoritmos que intervienen en este proceso.

## 4.1. Detección facial

Por lo que respecta a la detección facial, tal y como ya se ha mencionado, existen varios métodos para llevarla a cabo, pero en este caso se ha empleado el algoritmo de Viola-Jones.

### 4.1.1. Algoritmo de Viola-Jones

El algoritmo de Viola-Jones se describe en el artículo "*Rapid Object Detection using a Boosted Cascade of Simple Features*" (Viola & Jones, 2001). Se trata de un método basado en machine learning para la detección de objetos y que es capaz de procesar imágenes con una alta velocidad y con altas tasas de éxito. Su trabajo se distingue por la contribución de tres puntos clave.

El primero de ellos es la introducción del concepto de imagen integral, que principalmente consiste en analizar la imagen mediante segmentos rectangulares en vez de hacerlo píxel a píxel, por lo que el tiempo de procesado se disminuye de forma cuantiosa. La imagen integral en el punto x,y contiene la suma de los valores de cada uno de los píxeles que se encuentran por encima y a la izquierda de x,y:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{1}$$

donde $ii(x,y)$ es la imagen integral y $i(x,y)$ es la imagen original.

Esta contribución, además, se basa en el trabajo "*A General Framework for Object Detection*" realizado por Papageorgiu y otros en 1998 (Papageorgiou et al., 1998). En él se explica cómo a partir de la secuencia de funciones *Haar wavelets* se obtienen los descriptores Haar rectangulares, que combinados con el concepto de imagen integral, es posible procesarlos de manera mucho más rápida. El uso principal de los descriptores Haar se centra en la detección de objetos.

Cada uno de estos descriptores se traduce en un único valor, el cual es calculado al restar los píxeles de la imagen que están en la zona blanca y al sumar los que están en la zona oscura (Figura 7), o dicho de otra manera, se pretende hacer una búsqueda de los bits que son más oscuros que otros para encontrar las zonas de mayor contraste de una cara. Inicialmente, los descriptores Haar tenían una

medida fija, pero al introducir el concepto de imagen integral, se pueden calcular *Haar-like features* de cualquier tamaño, con los que se puede obtener el contorno de un rostro.

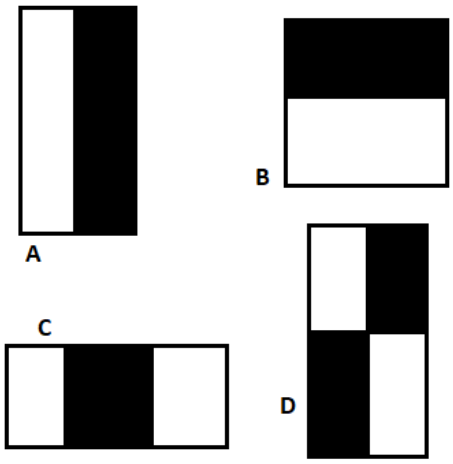


*Figura 7.- Descriptores Haar.* ***A, B:*** *Descriptores de dos rectángulos; su valor es la resta entre la suma de píxeles de cada rectángulo.* ***C:*** *Descriptor de tres rectángulos; su cómputo se realiza restando los rectángulos exteriores al rectángulo central.* ***D:*** *Descriptor de cuatro rectángulos; se calcula la diferencia entre dos pares de rectángulos diagonales. Fuente:* (Viola & Jones, 2001)

La segunda contribución se trata de un método para construir un clasificador seleccionando una pequeña cantidad de descriptores rectangulares utilizando AdaBoost (Freund & Schapire, 1995). Una imagen cualquiera puede contener tantos descriptores Haar como píxeles, o incluso más. En una simple imagen de 24 x 24 píxeles existen más de 180.000 posibles combinaciones de descriptores Haar. Para evitar esto y garantizar un procesado rápido, mediante el clasificador, compuesto de clasificadores simples (uno por cada descriptor Haar distinto), se excluyen la mayoría de los descriptores disponibles, dejando solamente los que afectan a las regiones de mayor interés para la identificación de un rostro.

Para cada descriptor, se crea un clasificador simple que, mediante un función de clasificación, determina el umbral óptimo que clasifica un menor número de imágenes erróneamente. De esta manera, cada clasificador simple, $h_j(x)$, es una función binaria construida a partir de un umbral $\theta_j$ y de un descriptor Haar $f_j(x)$:

$$h_j(x) = \begin{cases} 1 & si \;\; f_j(x) > \theta_j \\ 0 & otros \end{cases} \qquad (2)$$

donde; $j$ representa un descriptor

$x$ es una sección de la imagen de 24 x 24 píxeles.

Este algoritmo está diseñado para seleccionar el descriptor único que mejor separa las muestras negativas y postivas. De esta manera, el primer descriptor Haar rectangular será el más inclusivo, y se

basa en el contraste entre los ojos y las mejillas (se puede observar en la Figura 8). Los siguientes descriptores serán cada vez más restrictivos y se ajustarán cada vez más a las características faciales.

*Figura 8.-* ***De izquierda a derecha:*** *Imagen utilizada para el test, primer descriptor seleccionado por AdaBoost, segundo descriptor seleccionado por AdaBoost. Fuente:* (Viola & Jones, 2001)

Los primeros experimentos demostraron que un clasificador basado en 200 descriptores rectangulares tiene una tasa de éxito del 95%.

La tercera contribución y la mayor de ellas se centra en un método para combinar clasificadores AdaBoost sucesivamente más complejos en una estructura en cascada (Figura 9) que sirve para aumentar la velocidad de detección al conseguir una focalización en las regiones de mayor interés, descartando las imágenes que no poseen características propias de una cara.

En el clasificador inicial se eliminan una gran cantidad de muestras negativas con muy poco procesamiento, y a medida que se avanza se requiere cada vez más tiempo en realizar el proceso. Cuando la imagen llega al final del recorrido de la cascada, significa que se ha detectado una cara. El objetivo de este algoritmo en cascada es el de rechazar tantos negativos como sea posible y en la etapa más temprana posible.

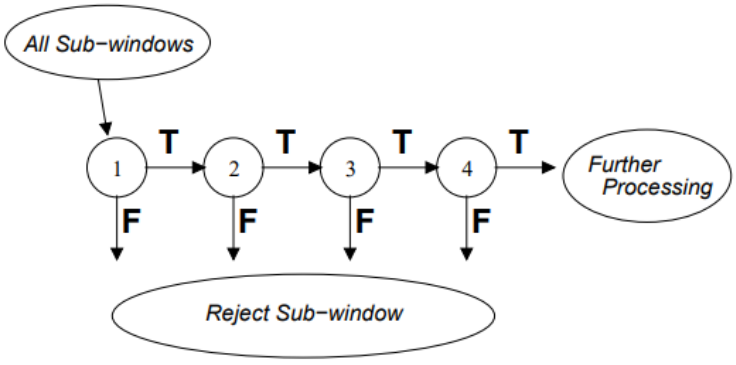


*Figura 9.- Esquemático de la detección en cascada.* ***F*** *y* ***T*** *representan Falso y Verdadero, respectivamente. Las secciones de la imagen que obtienen T (resultado positivo), pasan a la siguiente fase del procesado, mientras que las que obtienen F, son rechazadas. En cada fase interviene un clasificador simple. Fuente:* (Viola & Jones, 2001)

Posteriormente, en el año 2003, Viola y Jones presentaron una extensión a su trabajo inicial (Jones & Viola, 2003). En esta extensión se añadió la funcionalidad de detectar caras de perfil o con el rostro girado.

Finalmente, cabe destacar que este algoritmo se trata de un sistema de detección facial que es 15 veces más rápido que cualquier otro sistema desarrollado hasta ese momento, con el añadido de que presenta una gran efectividad.

## 4.2. Reconocimiento facial

Para el proceso de reconocimiento facial se han empleado los algoritmos Eigenfaces, Fisherfaces y LBPH, los cuales van a ser explicados en detalle a continuación.

### 4.2.1. Eigenfaces

El método Eigenfaces se basa en el Análisis de Componentes Principales o PCA (*Principal Component Analysis*), una técnica ampliamente utilizada en estadística para describir un conjunto de datos mediante nuevas variables no correlacionadas con el objetivo de reducir la dimensionalidad de dicho conjunto.

La técnica PCA fue inventada en 1901 por Karl Pearson (Pearson, 1901) y posteriormente desarrollada por Hotelling en 1933 (Hotelling, 1933).

Así, el Análisis de Componentes Principales tiene por objetivo describir con precisión los valores de $p$ variables a través de un subconjunto $r < p$, de manera que se reduce la dimensión del problema pasando por una pequeña pérdida de información.

Suponiendo que se tienen los valores de $p$-variables en $n$ elementos de una población dispuestos en una matriz $X$ de dimensiones $n \times p$, se desea encontrar un sub-espacio de dimensión menor que $p$ de manera que al proyectar sobre él los puntos, se conserve la mayor información posible. Considerando un caso de dos dimensiones ($p = 2$), esto se traduce en trazar una recta que recoja la máxima variación de los datos, es decir, se pide que las distancias entre los puntos originales y sus proyecciones sobre la recta sean mínimas, lo cual se puede expresar de la siguiente manera:

$$\min \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} |x_i - z_i \boldsymbol{a_1}|^2 \qquad (3)$$

donde $x_i$ es un punto en el espacio

$z_i \boldsymbol{a_1}$ es el vector que representa la proyección de dicho punto

$r_i$ es la distancia entre el punto $x_i$ y la dirección $\boldsymbol{a_1}$

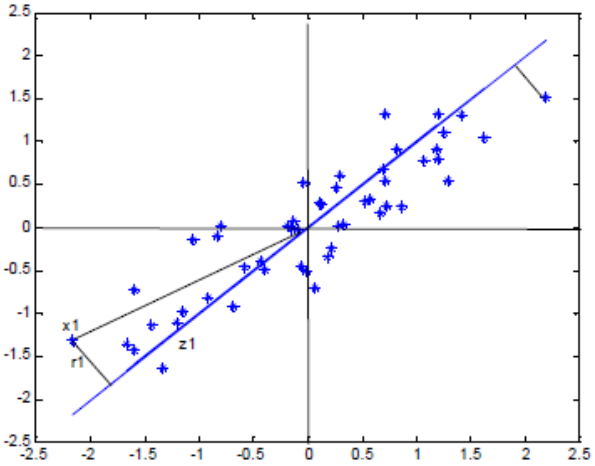


*Figura 10.- Recta que minimiza las distancias de los puntos respecto a ella (**azul**). Para el caso de estudio presente, cada punto representa una imagen y cada eje es la variable de un pixel de la imagen. Escala arbitraria. Fuente:* (Peña, 2002)

A partir del teorema de Pitágoras, de la Figura 10 y para todos los puntos, se puede escribir:

$$\sum_{i=1}^{n} x_i' x_i = \sum_{i=1}^{n} r_i^2 + \sum_{i=1}^{n} z_i^2 \tag{4}$$

Como la distancia desde el origen al punto, $x_i' x_i$, se mantiene constante, se cumple el hecho de que minimizar $\sum_{i=1}^{n} r_i^2$ es equivalente a maximizar $\sum_{i=1}^{n} z_i^2$. Esto se traduce en que para conseguir los componentes principales se traza una recta (la cual define un vector propio del nuevo espacio) que minimiza la distancia entre el punto y dicha recta y maximiza la longitud de la proyección del punto sobre la recta.

El primer componente principal es la combinación lineal de las variables originales que presente mayor varianza. Los valores de este primer componente vienen dados por el vector $\boldsymbol{z_1} = \boldsymbol{X a_1}$. Su varianza se expresa de la siguiente manera:

$$Var(\boldsymbol{z_1}) = \boldsymbol{a_1' S a_1} \tag{5}$$

donde $\boldsymbol{S}$ es la matriz de varianzas y covarianzas. Imponiendo que $\boldsymbol{a_1' a_1} = 1$ y maximizando la expresión de varianza empleando el multiplicador de Lagrange como método de optimización de funciones con restricciones, se obtiene:

$$\boldsymbol{S a_1} = \lambda \boldsymbol{a_1} \tag{6}$$

lo que implica que $\boldsymbol{a_1}$es un vector propio de la matriz $\boldsymbol{S}$ y $\lambda$ es su correspondiente valor propio.

Para el segundo componente, tras realizar los cálculos respectivos, se obtiene:

$$\boldsymbol{S a_1} = \lambda_1 \boldsymbol{a_1} \tag{7}$$

$$\boldsymbol{S a_2} = \lambda_2 \boldsymbol{a_2} \tag{8}$$

por lo que $\boldsymbol{a_1}$ y $\boldsymbol{a_2}$ son los vectores propios de $\boldsymbol{S}$ y $\lambda_1, \lambda_2$ son los dos mayores valores propios. De esta manera, estos vectores, que son ortogonales entre sí por definición, son los que definen el nuevo espacio.

Análogamente, el espacio de dimensión $r < p$ que mejor representa un conjunto de puntos viene definido por los vectores propios asociados a los $r$ mayores valores propios de $S$, dando lugar a unas nuevas variables llamadas componentes principales. Calcular dichos componentes principales equivale a aplicar una transformación ortogonal a las variables originales para obtener unas nuevas variables no correlacionadas entre sí (Peña, 2002).

En el año 1991 Turk y Pentland (M.A. Turk & A.P. Pentland, 1991) (Turk & Pentland, 1991) acuñaron por primera vez el nombre de Eigenfaces en sus dos artículos técnicos "*Face Recognition Using Eigenfaces*" y "*Eigenfaces for Recognition*", en los cuales desarrollaron y describieron por primera vez un algoritmo para el reconocimiento facial eficaz y en tiempo real, basado en vectores propios o *eigenvectors* (de ahí el nombre Eigenfaces). Su trabajo estuvo motivado por (Sirovich & Kirby, 1987), dónde se emplea por primera vez con éxito el análisis de componentes principales para la representación de imágenes de rostros.

A grandes rasgos, explicado desde el punto de vista del lenguaje de la teoría de la información, lo que se pretende es extraer la información relevante de la imagen de una cara, codificarla de la manera más eficiente posible y comparar la codificación de ese rostro con la base de datos.

Si se traduce a términos matemáticos, el objetivo se centra en buscar los componentes principales de la distribución de las caras, o dicho de otra manera, en buscar los vectores propios de la matriz de covarianza del conjunto de imágenes. El número máximo posible de componentes principales o eigenfaces es igual al número de imágenes en el conjunto de entrenamiento. Sin embargo, las caras pueden ser aproximadas también utilizando solamente los eigenfaces más representativos, es decir, utilizando los que acumulan la mayoría de la variación de los datos. Así, cada imagen puede ser representada como una combinación lineal de los eigenfaces.

Se tiene una imagen $I(x, y)$ compuesta por una matriz de dos dimensiones, $N \times N$ dónde los valores poseen la intensidad de cada píxel, que varía entre 0 y 255. Esta imagen puede ser considerada también

como un vector de dimensión $N^2$, por lo que una imagen cualquiera de 250 x 250 se convierte en un vector de 62.500 dimensiones, o, de manera equivalente, en un punto en el espacio de 62.500 dimensiones. De esta manera, un conjunto de imágenes se corresponden a un conjunto de puntos en este enorme espacio multidimensional. Como todas las imágenes de caras poseen características similares, no estarán distribuidas de manera aleatoria dentro de este espacio, por lo que mediante el análisis de componentes principales se obtendrá un sub-espacio de menor dimensionalidad, definido por un conjunto de vectores propios y unitarios. Estos vectores definen lo que se conoce como un "*face space*" (Figura 11).

Para llevar a cabo esto, se parte de un conjunto de imágenes $\boldsymbol{\Gamma_1}, \dots, \boldsymbol{\Gamma_M}$, a partir de las cuales se obtiene la representación facial media definida como $\boldsymbol{\psi} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{\Gamma_i}$. Cada representación facial individual difiere de la media según el vector $\boldsymbol{\phi_i} = \boldsymbol{\Gamma_i} - \boldsymbol{\psi}$, con lo que finalmente se obtiene la matriz de covarianza:

$$\boldsymbol{C} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{\phi_i}\boldsymbol{\phi_i^T} \tag{9}$$

A partir de la matriz de covarianza $C$, se obtienen los valores propios $\lambda_i$ y los vectores propios $\boldsymbol{a_i}$ que definen el sub-espacio *face space*:

$$\boldsymbol{C a_i} = \lambda_i \boldsymbol{a_i}, i = 1,2, \dots, M \tag{10}$$

Se ordenan los vectores propios en orden descendiente a partir del valor $\lambda_i$. Los $k$ componentes principales son los vectores propios correspondientes a los $k$ valores propios más grandes. De esta manera, se obtiene un sub-espacio $k$-dimensional que contiene la información más relevante o que más variación acumula de las imágenes, consiguiendo que $k < M^2 \ll N^2$.

Como aclaración, es necesario añadir que debido a que tratar con vectores de dimensión $N^2$ resulta poco práctico e inviable a nivel computacional, se resuelve el problema con una matriz $M \times M$ y utilizando combinaciones lineales de los vectores resultantes. De esta manera el análisis pasa de ser del orden del número de píxeles de una imagen, al orden del número de imágenes en el conjunto de entrenamiento.

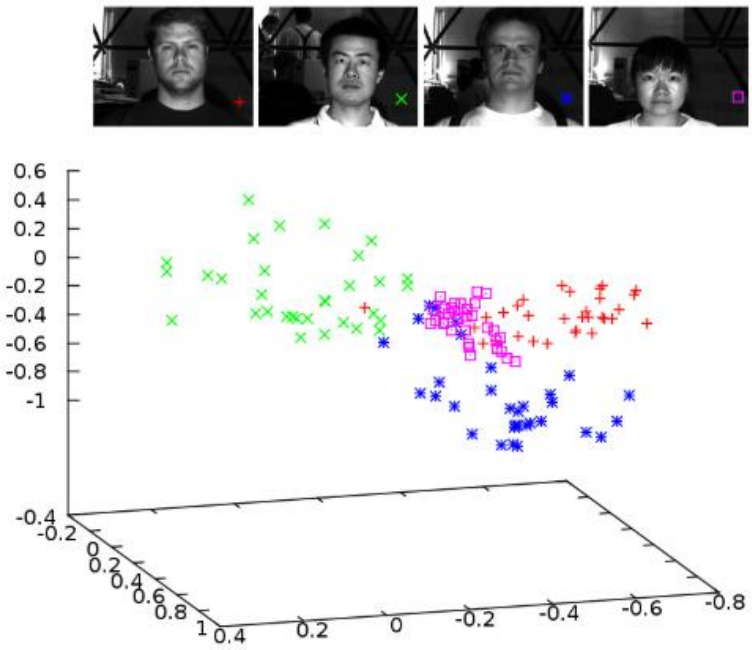


*Figura 11.- Representación en tres dimensiones del sub-espacio facial. Clustering de cuatro personas diferentes. Cada color representa una identidad diferente. Cada punto representa una imagen y cada eje es la variable de un pixel de la imagen (escalado a la unidad porque* $\boldsymbol{a_1'a_1} = 1$ *) Fuente:* (*Plotting face space – dave's blog of art and programming*, n.d.)

Finalmente, una vez se han obtenido los eigenfaces a partir de la matriz de covarianza (se puede ver una representación gráfica de los eigenfaces o componentes principales que agrupan una mayor información en la Figura 12), ya solo queda por hacer la tarea de identificación, la cual se centra en el reconocimiento de patrones y clustering. Para determinar qué identidad se corresponde con una imagen nueva bajo test, se proyectan las imágenes en el sub-espacio facial obtenido y se busca cuál de las caras de la base de datos tiene la menor distancia euclídea respecto a la imagen en cuestión. Esto se puede expresar de la siguiente manera:

$$DIFS = \|\Omega - \Omega_{\boldsymbol{c}}\| \tag{11}$$

donde DIFS (*distance in face space*) es la distancia entre la imagen proyectada $\Omega$ y la representación facial media de cada clase o población $\boldsymbol{\Omega_c}$.

Para más detalles consultar el artículo técnico "*Face Recognition Using Eigenfaces*" (M.A. Turk & A.P. Pentland, 1991).

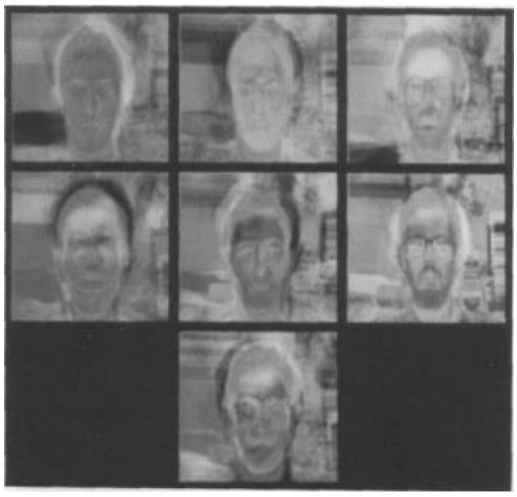

*Figura 12.- Representación de 7 eigenfaces calculados a partir de un conjunto de imágenes. Fuente:* (Turk & Pentland, 1991)

### 4.2.2. Fisherfaces

En el artículo técnico "*Eigenfaces vs Fisherfaces: Recognition Using Class Specific Linear Projection*" (Belhumeur et al., 1997) se describe un nuevo algoritmo de reconocimiento facial llamado Fisherfaces. Este método deriva de Eigenfaces, ya que se basa en el mismo principio de funcionamiento, con la diferencia de que en este caso se pretende desarrollar una técnica de reconocimiento facial mejorada que sea insensible a la variación de iluminación y a las distintas expresiones faciales en las imágenes.

Fisherfaces se basa en el método *Fisher's Linear Discriminant* (FLD) (FISHER, 1936), el cual se generalizó posteriormente en el *Linear Discriminant Analysis* (LDA) o Análisis Discriminante Lineal. Este método, utilizado en estadística, machine learning y reconocimiento de patrones, se utiliza para encontrar una combinación lineal de características que distinguen dos o más clases de objetos.

A continuación, se desarrollará una breve explicación del funcionamiento del Análisis Discriminante Lineal para un caso sencillo con solamente dos clases o poblaciones diferentes.

Se tienen dos poblaciones, $P_1, P_2$ donde hay definida una variable aleatoria vectorial $\boldsymbol{x}$. Suponiendo que las funciones de densidad de ambas poblaciones $f_1, f_2$ son conocidas, se pretende clasificar un nuevo elemento, $\boldsymbol{x_0}$, con valores conocidos de las $p$ variables en una de estas dos poblaciones. Si se conocen las probabilidades a priori, $\pi_1, \pi_2$ con $\pi_1 + \pi_2 = 1$, de que el nuevo elemento pertenezca a una de las poblaciones, su distribución de probabilidad será la siguiente:

$$f(\boldsymbol{x}) = \pi_1 f_1(\boldsymbol{x}) + \pi_2 f_2(\boldsymbol{x}) \tag{12}$$

A continuación se puede calcular la probabilidad a posteriori, $P(i|\boldsymbol{x_0})$ para $i = 1, 2$ por el teorema de Bayes:

$$P(1|\boldsymbol{x_0}) = \frac{P(\boldsymbol{x_0}|1)\pi_1}{P(\boldsymbol{x_0}|1)\pi_1 + P(\boldsymbol{x_0}|2)\pi_2} = \frac{f_1(\boldsymbol{x_0})\pi_1}{f(\boldsymbol{x_0})} = \frac{f_1(\boldsymbol{x_0})\pi_1}{f_1(\boldsymbol{x_0})\pi_1 + f_2(\boldsymbol{x_0})\pi_2} \quad (13)$$

$$P(2|\boldsymbol{x_0}) = \frac{P(\boldsymbol{x_0}|2)\pi_1}{P(\boldsymbol{x_0}|1)\pi_1 + P(\boldsymbol{x_0}|2)\pi_2} = \frac{f_2(\boldsymbol{x_0})\pi_2}{f(\boldsymbol{x_0})} = \frac{f_2(\boldsymbol{x_0})\pi_2}{f_1(\boldsymbol{x_0})\pi_1 + f_2(\boldsymbol{x_0})\pi_2} \quad (14)$$

Como el denominador de las probabilidades es igual en ambos casos, se simplifican las expresiones anteriores, obteniendo una clasificación en $P_2$ si se cumple:

$$P(2|\boldsymbol{x_0}) > P(1|\boldsymbol{x_0}) \quad (15)$$

O, lo que es equivalente a:

$$f_2(\boldsymbol{x_0})\pi_2 > f_1(\boldsymbol{x_0})\pi_1 \quad (16)$$

A continuación, si se aplica el análisis anterior al caso en que $f_1, f_2$ son distribuciones normales cuya función de densidad es la siguiente:

$$f_i(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p}\sqrt{V}} e^{-\frac{(\boldsymbol{x}-\boldsymbol{\mu_i})'V^{-1}(\boldsymbol{x}-\boldsymbol{\mu_i})}{2}} \quad (17)$$

Suponiendo las probabilidades a priori $\pi_1, \pi_2$ iguales en todos los casos, y sustituyendo $f_i(x)$ en la ecuación (16), operando con logaritmos se llega a la siguiente expresión:

$$(\boldsymbol{x} - \boldsymbol{\mu_1})'V^{-1}(\boldsymbol{x} - \boldsymbol{\mu_1}) > (\boldsymbol{x} - \boldsymbol{\mu_2})'V^{-1}(\boldsymbol{x} - \boldsymbol{\mu_2}) \quad (18)$$

Sabiendo que la distancia de Mahalanobis entre el punto observado y la media de la población presenta la siguiente forma

$$D_i^2 = (\boldsymbol{x} - \boldsymbol{\mu_i})'V^{-1}(\boldsymbol{x} - \boldsymbol{\mu_i}) \quad (19)$$

se obtiene finalmente que si

$$D_1^2 > D_2^2 \quad (20)$$

se clasifica el elemento en la población 2. Es decir, se clasifica la observación en la población más cercana medida con la distancia de Mahalanobis. De esta manera, con el discriminante lineal se realiza una clasificación de las muestras en sus respectivas clases o poblaciones.

Cuando se utiliza el Análisis de Componentes Principales para la reducción de dimensionalidad, se produce un efecto no deseado al maximizar la dispersión de las clases, y obteniendo como resultado la retención de variaciones debido a la variación de iluminación y expresiones faciales.

De esta manera, para evitar esta problemática y aumentar la tasa de éxitos del reconocimiento, en el algoritmo Fisherfaces primero se realiza una reducción de dimensiones utilizando PCA, y, posteriormente, se utiliza el método FLD, con el cual se consigue minimizar la dispersión de cada una de las clases y maximizar la distancia entre las medias, es decir se obtiene una mejor clasificación y separación entre clases, que, en este caso, se tratan de identidades faciales. Este fenómeno se puede observar en la Figura 13, dónde se puede comparar el resultado de utilizar el método PCA y el método FLD en un ejemplo en dos dimensiones.

La forma en que (Belhumeur et al., 1997) realizan la clasificación de imágenes, se trata de una simplificación de la técnica LDA expuesta en las líneas previas.

Teniendo un conjunto de $N$ imágenes $\mathbf{x_1}, \dots, \mathbf{x_N}$, en un espacio $n$-dimensional de imágenes, y asumiendo que cada imagen pertenece a una de las $c$ clases $\mathbf{X_1}, \dots, \mathbf{X_c}$, se define la matriz de dispersión entre clases $S_B$ y la matriz de dispersión intra-clases $S_W$ con las siguientes expresiones:

$$S_B = \sum_{i=1}^{c} N_i(\boldsymbol{\mu_i} - \boldsymbol{\mu})(\boldsymbol{\mu_i} - \boldsymbol{\mu})^T \tag{21}$$

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (\boldsymbol{x_k} - \boldsymbol{\mu_i})(\boldsymbol{x_k} - \boldsymbol{\mu_i})^T \tag{22}$$

donde $\boldsymbol{\mu}$ es la media total:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x_i} \tag{23}$$

y $\boldsymbol{\mu_i}$ la media de cada clase $i \in \{1, \dots, c\}$:

$$\boldsymbol{\mu_i} = \frac{1}{|\boldsymbol{X_i}|} \sum_{x_k \in X_i} \boldsymbol{x_k} \tag{24}$$

A partir de estas matrices, se obtiene un ratio que relaciona la dispersión entre clases y la dispersión intra-clases:

$$W = \arg max_W \frac{|W^T S_B W|}{|W^T S_W W|} \tag{25}$$

Así, el objetivo se trata de maximizar el ratio $W$ (matriz con columnas ortogonales), con lo que se consigue aumentar la dispersión entre clases (distancia entre las medias de cada clase) y disminuir la dispersión intra-clases para conseguir una mejor clasificación.

Por lo que respecta al proceso de identificación, es equivalente al que se realiza en el método Eigenfaces. Para más información acerca del algoritmo Fisherfaces, consultar el artículo "*Eigenfaces vs Fisherfaces: Recognition Using Class Specific Linear Projection*" (Belhumeur et al., 1997).

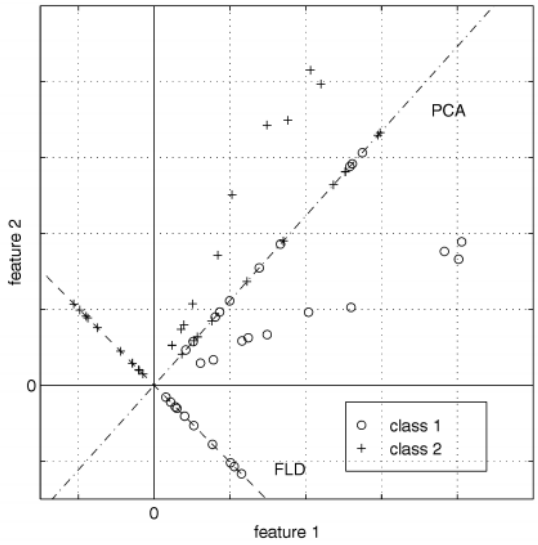


*Figura 13.- Clasificación de dos clases utilizando los métodos **PCA** y **FLD**. FLD ofrece una clasificación más precisa y eficiente. Fuente:* (Belhumeur et al., 1997)

Asimismo, para clarificar el proceso de reconocimiento para el algoritmo Fisherfaces se presenta la siguiente imagen (Figura 14), en la cual se observa el diagrama de bloques que sigue el proceso:

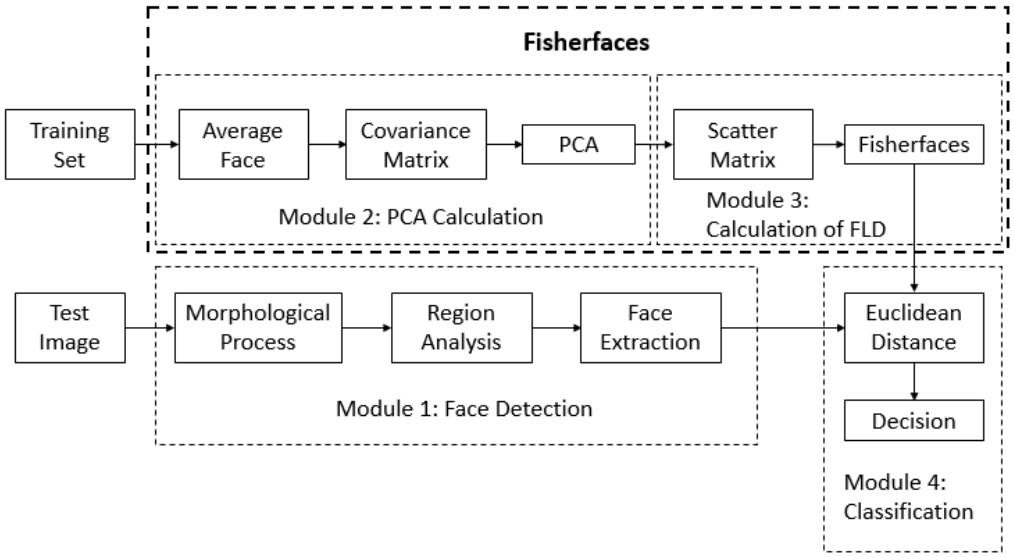


*Figura 14.- Diagrama de bloques del proceso de reconocimiento con Fisherfaces. Fuente:* (Wahyuningsih et al., 2019)

### 4.2.3. Local Binary Pattern Histograms

Dejando atrás los algoritmos de carácter holístico, a continuación se va a desarrollar la explicación del método LBPH (*Local Binary Pattern Histograms*), el cual se trata, como bien se ha explicado previamente, de un método de reconocimiento facial de tipo local.

En el año 2000 se presenta en el artículo "*Gray Scale and Rotation Invariant Texture Clasification with Local Binary Patterns*" (Ojala et al., 2000) una técnica eficiente para el análisis de texturas en escala de grises realizada con LBP. Posteriormente, y basándose en dicho artículo, Ahonen y otros desarrollaron en el año 2004 el trabajo "*Face Recognition with Local Binary Patterns*" (Ahonen et al., 2004), en el cual se describe la primera aproximación para el reconocimiento facial mediante el análisis local de patrones LBP.

La idea principal del algoritmo se centra en dividir la imagen en pequeñas regiones cuadradas de 3 x 3 píxeles. En cada una de estas regiones se aplica el análisis mediante el uso del Local Binary Pattern, el cual consiste en comparar cada píxel central con los que le rodean, de manera que si el valor central es mayor o igual al vecino, se le asigna un valor binario de "0", mientras que si es menor se le asigna un "1", tal como se puede apreciar en el ejemplo de la Figura 15. Con estos valores binarios se obtiene un número (orden según las agujas de un reloj), que pasado a decimal, se convierte en el descriptor LBP de cada uno de los píxeles de una imagen, con los cuales se obtiene la imagen LBP que sirve para acentuar los rasgos faciales (Figura 18).

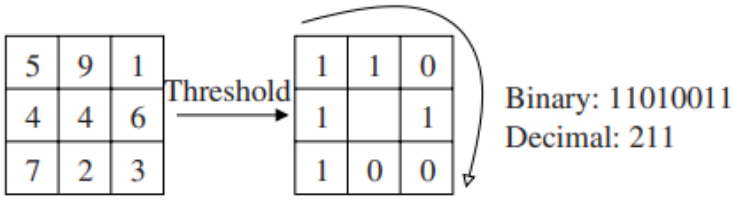


*Figura 15.- Ejemplo de aplicación del operador LBP. Fuente:* (Ahonen et al., 2004)

De esta manera, los píxeles cuyo valor binario sea 1, definen una frontera o una zona de contraste, con lo que se pueden distinguir bordes, puntos, líneas y zonas planas en la imagen.

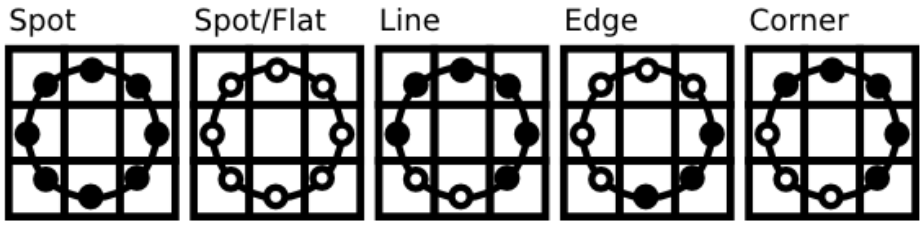


*Figura 16.- Zonas definidas por el operador LBP.* ***De izquierda a derecha:*** *zona oscura, zona clara, línea, borde, esquina. Fuente:* (*Face Recognition with OpenCV — OpenCV 2.4.13.7 documentation*, n.d.)

Al poco tiempo de publicar el operador LBP, se dieron cuenta de que utilizando una región 3 x 3 fija se generaban fallos de codificación en detalles de diferentes escalas, por lo que decidieron establecer un tamaño variable de la región de análisis (Figura 17). Para definir esta región se utiliza la notación (P, R), donde P hace referencia al número de muestras en un círculo de radio R. Cuando los puntos de muestreo no están en el centro de un píxel, se interpolan bilinealmente sus valores.

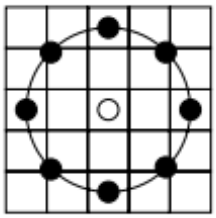

*Figura 17.- Ejemplo de una región circular (8,2). Fuente:* (Ahonen et al., 2004)

Una gran ventaja del operador LBP es que, por definición, es muy robusto ante variaciones en la intensidad lumínica de las imágenes en escala de grises. En la Figura 18, se puede observar como a pesar de haber modificado una misma imagen, el resultado de la imagen LBP es exactamente el mismo en todos los casos:

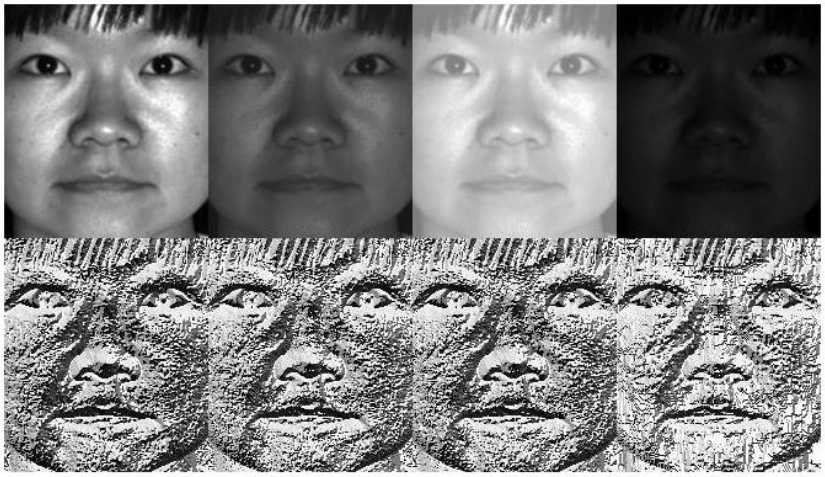

*Figura 18.- Imágenes obtenidas con LBP ante diferentes intensidades lumínicas. Fuente:* (*Face Recognition with OpenCV — OpenCV 2.4.13.7 documentation*, n.d.)

Posteriormente, se divide la imagen en varias celdas rectangulares, y de cada una de ellas se extrae un histograma de las frecuencias de los valores obtenidos con el operador LBP (varían entre 0 y 255), de manera que si se concatenan todos, se obtiene un histograma global de toda la imagen. Este histograma define la imagen completa, con lo que finalmente se tiene una descripción de la cara.

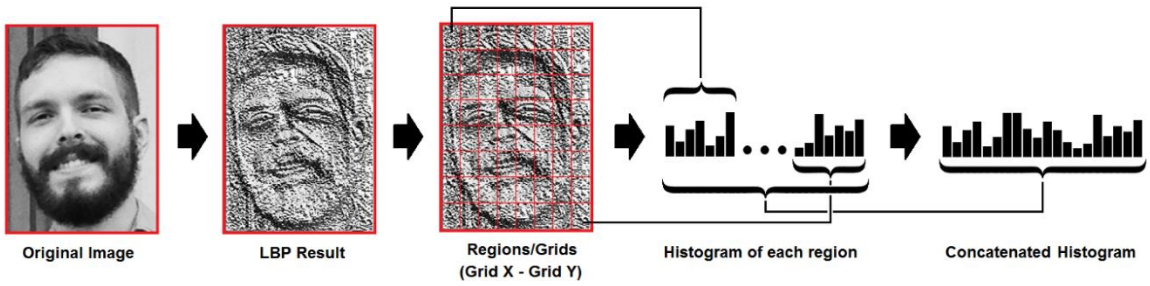


*Figura 19.- Proceso del algoritmo LBPH.* ***De izquierda a derecha:*** *Imagen original, imagen LBP, imagen dividida en 8 x 8 celdas, histograma de cada celda, histograma concatenado. Fuente: (Face Recognition: Understanding LBPH Algorithm - Towards Data Science*, n.d.)

En este ejemplo de la Figura 19, la imagen se divide en 8 x 8 celdas, por lo que el histograma final de la imagen posee un rango de 16.384 (8 x 8 x 256) datos.

Para la obtención de la identidad a partir de una imagen que está bajo test, se compara su histograma con los de la base de datos; los dos histogramas que más se asemejan (menor distancia euclídea entre ellos) determinan el resultado del reconocimiento facial.

## 4.3. Comparación de los métodos de reconocimiento facial

Se pretende hacer una comparación de los métodos de reconocimiento facial expuestos en el apartado anterior, basándose en los principios que rigen los algoritmos y la forma en que trabajan.

Dentro de los métodos holísticos, resulta evidente, que, a grande rasgos, Fisherfaces proporciona mejores resultados que Eigenfaces.

El principal motivo es que Fisherfaces se desarrolló en 1997, tomando como referencia el método de Eigenfaces creado en 1991, y con el objetivo de obtener un método más robusto ante las variaciones de iluminación y de cambio de pose. Esta mejora radica en el proceso de clustering, ya que Fisherfaces utiliza un discriminante lineal, por lo que las clases quedan más separadas entre sí, y, por lo tanto, mejor clasificadas, a diferencia de Eigenfaces que solamente se basa en el PCA (Figura 13).

Así mismo, en el propio artículo en el que se ha desarrollado el método Fisherfaces (Belhumeur et al., 1997), también se incluye una pequeña comparación entre los dos métodos. En dicho artículo se han realizado una serie de experimentos con dos bases de datos distintas, obteniendo las siguientes conclusiones:

- "Todos los algoritmos funcionan perfectamente cuando la iluminación es frontal. Sin embargo, a medida que la iluminación se descentraliza de los ejes, aparece una diferencia de rendimiento significativa entre los dos métodos."

- "En el método Eigenfaces, eliminar los tres primeros componentes principales da como resultado un mejor rendimiento bajo condiciones de iluminación variables." (Figura 20).
- "A pesar de que el método Eigenfaces presenta ratios de error competitivos con los del método Fisherfaces, requiere almacenar más del triple de recursos y toma el triple de tiempo."
- "En general, el método Fisherfaces presenta tasas de error más bajas que el método Eigenfaces y requiere de menos tiempo de computación."

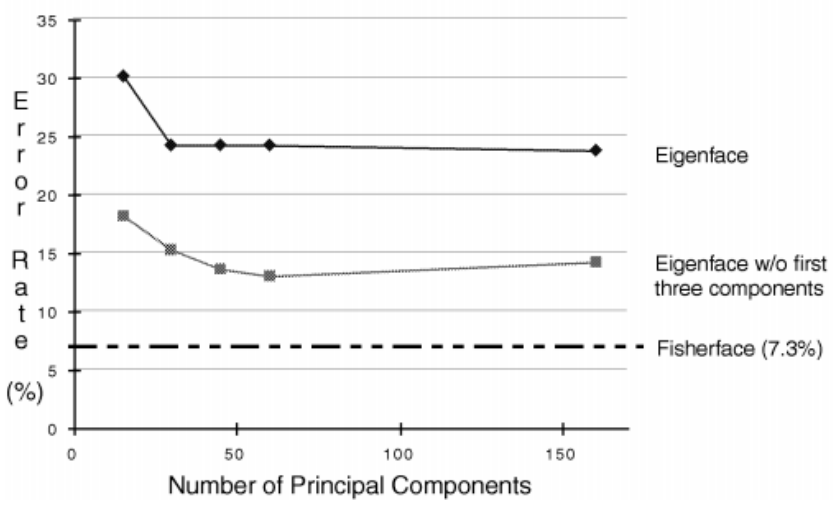


*Figura 20.- Tasa de error en función del número de componentes principales (para método **eigenfaces**, método **eigenfaces sin los tres primeros componentes** y método **fisherfaces**). Fisherfaces presenta la tasa de error más baja, y además es independiente del número de componentes principales. Fuente:* (Belhumeur et al., 1997)

Una vez comparados los dos primeros métodos de tipo holístico, se añade a la comparación el tercer método LBPH de tipo local o geométrico. A partir de aquí, la distinción al mejor método de reconocimiento facial de entre los que están bajo estudio ya no es tan clara como en el caso anterior, ya que se prevé que tanto el método Fisherfaces como LBPH presenten buenos resultados.

En el artículo técnico "*Comparison of Face Recognition Algorithms Using OpenCV for Attendance System*" (SudhaNarang et al., 2018) se realiza un estudio comparativo entre los métodos de reconocimiento facial que hay en la librería OpenCV. La conclusión de dicho estudio es que el método Local Binary Pattern Histograms es el que mayor eficiencia y tasa de éxito presenta.

Nicolas Delbiaggio realiza en su tesis (Delbiaggio, 2017) una comparación de cuatro métodos de reconocimiento facial, tres de ellos son de OpenCV (Fisherfaces, Eigenfaces y LBPH) mientras que el último se trata de OpenFace (*OpenFace*, n.d.), un software de uso libre para reconocimiento basado en redes neuronales. En este caso, el método que mejores resultados obtiene ante condiciones variables y situaciones diversas se trata del software OpenFace. El segundo puesto se debate entre Fisherfaces y LBPH, obteniendo resultados ligeramente parecidos, aunque variados en función de las condiciones.

En el trabajo "*A comparative analysis of face recognition algorithms in solving the problem of visual identification*" (Vladimir et al., 2017), se afirma, tras realizar varios experimentos, que el método LBPH es el que mejor tasa de éxitos proporciona, además de ser el que menor tiempo de cómputo necesita.

Sin embargo, a pesar de resultar evidente que tanto Fisherfaces como LBPH son mejores que Eigenfaces, en el artículo "*PCA versus LDA*" escrito por (Martinez & Kak, 2001) se pretende demostrar que esto no siempre es así. En este artículo se recalca el hecho de que siempre se menosprecia al método basado en PCA y que no hay que quitarle sus méritos. Tras desarrollar un estudio experimental con varias muestras y bajo diferentes condiciones, Martinez y Kak concluyen que "PCA supera a LDA cuando el número de muestras por clase es pequeño".

Observando varios estudios y múltiples bibliografías, la tendencia general afirma que los métodos Fisherfaces y LBPH son los mejores, pero cabe destacar que esto nunca puede ser una afirmación rotunda e inamovible, porque tal como se ha demostrado en varios trabajos, el rendimiento de estos algoritmos varía mucho en función de las condiciones; ya sea la variación de iluminación, el cambio de pose, el número de muestras o la parcial visibilidad del rostro.

Finalmente, resulta de notable interés observar que en ninguno de los trabajos consultados y expuestos en las líneas anteriores habla de los conceptos de overfitting y underfitting ni de cómo estos afectan a las tasas de éxito de reconocimiento. Es por esto que en el siguiente capítulo se intentará hablar de cómo influyen en los algoritmos.

## 4.4. Problemas y dificultades asociadas al reconocimiento facial

Una vez se ha analizado el funcionamiento de cada uno de los algoritmos, es importante también detectar los posibles problemas y dificultades que envuelven a cualquier sistema de reconocimiento facial.

### 4.4.1. Problemática: Factores que intervienen

El reconocimiento facial es una tarea compleja debido a múltiples factores. Un conjunto de caras de distintas personas puede presentar grandes similitudes, por lo que es todo un reto poder diferenciar correctamente la identidad de cada una de esas personas. Además, a medida que se trabaja con bases de datos más grandes, más complicada se vuelve la tarea de separar y clasificar las diferentes clases dentro del espacio facial. Todo esto, sin tener en cuenta otros factores ajenos al propio algoritmo o a la naturaleza de las imágenes.

En el trabajo "*A Survey of Face Recognition Techniques*" (Jafri & Arabnia, 2009) se elabora una clasificación de los factores que intervienen en la problemática del reconocimiento facial.

- **Factores intrínsecos:** Los factores intrínsecos se corresponden puramente a la naturaleza física y fisiológica de la cara, son independientes del observador. A su vez, este tipo de factores puede ser dividido en dos tipos: los intrapersonales y los interpersonales. Los primeros se corresponden a la variación de aspecto en una misma persona, y pueden ser factores como la edad, la expresión facial, el peinado, el crecimiento de la barba, etc. En cambio, los factores interpersonales se corresponden a las variaciones faciales referentes a un conjunto de personas, tales como la etnia a la que se pertenece, la región de origen o el sexo.

- **Factores extrínsecos:** Los factores extrínsecos hacen referencia al cambio de aspecto de una cara debido a la interacción del observador, y comprenden aspectos como la iluminación, la orientación del rostro, la resolución y el enfoque de la imagen, interferencias de ruido, etc.

### 4.4.2. Violación de la privacidad

Un sistema de reconocimiento facial, por su propia naturaleza, requiere de nuestros datos y de nuestra identidad facial (generalmente expresada a través de imágenes digitales) para poder llevar a cabo su tarea. El problema es que al realizar esto se está vulnerando la privacidad de las personas, a no ser que las personas partícipes hayan expresado de manera formal su conformidad y que la legislación de la región lo permita. Por lo tanto, aquí se entra en un ámbito jurídico, ético y legal bastante complejo que no siempre agrada a todos.

Siempre que se trata con datos biométricos, se debe tener en cuenta que estos están protegidos por el Reglamento General de Protección de Datos (RGPD), de manera que cualquier sistema de reconocimiento debe cumplir con los requisitos exigidos por dicho reglamento. Se debe informar correctamente a los usuarios que hagan uso del sistema y se les debe solicitar su expreso consentimiento.

Cualquier tecnología que haga uso de este tipo de sistemas, debe cumplir, al menos, con los siguientes principios (*RGPD | Reglamento Europeo de Protección de Datos*, n.d.):

- Consentimiento
- Transparencia
- Seguridad de los datos
- Privacidad desde el diseño
- Integridad y acceso a los datos
- Responsabilidad proactiva (ej.: avisar de brechas de seguridad)
- Capacidad de eliminar completamente todos los datos

En España, el uso de sistemas de reconocimiento facial se utiliza en lugares públicos como centros comerciales, autobuses, metros, aeropuertos, bancos, etc. Esto está generando problemas de privacidad, y no solo en España, sino también en la Unión Europea y en todo el mundo.

Evidentemente, para la realización del presente proyecto se ha utilizado una base de datos diseñada con todas las garantías referentes a la privacidad. Se trata de una base de datos (*Extended Yale Face Database B (B+) | vision.ucsd.edu*, n.d.) que puede ser utilizada de forma libre para fines académicos y de investigación.

### 4.4.3. Almacenamiento masivo de datos

Otro de los problemas del reconocimiento facial es el almacenamiento masivo de datos. Cuando se habla de machine learning, siempre se debe tener en mente que para que los algoritmos ofrezcan resultados decentes se necesita una gran cantidad de datos para su entrenamiento. A ello hay que sumarle que esos datos se encuentran representados mediante imágenes digitales o incluso en vídeos, y es bien sabido que en función de la calidad de imagen y de otros factores, pueden llegar a ocupar inmensas cantidades de memoria, no aptas para cualquier dispositivo o sistema. Además, una gran cantidad de datos tiene como consecuencia directa un elevado tiempo de cómputo y procesado. Así, resulta evidente que para realizar esta tarea, se necesitan equipos y sistemas bastante potentes y con una gran capacidad de almacenamiento.

Sin ir más lejos, y para poner un ejemplo, Microsoft ha creado una base de datos con 10 millones de imágenes disponible para investigadores. Resulta más que evidente que para manejar dicha base de datos se requiere de una ingente cantidad de recursos de memoria y de tiempo de procesado.

### 4.4.4. Eficacia en el reconocimiento

Tal como ya se ha ido comentando, los métodos de reconocimiento no son siempre 100% eficaces y fiables. Existe la posibilidad de que un sistema de reconocimiento facial confunda la identidad de una persona con la de otra, generando posibles fallos de seguridad. Evidentemente, la eficacia en el reconocimiento depende de muchos factores, los cuales ya se han visto previamente. También depende en gran medida de la tecnología utilizada para llevar a cabo dicha función, por lo que justamente, el objetivo de este trabajo se basa en eso; presentar un análisis del comportamiento de los métodos de reconocimiento facial de OpenCV. Cabe añadir, pero, que en la actualidad ya se han desarrollado sistemas con un gran rendimiento y tasas de éxito muy elevadas, aptas para aplicaciones donde la seguridad es de vital importancia.

## 4.5. Seguridad frente a suplantación de identidad

Otra problemática asociada al reconocimiento facial viene dada por la suplantación de identidad por parte de hackers o personas interesadas en robarnos tanto información confidencial como cualquier otro servicio, poniendo en un alto riesgo a las personas víctimas de este suceso.

Ante un sistema de reconocimiento facial básico en 2D, con una simple fotografía o máscara se le puede robar la identidad a la persona autorizada, dejando al alcance de cualquier delincuente tanto datos e información, como servicios privados o confidenciales.

Para evitar este tipo de delitos, existen varios métodos de seguridad, de entre los cuales se expondrán algunos de ellos a continuación.

### 4.5.1. Eye Blink

Uno de los métodos para evitar la suplantación de identidad se trata del *Eye Blink*; un algoritmo en tiempo real que detecta el parpadeo de los ojos en una secuencia de video grabado con una cámara estándar. Este algoritmo tiene varias funciones dentro del reconocimiento facial, como por ejemplo, detectar si el conductor de un vehículo está dormido o para comprobar el nivel de fatiga de una persona.

En este caso, el algoritmo se utiliza para comprobar que la persona que está accediendo a un servicio mediante reconocimiento facial es real y no se trata de una imagen.

En el artículo "*Real-Time Eye Blink Detection using Facial Landmarks*" (Soukupová & Cech, 2016) se propone un algoritmo para la detección del parpadeo basado en el valor EAR (*Eye Aspect Ratio*), un indicador que proporciona la distancia entre los párpados (basado en 6 puntos) y que caracteriza la apertura del ojo a cada instante (Figura 21).

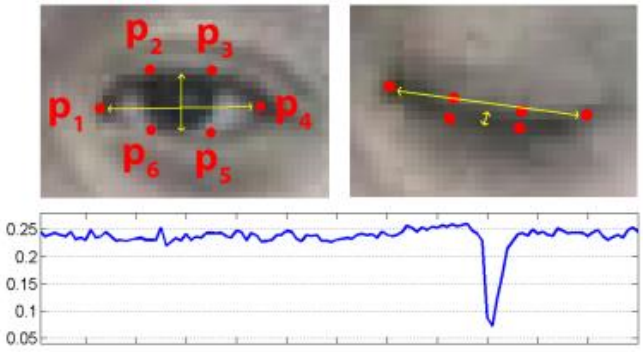


*Figura 21.- Representación del valor EAR en función de la apertura del ojo.* ***Arriba, izquierda:*** *Posiciones de los 6 puntos cuando el ojo está abierto.* ***Arriba, derecha:*** *Posiciones de los 6 puntos cuando el ojo está cerrado.* ***Abajo:*** *Valor del indicador EAR graficado. Eje vertical; valor EAR, Eje horizontal; apertura del ojo/tiempo. Se observa que cuando el ojo se cierra, este valor cae en picado. Fuente:* (Soukupová & Cech, 2016)

### 4.5.2. Uso de cámaras 3D

El uso de cámaras 3D es también otra solución al problema de la suplantación de identidad. Gracias al hecho de trabajar en 3 dimensiones, se puede analizar también la profundidad de los rasgos faciales, por lo que resulta en un método más eficiente y además más seguro porque en este caso no se puede suplantar la identidad con imágenes impresas en papel. Otra ventaja del reconocimiento en 3D es que también es capaz de reconocer un rostro que está de perfil o que se encuentra inclinado, a diferencia de los métodos tradicionales en 2D que solo permiten analizar la parte frontal de una cara.

### 4.5.3. Segundo sistema de control biométrico

La tercera opción propuesta es la de añadir un segundo, o incluso tercer sistema de control biométrico. Esto se traduce en la verificación de la identidad mediante el análisis de más de una característica física. Por ejemplo, un sistema así podría estar formado por una parte de reconocimiento facial y por otra parte de reconocimiento por huella dactilar, de manera que si las dos partes coinciden en la misma identidad, entonces se permite el acceso al servicio.

Sin lugar a duda este es uno de los métodos más seguros, debido a que es altamente complicado suplantar varios rasgos físicos a la vez.

# 5. Desarrollo de la fase experimental

A continuación se detalla el proceso seguido para el desarrollo del presente proyecto. En primer lugar, se describirán las especificaciones técnicas y herramientas utilizadas. En segundo lugar, se hablará de la base de datos, detallando sus características y su forma de uso. En tercer lugar, se expondrá el procedimiento seguido y las fases de desarrollo. Finalmente, se mostrarán los resultados experimentales obtenidos durante el proceso, así como un análisis detallado y minucioso de los mismos.

## 5.1. Especificaciones técnicas y herramientas utilizadas

### 5.1.1. Software

A continuación, en la Tabla 2, se detalla el software utilizado en la realización del TFG:

*Tabla 2.- Software utilizado*

| Software | Licencia de uso | Descripción |
|---|---|---|
| **PyCharm Community Edition 2019.3.3 x64** | Libre | Entorno de desarrollo integrado (IDE) para programación en Python |
| **Python 3.6.0** | Libre | Lenguaje de programación |
| **Excel 2013** | Comercial | Hoja de cálculo y análisis de datos |
| **Word 2013** | Comercial | Edición de documentos |
| **Mendeley** | Libre | Base de datos para citas y referencias bibliográficas |

El entorno de desarrollo *PyCharm* (Anexo A1) ha resultado muy útil y cómodo para la implementación del código de programación en *Python 3.6.0*. Mediante este IDE se ha creado un entorno virtual de trabajo para todo el proyecto.

Gracias a esto, se han organizado todos los archivos de trabajo y las bases de datos dentro de un mismo directorio. Además, ha permitido instalar todas las librerías que se han utilizado de forma aislada y permanente en dicho entorno de trabajo, evitando así interferencias con otras librerías o archivos externos de otros posibles trabajos. Otra ventaja que se ha aprovechado de *PyCharm*, es la posibilidad

de hacer copias de seguridad a todo el entorno virtual de trabajo, consiguiendo así que se puedan recuperar tanto los ficheros creados como las librerías asociadas de forma fácil sin tener que volver a repetir todo el proceso de instalación.

Respecto al resto de software, solamente cabe destacar que se ha decidido utilizar *Mendeley* debido a la facilidad que ofrece para gestionar la gran cantidad de referencias bibliográficas consultadas.

### 5.1.2. Hardware

En lo referente al hardware utilizado, como solamente se ha utilizado un ordenador portátil (con su WebCam) para la realización de todo el proyecto, en la Tabla 3 se describirán sus especificaciones técnicas.

*Tabla 3.- Especificaciones técnicas del hardware utilizado*

| Especificación | Propiedades |
|---|---|
| **Modelo** | Acer TravelMate P256-MG |
| **Procesador** | Intel Core i7-4510U<br>2,60 GHz |
| **Memoria RAM** | 8 GB |
| **Tipo de sistema** | Sistema operativo de 64 bits |
| **Sistema operativo** | Windows 10 Home |
| **Pantalla** | 15.6" HD (1366 x 768) |
| **WebCam** | HD WebCam Microsoft |

Cabe destacar que, a pesar del consumo de recursos al que ha sido sometido (mayormente en el procesado de una gran cantidad de imágenes), ha respondido de forma eficiente y ha presentado un buen rendimiento en todo momento.

### 5.1.3. Librerías de Python

A continuación se describirán brevemente las librerías de Python utilizadas. La más importante es OpenCV, por lo que se le ha dedicado un apartado entero. Posteriormente, se procede con el resto de librerías, ya que aunque no son menos importantes, solamente sirven como complemento para la librería principal.

Antes de empezar, es conveniente mencionar que todas las librerías se han instalado mediante el *Terminal* de *PyCharm*, utilizando el instalador *pip* (Package Installer for Python).

#### 5.1.3.1. OpenCV

OpenCV (Open Source Computer Vision Library) (*OpenCV*, n.d.) es una librería de software de visión por computador y machine learning que incorpora más de 2500 algoritmos optimizados, los cuales se pueden usar para detección y reconocimiento de caras, rastrear movimientos en video, identificar objetos, clasificar acciones humanas, y un largo etcétera.

Se trata de una librería de código abierto desarrollada en 1999 por Intel bajo licencia BSD, por lo que se puede usar y modificar el software libremente, siempre y cuando no se elimine el aviso de copyright. Debido a que se trata de un software ampliamente usado a nivel global (18 millones de descargas), hay mucha documentación y tutoriales accesibles, así como multitud de artículos técnicos disponibles en *IEEE Xplore*, por lo que se facilita su uso y aprendizaje.

#### 5.1.3.2. Otras librerías

Dentro de este sub-apartado, en la Tabla 4, se describen las librerías utilizadas en la realización del presente proyecto.

*Tabla 4.- Librerías de Python usadas*

| Librería | Descripción |
|---|---|
| **time** | Funciones relacionadas con el tiempo |
| **pickle** | Implementa protocolos binarios para serializar una estructura de objeto.<br>Ej.: Convertir diccionario de Python en archivo .yml |
| **cvlib** | Librería de código abierto para visión por computador |
| **os** | Relacionado con funcionalidades que involucran al sistema operativo.<br>Ej.: leer o escribir archivos, carpetas, etc |
| **numpy** | Paquete fundamental para procesado matemático |
| **matplotlib** | Librería para creación de gráficos a partir de datos |
| **pillow** | Añade soporte para abrir, manipular y guardar varios formatos de imagen diferentes |

En el apartado de anexos se puede observar el código implementado, dónde se puede apreciar con más detalle cómo se importa cada librería y en qué casos es necesario utilizarlas y en cuáles no.

## 5.2. Base de datos

En la realización del presente proyecto se ha utilizado la base de datos *Extended Yale Face Database* B (*Extended Yale Face Database B (B+) | vision.ucsd.edu*, n.d.), la cual es de uso libre y ha sido creada por un equipo de investigación (Georghiades et al., 2001) de la *U.C.S.D Computer Vision*. Originalmente, se constituye de un total de 16.128 imágenes en escala de grises (480 x 640 px) y con formato *.pgm*, es decir, que cada imagen se trata de una matriz de números naturales comprendidos entre 0 y 255 donde cada valor representa un píxel con su nivel de intensidad. La base de datos está formada por 28 sujetos diferentes, cada uno con 9 poses faciales diferentes y bajo 64 condiciones de iluminación.

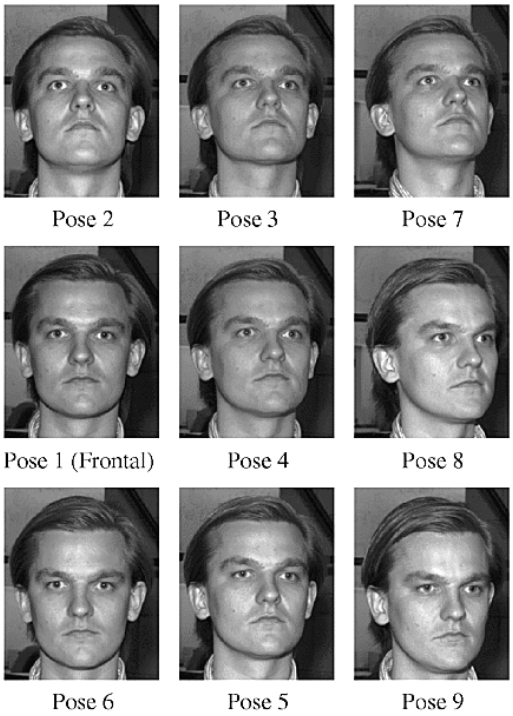


*Figura 22.- Imágenes de un sujeto con sus 9 poses e iluminación neutra. Fuente:* (Georghiades et al., 2001)

Cabe destacar que se trata de una base de datos compleja debido a la gran cantidad de iluminaciones diferentes que se utilizan. Tanto es así, que se ha observado (Georghiades et al., 2001) que "las variaciones entre las imágenes de una misma cara debido a la iluminación y a la dirección del rostro suelen ser mayores que las variaciones de la imagen debido al cambio de identidad de un cara".

Con el objetivo de reducir el tamaño de la base de datos (inicialmente tiene un tamaño de unos 5 GB) para poder trabajar con mayor facilidad y rapidez, además de evitar que suceda el fenómeno de confusión comentado en el párrafo anterior, se ha hecho una selección de imágenes. Para llevar a cabo esta tarea se ha creado el programa *elimina.py*, con el cual se ha automatizado el proceso de borrado de ciertas imágenes debido a que manualmente era totalmente inviable.

De esta manera, una vez realizada la selección, se ha obtenido la base de datos final que se ha utilizado. Se trata de un total de 27 sujetos, cada uno con 9 poses faciales diferentes y bajo 7 condiciones de iluminación diferentes. Es decir, hay 63 imágenes por sujeto, obteniendo la suma de 1.701 imágenes en total (510 MB).

Por lo que se refiere a la nomenclatura de cada una de las imágenes, presenta la siguiente forma:

yaleBx_P0yA+z.pgm

Dónde: x es el número del sujeto, comprendido entre 11 y 39 (excepto 14 y 16)
y es la pose facial, comprendida entre 0 y 8.
z es la iluminación

Así, las 7 iluminaciones diferentes para la primera pose del sujeto 11, se identifican:

yaleB11_P00A+000E+00.pgm
yaleB11_P00A+000E+20.pgm
yaleB11_P00A+000E-20.pgm
yaleB11_P00A+000E-35.pgm
yaleB11_P00A+005E+10.pgm
yaleB11_P00A+005E-10.pgm
yaleB11_P00A+010E+00.pgm

## 5.3. Procedimiento y fases de desarrollo de la fase experimental

Una vez descritas todas las herramientas para la realización de este proyecto, se hablará de las fases de desarrollo de la fase experimental del proyecto. De esta manera, se describirá, de forma no muy extensa, cuál ha sido el procedimiento llevado a cabo.

1. Análisis de las diferentes librerías y herramientas que se pueden utilizar.
2. Instalación del software (Python 3.6 y PyCharm) y creación del entorno virtual.
3. Instalación de librerías. Los problemas principales que se han encontrado en esta fase han sido ocasionados por la compatibilidad entre las versiones de las librerías y la de Python. Es por esto que se ha utilizado Python 3.6 a pesar de no ser la última versión.
4. Documentación y programas de prueba. Se trata de una primera toma de contacto con los programas y funciones que se van a utilizar.
5. Selección de una base de datos y adecuación de la misma.
6. Programación de los programas de training y test de los 3 métodos. En el anexo A se puede observar el código utilizado para ello.
7. Realización de test y obtención de resultados y conclusiones.

Por lo que respecta a la última fase, la de de realización de test y obtención de resultados, se ha intentado realizar una cantidad de pruebas que permita observar el comportamiento de cada método bajo diferentes condiciones pero que a la vez no suponga un consumo excesivo de tiempo debido a que se trata de un recurso limitado en la realización de este proyecto. Siguiendo estas directrices, las pruebas que se han realizado se estructuran de la siguiente manera (Figura 23):

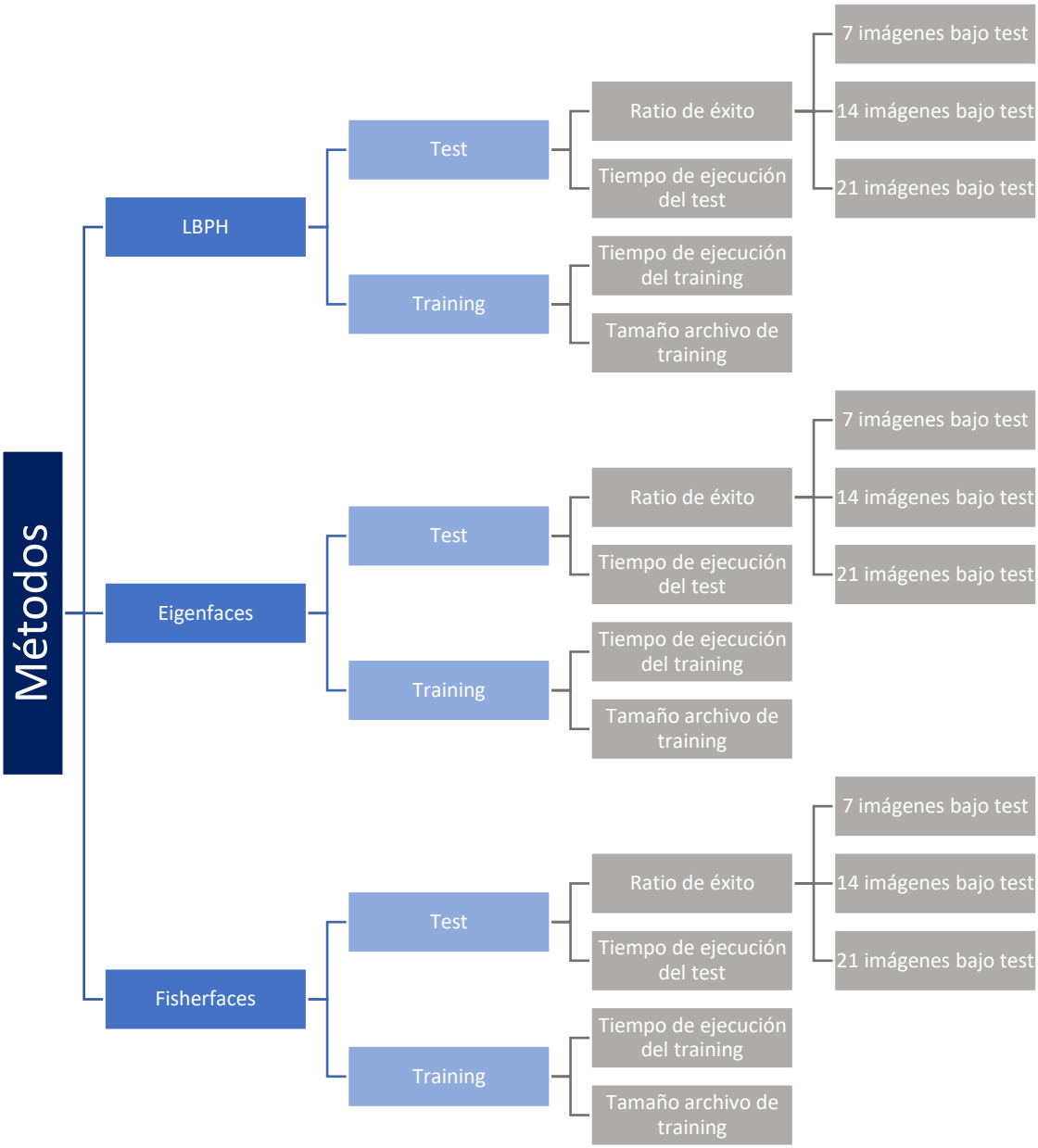


*Figura 23.- Procedimiento para la realización de pruebas en la fase experimental. Fuente: Propia*

Cabe añadir que todas estas pruebas se han realizado siendo cada elemento de estudio una variable dependiente del número de imágenes utilizadas en el training, siendo este último la variable independiente.

De esta manera, se ha analizado para cada uno de los tres métodos el ratio de éxito, el tiempo de ejecución del test, el tiempo de ejecución del training y el tamaño que ocupa el archivo de training.

Para el ratio de éxito, cuyo cálculo se basa en la relación entre el número de imágenes predichas correctamente y el número total de imágenes, se han realizado tres pruebas diferentes variando el número de imágenes que hay bajo test (imágenes nuevas de las que se predice la identidad) para ver cómo se comporta cada método ante estas variaciones. Además, para verificar y demostrar que no se trata de una tarea de reconocimiento sencilla para los algoritmos bajo estudio, en la Figura 24 se puede observar una muestra de imágenes escogidas al azar del sujeto 23, en las que se puede ver que tanto pose como iluminación sufren variaciones bastante notables. Con esto se podrá analizar la robustez de cada uno de los tres métodos abarcando una gran cantidad de situaciones diferentes en la toma de la imagen facial.

*Figura 24.- Imágenes escogidas al azar del sujeto número 23. Fuente: Propia*

## 5.4. Resultados experimentales

A continuación se procede a mostrar los resultados experimentales obtenidos siguiendo el modelo de la Figura 23. Es probable que alguna parte de los resultados experimentales que se presentan a continuación resulte repetitiva. El objetivo de ello es que los resultados que se han obtenido sean concluyentes y con una base sólida, o dicho de otra manera, se ha pretendido evitar que los resultados sean casuales o aleatorios. Todo esto se puede resumir con el concepto de repetibilidad, definido como la capacidad de obtener resultados consistentes al medir varias veces una variable bajo condiciones similares.

### 5.4.1. Método Eigenfaces

En primer lugar se exponen los resultados obtenidos para el método eigenfaces trabajando con 7 imágenes bajo test por cada sujeto (Tabla 5). En total se predicen 184 imágenes.

*Tabla 5.- Resultados para el método eigenfaces con 7 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_EigenFaces.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 102 | 9,07 | 18,09 | 42,39% | 6544 |
| **4** | 166 | 14,45 | 20,97 | 46,74% | 6864 |
| **7** | 314 | 28,36 | 27,6 | 51,09% | 6909 |
| **10** | 461 | 45,85 | 34,85 | 54,89% | 7071 |
| **15** | 694 | 81,14 | 46,23 | 59,24% | 7233 |
| **22** | 1010 | 147,58 | 63,34 | 66,85% | 7376 |
| **26** | 1200 | 188,71 | 72,27 | 67,39% | 7442 |
| **31** | 1450 | 261,07 | 84,45 | 62,50% | 7493 |
| **35** | 1650 | 335,48 | 95,56 | 63,59% | 7489 |
| **43** | 1980 | 495,67 | 111,71 | 63,04% | 7539 |
| **56** | 2590 | 869,29 | 147,18 | 63,04% | 7616 |

A partir de los datos obtenidos, se pueden graficar cada uno de los indicadores en función del número de imágenes bajo training.

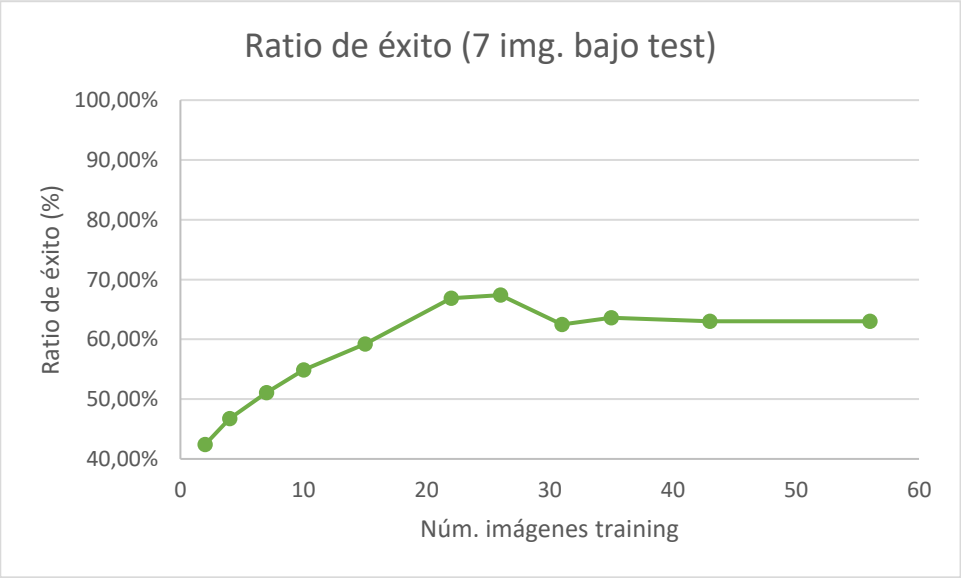


*Figura 25.- Ratio de éxito en función del número de imágenes en training. Método eigenfaces con 7 imágenes bajo test. Fuente: Propia*

Analizando este primer gráfico (Figura 25), se puede observar que a medida que aumenta el número de imágenes en el training, aumenta el ratio de éxito hasta llegar al máximo, situado en el 67,39%. A partir de este punto, la tasa de éxito empieza a disminuir a pesar de seguir aumentando el número de imágenes empleadas en el training. Esto se traduce en un caso de overfitting, que tal y como ya se ha explicado en capítulos previos, significa que a pesar de tener más información, no aumenta la tasa de éxito como sería de esperar, sino que más bien disminuye.

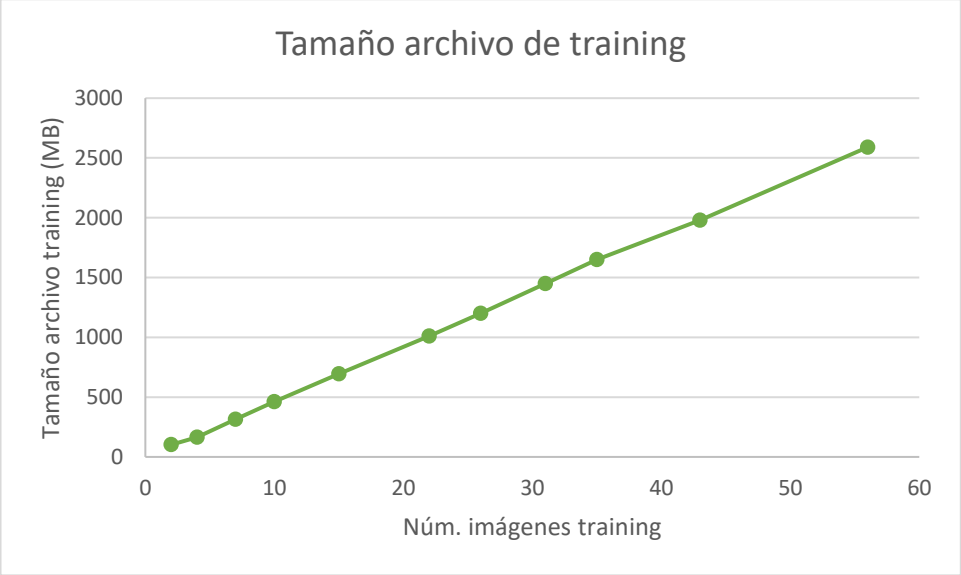


*Figura 26.- Tamaño del archivo de training en función del número de imágenes en training. Método eigenfaces con 7 imágenes bajo test. Fuente: Propia*

Por lo que respecta al tamaño del archivo de training (Figura 26), tal y como se puede observar, sigue una tendencia lineal prácticamente impecable. Cabe destacar el hecho de que cuando se trabaja con más de 50 imágenes por sujeto en el training, el espacio que ocupa supera los 2,5 GB, lo cual es una cantidad de recursos muy elevada a pesar de tratarse de un sencillo experimento práctico.


*Figura 27.- Tiempo de ejecución del training en función del número de imágenes en training. Método eigenfaces con 7 imágenes bajo test. Fuente: Propia*

El tiempo de ejecución que necesita la etapa de training, tal como se contempla en la Figura 27, presenta una tendencia ligeramente exponencial. Cuando se trabaja con 56 imágenes por sujeto en training, el tiempo de ejecución es de aproximadamente 870 segundos, es decir, de 14,5 minutos. Sabiendo que hay 27 sujetos, el total de imágenes es de 1512, por lo que realizando los cálculos pertinentes, se tiene que mediante el método eigenfaces, cada imagen tarda 0,57 segundos en procesarse. Mirando el otro extremo del gráfico, con tan solo 2 imágenes por sujeto en training, cada imagen tarda 0,17 segundos en procesarse. Por lo tanto, se observa que a medida que aumenta el número de imágenes en training, mayor es el tiempo de procesado de cada imagen.

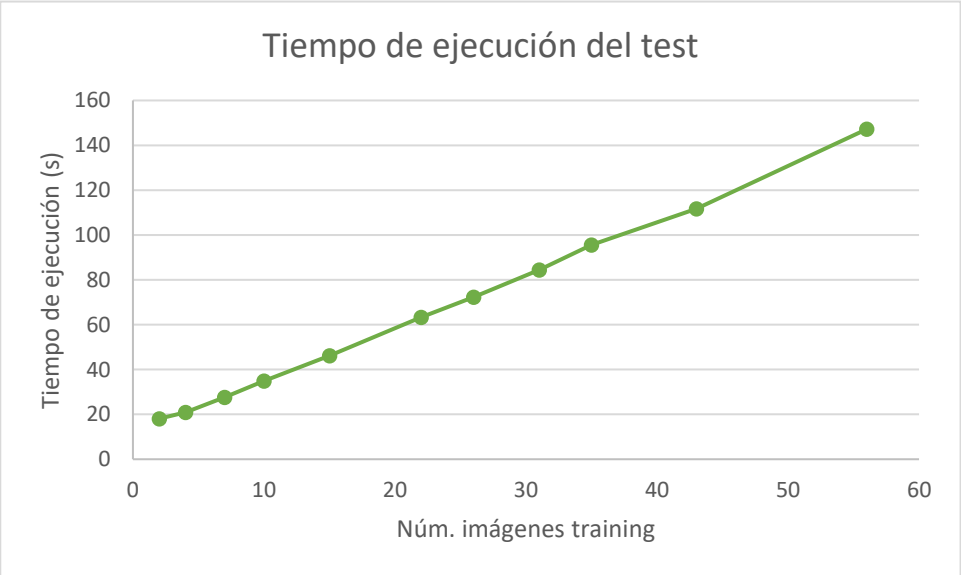


*Figura 28.- Tiempo de ejecución del test en función del número de imágenes en training. Método eigenfaces con 7 imágenes bajo test. Fuente: Propia*

En cambio, poniendo la vista sobre el tiempo de ejecución del test (Figura 28), se puede comprobar que al tener una evolución lineal, a pesar de que aumente el número de imágenes en training, el tiempo de detección o procesado individual de cada imagen se mantiene constante. Este tiempo es de aproximadamente 0,1 segundos por imagen.

Finalmente, antes de pasar al siguiente conjunto de resultados, es necesario hacer mención a la media de la distancia euclídea entre imágenes. Recordemos que, tal como se ha explicado en el capítulo 4, la identificación de cada imagen se realiza buscando la imagen de la base de datos que menor distancia euclídea presenta respecto a la imagen que está bajo test. Así, se ha calculado la media de esas distancias para cada número de imágenes en training y se ha utilizado como un indicador de precisión en el reconocimiento facial.

El comportamiento normal que debe presentar, a diferencia del que se ha obtenido para eigenfaces, se basa en la reducción de la media de distancia a medida que se tienen más imágenes en el training.

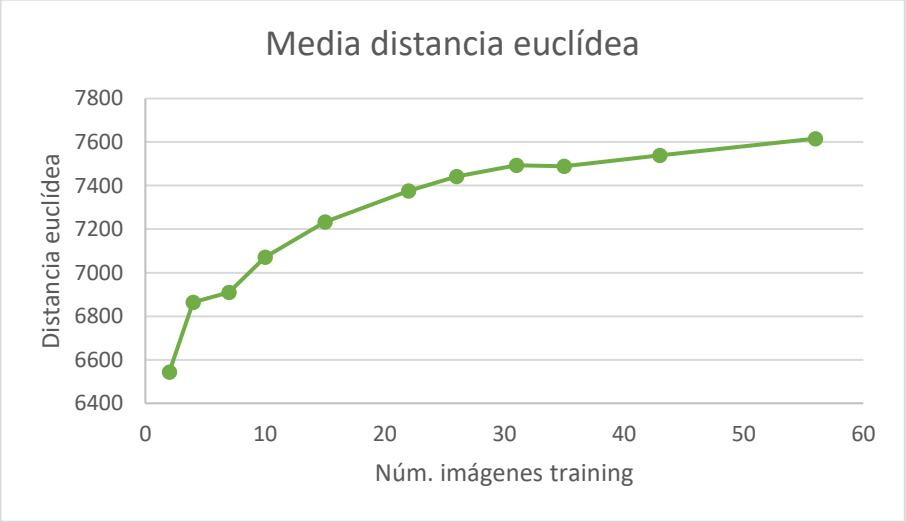


*Figura 29.- Media de la distancia euclídea en función del número de imágenes en training. Método eigenfaces con 7 imágenes bajo test. Fuente: Propia*

La media de la distancia euclídea debería disminuir al aumentar el porcentaje de aciertos, pero debido a que este porcentaje es tan bajo (en ningún caso se supera el 70% de éxitos), no tiene una influencia notable sobre la media. Lo único que se consigue es que cada vez que se aumenta el número de imágenes en el training aumente la distancia euclídea, mientras que el poco porcentaje de aciertos que aumenta no es suficiente para contrarrestar este efecto. Se han realizado pruebas con pocas imágenes, escogidas para obtener un ratio de éxito elevado a propósito y, sorprendentemente, este comportamiento se normaliza cuando las tasas de éxito se sitúan entorno del 90%.

Siguiendo con el método eigenfaces, a continuación, en la Tabla 6, se exponen los resultados obtenidos con 14 imágenes por sujeto bajo test. En total se predicen 357 imágenes.

*Tabla 6.- Resultados para el método eigenfaces con 14 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_EigenFaces.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 106 | 11,55 | 39,45 | 55,18% | 6096 |
| **4** | 174 | 13,42 | 33,3 | 58,82% | 6436 |
| **7** | 329 | 26,37 | 42,72 | 60,50% | 6741 |
| **10** | 477 | 42,1 | 52,52 | 65,55% | 6978 |
| **15** | 719 | 75,33 | 86,05 | 67,23% | 7163 |
| **18** | 864 | 118,88 | 96,19 | 69,47% | 7267 |
| **22** | 1040 | 143,52 | 91,17 | 67,51% | 7330 |
| **26** | 1230 | 184,54 | 100,36 | 67,23% | 7372 |
| **30** | 1430 | 237,85 | 111,34 | 67,23% | 7394 |
| **35** | 1620 | 311,69 | 119,77 | 67,23% | 7441 |

Debido a que los tiempos de ejecución y el tamaño del archivo de training presentan comportamientos similares a pesar de que varíe el número de imágenes bajo test, solamente se expondrá el resultado graficado para el ratio de éxito.

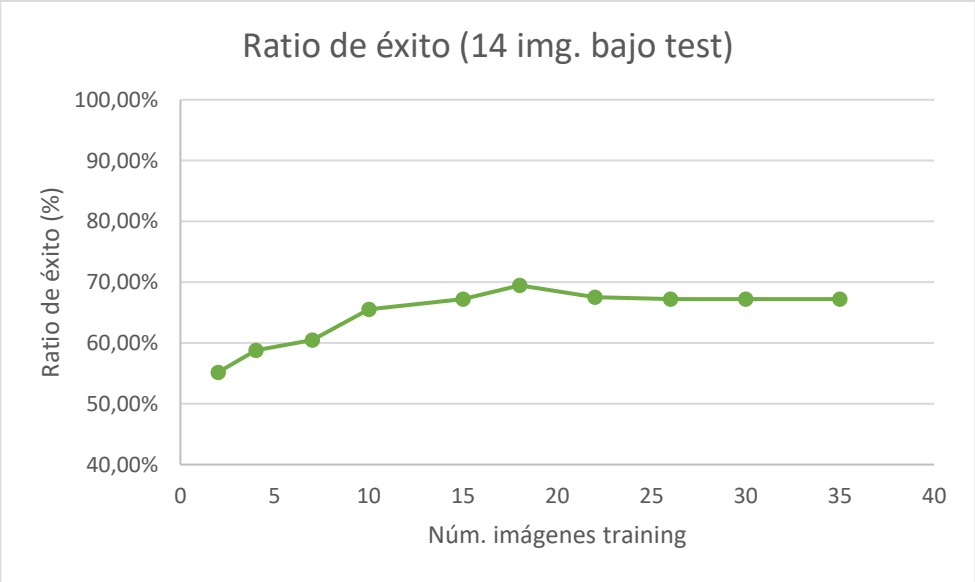


*Figura 30.- Ratio de éxito en función del número de imágenes en training. Método eigenfaces con 14 imágenes bajo test. Fuente: Propia*

Al aumentar el número de imágenes bajo test (Figura 30), el ratio de éxito ha aumentado ligeramente, pero sin cambios notables. Lo más destacable es que al trabajar con dos imágenes en training en ambos casos, el ratio de éxito ha pasado del 42,39% al 55,18%. El pico máximo se mantiene alrededor del 70%. El comportamiento general no varía, manteniéndose el efecto del overfitting.

Para finalizar con el método eigenfaces, se exponen en la Tabla 7 los resultados obtenidos con 21 imágenes por sujeto bajo test. En total se predicen 532 imágenes.

*Tabla 7.- Resultados para el método eigenfaces con 21 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_EigenFaces.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 104 | 7,55 | 44,22 | 52,44% | 6426 |
| **4** | 181 | 13,92 | 49,23 | 54,70% | 6513 |
| **7** | 389 | 33,31 | 64,55 | 59,59% | 6748 |
| **10** | 495 | 44 | 72,25 | 62,03% | 6887 |
| **15** | 741 | 79,15 | 90,8 | 61,46% | 6998 |
| **18** | 884 | 104,42 | 103,59 | 61,46% | 7059 |
| **22** | 1060 | 144,17 | 116,69 | 64,47% | 7042 |
| **26** | 1200 | 175,31 | 126,38 | 66,92% | 7040 |
| **30** | 1390 | 223,42 | 141,95 | 67,67% | 7099 |
| **35** | 1610 | 301,07 | 157,15 | 68,23% | 7152 |
| **43** | 1950 | 419,47 | 280,26 | 68,42% | 7221 |

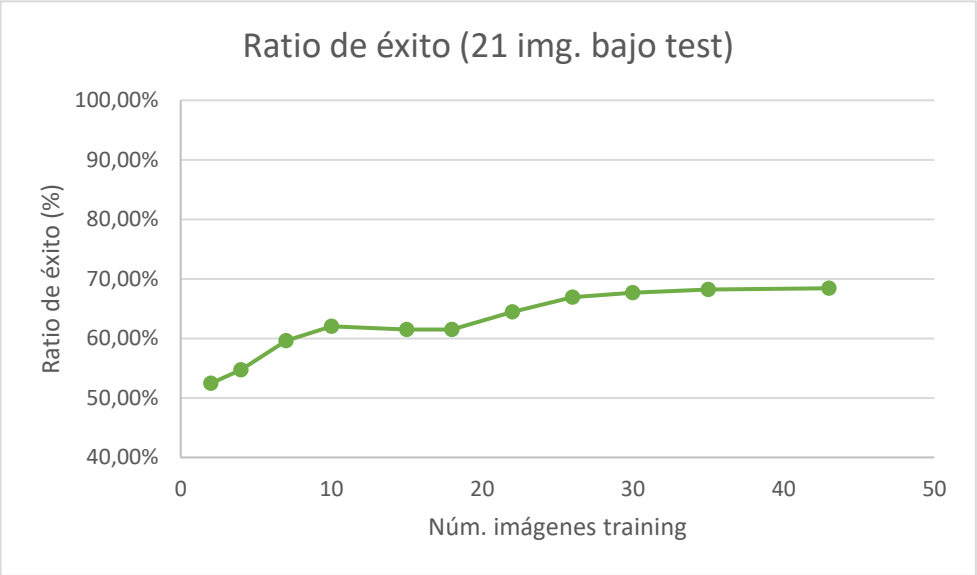


*Figura 31.- Ratio de éxito en función del número de imágenes en training. Método eigenfaces con 21 imágenes bajo test. Fuente: Propia*

Por lo que respecta al comportamiento del porcentaje de éxitos cuando hay 21 imágenes bajo test (Figura 31), apenas varía de los otros dos casos. La única diferencia es que no se puede llegar a apreciar el efecto del overfitting debido a que serían necesarias más imágenes en el training para poderlo observar. A pesar de esto, se puede comprobar que hay una zona, alrededor del 68%, que presenta claros indicios de saturación ya que la curva se estabiliza casi por completo.

### 5.4.2. Método Fisherfaces

Pasando al método fisherfaces, se tratarán en primer lugar los resultados obtenidos trabajando con 7 imágenes bajo test por cada sujeto (Tabla 8). En total se predicen 184 imágenes.

*Tabla 8.- Resultados para el método fisherfaces con 7 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_FisherFaces.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 48,2 | 9,01 | 18,33 | 50,00% | 4589 |
| **4** | 48,3 | 14,31 | 17,68 | 76,63% | 2957 |
| **7** | 48,3 | 26,07 | 18,81 | 82,07% | 2309 |
| **10** | 48,4 | 39,13 | 21,19 | 91,30% | 1505 |
| **15** | 48,5 | 63,8 | 20,63 | 95,65% | 1196 |
| **22** | 48,7 | 92,38 | 17,68 | 97,28% | 918 |
| **26** | 48,8 | 119,22 | 17,58 | 97,28% | 793 |
| **31** | 48,9 | 164,27 | 17,15 | 96,74% | 626 |
| **35** | 49 | 207,8 | 17,48 | 96,74% | 580 |
| **43** | 49,1 | 330,64 | 17,59 | 96,20% | 502 |
| **56** | 49,4 | 685,24 | 17,55 | 95,11% | 420 |

Procediendo de manera equivalente al método anterior, a partir de los datos obtenidos, se pueden graficar cada uno de los indicadores en función del número de imágenes bajo training.

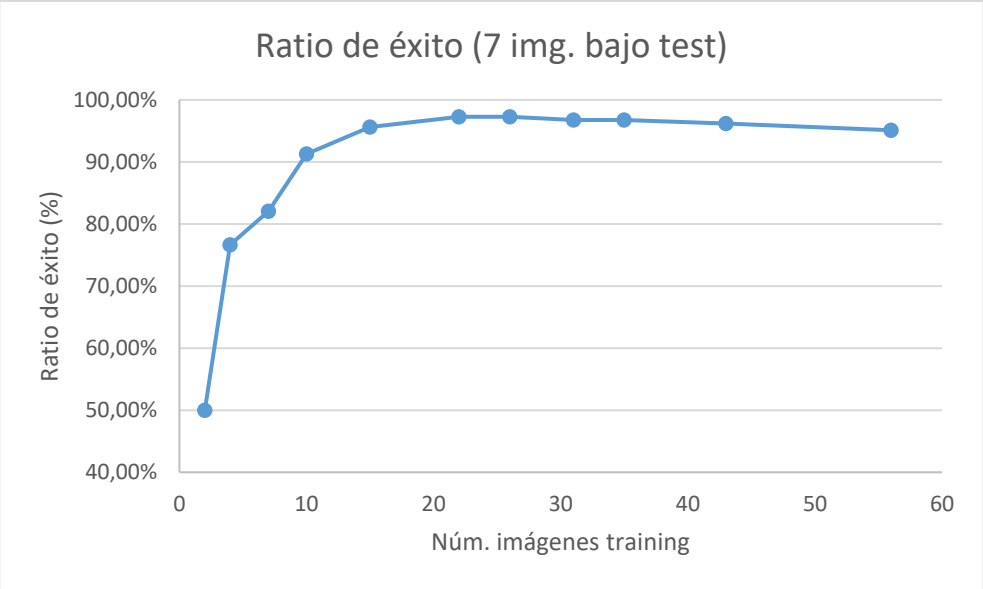


*Figura 32.- Ratio de éxito en función del número de imágenes en training. Método fisherfaces con 7 imágenes bajo test. Fuente: Propia*

En este caso (Figura 32), se puede observar un cambio de tendencia radical por lo que respecta al ratio de éxito si se compara con el método eigenfaces. El valor máximo obtenido es del 97,28%, lo que supone una precisión muy elevada si se habla de tasa de aciertos en la identificación facial. Además, si se recuerda la tasa de éxito máxima obtenida para el método eigenfaces, su valor nunca supera el 70%. El aumento de la precisión es de más del 27%, tomando siempre como referencia que los resultados son obtenidos utilizando la misma base de datos y el mismo conjunto de imágenes.

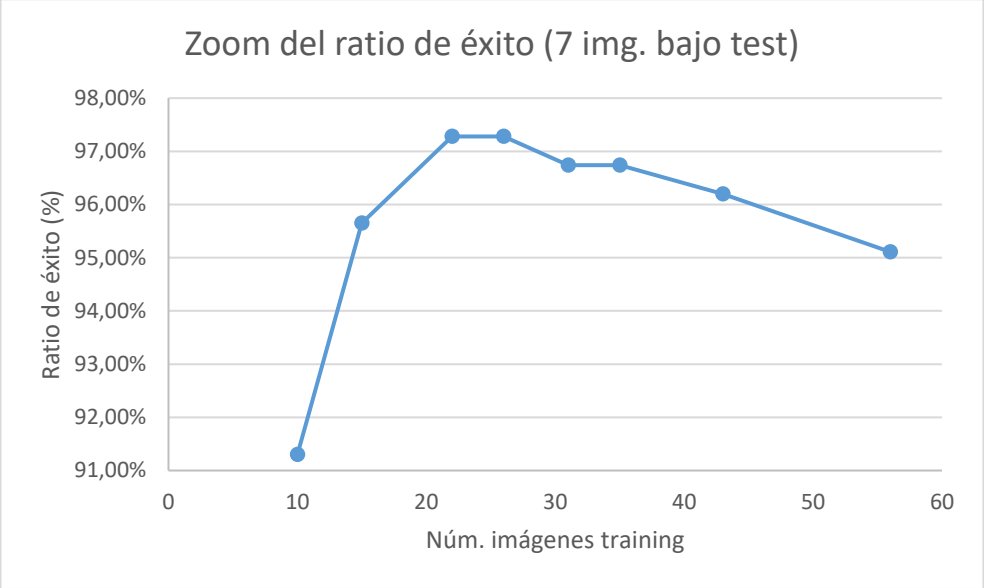


*Figura 33.- Zoom: Ratio de éxito en función del número de imágenes en training. Método fisherfaces con 7 imágenes bajo test. Fuente: Propia*

Cabe destacar, pero, que con este método, al trabajar con pocas imágenes se produce un underfitting muy pronunciado. Con 2 imágenes en training el porcentaje de aciertos es del 50%, mientras que al aumentar a 10 imágenes ya se obtiene un ratio de éxito que ronda el 90%. Por lo que respecta al overfitting, es importante notar que también se encuentra presente, pero no de forma tan pronunciada. Si se hace un zoom al gráfico se puede apreciar mejor. Así, observando la Figura 33, se puede observar como a partir de 26 imágenes en training, el ratio de éxito empieza a disminuir hasta llegar al 95,11% para 56 imágenes.

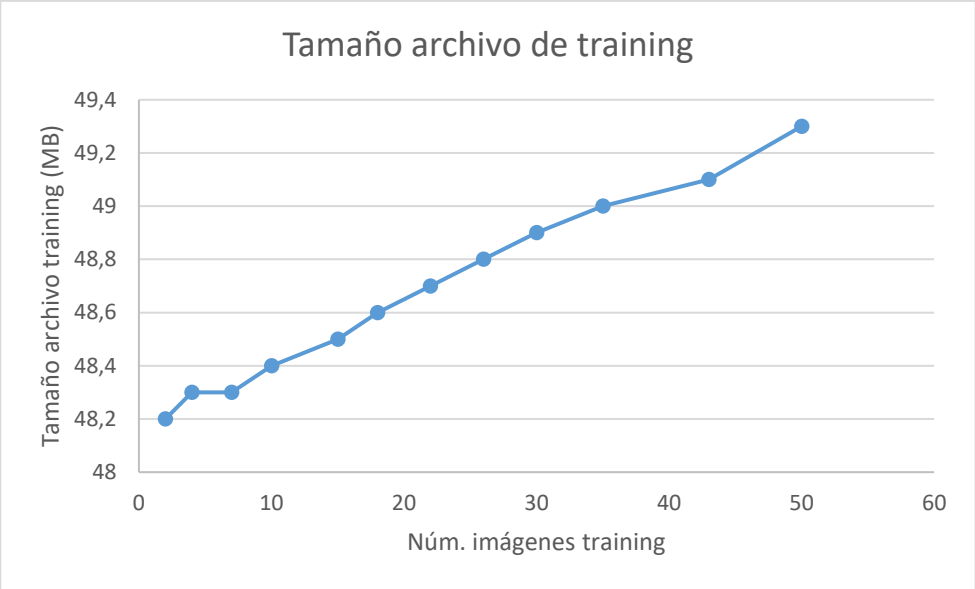


*Figura 34.- Tamaño del archivo de training en función del número de imágenes en training. Método fisherfaces con 7 imágenes bajo test. Fuente: Propia*

En la Figura 34 se grafica cómo evoluciona el tamaño del archivo de training en función del número de imágenes en training. Se observa una tendencia lineal, aunque con una pendiente muy baja, debido a que el tamaño solamente varía entre 48,2 MB y 49,4 MB. Se trata de un método altamente eficiente por lo que respecta al consumo de memoria, y más si se compara con eigenfaces, el cual llegaba a consumir 2,5 GB.

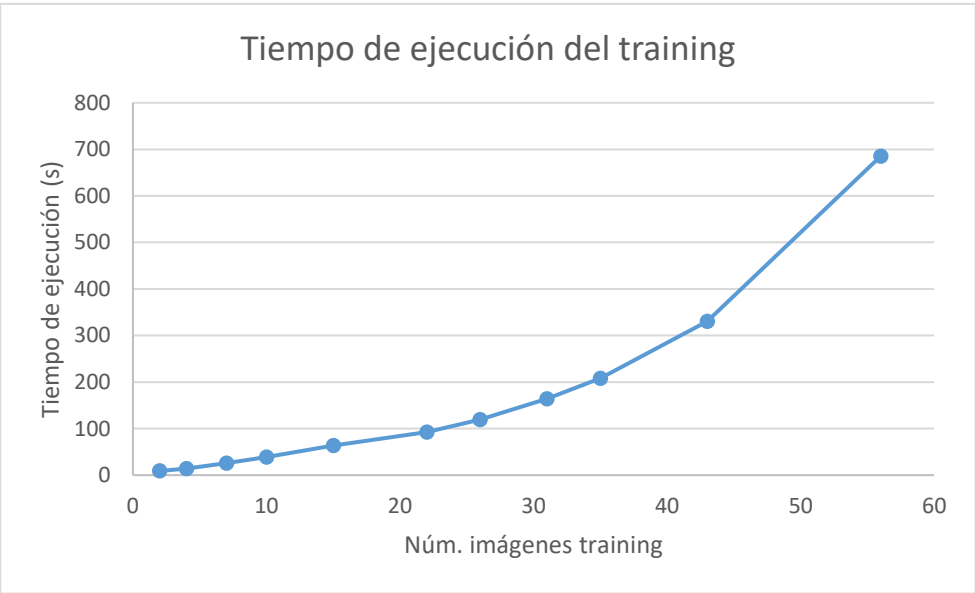


*Figura 35.- Tiempo de ejecución del training en función del número de imágenes en training. Método fisherfaces con 7 imágenes bajo test. Fuente: Propia*

El tiempo de ejecución del training (Figura 35), igual que en el caso anterior, presenta una tendencia aproximadamente exponencial. El tiempo de ejecución con 56 imágenes por sujeto en el training es de 685 segundos, lo que se traduce en 11,42 minutos. Así, el tiempo de procesado para cada imagen es de 0,45 segundos, mientras que cuando se trabaja con solamente dos imágenes en training, el tiempo de procesado cae a 0,17 segundos por imagen.

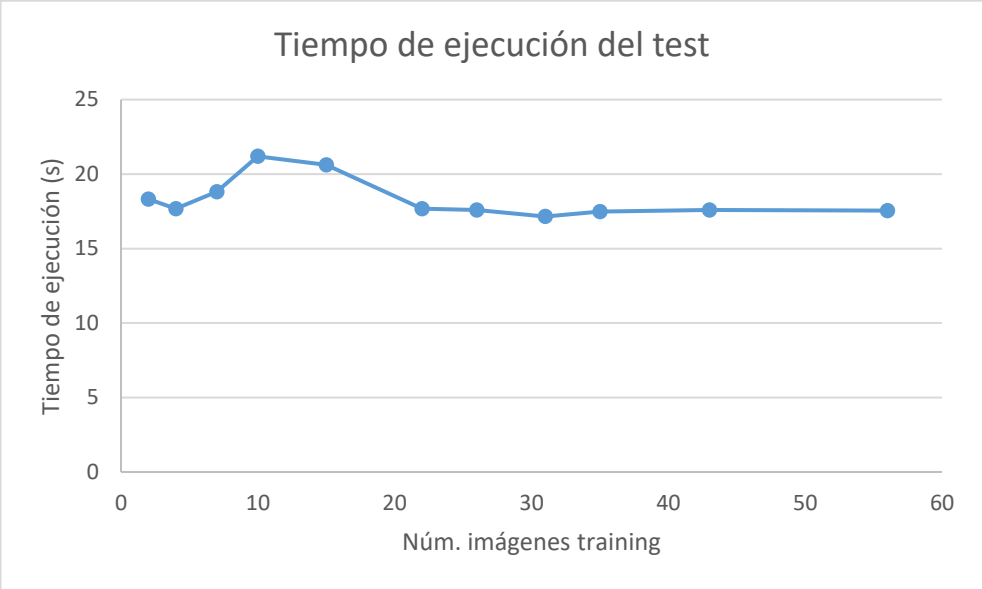


*Figura 36.- Tiempo de ejecución del test en función del número de imágenes en training. Método fisherfaces con 7 imágenes bajo test. Fuente: Propia*

Por lo que respecta al tiempo de ejecución del test (Figura 36), en este caso presenta un comportamiento lineal y con pendiente 0. Esto significa que el tiempo de ejecución del test es constante y no depende del número de imágenes en el training. Así, el tiempo medio de ejecución es de 18,33 segundos para 7 imágenes bajo test. Se concluye que para el método fisherfaces el tiempo de ejecución del test solamente depende de la cantidad de imágenes que hay bajo test, a diferencia del método eigenfaces. Para 14 imágenes bajo test el tiempo medio de ejecución es de 32,67 segundos, mientras que para 21 imágenes este tiempo es de 40,83 segundos.

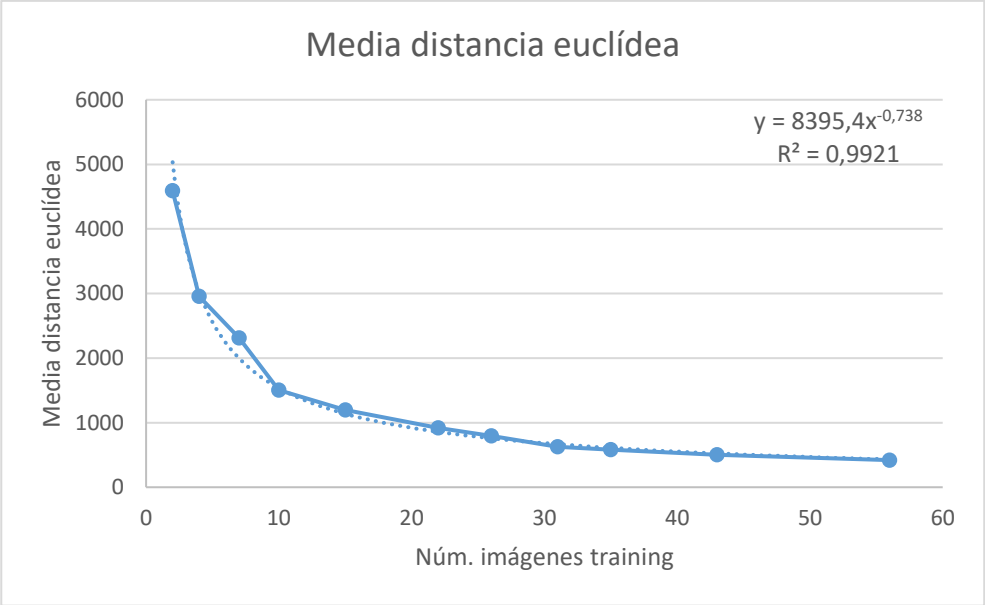


*Figura 37.- Media de la distancia euclídea en función del número de imágenes en training. Método fisherfaces con 7 imágenes bajo test. Arriba a la derecha se presenta la ecuación de la línea de tendencia que mejor se ajusta a la curva, en la que **y** representa la media de la distancia euclídea, mientras que **x** es el número de imágenes bajo training. Se presenta también su coeficiente de determinación. Fuente: Propia*

Para el método fisherfaces, la media de la distancia euclídea (Figura 37) sí que presenta un comportamiento normal. A medida que aumenta el número de imágenes bajo training, la distancia disminuye de forma potencial. Si bien podría tratarse de una regresión exponencial, en este caso la forma potencial es la que mejor se ajusta a la curva, presentando un coeficiente de determinación de 0,9921 (frente al valor de 0,8684 en el caso de la regresión exponencial). A grandes rasgos, esto significa que cuantas más imágenes hay en la base de datos como referencia, más confianza existe en la predicción.

Siguiendo con el método fisherfaces, en la Tabla 9 se exponen los resultados obtenidos con 14 imágenes por sujeto bajo test. En total se predicen 357 imágenes.

De igual manera al método anterior, solamente se procederá a graficar el ratio de éxito debido a que el resto de indicadores, aunque presentan variaciones (se pueden ver en la Tabla 9), poseen el mismo comportamiento general.

*Tabla 9.- Resultados para el método fisherfaces con 14 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_FisherFaces.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 48,2 | 8,12 | 41,86 | 56,86% | 4356 |
| **4** | 48,3 | 14,48 | 38,76 | 65,27% | 2854 |
| **7** | 48,3 | 26,08 | 35,97 | 69,19% | 2283 |
| **10** | 48,4 | 37,2 | 41,27 | 76,47% | 1829 |
| **15** | 48,5 | 61,85 | 37,1 | 81,79% | 1448 |
| **18** | 48,6 | 67,82 | 31,12 | 81,23% | 1271 |
| **22** | 48,7 | 90,39 | 28,23 | 84,87% | 1049 |
| **26** | 48,8 | 104,28 | 27,68 | 85,43% | 908 |
| **30** | 48,9 | 137,84 | 27,45 | 85,99% | 816 |
| **35** | 49 | 175,27 | 27,37 | 87,39% | 753 |
| **43** | 49,1 | 293,79 | 27,73 | 85,15% | 628 |
| **50** | 49,3 | 426,8 | 27,47 | 82,91% | 566 |

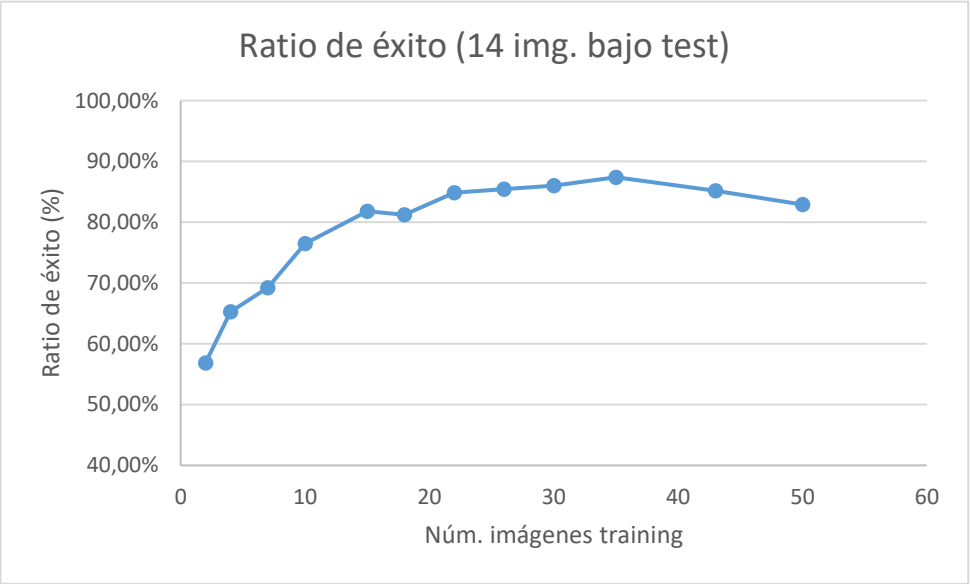


*Figura 38.- Ratio de éxito en función del número de imágenes en training. Método fisherfaces con 14 imágenes bajo test. Fuente: Propia*

Al duplicar el número de imágenes bajo test (Figura 38) se observa que el ratio de éxito máximo obtenido disminuye ligeramente, pasando de estar en un 97,28% cuando se trabaja con 7 imágenes en el test a un 87,39%. Por lo que respecta al resto, el comportamiento general resulta equivalente, teniendo lugar también los fenómenos de underfitting y overfitting, los cuales se pueden apreciar con más detalle en la Figura 39.

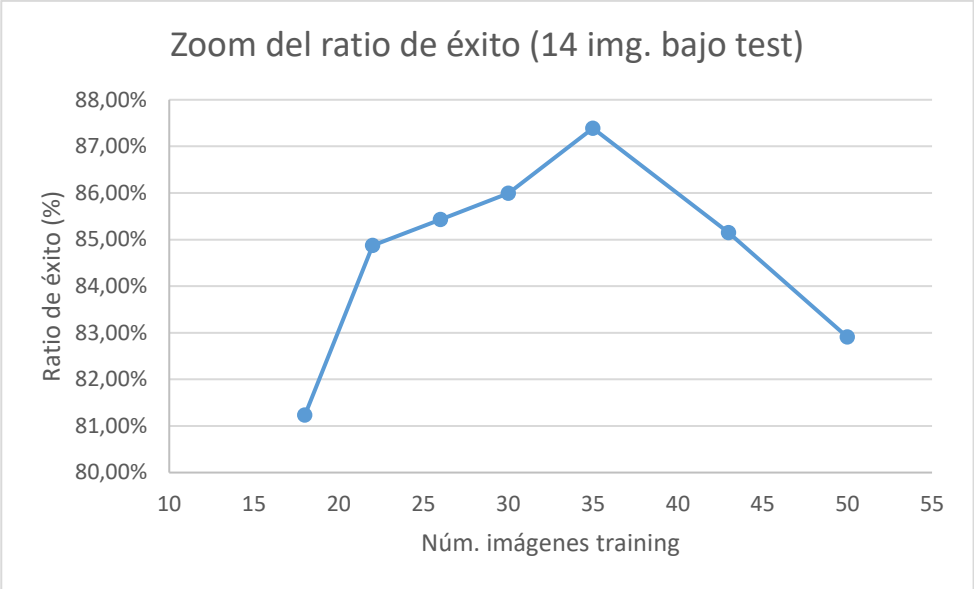


*Figura 39.- Zoom: Ratio de éxito en función del número de imágenes en training. Método fisherfaces con 14 imágenes bajo test. Fuente: Propia*

Para finalizar con el método fisherfaces, se exponen a continuación (Tabla 10) los resultados obtenidos con 21 imágenes por sujeto bajo test. En total se predicen 532 imágenes.

*Tabla 10.- Resultados para el método fisherfaces con 21 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_FisherFaces.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media valor estadístico |
|---|---|---|---|---|---|
| **2** | 48,2 | 6,84 | 44,89 | 55,83% | 4737 |
| **4** | 48,3 | 12,06 | 40,47 | 69,36% | 2811 |
| **7** | 48,4 | 21 | 40,04 | 81,20% | 1947 |
| **10** | 48,4 | 27,78 | 41,06 | 79,32% | 1767 |
| **15** | 48,5 | 46,26 | 41,38 | 81,39% | 1366 |
| **18** | 48,6 | 59,23 | 40,03 | 84,21% | 1179 |
| **22** | 48,7 | 83,41 | 40,03 | 87,41% | 999 |
| **26** | 48,8 | 101,78 | 39,55 | 87,59% | 879 |
| **30** | 48,9 | 131,25 | 40,27 | 87,78% | 780 |
| **35** | 49 | 181,97 | 40,08 | 86,09% | 705 |
| **43** | 49,1 | 290,05 | 41,33 | 86,09% | 606 |

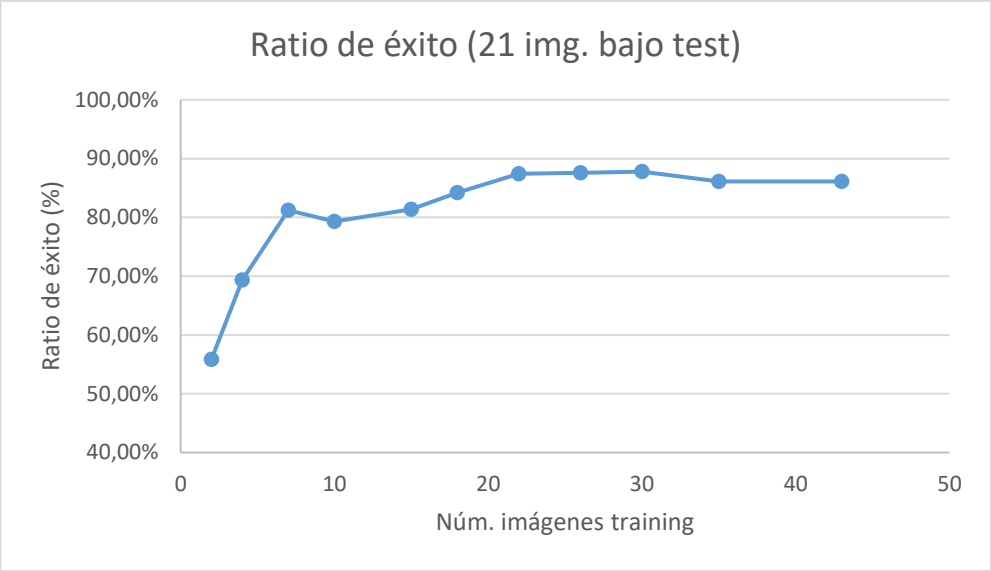


*Figura 40.- Ratio de éxito en función del número de imágenes en training. Método fisherfaces con 21 imágenes bajo test. Fuente: Propia*

El comportamiento de la evolución del ratio de éxito en función del número de imágenes bajo training es equivalente al de los otros dos casos del método fisherfaces. Asimismo, tanto el fenómeno de underfitting como el de overfitting siguen estando presentes.

### 5.4.3. Método LBPH

Se procede a mostrar los resultados obtenidos para el método LBPH. En primer lugar se exponen los resultados obtenidos trabajando con 7 imágenes bajo test por cada sujeto. Se predicen 184 imágenes.

*Tabla 11.- Resultados para el método LBPH con 7 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_LBPH.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 10 | 5,03 | 16 | 83,15% | 34,85 |
| **4** | 17,5 | 9,29 | 17,3 | 83,70% | 33,77 |
| **7** | 32,6 | 16,55 | 19,05 | 90,22% | 32,4 |
| **10** | 47,7 | 23,54 | 21,12 | 91,85% | 32,19 |
| **15** | 70,9 | 35,65 | 24,08 | 92,30% | 31,93 |
| **18** | 84,6 | 41,85 | 26,04 | 92,93% | 31,82 |
| **22** | 105 | 52,38 | 28,65 | 94,02% | 31,73 |
| **26** | 124 | 61,3 | 30,93 | 94,56% | 31,51 |
| **30** | 144 | 68,99 | 33,91 | 95,11% | 31,13 |
| **31** | 150 | 72,84 | 34,24 | 94,02% | 30,92 |
| **35** | 169 | 81,03 | 37,75 | 93,47% | 30,63 |
| **43** | 205 | 99,29 | 41,8 | 93,47% | 30,63 |
| **56** | 268 | 129,47 | 50 | 93,47% | 30,63 |

Tal y como se ha procedido previamente, a partir de los datos obtenidos, se pueden graficar cada uno de los indicadores en función del número de imágenes bajo training.

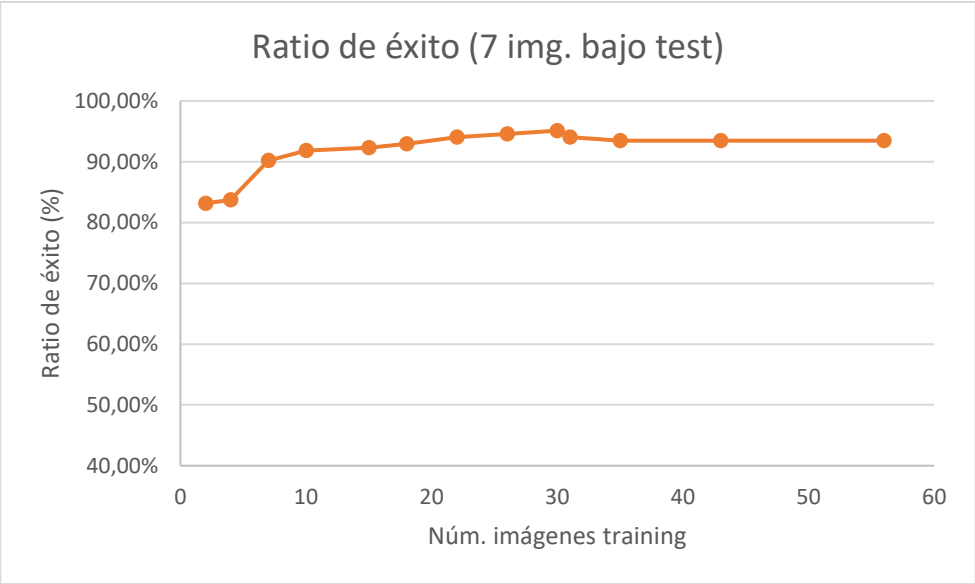


*Figura 41.- Ratio de éxito en función del número de imágenes en training. Método LBPH con 7 imágenes bajo test. Fuente: Propia*

En general, por lo que respecta al ratio de éxito del método LBPH (Figura 41), se observa un buen rendimiento. El pico máximo se sitúa en el 95,11%, mientras que el valor mínimo se encuentra a solamente 12 puntos por debajo, es decir, en el 83,15%. Esto significa que el rango de valores en los que se mueve el ratio de éxito es bastante estrecho, con el consiguiente de que incluso con 2 imágenes en training se obtengan buenos resultados. De esta manera, el fenómeno de underfitting es menos pronunciado que para el método fisherfaces. Se aprecia también la presencia de un overfitting, ya que a partir de 30 imágenes en training, el rendimiento empieza a disminuir.

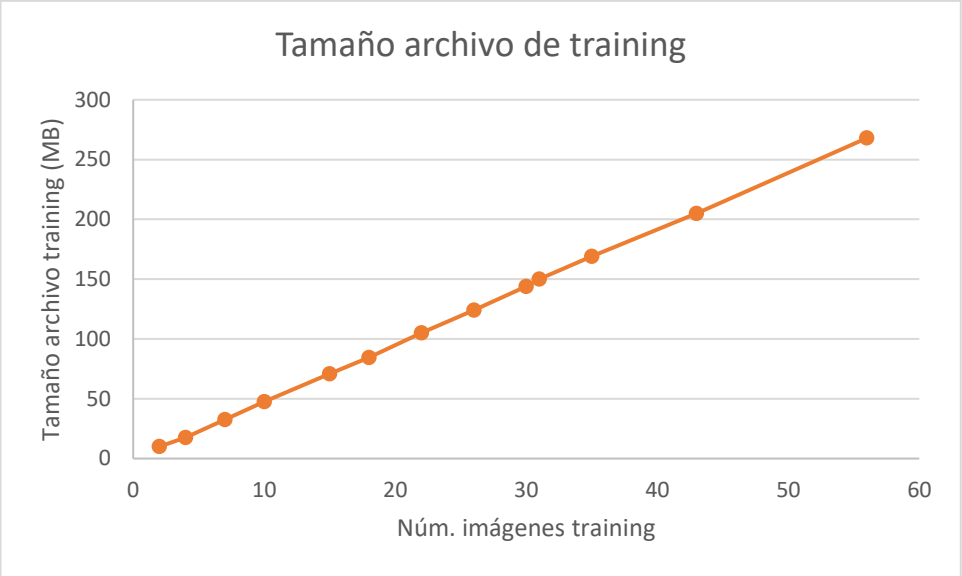


*Figura 42.- Tamaño del archivo de training en función del número de imágenes en training. Método LBPH con 7 imágenes bajo test. Fuente: Propia*

El tamaño del archivo de training (Figura 42) aumenta de manera lineal a medida que crece el número de imágenes bajo training. Para 56 imágenes por sujeto se ocupa un tamaño de 268 MB, o lo que es igual, cada imagen ocupa 0,18 MB relativos al almacenamiento de su extracción de características.

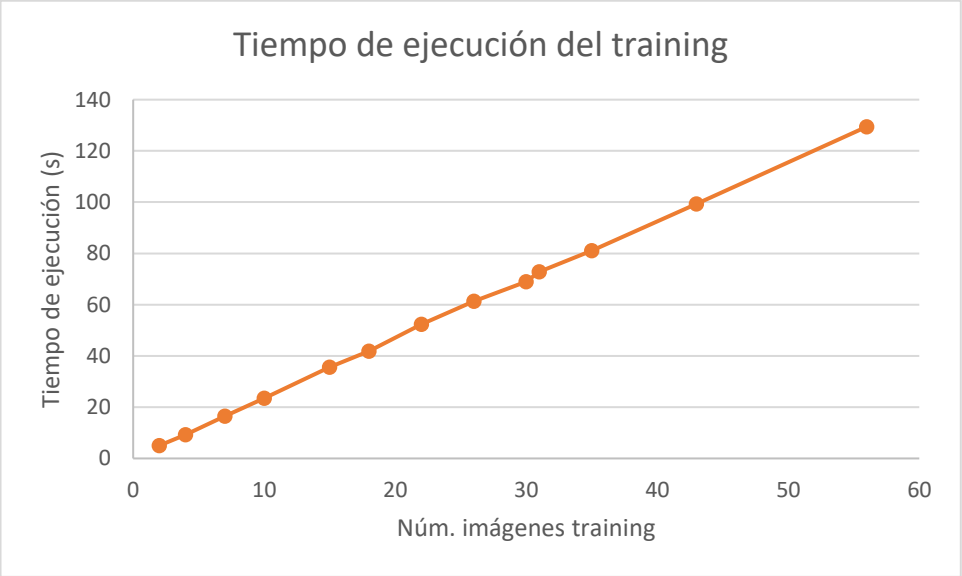


*Figura 43.- Tiempo de ejecución del training en función del número de imágenes en training. Método LBPH con 7 imágenes bajo test. Fuente: Propia*

El tiempo de ejecución del training (Figura 43), a diferencia de los dos métodos anteriores, presenta una tendencia lineal. Así, el tiempo de ejecución con 56 imágenes por sujeto en el training es de 129

segundos, lo que se traduce en 2,15 minutos. De esta manera, realizando los cálculos pertinentes, el tiempo de procesado y extracción de características para cada imagen es de 0,09 segundos.

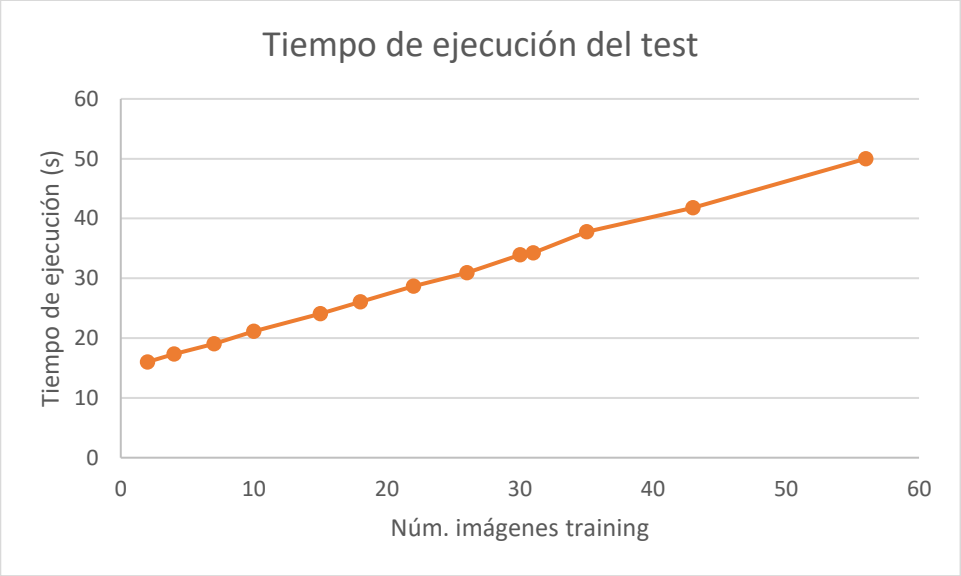


*Figura 44.- Tiempo de ejecución del test en función del número de imágenes en training. Método LBPH con 7 imágenes bajo test. Fuente: Propia*

Analizando el tiempo de ejecución del test (Figura 44), se puede comprobar que presenta una evolución lineal. El tiempo de detección o procesado individual de cada imagen se mantiene constante, el cual es de aproximadamente 0,03 segundos por imagen.

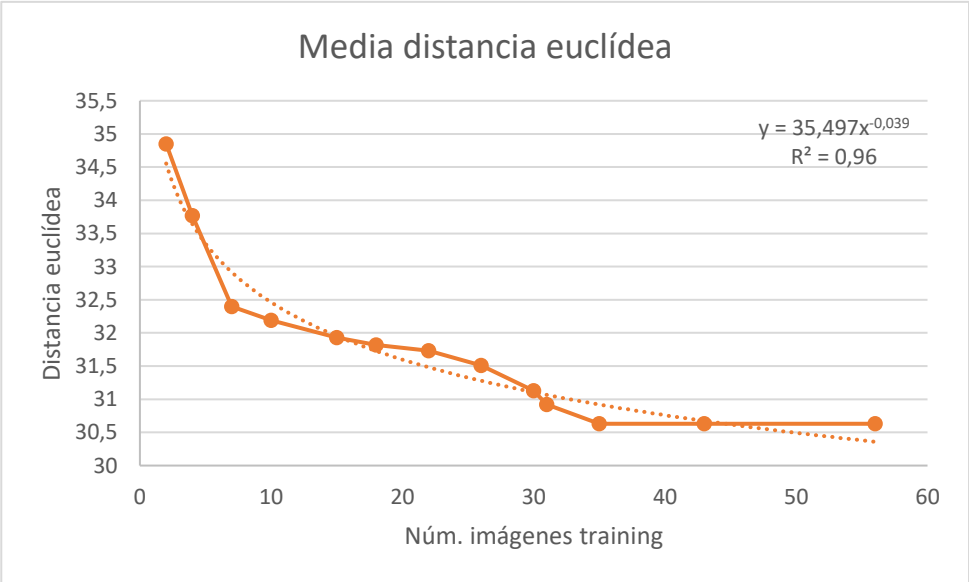


*Figura 45.- Media de la distancia euclídea en función del número de imágenes en training. Método LBPH con 7 imágenes bajo test. Arriba a la derecha se presenta la ecuación de la línea de tendencia que mejor se ajusta a la curva, en la que **y** representa la media de la distancia euclídea, mientras que **x** es el número de imágenes bajo training. Se presenta también su coeficiente de determinación. Fuente: Propia*

La media de la distancia euclídea (Figura 45) presenta una tendencia potencial descendiente respecto al número de imágenes en training (igual que en el método fisherfaces). Por lo tanto, se trata de un comportamiento normal, debido a que la distancia, utilizada como indicador de confianza, disminuye (la confianza aumenta) cuanta más información se tiene de cada sujeto.

Siguiendo con el siguiente conjunto de datos del método LBPH, a continuación se exponen los resultados obtenidos con 14 imágenes por sujeto bajo test (Tabla 12). En total se predicen 357 imágenes.

*Tabla 12.- Resultados para el método LBPH con 14 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_LBPH.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 10,3 | 4,7 | 30,41 | 80,39% | 36,83 |
| **4** | 17,5 | 8,85 | 30,58 | 82,07% | 35,42 |
| **7** | 33,1 | 15,4 | 33,88 | 83,19% | 35,11 |
| **10** | 48,3 | 22,41 | 37,2 | 84,31% | 34,97 |
| **15** | 72,6 | 33,3 | 41,7 | 87,68% | 34,71 |
| **18** | 87,1 | 38,05 | 44,15 | 89,64% | 34,41 |
| **22** | 107 | 47,1 | 49,51 | 89,35% | 33,82 |
| **26** | 127 | 55,37 | 51,95 | 85,99% | 33,3 |
| **30** | 147 | 68,88 | 57,27 | 86,83% | 33,17 |
| **35** | 167 | 76,07 | 59,62 | 87,11% | 33,17 |
| **50** | 236 | 121,58 | 72,15 | 87,11% | 33,17 |

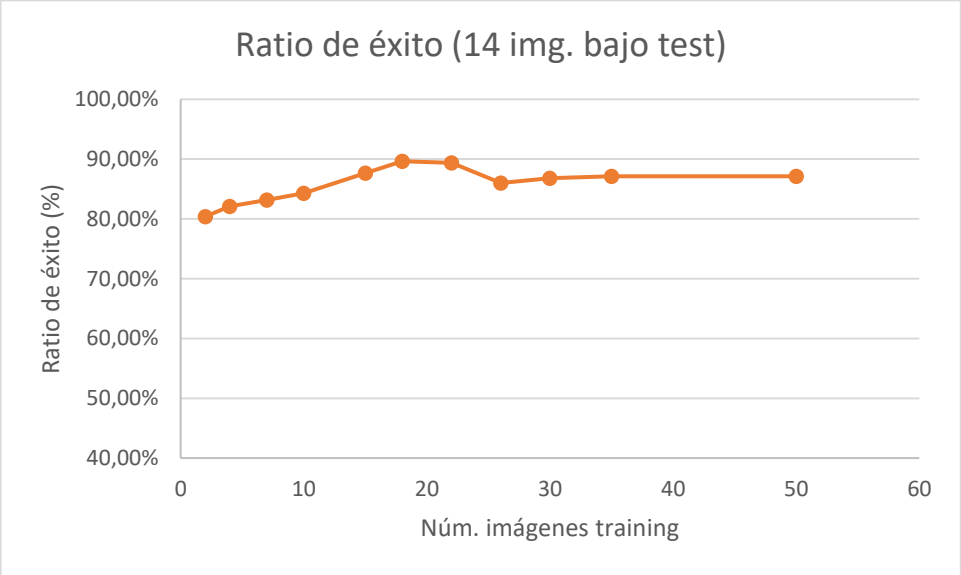


*Figura 46.- Ratio de éxito en función del número de imágenes en training. Método LBPH con 14 imágenes bajo test. Fuente: Propia*

Al aumentar la cantidad de imágenes bajo test (Figura 46), el ratio de éxito ha disminuido ligeramente respecto a los resultados anteriores. El pico máximo se sitúa en torno al 90%, mientras que el mínimo se encuentra alrededor del 80%. Siguiendo la tendencia del conjunto de datos anterior, el underfitting sigue sin ser muy pronunciado, mientras que, de manera bastante disruptiva, el overfitting que se ha generado resulta bastante considerable. Tanto es así, que, tomando el pico máximo como referencia, se disminuye la eficiencia de aciertos en casi la mitad, situándose sobre el 86%.

Para finalizar con el método LBPH, se expondrán en la Tabla 13 los resultados obtenidos con 21 imágenes por sujeto bajo test. En total se predicen 532 imágenes.

*Tabla 13.- Resultados para el método LBPH con 21 imágenes bajo test*

| Imágenes en training | Tamaño del archivo trainer_LBPH.yml (MB) | Tiempo de ejecución training (s) | Tiempo de ejecución test (s) | Ratio de éxito | Media distancia euclídea |
|---|---|---|---|---|---|
| **2** | 10,3 | 4,7 | 45,05 | 71,05% | 38,99 |
| **4** | 17,5 | 8,85 | 45,75 | 79,13% | 36,68 |
| **7** | 33,1 | 15,4 | 53,45 | 78,57% | 35,7 |
| **10** | 48,3 | 22,41 | 54,46 | 80,45% | 35,24 |
| **15** | 72,6 | 33,3 | 61,67 | 83,46% | 34,38 |
| **18** | 87,1 | 38,05 | 64,93 | 82,71% | 33,86 |
| **22** | 107 | 47,1 | 70,67 | 83,83% | 33,59 |
| **26** | 127 | 55,37 | 74,88 | 85,15% | 33,43 |
| **30** | 147 | 68,88 | 81,03 | 84,96% | 33,38 |
| **35** | 167 | 76,07 | 85,36 | 84,96% | 33,38 |
| **43** | 202 | 91,3 | 94,76 | 84,96% | 33,38 |

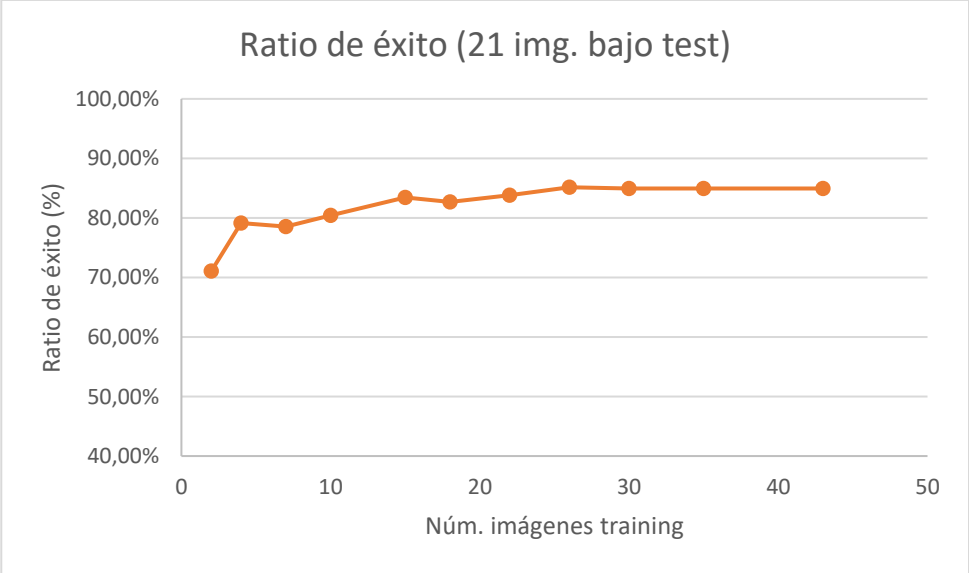


*Figura 47.- Ratio de éxito en función del número de imágenes en training. Método LBPH con 21 imágenes bajo test. Fuente: Propia*

A medida que se sigue aumentando el número de imágenes bajo test, el ratio de éxito sigue disminuyendo ligeramente. En este caso el máximo se sitúa en el 85%, mientras que el mínimo llega hasta el 71%. Además, se puede observar cómo, en este caso, no sucede el fenómeno de overfitting, pero sí que se llega a una zona de saturación. Esto puede ser debido a que no hay suficientes imágenes en el training para que haya un exceso de información.

### 5.4.4. Comparación de los resultados

Se muestra, en primer lugar, la gráfica comparativa de los tres métodos para el ratio de éxito con 7 imágenes por sujeto bajo test (Figura 48).

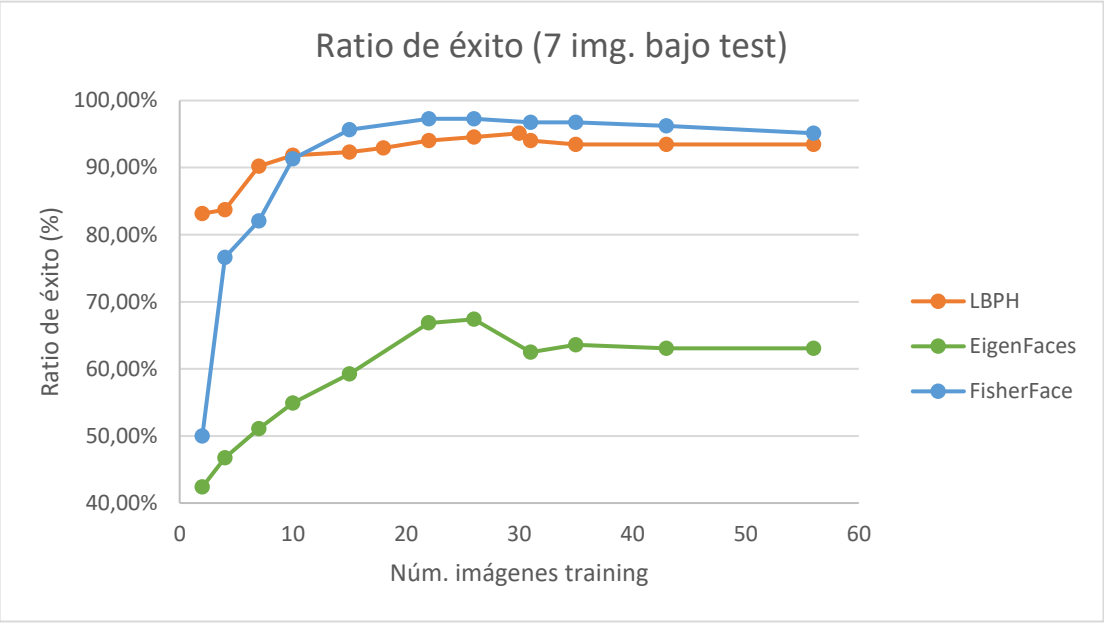


*Figura 48.- Ratio de éxito en función del número de imágenes en training. Comparativa de los tres métodos con 7 imágenes bajo test. Fuente: Propia*

En general, se observa que el método fisherfaces es el que mayores ratios de éxito proporciona, por lo que, a simple vista, se trata del más fiable. A pesar de esto, es el método que peor respuesta proporciona al trabajar con un número de imágenes en training bajo (menor a 10). En este sentido, el método LBPH se trata del más estable, proporcionando una buena tasa de éxitos incluso con un número bajo de imágenes en training. Esto, dicho con otras palabras, se puede expresar como que el método fisherfaces es el que mayor exposición presenta al underfitting. Por lo que respecta al overfitting, todos lo experimentan, pero en medida similar. Eigenfaces es el método que presenta peores resultados en general. Esto se debe a la propia naturaleza del funcionamiento del método, debido a que al trabajar con una base de datos que presenta una cierta complejidad en cuanto a iluminaciones y poses, se reduce considerablemente su rendimiento, tal y como se esperaba por lo que se ha expuesto en el capítulo 4.

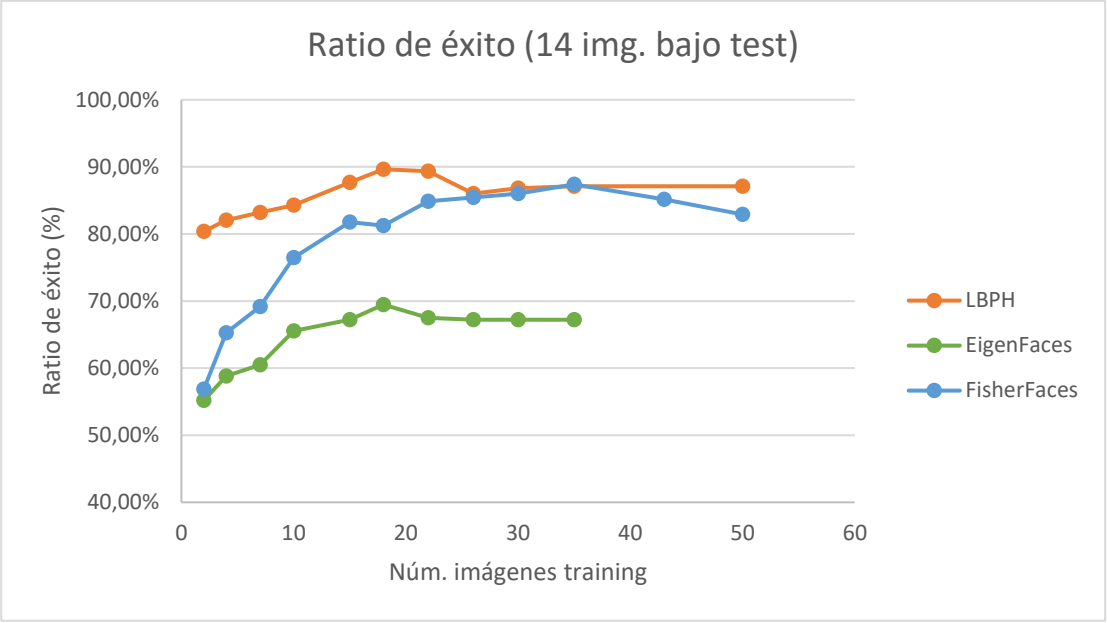


*Figura 49.- Ratio de éxito en función del número de imágenes en training. Comparativa de los tres métodos con 14 imágenes bajo test. Fuente: Propia*

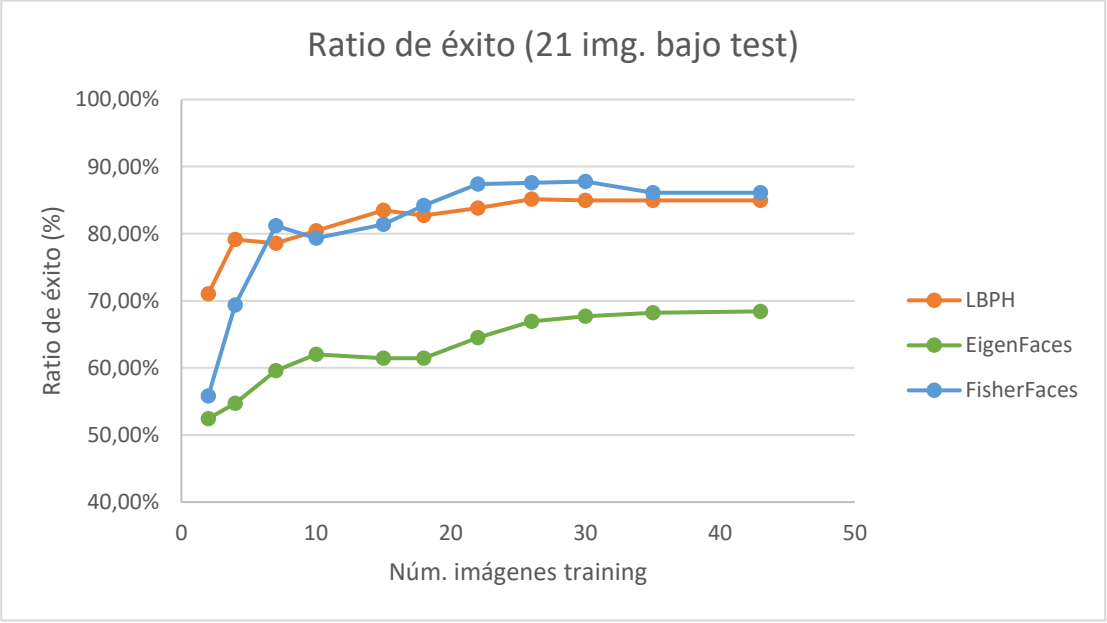


*Figura 50.- Ratio de éxito en función del número de imágenes en training. Comparativa de los tres métodos con 21 imágenes bajo test. Fuente: Propia*

Los ratios de éxito para los tres métodos se comportan de manera similar al aumentar el número de imágenes a predecir (Figura 49 y Figura 50). Lo único destacable, es que cada vez que se aumenta este número, el rendimiento en el porcentaje de éxitos baja ligeramente, debido a que se añade dificultad a la predicción.

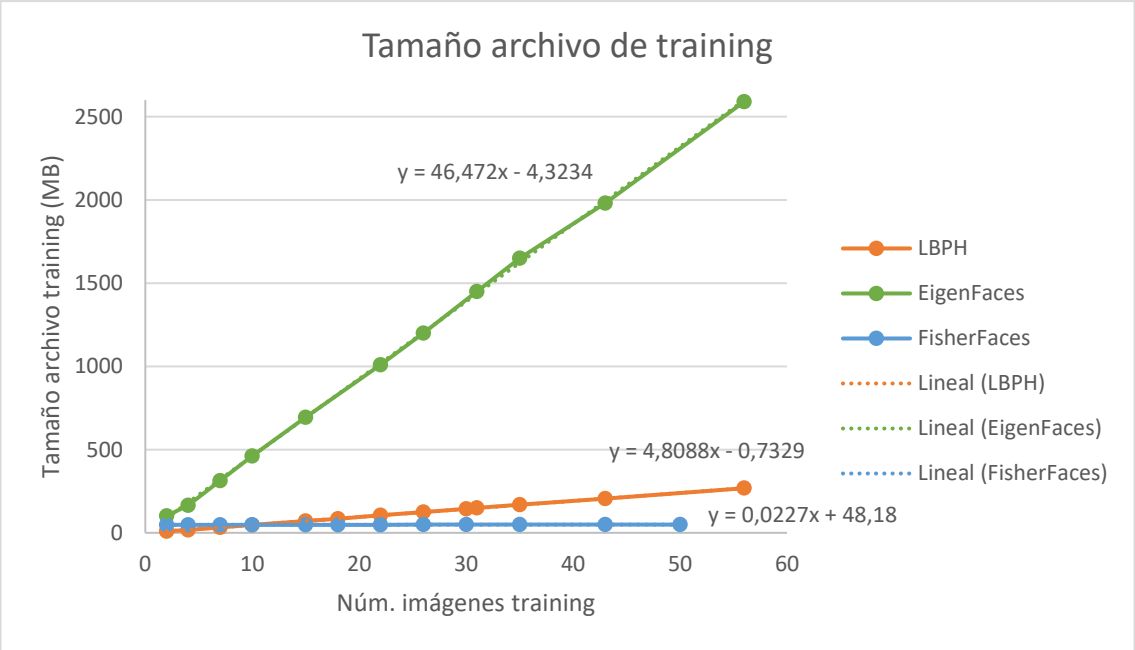


*Figura 51.- Tamaño del archivo de training en función del número de imágenes en training. Comparativa de los tres métodos. Junto a cada recta se presenta la ecuación de la línea de tendencia que mejor se ajusta a la curva, en la que **y** representa tamaño del archivo de training, mientras que **x** es el número de imágenes bajo training. Fuente: Propia*

Por lo que respecta al tamaño del archivo de training (Figura 51), resulta evidente que eigenfaces es el que mayores recursos de memoria necesita, llegando a sobrepasar los 2,5 GB para más de 50 imágenes en training. Los métodos LBPH y fisherfaces son los más eficaces, pero de entre ellos dos, tal como se puede apreciar en la Figura 51, fisherfaces se trata del método más óptimo en cuanto a uso de recursos de memoria. Además, por si gráficamente no queda claro, con la pendiente de cada recta se puede analizar cual tiene una tendencia de consumo de memoria mayor. Así, en orden ascendiente, fisherfaces presenta una pendiente de 0,0227, LBPH de 4,8088 y eigenfaces de 46,472. Esto se traduce en que los métodos LBPH y eigenfaces necesitan 211  y 2.047 veces más recursos de memoria que fisherfaces, respectivamente.


*Figura 52.- Tiempo de ejecución del training en función del número de imágenes en training. Comparativa de los tres métodos. Fuente: Propia*

El mejor tiempo de ejecución de training (Figura 52) es el del método LBPH, mientras que el que mayor número de recursos consume en cuanto a tiempo es eigenfaces. LBPH presenta una tendencia lineal, mientras que los otros dos métodos tienen una tendencia ligeramente exponencial.

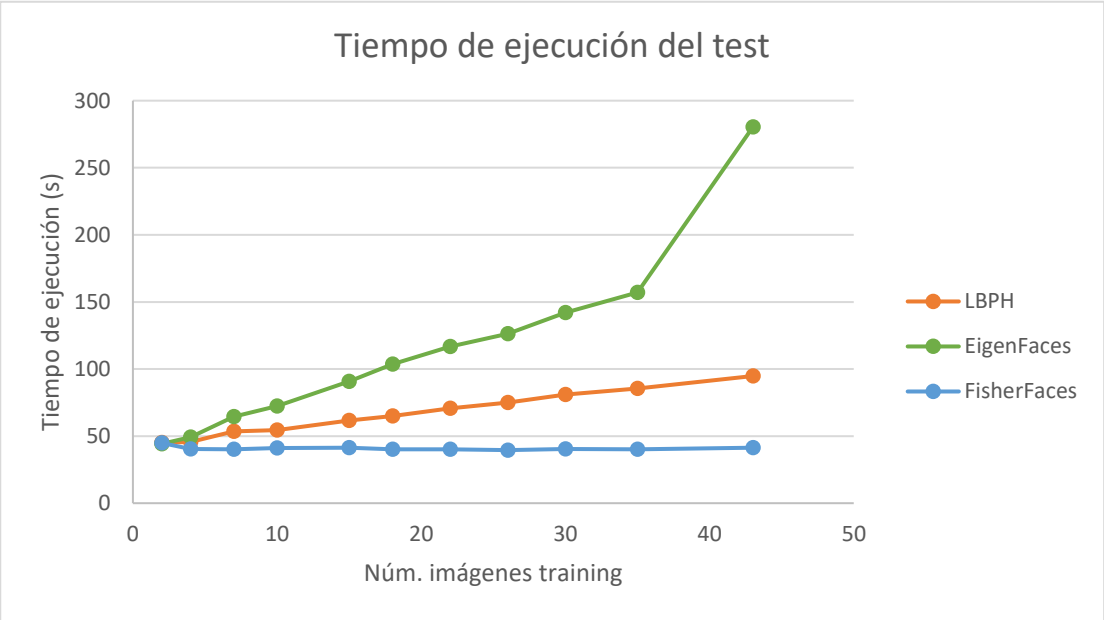


*Figura 53 Tiempo de ejecución del test en función del número de imágenes en training. Comparativa de los tres métodos. Fuente: Propia*

Hablando del tiempo de ejecución del test (Figura 53), fisherfaces es el mejor método. Además, cabe destacar que para este método, el tiempo se mantiene constante a pesar de que el número de imágenes en training aumente, de manera que solamente depende de la cantidad de imágenes que hay bajo test. En el lado opuesto se encuentra el método eigenfaces, teniendo el peor tiempo de ejecución del test.

Para resumir toda la información relativa a los tres métodos, se ha elaborado una tabla comparativa con los indicadores o elementos que se han analizado. Así, se asignará a cada método una puntuación comprendida entre 1 y 3, dónde el 3 refleja el método con mejores resultados y el 1, el peor. Además, como cada indicador tiene una importancia diferente, no resulta coherente obtener una suma simple de las valoraciones como indicador general. Para poner solución a esto, se ha propuesto ponderar la valoración de cada indicador en función de su importancia, siguiendo las siguientes pautas (Tabla 14):

*Tabla 14.- Ponderaciones propuestas para la valoración de cada indicador en función de su importancia*

| Indicador | Ponderación |
|---|---|
| **Ratio de éxito** | 60% |
| **Tamaño del archivo de training** | 5% |
| **Tiempo de ejecución del training** | 10% |
| **Tiempo de ejecución del test** | 12% |
| **Robustez frente a underfitting** | 10% |
| **Robustez frente a overfitting** | 3% |

*Tabla 15.- Tabla comparativa de los tres métodos*

| Indicador | LBPH | Fisherfaces | Eigenfaces |
|---|---|---|---|
| **Ratio de éxito** | 2 | 3 | 1 |
| **Tamaño del archivo de training** | 2 | 3 | 1 |
| **Tiempo de ejecución del training** | 3 | 2 | 1 |
| **Tiempo de ejecución del test** | 2 | 3 | 1 |
| **Robustez frente a underfitting** | 3 | 1 | 2 |
| **Robustez frente a overfitting** | 3 | 2 | 1 |
| **Total** | **2,23** | **2,67** | **1,1** |

De esta manera, una vez realizada la comparación de los tres métodos en la Tabla 15, se puede afirmar que el método fisherfaces es el que mejores resultados presenta empleando la base de datos *Extended Yale Face Database* B. Por el lado opuesto, eigenfaces se trata del peor método, obteniendo la peor puntuación en casi todos los indicadores.

Cabe destacar, pero, que tal y como se ha comentado previamente, estos resultados están sujetos a la tipología de la base de datos. Es decir, son resultados concluyentes cuando se tiene un conjunto de imágenes con ligeras variaciones en la iluminación y en la pose de la cara.

Como muestra de que estos resultados son concluyentes, se observa que, tal y como se explica en el capítulo 4, se confirma que el método eigenfaces es altamente sensible ante estas variaciones, mostrando que la base teórica se ajusta a la realidad. Esto significa que se tratan de unos resultados válidos y con una perspectiva dirigida hacía el mundo real, ya que, evidentemente, la realidad no es invariante, sino todo lo contrario, por lo que se ha pretendido utilizar unas imágenes que se ajustasen al máximo a este hecho. De esta manera, se puede reafirmar que fisherfaces se trata del mejor método de reconocimiento facial, seguido por LBPH.

# 6. Pliego de condiciones

En este capítulo se describen las condiciones necesarias que se deben cumplir para utilizar cualquier sistema de reconocimiento facial.

## 6.1. Condiciones generales

### 6.1.1. Legislación

La Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales (LOPDGDD, 3/2018) se trata de la adaptación del ordenamiento jurídico español al Reglamento UE 2016/679 del Parlamento Europeo por el cual se dictó el Reglamento General sobre Protección de Datos (RGPD). Con esto, se establece el derecho fundamental de las personas físicas a la protección de datos personales, amparado en el artículo 18.4 de la Constitución. Por lo tanto, la protección de datos se ejercerá en base a estas dos normativas, complementarias entre sí.

Las exigencias de esta nueva ley, que sustituye a la LOPD 15/1999, son las siguientes:

- **Registrar los tratamientos realizados en la empresa:** Se obliga a los responsables del tratamiento de los datos a registrar la siguiente información: identidad y datos de contacto del responsable, finalidad con la que se recogen los datos, descripción de las categorías de interesados y de las categorías de datos, si los datos van a ser cedidos a terceros, plazos para la eliminación de la información, descripción de las medidas técnicas que se adoptarán para garantizar la seguridad de la información.
- **Consentimiento inequívoco y explícito:** Obligación de obtener consentimiento inequívoco para poder recabar, almacenar y utilizar con cualquier fin los datos obtenidos.
- **Confidencialidad de los datos**
- **Obligación de dar más información:** Obligación de dar más información al interesado sobre las vías que dispone para ejercer sus derechos de acceso, rectificación, supresión, limitación del tratamiento, portabilidad y oposición.
- **Análisis del riesgo:** Se debe realizar un análisis del riesgo que puede provocar el tratamiento de determinados datos personales.
- **Comunicar los incidentes de seguridad**: Cualquier violación de la seguridad de los datos se debe comunicar a los afectados y a la Agencia Española de Protección de Datos (AEPD) en un plazo máximo de 72 horas.
- **Evaluación del impacto sobre la protección de datos**

- **Contratos de encargados del tratamiento:** El encargado del tratamiento es la persona que maneja los datos e información personal de los usuarios de una empresa. Esto se formaliza mediante un contrato entre ambas partes.

### 6.1.2. Estándares

El estándar internacional ISO/IEC 19794-5 tiene como objetivo establecer los requisitos de las imágenes de rostros para aplicaciones de reconocimiento facial y definir un formato para el almacenamiento e intercambio de dichas imágenes. Este estándar ha sido adoptado por la Organización Internacional de la Aviación Civil (ICAO). En el artículo técnico "*Evaluación de la calidad de las imágenes de rostros utilizadas para la identificación de las personas*" (Méndez-Vázquez et al., 2012) se realiza un estudio en detalle de los principales requisitos del estándar ISO/IEC 19794-5.

Las especificaciones principales de este estándar son las siguientes:

- **Posición del sujeto y fondo:** La cara debe estar centrada tanto horizontal como verticalmente. El fondo debe ser uniforme y claro. El largo de la cabeza debe comprender entre el 70% y el 80% del alto de la imagen.
- **Postura:** Postura frontal, dirigiendo la mirada al frente. Los ojos y la boca deben tener una apertura natural. El peinado no debe cubrir el rostro. El iris y la pupila deben ser claramente visibles.
- **Calidad de la fotografía:** Las imágenes no deben estar borrosas o desenfocadas. La distancia entre los centros de los ojos debe ser como mínimo de 60 píxeles para tener una buena resolución.
- **Colores e iluminación:** Imagen de color neutro. La luz debe estar distribuida de forma uniforme, evitando sombras, destellos o reflejos. No se admiten fotos con ojos rojos. Contraste y saturación adecuados.
- **Complementos:** No se aceptan elementos que cubran la cabeza, a excepción de que la autoridad de la región lo apruebe por causas religiosas o culturales. Se permiten las gafas, siempre que no oculten gran parte de la cara y no generen reflejos.

A continuación, en la Figura 54 se muestran algunos ejemplos de aplicación correcta de la norma ISO/IEC 19794-5.

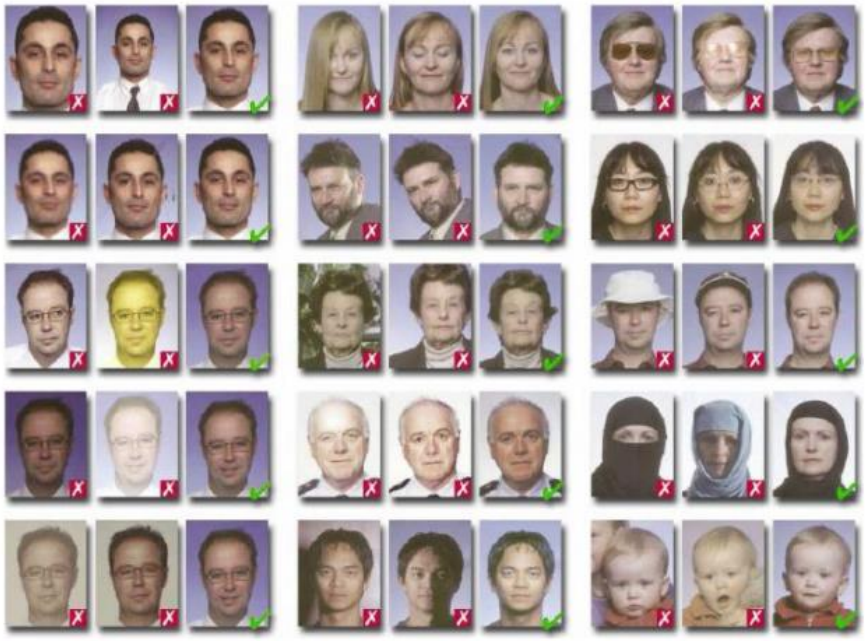

*Figura 54.- Ejemplos de aplicación de la norma ISO/IEC 19794-5. Fuente:* (Méndez-Vázquez et al., 2012)

A parte de esta normativa, cabe destacar la existencia de la ISO/IEC 19785, la cual se encarga de definir una forma de codificar y compartir datos de cualquier aplicación biométrica.

## 6.2. Condiciones técnicas

Por lo que respecta a las condiciones técnicas, se describen las licencias del software utilizado para la implementación del presente proyecto así como para la elaboración de esta memoria (Tabla 16). Para ver las condiciones técnicas del hardware consultar la Tabla 3.

*Tabla 16.- Licencias de uso del software utilizado*

| Software | Licencia de uso |
|---|---|
| **PyCharm Community Edition 2019.3.3 x64** | Libre, gratuita |
| **Python 3.6.0** | Libre, gratuita |
| **Excel 2013** | Comercial |
| **Word 2013** | Comercial |
| **Mendeley** | Libre, gratuita |
| **OpenCV** | Libre, licencia BSD |

La mayoría del software utilizado es de licencia libre porque se trata de un uso a nivel particular.

# 7. Impacto medioambiental

La UPC tiene un compromiso con la comunidad educativa de informar, infundir y promover una gestión sostenible de cualquier servicio o producto diseñado a todos sus estudiantes de ingeniería. Es un deber social velar por el uso racional y responsable de la tecnología que nos rodea. Las acciones del ser humano pueden tener consecuencias muy graves sobre la naturaleza, la fauna y la flora, el clima, y hasta sobre nosotros mismos. Estas consecuencias son muchas, pero algunas de ellas pueden ser la contaminación del suelo, del aire y del agua, la pérdida de la biodiversidad, la aparición de enfermedades en los seres humanos, la desertificación o el cambio climático. Es evidente que está en nuestras manos evitar todo esto, poniendo cada una de las personas de este mundo su granito de arena y concienciando al resto de la sociedad.


*Figura 55.- Compromiso medioambiental UPC. Fuente: UPC*

Siguiendo este compromiso, se ha incluido un estudio del impacto medioambiental generado en la realización de este proyecto. A pesar de tratarse de un proyecto intangible, basado en software, existe una carga sobre la naturaleza derivada del desarrollo del proyecto utilizando como herramienta principal un ordenador. Lo que se pretende es analizar en qué medida la realización de este proyecto ha tenido impacto sobre el medio ambiente.

Debido a que la única herramienta que se ha utilizado para la realización de este proyecto es un portátil, como bien se ha mencionado anteriormente, se estudiará el impacto medioambiental que este genera, tanto en consumo de energía como en su fabricación, uso de materiales y posterior reciclaje.

El tiempo dedicado a la realización del TFG es de 420 horas, mientras que la potencia del ordenador es de 65 W. Con esto, se calcula el consumo del PC de la siguiente manera:

$$Consumo\ PC = Horas\ de\ trabajo \cdot Potencia = 420 \cdot 65 = 27{,}3\ kWh \qquad (26)$$

A continuación, para calcular la huella de carbono equivalente emitida por el consumo del PC se ha consultado el informe "*Factores de Emisión: Registro de Huella de Carbono, Compensación y Proyectos de Absorción de Dióxido de Carbono*" (Ministerio para la Transición Ecológica y el Reto Demográfico, 2020). Se ha obtenido que para la empresa suministradora de energía en cuestión, Endesa Energía, S.A.U., el factor para la huella de carbono es de 0,38 kg $CO_2$/kWh en el año 2018. Aplicando este factor, se obtiene un total de 10,37 kg de $CO_2$ equivalentes emitidos a la atmosfera.

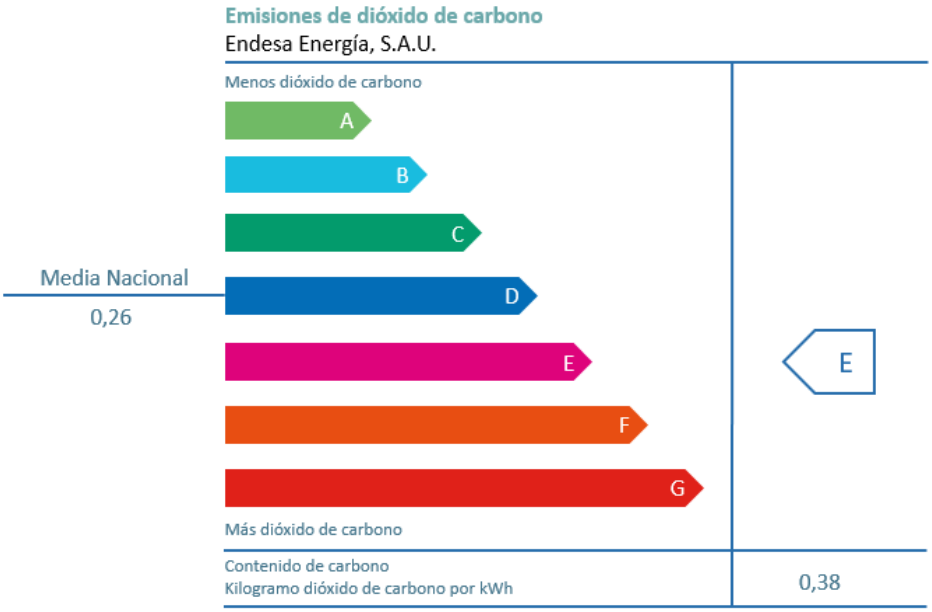


*Figura 56.- Emisiones de dióxido de carbono. Fuente: Endesa Energía, S.A.U.*

El impacto ambiental de la electricidad depende de las fuentes energéticas utilizadas para su generación. En la Figura 56 se muestra una escala de la A a la G donde A indica el mínimo impacto ambiental y G el máximo. El valor de la media nacional se sitúa en la D, mientras que la energía consumida durante la realización de este proyecto se encuentra en la E.

Desgranando al detalle el origen de esta energía consumida, se tiene que solamente el 10,5% proviene de energías renovables, mientras que el resto tiene su origen en energía nuclear, carbón, gas natural, y fuel, entre otros. Se puede observar con detalle el origen de la electricidad consumida en la Figura 57. Por otro lado, dentro del sistema eléctrico español, el uso de las energías renovables supone un 38,2%, por lo que, en comparación, Endesa se encuentra muy por debajo de la media. Cabe destacar también, que el sistema eléctrico nacional ha importado un 4,3% de producción neta total.

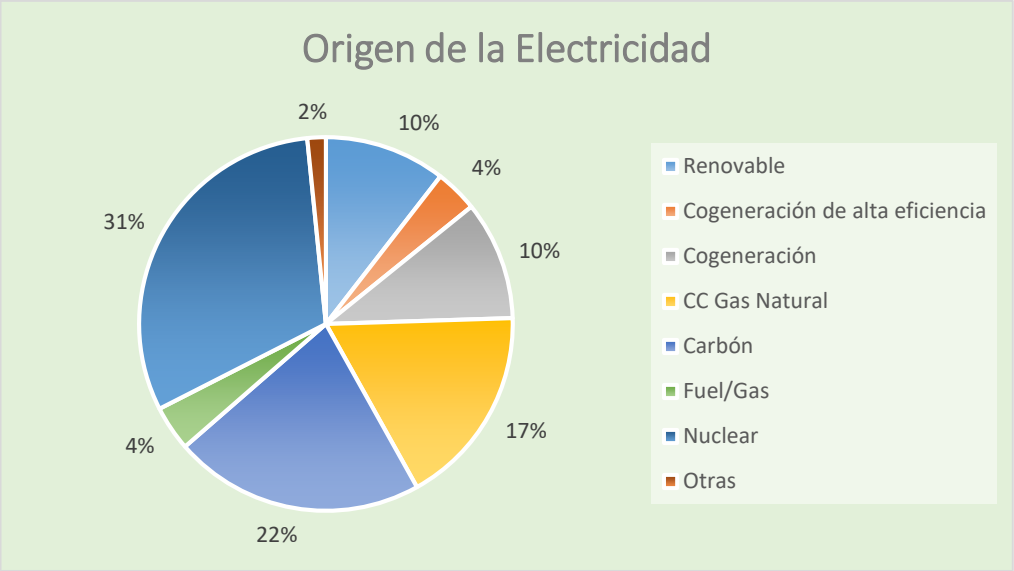


*Figura 57.- Origen de la electricidad generada por Endesa. Fuente: Endesa Energía, S.A.U.*

Una vez analizada la huella de carbono generada por el tiempo de trabajo en el proyecto, es importante destacar que, tanto la fabricación como el reciclaje del ordenador también conllevan un importante impacto medioambiental.

En el caso de un portátil, la energía necesaria para su fabricación y distribución genera aproximadamente 900 kg de $CO_2$ (*Qué impacto tiene tu PC en el medio ambiente | Reketec*, n.d.), lo que supone más del doble de la cantidad de dióxido de carbono que produce el uso de dicho portátil a lo largo de toda su vida útil. Un ordenador está compuesto por una gran cantidad de materiales, que pueden ser de origen minero o petroquímico, cuya obtención también tiene un gran impacto sobre el medio ambiente. Además, en muchos casos, la extracción de la materia prima tiene lugar en zonas conflictivas, lo cual desemboca en guerras, violaciones de los derechos humanos, pobreza y explotación laboral. Cabe tener en cuenta, también, que todos estos procesos conllevan un consumo de agua abismal. Los sectores industriales y energéticos son responsables del consumo de un 20% del agua total disponible en el planeta.

Finalmente, cuando se acaba el ciclo de vida útil de un portátil, éste debe ser llevado a un punto de reciclaje especial, donde se procederá a separar todos sus componentes para poder reutilizar su material en futuros electrodomésticos. Si los equipos no son tratados de forma correcta, se pueden liberar contaminantes que afectan al suelo, al mar y a los ríos, contribuyendo, además, al aumento de lo que se conoce como basura electrónica. Esto tiene consecuencias muy graves para el medio ambiente, para la biodiversidad y para las personas que se encuentran cerca de los vertederos donde se acumula este tipo de basura. Es por todos estos motivos que se debe hacer un uso responsable de lo que se utiliza a diario, intentando siempre que el impacto medioambiental sea el menor posible.

# 8. Análisis económico

En este capítulo se realiza un estudio económico en el que se detallan los costes de realización del proyecto. Estos costes se dividen en recursos humanos y en equipos.

## 8.1. Coste de los recursos humanos

El coste de los recursos humanos incluye el salario correspondiente a todas las horas de trabajo que han sido necesarias para llevar a cabo este TFG. Así, se han contabilizado las horas realizadas por el director del proyecto, a un precio de 50 €/h y las horas realizadas por el ingeniero júnior, a un precio de 25 €/h.

*Tabla 17.- Coste de los recursos humanos*

| Cargo | Descripción | Duración (h) | Precio (€/h) | Importe (€) |
|---|---|---|---|---|
| **Ingeniero Júnior** | Análisis del problema | 20 | 25 | 500,00 € |
| | Programación en Python | 80 | 25 | 2.000,00 € |
| | Realización de pruebas y obtención de resultados experimentales | 60 | 25 | 1.500,00 € |
| | Documentación y análisis teórico | 80 | 25 | 2.000,00 € |
| | Redacción memoria | 180 | 25 | 4.500,00 € |
| **Director del proyecto** | Asesoramiento y revisiones | 50 | 50 | 2.500,00 € |
| | | | **Subtotal** | 13.000,00 € |
| | | | **IVA (21%)** | 2.730,00 € |
| | | | **Total** | **15.730,00 €** |

Así, se obtiene un importe total de 15.730,00 € para los costes de recursos humanos.

## 8.2. Coste de los equipos

El coste de los equipos incluye tanto el material utilizado para la realización del proyecto como las licencias del software utilizado que no es de uso libre.

Por lo que respecta al coste del ordenador portátil utilizado, dado que su tiempo de vida útil es superior al tiempo de realización de este proyecto, para el cómputo de su coste, se ha realizado su amortización equivalente al año natural en que se ha desarrollado el TFG.

Para el equipo informático (Acer TravelMate P256-MG), consultando la tabla de coeficientes de amortización lineal (*Tabla de coeficientes de amortización lineal. - Agencia Tributaria*, n.d.), se tiene que para la categoría de "Equipos para procesos de la información" el coeficiente lineal máximo es del 25% y el período de años máximo es de 8. Utilizando la siguiente expresión;

$$\frac{100}{Años} \leq Tasa\ de\ amortización\ (\%) \leq Coeficiente\ lineal\ máximo\ (\%) \tag{27}$$

$$\frac{100}{8} = 12{,}5 \leq Tasa\ de\ amortización\ (\%) \leq 25\% \tag{28}$$

se tiene que la tasa de amortización del inmovilizado es de entre el 12,5% y el 25%. Así, se escoge un valor para la tasa de amortización del 20%, lo que implica que el ordenador portátil se amortiza a lo largo de 5 años. De esta manera, el valor de la amortización anual del equipo informático es de 129,80 €.

*Tabla 18.- Coste de los equipos*

| Descripción | Cantidad (ud.) | Precio (€/ud.) | Importe (€) |
|---|---|---|---|
| **Amortización anual Acer TravelMate P256-MG** | 1 | 649 | 129,80 € |
| **Licencia anual de Microsoft 365 Personal** | 1 | 69 | 69,00 € |
| | | **Subtotal** | 198,80 € |
| | | **IVA (21%)** | 41,75 € |
| | | **Total** | **240,55 €** |

Finalmente, si se realiza la suma de los dos presupuestos, se obtiene un coste total para el proyecto de 15.970,55 €.

# Conclusiones

Mediante la realización de este proyecto se han conseguido llevar a cabo, y, con éxito, los objetivos propuestos inicialmente. Se ha realizado un análisis en detalle, tanto a nivel teórico como experimental, de tres métodos utilizados en el reconocimiento facial con la finalidad de observar cuál es su rendimiento y su comportamiento bajo distintas situaciones. Para llevar a cabo dicho análisis, se han programado los tres métodos utilizando el lenguaje de programación Python y la librería de machine learning OpenCV. Una vez realizadas todas las pruebas, se ha llegado a las siguientes conclusiones:

- El método que mejor porcentaje de aciertos ofrece ante una situación variable en cuanto a iluminación y poses faciales es fisherfaces. El mayor ratio de éxito (con 7 imágenes bajo test) obtenido para cada uno de los tres métodos es: 97,28% para fisherfaces, 95,11% para LBPH, y 67,39% para eigenfaces.
- El método que menor cantidad de recursos de memoria necesita para realizar el reconocimiento facial es fisherfaces. Los métodos LBPH y eigenfaces necesitan 211 y 2.047 veces más recursos de memoria que fisherfaces, respectivamente
- El método más óptimo en el uso de recursos de tiempo que necesita para el training es LBPH.
- El método más óptimo en el uso de recursos de tiempo que necesita para el test es fisherfaces.
- El método que mayor robustez presenta ante el underfitting es LBPH. Esto significa que se trata del método más eficaz cuando se trabaja con pocas imágenes en training y del más estable si se habla en líneas generales.
- Todos los métodos presentan overfitting; llega un punto en que a pesar de que se le proporcione más información al algoritmo de reconocimiento (aumentar número de imágenes en traininig) no aumenta el porcentaje de aciertos, sino que sucede todo lo contrario, empieza a disminuir.
- En general, teniendo en cuenta todos los indicadores analizados en este estudio, el método más eficiente y fiable es fisherfaces, mientras que eigenfaces se trata del que peores resultados ha dado.

Para finalizar, cabe destacar que los resultados que se han obtenido de manera experimental, coinciden con lo que se esperaba de ellos al analizar de forma teórica los algoritmos y principios que rigen su comportamiento. Por lo tanto, se puede verificar que las pruebas realizadas se han llevado a cabo de manera correcta y fiable, pudiendo concluir así que se han superado todos los objetivos propuestos; aportando un conjunto de resultados que se han obtenido de forma empírica, demostrable, y que, además, son concluyentes.

# Líneas futuras de trabajo

Existen numerosas formas de aumentar tanto el rendimiento como la eficiencia de los métodos de reconocimiento facial expuestos en este proyecto. Una de estas formas es la combinación de varios métodos entre sí, consiguiendo que cada uno de ellos aporte sus mejores propiedades al método final. Otra forma de mejorar la eficacia del reconocimiento facial sería con el uso de cámaras de profundidad o con el uso de cámaras infrarrojas para permitir el reconocimiento en ausencia de luz.

Como propuesta, también cabe la posibilidad de ampliar la funcionalidad del sistema de reconocimiento facial, permitiendo detectar a varias personas a la vez, ya sea en la misma imagen o a través de múltiples cámaras. Esto permitiría realizar la tarea de identificación de varias personas de manera simultánea, permitiendo un gran ahorro de tiempo.

La siguiente propuesta de mejora se centra en la posibilidad de crear un sistema que genere de forma automática una base de datos con las identidades de las personas con las que se quiera trabajar. Con esto se pretende que, si por ejemplo, el reconocimiento facial es aplicado en el acceso a un edificio, no sea necesaria la intervención de una persona para la incorporación de personas en la base de datos. Esto incluiría el desarrollo de una interfaz gráfica con la que los usuarios pudieran interactuar y asignar un nombre a la identidad de nuevo acceso. Además, se permitiría interactuar con la base de datos, pudiendo ver exactamente qué datos tiene del usuario o incluso actualizando las imágenes de referencia si se produce un cambio de aspecto considerable. Finalizando con esta propuesta, si se quisiera llegar más lejos, dicho sistema podría incluir también la funcionalidad de cámara de video vigilancia, haciendo que si la identidad de una persona que intenta acceder no está permitida, avise automáticamente a la policía.

Como propuesta final, dada la situación que se está viviendo en el 2020 a causa del virus Covid-19, sería de gran interés estudiar la posibilidad de añadir a cualquier sistema de reconocimiento biométrico una cámara termográfica para medir la temperatura corporal de las personas. Así, en los lugares donde el acceso esté restringido a ciertas personas, se llevarían a cabo dos procesos; en primer lugar se verifica la identidad de la persona para determinar si tiene el acceso permitido, y, en segundo lugar, se realiza la medida de temperatura de dicha persona, y si supera el límite de temperatura, no se le permitirá el acceso, pudiendo evitar la aparición de un potencial foco de contagio del virus.

# Bibliografía

*Advancements in Computer based Facial recognition systems*. (n.d.). Retrieved April 14, 2020, from https://medium.com/coinmonks/from-the-rand-tablet-to-differentiating-identical-twins-aa4ba6031bb0

Ahonen, T., Hadid, A., & Pietikäinen, M. (2004). Face recognition with local binary patterns. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3021*, 469–481. https://doi.org/10.1007/978-3-540-24670-1_36

Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(7), 711–720. https://doi.org/10.1109/34.598228

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, *I*, 886–893. https://doi.org/10.1109/CVPR.2005.177

Delbiaggio, N. (2017). *A comparison of facial recognition's algorithms*.

*Extended Yale Face Database B (B+) | vision.ucsd.edu*. (n.d.). Retrieved March 7, 2020, from http://vision.ucsd.edu/content/extended-yale-face-database-b-b

*Face Recognition: Understanding LBPH Algorithm - Towards Data Science*. (n.d.). Retrieved April 20, 2020, from https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

*Face Recognition with OpenCV — OpenCV 2.4.13.7 documentation*. (n.d.). Retrieved April 19, 2020, from https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html

FISHER, R. A. (1936). THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. *Annals of Eugenics*, *7*(2), 179–188. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *904*, 23–37. https://doi.org/10.1007/3-540-59119-2_166

Georghiades, A. S., Belhumeur, P. N., & Kriegman, D. J. (2001). From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(6), 643–660. https://doi.org/10.1109/34.927464

Goldstein, A. J., Harmon, L. D., & Lesk, A. B. (1971). Identification of Human Faces. *Proceedings of the IEEE*, *59*(5), 748–760. https://doi.org/10.1109/PROC.1971.8254

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*(6), 417–441. https://doi.org/10.1037/h0071325

*index | TIOBE - The Software Quality Company*. (n.d.). Retrieved April 2, 2020, from

https://www.tiobe.com/tiobe-index/

Jafri, R., & Arabnia, H. R. (2009). A Survey of Face Recognition Techniques. *Journal of Information Processing Systems*, *5*(2), 41–68. https://doi.org/10.3745/jips.2009.5.2.041

Jones, M. ;, & Viola, P. (2003). *Fast Multi-view Face Detection*. http://www.merl.com

*Las aplicaciones más rentables de la inteligencia artificial | Statista*. (n.d.). Retrieved April 9, 2020, from https://es.statista.com/grafico/9437/las-aplicaciones-mas-rentables-de-la-inteligencia-artificial/

M.A. Turk, & A.P. Pentland. (1991, June). Face recognition using eigenfaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/document/139758

Martinez, A. M., & Kak, A. C. (2001). PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(2), 228–233. https://doi.org/10.1109/34.908974

Méndez-Vázquez, H., Chang, L., Rizo-Rodríguez, D., & Morales-González, A. (2012). *Evaluación de la calidad de las imágenes de rostros utilizadas para la identificación de las personas Face Image Quality Evaluation for Person Identification*. *16*(2), 147–165. http://www.scielo.org.mx/pdf/cys/v16n2/v16n2a3.pdf

Ministerio para la Transición Ecológica y el Reto Demográfico, G. de E. (2020). *FACTORES DE EMISIÓN REGISTRO DE HUELLA DE CARBONO, COMPENSACIÓN Y PROYECTOS DE ABSORCIÓN DE DIÓXIDO DE CARBONO*. https://www.miteco.gob.es/es/cambio-climatico/temas/mitigacion-politicas-y-medidas/factores_emision_tcm30-479095.pdf

Ojala, T., Pietikäinen, M., & Mäenpää, T. (2000). Gray scale and rotation invariant texture classification with local binary patterns. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1842*, 404–420. https://doi.org/10.1007/3-540-45054-8_27

*OpenCV*. (n.d.). Retrieved March 27, 2020, from https://opencv.org/

*OpenFace*. (n.d.). Retrieved April 21, 2020, from https://cmusatyalab.github.io/openface/

Papageorgiou, C. P., Oren, M., & Poggio, T. (1998). General framework for object detection. *Proceedings of the IEEE International Conference on Computer Vision*, 555–562. https://doi.org/10.1109/iccv.1998.710772

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space . *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572. https://doi.org/10.1080/14786440109462720

Peña, D. (2002). *Análisis de datos multivariantes*.

Phillips, P. Jonathon, Flynn, P. J., Scruggs, T., Bowyer, K. W., Chang, J., Hoffman, K., Marques, J., Min, J., & Worek, W. (2005). Overview of the face recognition grand challenge. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, *I*, 947–954. https://doi.org/10.1109/CVPR.2005.268

Phillips, P.J., Grother, P., Micheals, R., Blackburn, D. M., Tabassi, E., & Bone, M. (2002). *Face recognition vendor test 2002*. 44. https://doi.org/10.1109/amfg.2003.1240822

*Plotting face space – dave's blog of art and programming*. (n.d.). Retrieved April 18, 2020, from http://davesblog.fo.am/2009/11/plotting-face-space/

*Qué impacto tiene tu PC en el medio ambiente | Reketec*. (n.d.). Retrieved May 21, 2020, from https://www.reketec.com/blog/reketec-1/post/que-impacto-tiene-tu-pc-en-el-medio-ambiente-20

*RGPD | Reglamento Europeo de Protección de Datos*. (n.d.). Retrieved April 22, 2020, from https://ayudaleyprotecciondatos.es/guia-rgpd/

Sirovich, L., & Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, *4*(3), 519. https://doi.org/10.1364/josaa.4.000519

Soukupová, T., & Cech, J. (2016). *Real-Time Eye Blink Detection using Facial Landmarks*. https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf

SudhaNarang, Jain, K., MeghaSaxena, & AashnaArora. (2018). Comparison of Face Recognition Algorithms Using Opencv for Attendance System. *International Journal of Scientific and Research Publications*, *8*(2), 268. www.ijsrp.org

*Tabla de coeficientes de amortización lineal. - Agencia Tributaria*. (n.d.). Retrieved May 25, 2020, from https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml

Turk, M., & Pentland, A. (1991). *Eigenfaces for Recognition*. https://www.mitpressjournals.org/doi/pdfplus/10.1162/jocn.1991.3.1.71

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *1*. https://doi.org/10.1109/cvpr.2001.990517

Vladimir, G., Dmitriy, B., Win, T. N., & Htet, N. W. (2017). A comparative analysis of face recognition algorithms in solving the problem of visual identification. *Proceedings of the 2017 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conference, ElConRus 2017*, 666–668. https://doi.org/10.1109/EIConRus.2017.7910644

Wahyuningsih, D., Kirana, C., Sulaiman, R., Hamidah, & Triwanto. (2019, November 1). Comparison of the Performance of Eigenface and Fisherface Algorithm in the Face Recognition Process. *2019 7th International Conference on Cyber and IT Service Management, CITSM 2019*. https://doi.org/10.1109/CITSM47753.2019.8965345

# Anexo A

## A1. Entorno de trabajo en PyCharm

En la Figura 58 se puede observar el entorno de trabajo general en PyCharm:

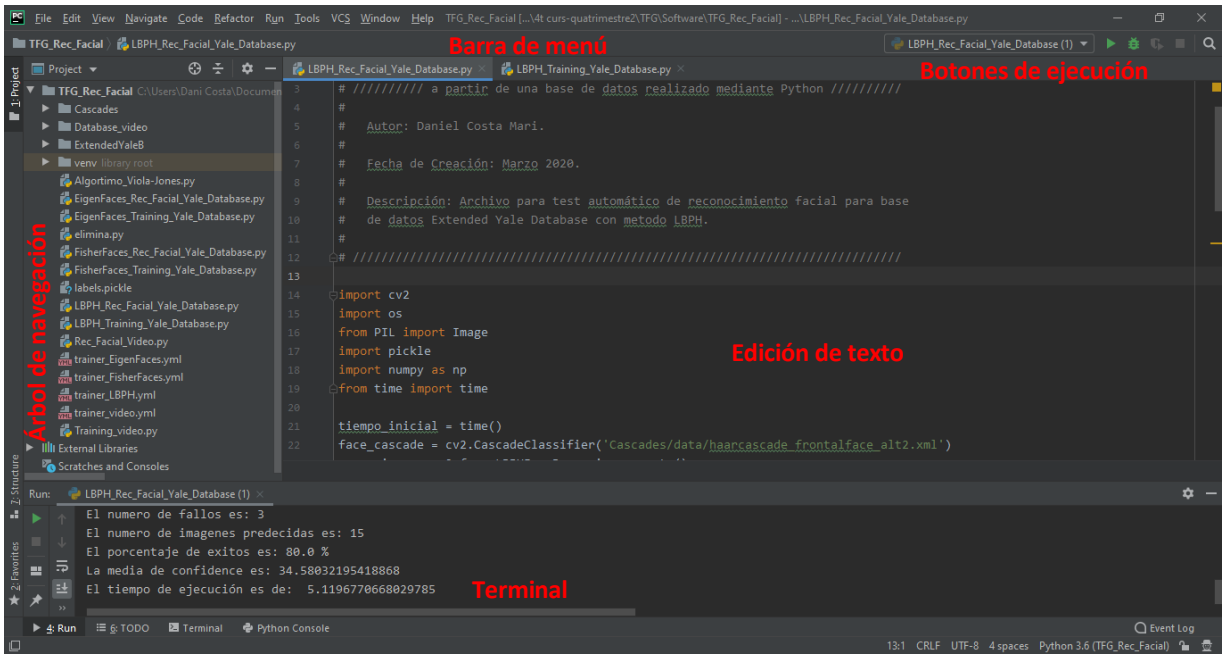


*Figura 58.- Entorno de trabajo en PyCharm. Fuente: Propia*

- **Árbol de navegación:** Permite trabajar con los distintos archivos que contiene el directorio general del proyecto.
    - ***Cascades*:** Contiene los archivos utilizados para la detección facial
    - ***Database_video*:** Es la base de datos que contiene las imágenes para el reconocimiento facial en vídeo
    - ***ExtendedYaleB*:** Es la base de datos que se ha utilizado en el proyecto; *Extended Yale Face Database* B
    - ***venv*:** Contiene las librerías y archivos del entorno virtual con el que se ha trabajado en este proyecto
    - ***Algoritmo_Viola-Jones.py*:** Archivo para la detección de caras
    - ***EigenFaces_Rec_Facial_Yale_Database.py*:** Archivo para reconocimiento facial con método Eigenfaces
    - ***EigenFaces_Training_Yale_Database.py*:** Archivo para training automático con método Eigenfaces

- ***elimina.py*:** Programa con el cual se ha automatizado el proceso de borrado de ciertas imágenes debido a que manualmente era totalmente inviable
- ***FisherFaces_Rec_Facial_Yale_Database.py*:** Archivo para reconocimiento facial con método Fisherfaces
- ***FisherFaces_Training_Yale_Database.py*:** Archivo para training automático con método Fisherfaces
- ***labels.pickle*:** Archivo generado automáticamente para etiquetar y guardar la identidad de cada sujeto
- ***LBPH_Rec_Facial_Yale_Database.py*:** Archivo para reconocimiento facial con método LBPH
- ***LBPH_Training_Yale_Database.py*:** Archivo para training automático con método LBPH
- ***Rec_Facial_Video.py*:** Archivo para reconocimiento facial en video
- ***trainer_EigenFaces.yml*:** Archivo donde se guardan las características extraídas en el training para el reconocimiento facial
- ***trainer_FisherFaces.yml*:** Archivo donde se guardan las características extraídas en el training para el reconocimiento facial
- ***trainer_LBPH.yml*:** Archivo donde se guardan las características extraídas en el training para el reconocimiento facial
- ***trainer_video.yml*:** Archivo donde se guardan las características extraídas en el training para el reconocimiento facial
- ***Training_video.py*:** Archivo para training de video

- **Edición de texto:** Se escriben y editan los programas. Se pueden añadir *breakpoints* para analizar qué sucede en una línea concreta cuando se realiza un *Debug*.
- **Terminal:** Muestra el resultado de ejecutar el código. Se puede interactuar con el programa y con el entorno virtual a través del terminal y de la consola. Muestra el proceso de *Debug*.
- **Barra de menú**: Permite seleccionar opciones, configurar el programa, realizar acciones rápidas con el código, cambiar de vista, utilizar herramientas y trabajar con los archivos.
- **Botones de ejecución**: Para ejecutar o debugar el programa.

# A2. Programación del método Eigenfaces

***EigenFaces_Training_Yale_Database.py*:**

```
# ////////////////////////////////////////////////////////////////////////////
# //////////  TFG: Análisis de un sistema de reconocimiento facial   //////////
# ////////// a partir de una base de datos realizado mediante Python //////////
#
#   Autor: Daniel Costa Mari.
#
#   Fecha de Creación: Marzo 2020.
#
#   Descripción: Archivo para training automático de la base
#   de datos Extended Yale Database para método EigenFaces.
#
# ////////////////////////////////////////////////////////////////////////////

import os
import numpy as np
from PIL import Image
import cv2
import pickle
from time import time

# se inicia el temporizador
tiempo_inicial = time()

# se selecciona el directorio de la base de datos
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
image_dir = os.path.join(BASE_DIR, "ExtendedYaleB")

# se escoge el detector facial de la carpeta "Cascades"
face_cascade =
cv2.CascadeClassifier('Cascades/data/haarcascade_frontalface_alt2.xml')
# se crea el algoritmo de reconocimiento facial
recognizer = cv2.face.EigenFaceRecognizer_create()

# se inician las variables
current_id = 0
label_ids = {}  # se crea un diccionario vacio
y_labels = []
x_train = []

# bucle para todas las imagénes del directorio seleccionadas
for root, dirs, files in os.walk(image_dir):
    for file in files[1:5]:
        if file.endswith("pgm"):
            path = os.path.join(root, file)
            label = os.path.basename(root).replace(" ", "-").lower()
            print(label, path)
            if label not in label_ids:
                label_ids[label] = current_id
                current_id += 1
            id_ = label_ids[label]
            print(label_ids)
```

```
            pil_image = Image.open(path).convert("L")  # mode L: 8-bit pixels,
black and white -> una capa en blanco y negro
            image_array = np.array(pil_image, "uint8")  # convierte imágenes en
array de números, unsigned integer (0 a 255)
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.3,
minNeighbors=4)  # detección de caras

            for (x, y, w, h) in faces:
                region_of_interest_gray = cv2.resize((image_array[y: y+h, x:
x+w]), (260, 260))  # región de interés de la imagen
                x_train.append(region_of_interest_gray)  # se crea un array con
las matrices de valores de cada imagen
                y_labels.append(id_)  # se crea un array con las id de cada
imagen -> [0, 1, 2, etc]

# escribe la representación serializada de "labels_ids" en el archivo file (f)
que está abierto
with open("labels.pickle", 'wb') as f:
    pickle.dump(label_ids, f)

# training con los datos
recognizer.train(x_train, np.array(y_labels))
# se guardan los datos del training en el archivo correspondiente
recognizer.save("trainer_EigenFaces.yml")

# fin del temporizador
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print("El tiempo de ejecución es de: ", tiempo_ejecucion)
```

***EigenFaces_Rec_Facial_Yale_Database.py*:**

```
# ////////////////////////////////////////////////////////////////////////////
# //////////  TFG: Análisis de un sistema de reconocimiento facial   //////////
# ////////// a partir de una base de datos realizado mediante Python //////////
#
#   Autor: Daniel Costa Mari.
#
#   Fecha de Creación: Marzo 2020.
#
#   Descripción: Archivo para test automático de reconocimiento facial para
#   base de datos Extended Yale Database con método EigenFaces.
#
# ////////////////////////////////////////////////////////////////////////////

import cv2
import os
from PIL import Image
import pickle
import numpy as np
from time import time

# se inicia el temporizador
```

```
tiempo_inicial = time()

# se escoge el detector facial de la carpeta "Cascades"
face_cascade =
cv2.CascadeClassifier('Cascades/data/haarcascade_frontalface_alt2.xml')
# se crea el algoritmo de reconocimiento facial
recognizer = cv2.face.EigenFaceRecognizer_create()
# se lee el archivo donde se ha guardado la información del training
recognizer.read("trainer_EigenFaces.yml")

# etiquetado y lectura de los nombres de cada identidad
labels = {"person_name": 1}
with open("labels.pickle", 'rb') as f:
    og_labels = pickle.load(f)
    labels = {v: k for k, v in og_labels.items()}

# se selecciona el directorio de la base de datos
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
image_dir = os.path.join(BASE_DIR, "ExtendedYaleB")

# se inicializan los contadores de aciertos, fallos y suma_conf a 0
aciertos = 0
fallos = 0
suma_conf = 0

# bucle para todas las imagénes del directorio seleccionadas
for root, dirs, files in os.walk(image_dir):
    for file in files[0:1]:
        # se lee el archivo si acaba en .pgm (formato de las imagénes)
        if file.endswith("pgm"):
            path = os.path.join(root, file)
            image = Image.open(path).convert("L") # mode L: 8-bit pixels, black
and white -> una capa en blanco y negro
            image_array = np.array(image, "uint8")  # convierte imágenes en
array de números, unsigned integer (0 a 255)
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.2,
minNeighbors=4) # detección de caras
            for (x, y, w, h) in faces:
                region_of_interest_gray = cv2.resize(image_array[y:y+h, x:x+w],
(260, 260))  # región de interés de la imagen
                id_, conf = recognizer.predict(region_of_interest_gray)  #
predicción de la identidad
                suma_conf = suma_conf + conf
                if conf > 0:
                    if (file[0:7].lower()) == labels[id_]:
                        aciertos += 1
                    else:
                        fallos += 1
                    print("Imagen bajo test:", file, "Prediccion:",
labels[id_], "Confidence:", conf)

# se muestran los resultados en el terminal
print("El numero de aciertos es:", aciertos)
print("El numero de fallos es:", fallos)
print("El numero de imagenes predecidas es:", (aciertos+fallos))
print("El porcentaje de exitos es:", ((aciertos/(aciertos+fallos))*100), "%")
```

```
print("La media de confidence es:", (suma_conf/(aciertos+fallos)))

# fin del temporizador
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print("El tiempo de ejecución es de: ", tiempo_ejecucion)
```

# A3. Programación del método Fisherfaces

***FisherFaces_Training_Yale_Database.py*:**

```
# ////////////////////////////////////////////////////////////////////////////
# //////////  TFG: Análisis de un sistema de reconocimiento facial   //////////
# ////////// a partir de una base de datos realizado mediante Python //////////
#
#   Autor: Daniel Costa Mari.
#
#   Fecha de Creación: Marzo 2020.
#
#   Descripción: Archivo para training automático de la base
#   de datos Extended Yale Database para método FisherFaces.
#
# ////////////////////////////////////////////////////////////////////////////

import os
import numpy as np
from PIL import Image
import cv2
import pickle
from time import time

# se inicia el temporizador
tiempo_inicial = time()

# se selecciona el directorio de la base de datos
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
image_dir = os.path.join(BASE_DIR, "ExtendedYaleB")

# se escoge el detector facial de la carpeta "Cascades"
face_cascade =
cv2.CascadeClassifier('Cascades/data/haarcascade_frontalface_alt2.xml')
# se crea el algoritmo de reconocimiento facial
recognizer = cv2.face.FisherFaceRecognizer_create()

# se inician las variables
current_id = 0
label_ids = {}  # se crea un diccionario vacio
y_labels = []
x_train = []

# bucle para todas las imagénes del directorio seleccionadas
for root, dirs, files in os.walk(image_dir):
    for file in files[1:5]:
        if file.endswith("pgm"):
            path = os.path.join(root, file)
            label = os.path.basename(root).replace(" ", "-").lower()
            print(label, path)
            if label not in label_ids:
                label_ids[label] = current_id
                current_id += 1
            id_ = label_ids[label]
            print(label_ids)
```

```
            pil_image = Image.open(path).convert("L")  # mode L: 8-bit pixels,
black and white -> una capa en blanco y negro
            image_array = np.array(pil_image, "uint8")  # convierte imágenes en
array de números, unsigned integer (0 a 255)
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.2,
minNeighbors=4)  # detección de caras

            for (x, y, w, h) in faces:
                region_of_interest_gray = cv2.resize((image_array[y: y+h, x:
x+w]), (250, 250))  # región de interés de la imagen
                x_train.append(region_of_interest_gray)  # se crea un array con
las matrices de valores de cada imagen
                y_labels.append(id_)  # se crea un array con las id de cada
imagen -> [0, 1, 2, etc]

# escribe la representación serializada de "labels_ids" en el archivo file (f)
que está abierto
with open("labels.pickle", 'wb') as f:
    pickle.dump(label_ids, f)

# training con los datos
recognizer.train(x_train, np.array(y_labels))
# se guardan los datos del training en el archivo correspondiente
recognizer.save("trainer_FisherFaces.yml")

# fin del temporizador
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print("El tiempo de ejecución es de: ", tiempo_ejecucion)
```

***FisherFaces_Rec_Facial_Yale_Database.py*:**

```
# ////////////////////////////////////////////////////////////////////////////
# //////////  TFG: Análisis de un sistema de reconocimiento facial   //////////
# ////////// a partir de una base de datos realizado mediante Python //////////
#
#   Autor: Daniel Costa Mari.
#
#   Fecha de Creación: Marzo 2020.
#
#   Descripción: Archivo para test automático de reconocimiento facial para
#   base de datos Extended Yale Database con método FisherFaces.
#
# ////////////////////////////////////////////////////////////////////////////

import cv2
import os
from PIL import Image
import pickle
import numpy as np
from time import time

# se inicia el temporizador
```

```
tiempo_inicial = time()

# se escoge el detector facial de la carpeta "Cascades"
face_cascade =
cv2.CascadeClassifier('Cascades/data/haarcascade_frontalface_alt2.xml')
# se crea el algoritmo de reconocimiento facial
recognizer = cv2.face.FisherFaceRecognizer_create()
# se lee el archivo donde se ha guardado la información del training
recognizer.read("trainer_FisherFaces.yml")

# etiquetado y lectura de los nombres de cada identidad
labels = {"person_name": 1}
with open("labels.pickle", 'rb') as f:
    og_labels = pickle.load(f)
    labels = {v: k for k, v in og_labels.items()}

# se selecciona el directorio de la base de datos
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
image_dir = os.path.join(BASE_DIR, "ExtendedYaleB")

# se inicializan los contadores de aciertos, fallos y suma_conf a 0
aciertos = 0
fallos = 0
suma_conf = 0

# bucle para todas las imagénes del directorio seleccionadas
for root, dirs, files in os.walk(image_dir):
    for file in files[0:1]:
        # se lee el archivo si acaba en .pgm (formato de las imagénes)
        if file.endswith("pgm"):
            path = os.path.join(root, file)
            image = Image.open(path).convert("L")  # mode L: 8-bit pixels,
black and white -> una capa en blanco y negro
            image_array = np.array(image, "uint8")  # convierte imágenes en
array de números, unsigned integer (0 a 255)
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.2,
minNeighbors=4)  # detección de caras
            for (x, y, w, h) in faces:
                region_of_interest_gray = cv2.resize(image_array[y:y+h, x:x+w],
(250, 250))  # región de interés de la imagen
                id_, conf = recognizer.predict(region_of_interest_gray)  #
predicción de la identidad
                suma_conf = suma_conf + conf
                if conf > 0:
                    if (file[0:7].lower()) == labels[id_]:
                        aciertos += 1
                    else:
                        fallos += 1
                    print("Imagen bajo test:", file[0:7], "Prediccion:",
labels[id_], "Confidence:", conf)

# se muestran los resultados en el terminal
print("El numero de aciertos es:", aciertos)
print("El numero de fallos es:", fallos)
print("El numero de imagenes predecidas es:", (aciertos+fallos))
print("El porcentaje de exitos es:", ((aciertos/(aciertos+fallos))*100), "%")
```

```
print("La media de confidence es:", (suma_conf/(aciertos+fallos)))

# fin del temporizador
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print("El tiempo de ejecución es de: ", tiempo_ejecucion)
```

# A4. Programación del método LBPH

***LBPH_Training_Yale_Database.py*:**

```
# ////////////////////////////////////////////////////////////////////////////
# //////////  TFG: Análisis de un sistema de reconocimiento facial   //////////
# ////////// a partir de una base de datos realizado mediante Python //////////
#
#   Autor: Daniel Costa Mari.
#
#   Fecha de Creación: Marzo 2020.
#
#   Descripción: Archivo para training automático de la base
#   de datos Extended Yale Database con metodo LBPH.
#
# ////////////////////////////////////////////////////////////////////////////

import os
import numpy as np
from PIL import Image
import cv2
import pickle
from time import time

# se inicia el temporizador
tiempo_inicial = time()

# se selecciona el directorio de la base de datos
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
image_dir = os.path.join(BASE_DIR, "ExtendedYaleB")

# se escoge el detector facial de la carpeta "Cascades"
face_cascade =
cv2.CascadeClassifier('Cascades/data/haarcascade_frontalface_alt2.xml')
# se crea el algoritmo de reconocimiento facial
recognizer = cv2.face.LBPHFaceRecognizer_create()

# se inician las variables
current_id = 0
label_ids = {}  # se crea un diccionario vacio
y_labels = []
x_train = []

# bucle para todas las imagénes del directorio seleccionadas
for root, dirs, files in os.walk(image_dir):
    for file in files[1:5]:
        if file.endswith("pgm"):
            path = os.path.join(root, file)
            label = os.path.basename(root).replace(" ", "-").lower()
            print(label, path)
            if not label in label_ids:
                label_ids[label] = current_id
                current_id += 1
            id_ = label_ids[label]
            print(label_ids)
```

```
            pil_image = Image.open(path).convert("L")  # mode L: 8-bit pixels,
black and white -> una capa en blanco y negro
            image_array = np.array(pil_image, "uint8")  # convierte imágenes en
array de números, unsigned integer (0 a 255)
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.2,
minNeighbors=4)  # detección de caras

            for (x, y, w, h) in faces:
                region_of_interest_gray = image_array[y:y + h, x:x + w]  #
región de interés de la imagen
                x_train.append(region_of_interest_gray)  # se crea un array con
las matrices de valores de cada imagen
                y_labels.append(id_)  # se crea un array con las id de cada
imagen -> [0, 1, 2, etc]

# escribe la representación serializada de "labels_ids" en el archivo file (f)
que está abierto
with open("labels.pickle", 'wb') as f:
    pickle.dump(label_ids, f)

# training con los datos
recognizer.train(x_train, np.array(y_labels))
# se guardan los datos del training en el archivo correspondiente
recognizer.save("trainer_LBPH.yml")

# fin del temporizador
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print("El tiempo de ejecución es de: ", tiempo_ejecucion)
```

***LBPH_Rec_Facial_Yale_Database.py*:**

```
# ////////////////////////////////////////////////////////////////////////////
# //////////  TFG: Análisis de un sistema de reconocimiento facial   //////////
# //////////  a partir de una base de datos realizado mediante Python //////////
#
#   Autor: Daniel Costa Mari.
#
#   Fecha de Creación: Marzo 2020.
#
#   Descripción: Archivo para test automático de reconocimiento facial para
base
#   de datos Extended Yale Database con metodo LBPH.
#
# ////////////////////////////////////////////////////////////////////////////

import cv2
import os
from PIL import Image
import pickle
import numpy as np
from time import time
```

```
# se inicia el temporizador
tiempo_inicial = time()

# se escoge el detector facial de la carpeta "Cascades"
face_cascade =
cv2.CascadeClassifier('Cascades/data/haarcascade_frontalface_alt2.xml')
# se crea el algoritmo de reconocimiento facial
recognizer = cv2.face.LBPHFaceRecognizer_create()
# se lee el archivo donde se ha guardado la información del training
recognizer.read("trainer_LBPH.yml")

# etiquetado y lectura de los nombres de cada identidad
labels = {"person_name": 1}
with open("labels.pickle", 'rb') as f:
    og_labels = pickle.load(f)
    labels = {v: k for k, v in og_labels.items()}

# se selecciona el directorio de la base de datos
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
image_dir = os.path.join(BASE_DIR, "ExtendedYaleB")

# se inicializan los contadores de aciertos, fallos y suma_conf a 0
aciertos = 0
fallos = 0
suma_conf = 0

# bucle para todas las imagénes del directorio seleccionadas
for root, dirs, files in os.walk(image_dir):
    for file in files[10:11]:
        # se lee el archivo si acaba en .pgm (formato de las imagénes)
        if file.endswith("pgm"):
            path = os.path.join(root, file)
            image = Image.open(path).convert("L")  # mode L: 8-bit pixels,
black and white -> una capa en blanco y negro
            image_array = np.array(image, "uint8")  # convierte imágenes en
array de números, unsigned integer (0 a 255)
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.2,
minNeighbors=4)  # detección de caras
            for (x, y, w, h) in faces:
                region_of_interest_gray = image_array[y:y+h, x:x+w]  # región
de interés de la imagen
                id_, conf = recognizer.predict(region_of_interest_gray)  #
predicción de la identidad
                suma_conf = suma_conf + conf
                if conf > 0:
                    if (file[0:7].lower()) == labels[id_]:
                        aciertos += 1
                    else:
                        fallos += 1
                    print("Imagen bajo test:", file, "Prediccion:",
labels[id_], "Confidence:", conf)

# se muestran los resultados en el terminal
print("El numero de aciertos es:", aciertos)
print("El numero de fallos es:", fallos)
print("El numero de imagenes predecidas es:", (aciertos+fallos))
```

```
print("El porcentaje de exitos es:", ((aciertos/(aciertos+fallos))*100), "%")
print("La media de confidence es:", (suma_conf/(aciertos+fallos)))

# fin del temporizador
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print("El tiempo de ejecución es de: ", tiempo_ejecucion)
```