Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Reconocimiento facial basado en redes neuronales convolucionales

Autor: Ildefonso Jiménez Silva

Tutores: Rubén Martín Clemente

Sergio Antonio Cruces Álvarez

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Reconocimiento facial basado en redes neuronales convolucionales

Autor:

Ildefonso Jiménez Silva

Tutores:

Rubén Martín Clemente

Profesor Titular

Sergio Antonio Cruces Álvarez

Catedrático de Universidad

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado: Reconocimiento facial basado en redes neuronales convolucionales

Autor: Ildefonso Jiménez Silva
Tutores: Rubén Martín Clemente
Sergio Antonio Cruces Álvarez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres por su continuo apoyo en todo momento y por confiar siempre en mí.

También me gustaría dar las gracias a Cristina quien me ha acompañado en tantos momentos difíciles y me ha conseguido aportar motivación siempre que la he necesitado.

No puedo olvidarme de los compañeros con los que he tenido el gusto de compartir camino de estudios: Daniel, Javier, Jesús, Sergio y mis compañeros de intensificación Abraham, Diego y Guillermo.

Finalmente, me gustaría agradecer a mi tutor, Rubén, quien además de aconsejarme y darme orientaciones para completar este proyecto, me ayudó a tener un primer contacto con un tema que desde entonces me ha llamado mucho la atención: las redes neuronales.

Ildefonso Jiménez Silva

Sevilla, 2018

Resumen

El reconocimiento facial representa un importante campo de estudio en la actualidad debido a la variedad de aplicaciones para las que puede ser empleado, que van desde la mejora de la seguridad o aplicaciones industriales hasta utilidades en dispositivos personales.

Por otro lado, las redes neuronales han experimentado en los últimos años un incremento de su uso en todo tipo de sectores gracias a su adaptación a diferentes problemas.

En este proyecto se pretende hacer un estudio de algunos de los algoritmos que usan los sistemas de reconocimiento facial y, posteriormente, implementar un sistema que aúne este área con la de las redes neuronales, donde estas sean de utilidad en algunas fases que componen todo el proceso, así como realizar una evaluación de los resultados.

Abstract

Facial recognition is currently an important field of study due to the variety its applications. They can be applied to security systems, industrial purposes or even in mobile applications.

On the other hand, the use of neural networks has increased in several areas in the last years thanks to their adjustment capacity to different issues.

This project intends to study some of the algorithms used by facial recognition systems and, in addition, to implement a system that combines some methods from this area with some neural networks and evaluate the results.

Índice

Notación

$\mathbb{R}$	Cuerpo de los números reales
$\|\mathbf{v}\|$	Norma del vector $\mathbf{v}$
$\mathbf{A}^\top$	Transpuesto de $\mathbf{A}$
e.o.c.	En cualquier otro caso
e	número e
sen	Función seno
arc tg	Función arco tangente
cot	Función cotangente
senh	Función seno hiperbólico
cosh	Función coseno hiperbólico
tanh	Función tangente hiperbólica
sgn	Función signo
$\frac{\partial y}{\partial x}$	Derivada parcial de y respecto a x
x°	Notación de grado, x grados.
σ_X^2	Varianza de la variable aleatoria X
$\gtrless$	Menor o mayor
$\lfloor x \rfloor$	Función piso

1 Introducción

1.1 Motivación del proyecto

En la sociedad actual se está produciendo un continuo desarrollo de las tecnologías de automatización. Los sistemas automáticos se han ido utilizando poco a poco en una gran cantidad de aplicaciones, tanto industriales, como en el hogar.

La biometría también ha sido un campo que ha experimentado un gran avance tecnológico, estando cada vez más presente en la sociedad actual donde la importancia de la seguridad está siendo un factor clave. Es por este motivo que los sistemas de verificación biométrica estén en continuo desarrollo, surgiendo sistemas de reconocimiento de huella dactilar, de voz o incluso de retina.

En este proyecto, se va a realizar un estudio de algunas de las tecnologías utilizadas en el desarrollo de uno de los sistemas biométricos que mayor desarrollo están mostrando actualmente, los sistemas de reconocimiento facial.

Los sistemas de reconocimiento facial emplean algoritmos para reconocer la identidad de personas que aparecen en imágenes o en vídeos a partir de sus características fisiológicas. Esto permite que cuenten con una amplia variedad de posibilidades de uso, como pueden ser la identificación de personas para sistemas de vigilancia, control de acceso o desbloqueo de dispositivos.

El reconocimiento facial representa un campo que cuenta con el interés de una gran cantidad de grandes compañías. Ya la empresa Apple ha incluido un sistema de reconocimiento facial en su iPhone X para el desbloqueo del dispositivo o la autenticación de usuario; Google cuenta con un sistema capaz de reconocer rostros en las imágenes almacenadas y Facebook ha implementado un sistema para la identificación de personas en fotografías donde no han sido etiquetadas.

En definitiva, un sistema de reconocimiento facial pretende imitar la capacidad de un ser humano para diferenciar rostros. Y es en este punto donde entra en común con las redes neuronales, las cuales tienen como objetivo desarrollar cierta inteligencia artificial y una capacidad de análisis de los datos, a partir de imitar el funcionamiento del cerebro humano.

El interés de aunar ambos conceptos representa uno de los mayores motivos por los que este proyecto ha sido llevado a cabo, donde una serie de redes neuronales serán implementadas para conseguir un sistema capaz de diferenciar personas tanto en imagen como en vídeo imitando, en cierto modo, el comportamiento de un ser humano.

1.2 Objetivos

El objetivo de este proyecto es implementar un sistema de reconocimiento facial que haga uso de redes neuronales en algunas de sus etapas y luego evaluar su funcionamiento. Se desarrollará una interfaz que permita hacer uso de los algoritmos establecidos para analizar tanto imágenes como vídeos.

El sistema presentará una fase previa a su funcionamiento en la que será necesario un entrenamiento donde analice una fuente de imágenes que le servirán de referencia para la posterior identificación de los rostros que sean analizados.

Para desarrollar el sistema, será necesaria una documentación técnica de diferentes métodos usados en el reconocimiento facial, así como una introducción a los fundamentos de redes neuronales, que tomarán importancia en ciertas etapas del sistema.

1.3 Estructura de la memoria

La presente memoria se ha organizado en un total de seis capítulos:

1. **Introducción:** Que recoge la motivación y los objetivos de este proyecto, así como su estructura.
2. **Estado de la tecnología del reconocimiento facial:** Capítulo en el que se hará una revisión de la tecnología del reconocimiento facial, pasando por cada una de las etapas que la componen, introduciendo ejemplos de algunos métodos de especial relevancia.
3. **Redes neuronales artificiales:** Capítulo que recoge una introducción a las redes neuronales artificiales, haciendo un estudio de su estructura general y un repaso de algunos casos concretos que tendrán un papel importante en la implementación del sistema.
4. **Sistema propuesto:** En esta sección se describirán las herramientas de desarrollo del sistema, así como la serie de algoritmos que son implementados.
5. **Validación experimental:** Se realizará una serie de pruebas para fijar algunos parámetros del sistema y analizar los resultados obtenidos.
6. **Conclusiones:** Última sección donde se recogerán los conceptos más importantes que han resultado del proyecto y una visión de las líneas futuras para el sistema propuesto.

2 Estado de la tecnología del reconocimiento facial

2.1 Introducción histórica

Los sistemas de reconocimiento facial se podrían considerar como relativamente modernos. Uno de sus pioneros fue Woodrow Bledsoe, quien junto con su grupo de investigación desarrolló una técnica que fue denominada "reconocimiento facial hombre-máquina", a mediados de los años 60. Esta técnica requería una clasificación de las fotografías de las caras que eran digitalizadas manualmente. Además, también era necesario indicar las coordenadas donde se encontraban los rasgos faciales dentro de cada imagen, entre los que destacaban los ojos, la nariz y la línea del pelo. Posteriormente, era necesario registrar los datos recogidos junto con el nombre de la persona en una base de datos. Finalmente, con una fotografía de una cara desconocida, el sistema utilizaría un método basado en distancias euclídeas para obtener la imagen en la base de datos que más se aproxime a la fotografía en cuestión [1].

Un acontecimiento importante en el campo del reconocimiento facial se dio en 1988, cuando L. Sirobich y M. Kirby [2] aplicaron la técnica del análisis de componentes principales (PCA) al problema del reconocimiento facial y del que posteriormente surgiría el método de *Eigenfaces* de la mano de Turk y Pentland en 1991 [3].

Aunque hubo unos años en los que el campo del reconocimiento facial no presentó importantes desarrollos, en los años 90 se convirtió en una actividad en crecimiento debido al aumento de la investigación comercial, así como el alza de los clasificadores basados en redes neuronales en tiempo real [4]. Desde entonces, han surgido numerosos tipos de algoritmos, así como variaciones de los primeros, que aún en la actualidad siguen utilizándose.

2.2 Procesamiento de reconocimiento facial

El problema del reconocimiento facial se basa en una identificación de patrones visuales. Generalmente, un sistema de reconocimiento facial se divide en cuatro fases (ver Figura 2.1) detección, preprocesado, extracción de características y comparación y clasificación.

La primera fase de detección facial consiste en localizar el rostro en la imagen. A continuación, se realiza un preprocesado a la imagen para alinear y normalizar los rostros, de manera que se obtengan unas características geométricas en común con todas las imágenes que se procesen. Luego, se lleva a cabo una extracción de características faciales para obtener información útil que será usada para distinguir unas caras de otras. Finalmente, el vector de características obtenido será comparado con la base de datos para decidir a quién pertenece el rostro [5].

Todas y cada una de las etapas presentan una gran importancia ya que el resultado final dependerá de la precisión con la que se han obtenido las características, lo cual depende tanto de la localización del rostro en la imagen como de la normalización de la imagen.

En los siguientes apartados se hará un estudio individual de las fases que se llevan a cabo a lo largo del funcionamiento de un sistema de reconocimiento facial.

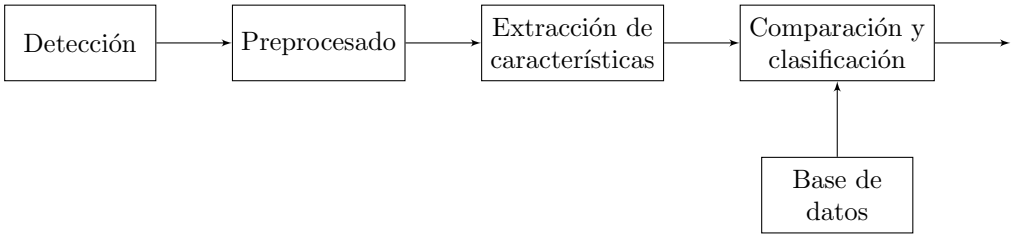


Figura 2.1 Diagrama de un sistema de reconocimiento facial.

2.3 Detección facial

La detección facial es el proceso mediante el cual el sistema localiza la posición de los rostros de personas en una imagen o fotograma[1].

Aunque en los inicios del reconocimiento facial, el proceso de localización de la cara en la imagen tenía que ser llevada a cabo manualmente, en la actualidad existen diferentes técnicas de detección automática, con diferentes fundamentos. Aunque en algunas ocasiones es difícil clasificar algunos métodos, existen autores que han realizado clasificaciones para diferenciar entre algunos métodos y otros, según su fundamento. En el trabajo de Yang *et al.* [6] se distinguen los métodos que se basan en el conocimiento, los basados en características invariantes, los que hacen uso de plantillas o modelos y los que se basan en apariencia.

Como no es posible nombrar y describir todos los métodos de detección existentes debido a la gran variedad de algoritmos (que además a su vez presentan variantes), se van a nombrar tres que se han considerado de relevancia en este proyecto. Para los dos primeros se hará un breve repaso de su funcionamiento y para el último se entrará en profundidad ya que será el método que se utilice para implementar el sistema de reconocimiento facial de este proyecto.

El primer método será el propuesto por Viola y Jones, que representa el algoritmo pionero en detección facial. A continuación, se hará un repaso de un método que hace uso de redes neuronales con el objetivo de ver un ejemplo sobre cómo se pueden implementar estas para la tarea específica de la detección facial. Finalmente, se estudiará con mayor detalle el método HOG, uno más moderno que puede ser utilizado para la detección de cualquier tipo de objeto en una imagen a partir de unas plantillas.

2.3.1 Algoritmo de Viola-Jones

El algoritmo que fue propuesto por Paul Viola y Michael Jones [7] fue la primera técnica de detección facial que, además, presentaba una rapidez y precisión apta para su uso en tiempo real.

Este método se basa en utilizar una serie de características denominadas *Haar-like features*, que son obtenidas a partir del producto escalar entre una imagen y un patrón simple del mismo tamaño que la imagen, como se muestra en la Figura 2.2. Este producto escalar tendrá signo positivo o negativo según cómo se describa el propio patrón, del que existe una gran variedad. Este resultado será el que se compare para determinar si se encuentra o no una cara en una zona determinada de la imagen.

Dado un conjunto de Haar-like features y un conjunto de imágenes positivas y negativas[2], Viola y Jones implementaron una variante de AdaBoost para el entrenamiento de su clasificador. Este se trata de un algoritmo de aprendizaje creado por Freund y Schapire [8] que consiste en obtener un clasificador fuerte a partir de varios clasificadores débiles que fijarán unos umbrales que, puestos en forma de cascada como se muestra en la Figura 2.3, formarán la región delimitada para una clase. Una vez haya sido entrenado este clasificador, será posible distinguir si el resultado de calcular las Haar-like features se corresponde con un rostro en cierto lugar de la imagen.

2.3.2 Redes Neuronales Convolucionales

Como se verá en el Capítulo 3, las redes neuronales representan una herramienta que tiene infinidad de aplicaciones, y aunque en este proyecto no se van a utilizar para la tarea de la detección facial, se va a hacer

[1] Un fotograma se puede definir como cada una de las imágenes de las que se suceden en una película cinematográfica.

[2] En el ámbito de machine learning, se conocen como datos positivos y negativos aquellos que describen o no una clase para la que se va a entrenar un modelo, respectivamente.

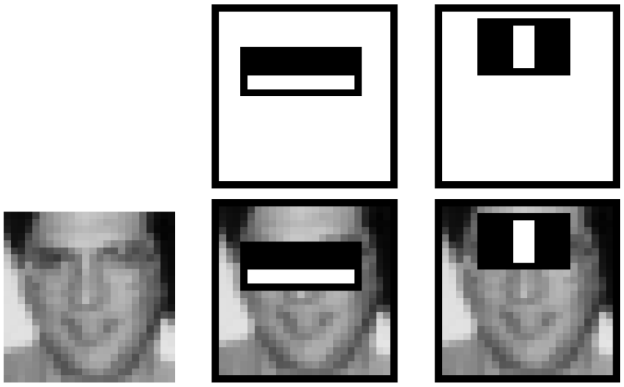


Figura 2.2 Aplicación de Haar-like features a una imagen [7].

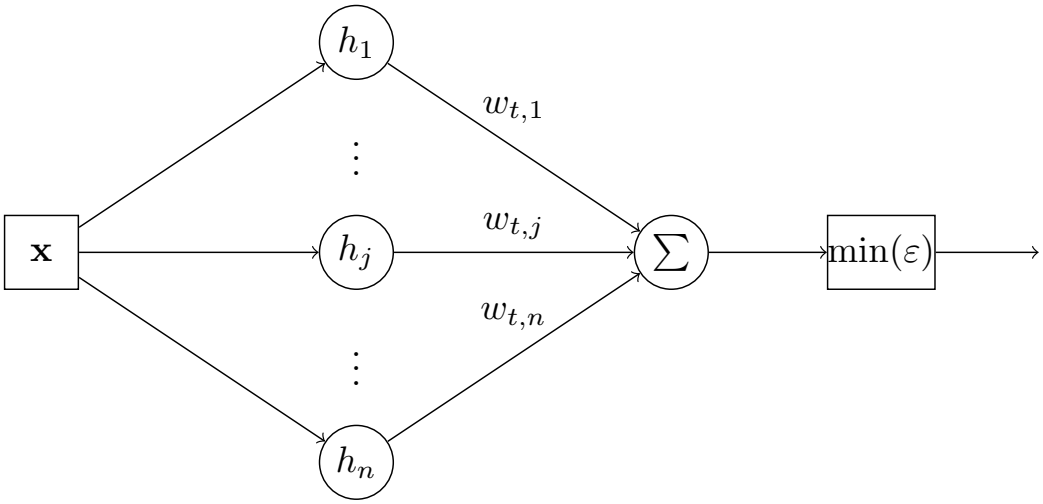


Figura 2.3 Diagrama simplificado del proceso de AdaBoost.

un breve repaso a título informativo de un método que hace uso de redes neuronales convolucionales (CNN) para detectar caras en imágenes.

A diferencia de en el último paso del reconocimiento facial, que consiste en la clasificación de rostros, el uso de redes neuronales para la detección de rostros en imágenes se trata de una tarea más difícil de llevar a cabo [9].

En la actualidad existen diferentes trabajos en los que se han implementado redes neuronales para la detección facial [9] [10], y en este apartado se hará un breve repaso del trabajo presentado en 2015 por H. Li *et al.* [11], donde proponen aplicar una serie de CNN en forma de cascada para detectar rostros directamente de la propia imagen, aprendiendo automáticamente las características en las que se tiene que basar.

Concretamente, en su trabajo describen 6 CNN, 3 de las cuales son de detección y 3 de calibración, estando intercaladas unas con otras. Las redes de detección, especializadas en 12, 24 y 48 píxeles de resolución respectivamente, son utilizadas para localizar posibles caras en la imagen. Luego, la correspondiente red de calibración se encargará de procesar las ventanas detectadas en las imágenes para ajustar su tamaño y localización para acercarse a una cara potencial cercana. En definitiva, se encarga de calibrar los recuadros que rodean las caras para ajustarlos a la cara que es posible que puedan contener.

De esta forma, la cascada irá acercándose a las localizaciones de los rostros en la imagen hasta detectarlas por completo. Este proceso se ve representado en la Figura 2.4.

En capítulos posteriores se estudiarán con mayor detalle la estructura y funcionamiento de las redes neuronales así como algunas formas de entrenarlas para conseguir buenos resultados con imágenes y datos que no hayan sido previamente analizados.

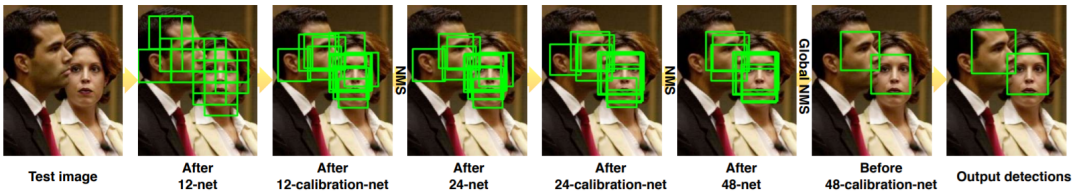


Figura 2.4 Estructura del detector facial basado en CNN en cascada [11].

2.3.3 Método de los histogramas de gradientes orientados

A diferencia del algoritmo de Viola-Jones, existen algoritmos que no han sido desarrollados específicamente para la detección facial, sino para la detección de objetos en general. Sin embargo, estos también pueden presentar un buen comportamiento si se ajustan al problema de la detección de rostros. Uno de estos métodos puede ser el de los histogramas de gradientes orientados (HOG), que fue presentado por Navneet Dalal y Bill Triggs en 2005 [12] y que, a pesar de que lo utilizaron para la detección de peatones en imágenes, es un método que puede ser utilizado para la detección de cualquier objeto, según se modele.

Este método se basa en la evaluación de histogramas locales normalizados de la orientación del gradiente de una imagen dividida en cuadrículas (que presentan un determinado solape entre ellas), que serán llamadas células. Para cada una de las divisiones de la imagen, se calcula el histograma de las direcciones de gradiente y, combinándolos, se obtiene la representación completa de la imagen. Además, con el objetivo de mejorar la invarianza frente a las condiciones de iluminación y sombras, se realiza una normalización de contraste a partir del cálculo de la energía de los histogramas locales de grandes regiones espaciales denominadas bloques. Los histogramas normalizados de los bloques son los conocidos como descriptores HOG y son los que proporcionan la información sobre cómo cambia la intensidad de la imagen debido a los bordes contenidos en ella.

Se verá a continuación la relación de los pasos que se sigue con el método HOG.

Transformación a escala de grises

El primer paso para llevar a cabo el método de HOG es disponer de una imagen de entrada que esté representada por un único canal de color, es decir, en escala de grises.

Si la imagen está en escala de grises por defecto, este paso será inmediato. En cambio, si la imagen inicial es a color, habrá que recurrir a la Ecuación (2.1) para realizar la conversión [13].

$$Y = 0.299R + 0.588G + 0.114B \tag{2.1}$$

Donde Y representa el valor de un píxel en escala de grises, y R, G y B, son los valores de intensidad de los canales rojo, verde y azul en dicho punto, respectivamente.

Cálculo de gradientes

El vector gradiente de una magnitud indica la dirección de máxima variación y su módulo indica el grado de variación de la misma. Para cada píxel, se asignará un valor de gradiente, en función del valor de los píxeles que los rodean, generalmente, los situados arriba, abajo, a la izquierda y a la derecha del píxel en cuestión. En una imagen, el gradiente ∇f indica el nivel de variación de la intensidad de la imagen y su dirección. Por definición, se sabe que el gradiente es:

$$\nabla f = \begin{bmatrix} \mathbf{G_x} \\ \mathbf{G_y} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.2}$$

Por lo que a partir de ello se podrá obtener la magnitud del gradiente $M(x,y)$ para un píxel (x,y):

$$\mathbf{M}(x,y) = \|\nabla f\| = \sqrt{\|\mathbf{G_x}\|^2 + \|\mathbf{G_y}\|^2} \approx \|\mathbf{G_x}\| + \|\mathbf{G_y}\| \tag{2.3}$$

Donde $\mathbf{G_x}$, $\mathbf{G_y}$ y $\mathbf{M}(x,y)$ son matrices del mismo tamaño que la imagen f original $(M \times N)$.

La dirección del gradiente $\alpha(x,y)$ dado un píxel (x,y) se define como:

$$\alpha(x,y) = \text{arc tg}\left(\frac{\mathbf{G_x}}{\mathbf{G_y}}\right) \tag{2.4}$$

El cálculo de las derivadas para una imagen completa puede ser implementado usando máscaras de diferentes tamaños, existiendo algunas ya predefinidas como la de Roberts, la de Prewitt o la de Sobel [14].

División en bloques

Con los gradientes de la imagen completa calculados, se procede a hacer una división de la imagen en celdas de un tamaño variable que puede ser, entre otros, de 6×6, 8×8 o 16×16 píxeles, según el objetivo fijado.

Por otro lado, hay que designar un rango de orientaciones en un intervalo concreto. Se puede usar un gradiente "sin signo" con un intervalo de 0°a 180°, o gradiente "con signo" con un intervalo de 0°a 360°. Para el intervalo fijado, se hace una partición en subregiones que, tras varias pruebas, se comprobó que el número que presenta mejores resultados es un total de 9 particiones para un gradiente sin signo. Luego, para cada celda de la imagen, se hace un recuento de cada uno de los gradientes, obteniendo un histograma con el número de gradientes situados en cada subregión. Por ejemplo, para el caso de gradientes sin signo con 9 particiones, se contarán cuántos puntos de cada bloque presentan un gradiente entre 0°y 20°, cuántos entre 20°y 40°, etc, obteniendo un histograma por cada bloque en los que se ha dividido la imagen [12].

Normalización de histogramas

El último paso antes de la detección del rostro será la normalización de los histogramas calculados anteriormente. Esto es llevado a cabo con el objetivo de paliar las variaciones indeseadas producidas por los diferentes niveles de iluminación, ya que es muy común que una imagen presente zonas más oscuras o con más sombras que otras.

Por este motivo, la normalización de los histogramas habrá que hacerla adaptándose a la iluminación de las diferentes partes de la imagen. Para ello, se toman los histogramas calculados para cada celda y se agrupan en bloques para trabajar con cada uno de ellos individualmente.

Para cada bloque, se concatenan sus histogramas de forma que se obtenga un vector v que represente dicho bloque. Una vez se cuenta con tantos vectores v como número de bloques, se lleva a cabo la normalización de estos de manera individual, siguiendo una de las normas siguientes:

$$\text{Norma L2: } v \rightarrow v/\sqrt{||v||_2^2 + \varepsilon^2} \tag{2.5a}$$

$$\text{Norma L1: } v \rightarrow v/(||v||_1 + \varepsilon) \tag{2.5b}$$

$$\text{Raíz cuadrada de norma L1: } v \rightarrow \sqrt{v/(||v||_1 + \varepsilon)} \tag{2.5c}$$

Donde ε representa una constante de valor pequeño. También se puede usar la norma L2-Hys, que consiste en limitar los valores de v a 0,2 y renormalizar. La norma L1 funciona un 5 % peor que las demás, y no aplicar el proceso de normalización hace que los resultados sean aproximadamente un 27 % peores [12].

En la Figura 2.5, se puede observar una imagen original en escala de grises y, a la derecha, el resultado de calcular la imagen HOG siguiendo el método descrito, en la que se han tomado celdas de 16×16 píxeles de tamaño y una celdas por bloque, es decir, hay el mismo número de celdas que de bloques.

(a) Imagen original. **(b)** Imagen HOG.

Figura 2.5 Ejemplo del cálculo de una imagen HOG.

Localización del objetivo

Cuando la imagen HOG es obtenida, el último será paso buscar la localización del objeto si es que existe. Para ello, se utilizan modelos o plantillas generados previamente. En el caso del reconocimiento facial, el objetivo siempre será una cara, por lo que el modelo deberá ser una imagen HOG que presente las características de una cara genérica, como la que se representa en la Figura 2.6. Esta plantilla será pasada por diferentes zonas de la imagen. Si existe cierta correlación entre el modelo y la imagen en una región concreta, significará que en dicha zona se ha encontrado una cara.

En el Capítulo 4 se utilizará esta técnica utilizada para detectar los rostros en este proyecto.

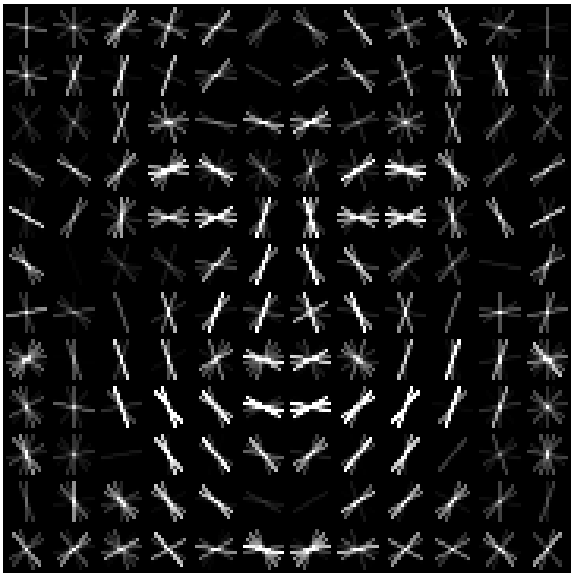


Figura 2.6 Detector HOG para encontrar rostros en una imagen [15].

2.4 Preprocesado

Mediante alguna de las técnicas de detección, se obtiene la localización de las caras dentro de la imagen en cuestión. En la siguiente etapa de los sistemas de reconocimiento facial, es necesario realizar un preprocesado de las imágenes, para prepararlas para el siguiente paso.

En muchas ocasiones, el preprocesado de una imagen intenta mejorar el resultado final de una imagen capturada, por lo que existen algoritmos que permiten eliminar ruido, mejorar la intensidad, retocar el contraste, etc [16]. Sin embargo, en el caso de esta etapa del reconocimiento facial, la mayoría de algoritmos de preprocesado va a consistir en transformaciones geométricas, como rotación o escalado, que conserven las líneas paralelas de la imagen, lo que se conoce como transformaciones afines.

Todos los algoritmos de transformaciones geométricas se basan en redistribuir los píxeles según su objetivo, por lo que el proceso consiste en determinar las nuevas coordenadas de cada píxel y luego, calcular los valores de los píxeles en la imagen destino. Este proceso se denomina interpolación [16].

Aunque existen diferentes formas de realizar transformaciones en imágenes, se va a hacer un repaso de algunas de las técnicas más generales [16].

Rotación

La rotación de una imagen es habitual en el reconocimiento facial, pues normalmente las caras no se van a encontrar siempre con el mismo grado de inclinación. Por este motivo, se requieren algoritmos de rotación que normalicen la posición de la cara en la imagen. Dada una imagen $i(x,y)$, las nuevas coordenadas de los puntos que forman la nueva imagen x' e y' serán calculadas como:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta \\ \operatorname{sen}\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.6}$$

Donde el ángulo de rotación viene definido por θ y, por lo tanto, la imagen girada $i_R(x,y)$ será:

$$i_R(x,y) = i(x',y') = i(x\cos\theta - y\,\text{sen}\,\theta, x\,\text{sen}\,\theta + y\cos\theta) \tag{2.7}$$

En reconocimiento facial, el cálculo del ángulo θ se puede calcular de diferentes formas, siendo muy común su obtención a partir de la posición de los ojos, labios y otros rasgos faciales.

Escalado

Los algoritmos de escalado permiten reducir o aumentar el tamaño de la imagen, así como hacer zoom en ciertas partes de esta.

Dado un factor de escalado α para las coordenadas x y otro β para las coordenadas y, la nueva imagen $i_E(x,y)$ sería:

$$i_E(x,y) = i(\alpha x, \beta y) \tag{2.8}$$

Teniendo en cuenta que si x/α y y/β no son enteros, hay que recurrir a una interpolación.

Deformación o inclinación

La deformación de un objeto puede ser muy útil para adaptar un rostro que no esté de frente a la cámara, puesto que puede ser equivalente a un cambio de perspectiva. Una forma sencilla de deformación para el eje x (siendo equivalente para el eje y cambiando las variables) vendría dada por:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \cot\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.9}$$

Donde θ representa el ángulo entre el eje x y el nuevo punto, tal como se muestra en la Figura 2.7.

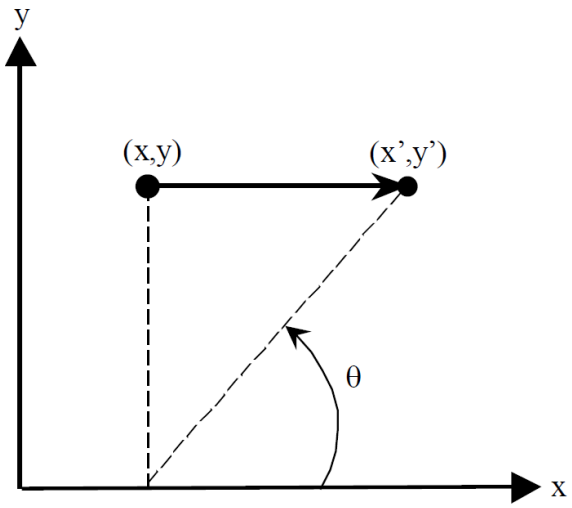


Figura 2.7 Deformación de un objeto en el eje x [16].

Por tanto, la nueva imagen podrá ser definida según la Ecuación (2.10), teniendo que ser truncada la coordenada $x + y\cot\theta$.

$$i_D(x,y) = i(x',y') = i(x + y\cot\theta, y) \tag{2.10}$$

Recorte

El recorte sirve para tomar únicamente la región de la imagen donde se encuentra un rostro. El procedimiento resulta sencillo teniendo en cuenta que una imagen en realidad se trata de una matriz numérica. Por tanto, se puede obtener el recorte tomando una submatriz a partir de los datos de la localización de la cara, que suelen ser el punto (x,y) correspondiente con la esquina superior izquierda de la imagen, y el tamaño de la ventana de detección.

Ecualización del histograma

Aunque no es una transformación geométrica, hay métodos de reconocimiento facial que son muy sensibles a las condiciones de luminosidad, por lo que una ecualización del histograma puede ser clave en estos casos. Esta técnica consiste en repartir los píxeles de forma que su luminancia se encuentre más distribuida, por lo que se aumentará el contraste y mejorará la distinción de objetos.

Para ello, dada una imagen i con niveles de intensidad de píxel que van desde 0 hasta $L-1$ (donde normalmente $L = 256$), su histograma normalizado p_n seguirá la Ecuación (2.11):

$$p_n = \frac{N_n}{N} \quad n = 0,1,\ldots,L-1 \tag{2.11}$$

Donde N_n es el número de píxeles con intensidad n y N es el número total de píxeles.

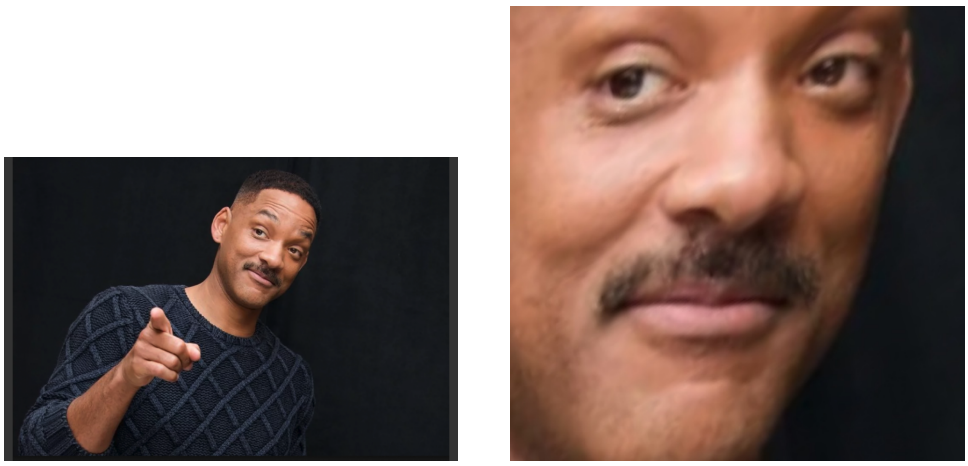
Luego, se calcula el histograma ecualizado g de la imagen tal que:

$$g_{x,y} = \lfloor (L-1) \sum_{n=0}^{i_{x,y}} p_n \rfloor \tag{2.12}$$

Donde $\lfloor x \rfloor$ denota la función piso, que redondea su argumento hacia el menor número entero más cercano y (x,y) son las coordenadas de cada píxel.

De esta forma, esto será equivalente a aplicar la función de transformación $T(k)$ de las intensidades de píxeles k de la imagen i:

$$T(k) = \lfloor (L-1) \sum_{n=0}^{k} p_n \rfloor \tag{2.13}$$



(a) Imagen original. **(b)** Imagen preprocesada.

Figura 2.8 Ejemplo del preprocesamiento de una imagen.

2.5 Extracción de características

La extracción de características consiste en la aplicación de algoritmos a imágenes digitales para reducir la redundancia y las irrelevancias que esta presenta [17].

Para reconocer e identificar los rostros en las imágenes de una forma eficiente, es necesario obtener una serie de características que los describan y representen. Por esta razón, es muy importante que esta etapa se realice con precisión y unas reglas bien definidas. Además, puesto que es una fase muy presente en campos como el análisis de datos, clasificación de patrones, biometría, visión artificial, machine learning o data mining, existe una gran variedad de técnicas de extracción de características [17].

Existen dos tipos de características presentes en las imágenes que pueden ser utilizadas para describir rostros: las geométricas y las analíticas o basadas en apariencia [18].

- **Características geométricas:** son aquellas que miden las distancias entre ciertos rasgos faciales, como los ojos, la nariz o la boca, así como su localización. Estos componentes o puntos de rasgos faciales son extraídos para formar un vector de características. Los métodos que se basan en estas características suelen buscar puntos concretos en las caras para hacer mediciones y obtener dicho vector.
- **Características de apariencia:** estas describen el cambio en la textura de la cara en función de arrugas, regiones alrededor de la boca, los ojos, y otras propiedades globales de un rostro humano. Normalmente, los métodos holísticos que se basan en estas características codifican la matriz de intensidad de los píxeles sin basarse en rasgos faciales concretos. Además, suelen hacer uso de técnicas que convierten la imagen en un espacio de características de baja dimensión, puesto que las imágenes de caras contienen redundancias o regularidades estadísticas [17].

Adicionalmente, aunque en este proyecto no se entrará en profundidad, existen técnicas que hacen uso de ambos tipos de características, por lo que son llamados **métodos híbridos** [17].

Según la forma en la que las caras son representadas, las técnicas de reconocimiento facial pueden ser divididas en cuatro grupos: basados en apariencia, en modelos, por comparación de plantillas y técnicas de redes neuronales [19]. Basándose en esta clasificación, se muestra en la Figura 2.9 un esquema con algunos ejemplos de estos algoritmos.

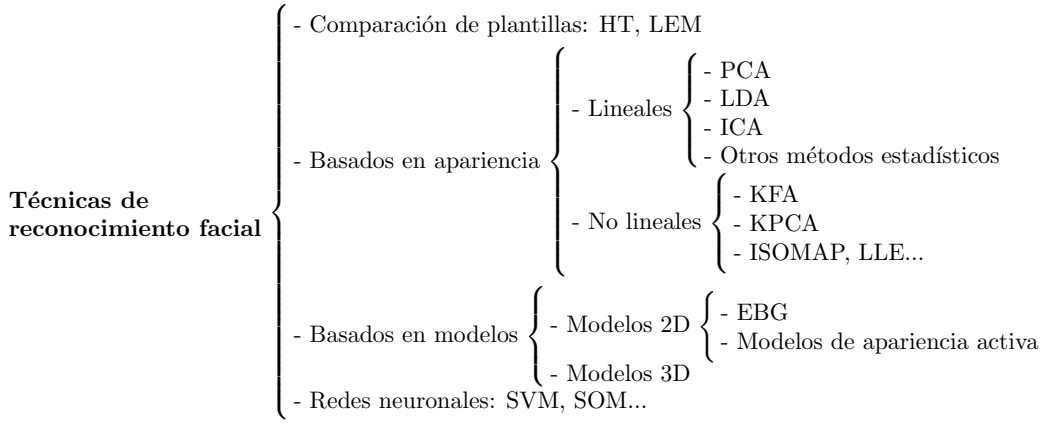


Figura 2.9 Clasificación de las técnicas de reconocimiento facial.

A continuación, se pretende hacer un breve repaso de algunas técnicas de extracción de características más utilizadas en el campo del reconocimiento facial.

2.5.1 Análisis de Componentes Principales

El análisis de componentes principales (PCA) consiste en un método de reducción de la dimensionalidad de una gran cantidad de datos formados por un gran número de variables interrelacionadas. Esto se realiza transformando estos datos a un nuevo conjunto de variables, los componentes principales (PC), que son no correlacionados y que son ordenados de modo que los primeros conservan la mayoría de la variación de las variables originales [20].

Esta técnica es utilizada en una gran variedad de aplicaciones y desde que Turk y Pentland lo aplicaron al campo del reconocimiento facial se le conoce como *Eigenfaces* [21].

2.5.2 Análisis Discriminante Lineal

El análisis discriminante lineal (LDA) se trata de un caso general del Discriminante Lineal de Fisher, el cual fue utilizado como base para el método conocido como *Fisherfaces* [22]. Este método representa una mejora del método de Eigenfaces ya que pretende maximizar la dispersión tanto entre clases diferentes, como dentro de una misma clase, lo que aportará mayor capacidad de distinción entre unos rostros y otros. Además, este método presenta un mejor comportamiento ante diferentes cambios de iluminación en una imagen y es invariante frente a diferentes expresiones faciales [23].

2.5.3 Local Binary Patterns

La idea principal del método de patrones locales binarios (LBP) es que la imagen de una cara puede ser analizada como una composición de pequeños patrones que se comportan de forma invariante respecto a las transformaciones en escala de grises. La imagen se divide en pequeñas regiones, a partir de las cuales son extraídas las características de LBP para ser concatenadas en un único histograma que represente la imagen de la cara de manera eficiente [24].

Este método parte del operador LBP, introducido por Ojala *et al.* [25], cuyo objetivo es analizar la textura en la vecindad de un píxel. Haciendo uso de este operador se obtienen histogramas que contienen información sobre bordes, puntos y zonas planas de la imagen. Posteriormente, esta información puede ser utilizada para comparar y hacer mediciones de similitud entre histogramas y, por tanto, entre diferentes caras.

2.6 Comparación y clasificación

A lo largo de todo el proceso se han ido utilizando diferentes técnicas y algoritmos para obtener, finalmente, una representación fiel de la cara que aparece en una imagen. La última fase de un sistema de reconocimiento facial consistirá en la evaluación de dicha representación, de modo que pueda ser comparada con la base de datos y poder llegar a lograr una clasificación, determinando si se corresponde con alguna de la base de datos o si, por el contrario, se trata de una cara desconocida.

Según los métodos que hayan sido utilizados en los procesos anteriores, será conveniente el uso de unas técnicas de comparación u otras. A continuación, se hará un repaso de algunas técnicas de comparación y clasificación de datos.

2.6.1 Distancia euclídea

La distancia euclidiana es una de las herramientas más antiguas de medición de distancias, así como la más simple. Aunque se fundamenta en el Teorema de Pitágoras, puede ser generalizada para un espacio de N dimensiones. De esta forma, dados dos vectores $X = [x_1, x_2, \ldots, x_N]$ e $Y = [y_1, y_2, \ldots, y_N]$, la distancia euclídea se define como:

$$d(X,Y) = \sqrt{\sum_{i=1}^{N}(y_i - x_i)^2} = \sqrt{(y_1 - x_1)^2 - (y_2 - x_2)^2 + \cdots + (y_N - x_N)^2} \tag{2.14}$$

2.6.2 K-Nearest Neighbors

El algoritmo de los K vecinos más cercanos (KNN) [26] se trata de un método de clasificación mediante aprendizaje supervisado que consiste en identificar cada dato nuevo en una clase, dependiendo de las distancia respecto a los ya existentes.

El funcionamiento del algoritmo es el siguiente, para un conjunto de datos inicial $D = \{(\mathbf{x_1}, c_1), \ldots, (\mathbf{x_N}, c_N))$:

1. Sea $\mathbf{x} = (x_1, \ldots, x_n)$ un nuevo dato que se quiere clasificar.
2. Para cada dato ya clasificado $(\mathbf{x_i}, c_i)$, se calcula la distancia con el dato sin clasificar. Normalmente, se suele usar la distancia euclídea para ello: $d_i = d(\mathbf{x_i}, \mathbf{x})$.
3. Ordenar las distancias d_i en orden ascendente.
4. Tomar los K datos $D_{\mathbf{x}}^K$ ya clasificados de menor distancia (los K más cercanos).
5. Asignar a nuevo caso $\mathbf{x}$ la clase más repetida de los $D_{\mathbf{x}}^K$.

En la Figura 2.10 se representa un sencillo ejemplo del funcionamiento del algoritmo KNN en la que se habría fijado un valor de $K = 5$. Observando el resultado, debido a que hay más elementos de la clase 1 en su vecindad, el dato en cuestión será identificado como clase 1.

2.6.3 Support Vector Machines

Las máquinas de soporte vectorial (SVM) [27] son un método de aprendizaje supervisado que genera funciones de clasificación a partir de un conjunto de datos de entrenamiento etiquetados. En sus inicios, el SVM estaba orientado a problemas de clasificación binaria, es decir, distinguir entre dos clases, aunque posteriormente se ha investigado en su aplicación multiclase [28]. Debido a su buen comportamiento, es una herramienta muy

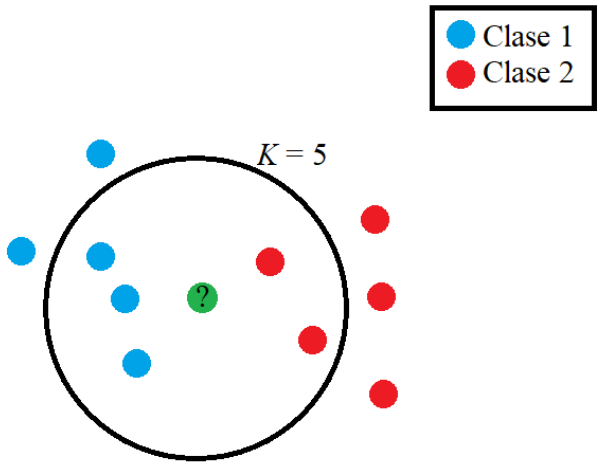


Figura 2.10 Representación gráfica del algoritmo KNN.

utilizada en múltiples aplicaciones de regresión y de clasificación de patrones, perteneciendo a la categoría de clasificadores lineales. Esto se debe a que se basan en calcular funciones lineales o hiperplanos que separen de manera óptima dos clases, dejando el mayor margen entre los datos de entrenamiento. Esto puede verse representado en la Figura 2.11 para el caso simplificado de dos dimensiones [29].

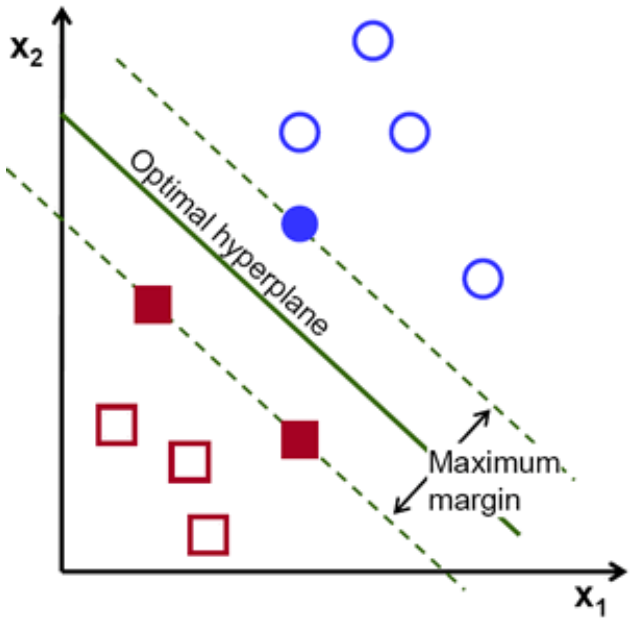


Figura 2.11 Representación de un hiperplano óptimo en un problema de clasificación [29].

2.6.4 Redes neuronales

Una red neuronal se puede definir como un modelo matemático compuesto por un gran número de elementos de procesado organizados en niveles, estando basada en el funcionamiento de las neuronas biológicas. Se trata de una tecnología computacional emergente de la que existen una gran diversidad de tipos y aplicaciones [30].

Como se ha podido comprobar, las redes neuronales pueden ser utilizadas para la tarea de la detección de caras en una imagen. No obstante, son una herramienta que puede ser empleada en una gran variedad de aplicaciones. Una de las más comunes es la clasificación de datos y patrones, por lo que también será posible emplearlas para la clasificación de caras una vez se hayan obtenido los vectores de características en la etapa anterior previa. De hecho, por este motivo hay quien considera las SVM como un tipo de red neuronal.

En los siguientes capítulos se hará una revisión de las redes neuronales debido a que serán usadas en el

proyecto, por lo que se reservará la descripción de su funcionamiento dedicado a la clasificación para el desarrollo del Capítulo 3.

3 Redes neuronales artificiales

Las redes neuronales artificiales (RNA) representan un campo de la inteligencia artificial (IA) que en la actualidad se encuentra en pleno auge debido a la enorme variedad de posibilidades que pueden llegar a ofrecer. Inspiradas en el propio comportamiento de un sistema nervioso biológico, las RNA son capaces de aprender a realizar diferentes tareas entre las que, para el desarrollo de este proyecto, se destacará su capacidad de clasificación.

3.1 Introducción histórica

Aunque las redes neuronales artificiales ha presentado una importante evolución en los últimos años, ya en 1936 Alan Turing empezó a estudiar el funcionamiento del cerebro desde el punto de vista de la computación y en 1943 W. McCulloch y W. Pitts lanzaron una teoría sobre la forma de trabajar de las neuronas, modelando su propia red neuronal simple con circuitos eléctricos.

En 1957, F. Rosenblatt inició el desarrollo del Perceptrón, la primera red neuronal, capaz de reconocer y clasificar patrones sin antes haberlos analizado.

En 1959, B. Widrow y M. Hoff desarrollaron el modelo ADALINE, la primera red neuronal aplicada a un problema real.

En 1969, M. Misnky y S. Papert publicaron su libro *Perceptrons*, donde se introducía el concepto de percetrones multinivel y, posteriormente, K. Fukushima propuso en 1980 el Neocognitrón, usado para reconocimiento de caracteres manuscritos.

Desde entonces, las redes neuronales artificiales han seguido desarrollándose, hasta llegar a la publicación de revistas especializadas como *Neural Networks* de la INNS o *IEEE Transactions on Neural Networks* del *IEEE Neural Networks Council* [30].

3.2 Estructura de una red neuronal artificial

Las redes neuronales artificiales presentan una serie de componentes que hacen que su estructura sea muy variable, según la configuración elegida.

3.2.1 Neurona artificial

Las neuronas artificiales U_j son las unidades de proceso y su función es recibir las entradas de células vecinas, calcular un valor de salida y enviarlo a otras células.

En una red neuronal se suelen diferenciar tres tipos de unidades: las de entrada, de salida y ocultas [30].

1. **Neuronas de entrada:** son aquellas que reciben las señales o información de entrada a la red neuronal.
2. **Neuronas ocultas:** se ocupan del procesado de la información y no tienen contacto con el exterior, es decir, tanto sus entradas como sus salidas pertenecen al sistema.
3. **Neuronas de salida:** son aquellas que se encargan de ofrecer la respuesta del sistema tras el procesado de la información.

Además, estas se organizan en capas o niveles, es decir, en grupos de neuronas cuyas entradas provienen de la misma fuente (o capa) y cuyas salidas se dirige al mismo destino. Al igual que las neuronas, las capas se dividen en capas de entrada, ocultas o de salida, según las neuronas que forman parte de estas.

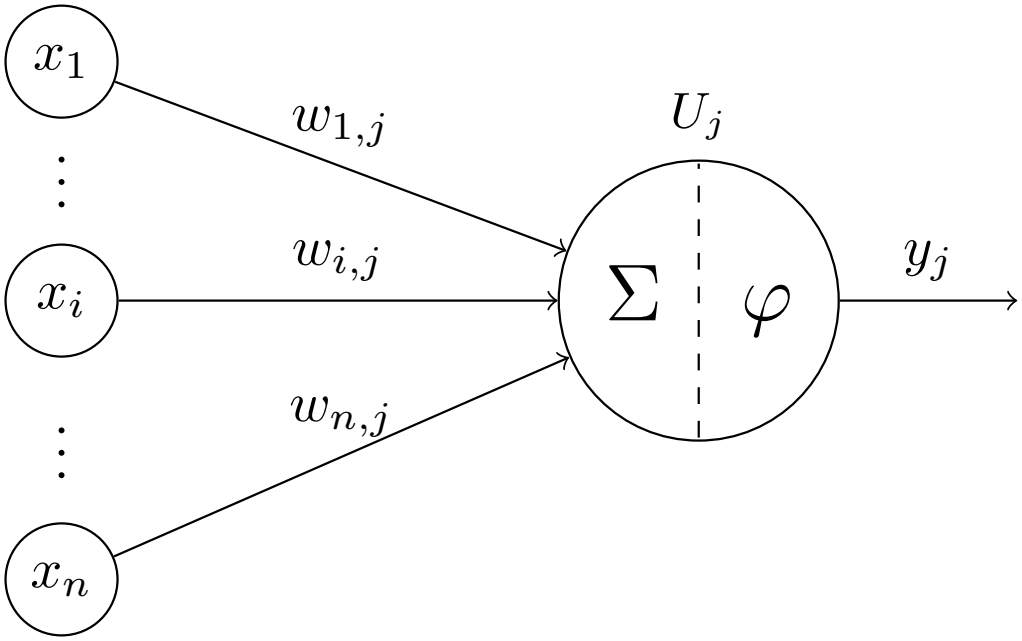


Figura 3.1 Representación de una neurona artificial genérica.

3.2.2 Conexión entre neuronas

Las neuronas deben estar conectadas entre ellas para transmitir la información a través de las diferentes capas. Si ninguna de las salidas de las neuronas hacen de entrada en neuronas de la misma capa o de niveles anteriores, la red se define como de propagación hacia delante. Si por el contrario, algunas salidas de neuronas entran en capas anteriores o en una neurona de la propia capa, la red presenta propagación hacia atrás [30].

Por otro lado, cada conexión entre una neurona i y una neurona j tiene asociado un peso $w_{i,j}$, tal como se observa en la Figura 3.1. Si este es positivo, la conexión o sinapsis se denomina excitadora; si es negativo, la conexión será inhibidora; y si su valor es cero, supondrá que la conexión entre ambas neuronas es nula [30]. Además, se suele simplificar la entrada neta de una neurona en función de los pesos $w_{i,j}$ y de las salidas de las neuronas que se conectan a ella y_i:

$$\text{Net}_j = \sum_{i=0}^{n} w_{i,j} y_i \tag{3.1}$$

3.2.3 Estado de activación

El estado de activación de la red se define como un vector $\mathbf{A}(\mathbf{t})$ de N números reales del cual cada elemento $a_i(t)$ representa la activación de una neurona U_i en función del tiempo t. Cada neurona puede presentar un estado de reposo o excitado y puede ser representado con valores continuos o discretos [30].

3.2.4 Función de salida

Cada unidad neuronal está asociada a una función de salida $\varphi_i(a_i(t))$ que transforma el estado actual de activación $a_i(t)$ en una señal de salida:

$$y_i(t) = \varphi_i(a_i(t)) \tag{3.2}$$

Las funciones de salida o de transferencia más comunes en las redes neuronales pueden ser de tipo escalón, lineal, sigmoidal o Gaussiana [30].

3.2.5 Regla de activación

La regla o función de activación es aquella que produce un nuevo estado de activación a partir del anterior y de las entradas con los pesos de las conexiones. Dada una neurona U_i con estado de activación $a_i(t)$ y su entrada total Net_i, la función de activación F producirá un nuevo estado:

$$a_i(t+1) = F(a_i(t), \text{Net}_i) \tag{3.3}$$

Normalmente la regla de activación es la función de identidad por lo que el nuevo estado de activación $a_i(t+1)$ de la neurona j coincidirá con su entrada total Net_j en el instante anterior. De este modo, la función de salida de dicha neurona recibirá la propia entrada a esta Net_j, sin tener en cuenta el estado de activación anterior, por lo que se suele considerar una única función en la neurona, la función de transferencia o de activación [30]. De esta forma, la salida de una neurona y_j en el instante $t+1$ seguirá la Ecuación (3.4).

$$y_j(t+1) = \varphi_j(\text{Net}_j - \theta_j) = \varphi_j(\sum_{i=0}^{n} w_{i,j} y_i(t) - \theta_j) \tag{3.4}$$

Donde θ_i representa un umbral de desplazamiento de la función de activación.

3.2.6 Regla de aprendizaje

La regla de aprendizaje es aquella que fija el proceso mediante el que la red neuronal modifica los pesos de sus conexiones en función de la información de entrada.

Existen dos formas principales de clasificar las redes neuronales según su regla de aprendizaje.

En primer lugar, una red neuronal puede presentar aprendizaje supervisado o no supervisado, dependiendo si necesita o no de un agente externo que controle el proceso, respectivamente.

Por otro lado, en función de si se distingue una fase de aprendizaje y otra de funcionamiento, el aprendizaje será Offline, u Online si la red puede aprender a la vez que funciona [30].

3.3 El Perceptrón

El Perceptrón es la red neuronal artificial más antigua, siendo capaz de aprender a reconocer patrones sencillos y realizar una clasificación binaria.

La estructura del Perceptrón presenta una única neurona de salida, la cual obtiene la suma ponderada de sus entradas, calcula su diferencia con el umbral θ y le aplica una función escalón o signo sgn como función de transferencia, tal como se muestra en la Ecuación (3.5).

$$y(t) = \text{sgn}\left(\sum_{i=1}^{N} w_i(t) x_i(t) - \theta\right) \tag{3.5}$$

De esta forma, el Perceptrón calcula la ecuación de un hiperplano (Ecuación (3.6)) que separa las regiones de cada clase, a partir de los valores de los pesos $\mathbf{w}^\top$ y del valor umbral θ de la función de activación. Sin embargo, al estar compuesto por una única capa de entrada y otra de salida con una sola neurona, solo puede diferenciar patrones sencillos linealmente separables [30], tales que:

$$\mathbf{w}^\top \mathbf{x} - \theta \gtrless 0 \tag{3.6}$$

3.3.1 Regla de aprendizaje

El aprendizaje del Perceptrón es de tipo supervisado, por lo que cuando se evalúen los resultados obtenidos, se deben hacer unas modificaciones del sistema hasta que este se ajuste correctamente al problema. Como se ha visto, la salida del sistema depende de los pesos de las conexiones, por tanto, estos valores serán modificados para ajustar el sistema. Para ello, el algoritmo es el siguiente [30]:

1. Inicialización de los pesos w_i y el umbral $\theta = -w_0$.
2. Presentación de los datos de entrada $\mathbf{X}_p = (x_1, x_2, \ldots, x_N)$ con la salida esperada $d(t)$.
3. Cálculo de la salida del sistema según la Ecuación (3.5).

4. Adaptación de los pesos según la Ecuación (3.7).

$$w_i(t+1) = w_i(t) + \alpha\,[d(t) - y(t)]\,x_i(t) \quad 0 \leq i \leq N \tag{3.7}$$

Donde $\alpha \in [0,1]$ es un factor de aprendizaje.

5. Repetición desde el paso 2 hasta que el sistema sea capaz de clasificar sus entradas y se de por terminado el aprendizaje.

3.4 El Perceptrón Multicapa

Como se ha visto, el Perceptrón no es capaz de realizar clasificaciones que no puedan separarse de forma lineal. De la necesidad de establecer regiones más complejas nace el perceptrón multicapa (MLP). Como su propio nombre indica, el MLP es una red neuronal de tipo feedforward compuesta por una o más capas ocultas [30], cuya estructura se representa en la Figura 3.2.

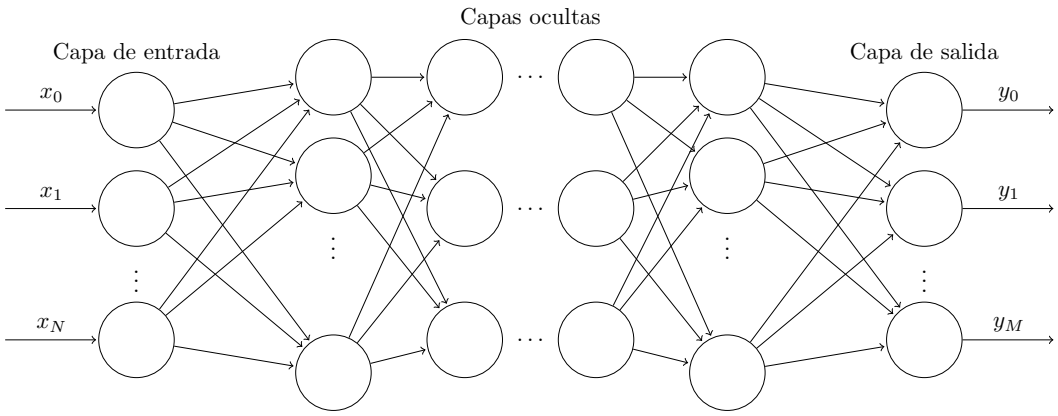


Figura 3.2 Estructura de un Perceptrón Multicapa.

El MLP presenta los tres tipos de capas descritas anteriormente: la capa de entrada, encargada de recibir los parámetros hacia el interior de la red; las capas ocultas que realizan el procesamiento de la información; y las de salida, que entregan el resultado obtenido por el sistema.

3.4.1 Regla de aprendizaje

Como se ha dicho, el MLP es una red de tipo feedforward y su técnica de aprendizaje es una de las más populares, la *back-propagation*. El funcionamiento de este método es similar al algoritmo de aprendizaje del Perceptrón, pero con el añadido de dividir la contribución del error para cada peso de la red. Así, su procedimiento es el siguiente [31]:

1. Inicialización del valor de los pesos de la red.

2. Presentación de los datos de entrada $\mathbf{X_p} = (x_1, x_2, \ldots, x_N)$ con la salida esperada $d(t)$.

3. Comparación de la salida esperada con la salida del sistema $y(t)$ y calcular el error: $\varepsilon = d(t) - y(t)$.

4. Actualización de los pesos de la capa de salida:

$$w_{i,j}(t+1) = w_{i,j}(t) + \alpha a_j \varepsilon_j F'(\mathrm{Net}_j) \tag{3.8}$$

Donde F' representa la derivada de la función de activación F (que deberá ser una función derivable como la rampa, tangente hiperbólica, logística, etc.) y $a_j = F(\mathrm{Net}_j)$ es el estado de activación de la neurona.

5. Definiendo el término de error de una capa j:

$$\Delta_j = \varepsilon_j F'(\text{Net}_j) \tag{3.9}$$

La regla de propagación de los valores de error será:

$$\Delta_i = F'(\text{Net}_i) \sum_i w_{i,j} \Delta_j \tag{3.10}$$

6. Calcular los nuevos pesos de la capa anterior k:

$$w_{k,i}(t+1) = w_{k,i}(t) + \alpha a_k \Delta_i \tag{3.11}$$

Repitiendo este proceso las veces que sea requerido, se pueden configurar todos los pesos de la red para ajustarla a una aplicación concreta.

El Perceptrón multicapa será usado en este proyecto para la tarea de clasificación en el sistema de reconocimiento facial, por lo que en posteriores secciones serán detalladas las características de configuración, así como un método para evaluar su comportamiento.

3.5 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) son un tipo concreto de red neuronal artificial que son capaces de aprender a distinguir características en un conjunto de datos a través del cálculo de convoluciones. Es por ello que este tipo de redes sean muy utilizadas en el reconocimiento de objetos en imágenes [31].

El funcionamiento de una CNN consiste en transformar los datos que obtiene mediante la capa de entrada en un conjunto de valores calculados por capas ocultas completamente conectadas. De esta forma, la estructura más general de una CNN se divide en tres tipos de capas: una de entrada, una de extracción de características y otra de clasificación (ver Figura 3.3).

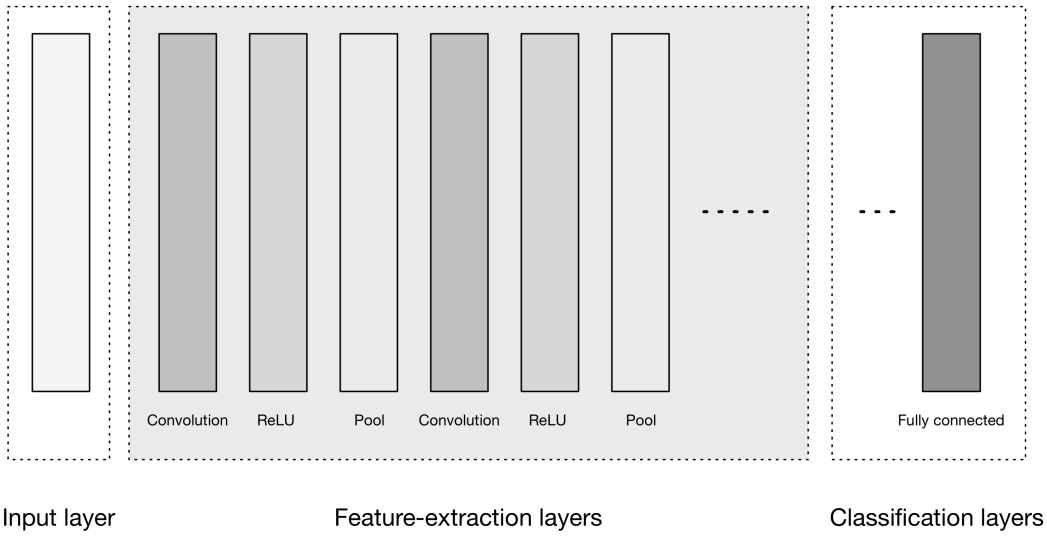


Figura 3.3 Representación general de la estructura de una CNN [31].

3.5.1 Capa de entrada

La capa de entrada es aquella que recibe los datos provenientes del exterior de la CNN. En el caso de que estos datos sean imágenes, las entradas serán tridimensionales cuyas dimensiones son el ancho, el alto y los canales de color como, por ejemplo, los tres valores RGB de cada píxel [31].

3.5.2 Capas de extracción de características

Son una serie de capas que se encargan de obtener información relevante para construir progresivamente características de orden superior. A su vez, las capas de extracción de características se pueden diferenciar en dos tipos: las capas de convolución, encargadas de calcular dicha función, y las capas de pooling que, a continuación, reúnen los resultados obtenidos por las anteriores capas [31].

Capas convolucionales

Las capas convolucionales representan la parte más importante de una CNN. Estas tienen como objetivo transformar los datos de entrada mediante un conjunto de neuronas que son conectadas localmente desde la capa anterior. Esta capa calculará un producto de puntos entre una región concreta de la capa de entrada y los pesos que la conectan a la capa de salida. Normalmente, este proceso suele mantener las mismas dimensiones espaciales.

La detección de características de la CNN es realizada mediante la de convolución, operación que es realizada en este tipo de capas. La entrada de esta pueden ser los propios datos brutos de entrada, o una salida proveniente de otra convolución anterior. En muchas ocasiones, esta operación se interpreta como un filtrado de los datos de entrada, donde un kernel encargado de filtrar se suele corresponder con el conjunto de pesos de la capa convolucional [31].

En la Figura 3.4 se muestra cómo un kernel se va desplazando a lo largo de unos datos de entrada. En cada paso, se multiplica el filtro por los valores de los datos de entrada, creando una nueva característica de salida.

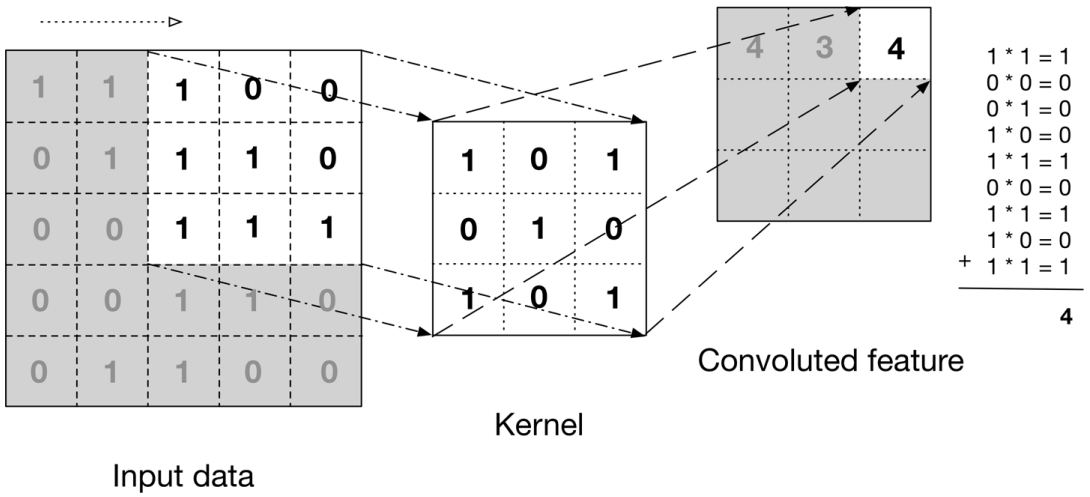


Figura 3.4 Representación de la operación de convolución con un kernel [31].

Como se muestra en la Figura 3.3, las capas convolucionales pueden ir seguidas de unas capas ReLU, las cuales tendrán el objetivo de aplicar una función de activación a modo de umbral en cero a sus datos de entrada, por lo que estas capas podrán cambiar algunos valores, pero no la dimensionalidad inicial de los datos.

Capas de pooling

Las capas de pooling son situadas entre capas convolucionales para reducir la dimensionalidad de los datos que pasan a través de las diferentes capas. Para ello, las capas de pooling suelen usar un filtro de, por ejemplo, 2×2. Este filtro es aplicado a los datos, de modo que los cuatro píxeles que sean filtrados serán sustituidos por un único valor, que se corresponderá con el máximo valor de los píxeles filtrados [31], según se muestra en la Figura 3.5.

3.5.3 Capa de clasificación

La capa de clasificación representa la última capa de la CNN, aunque en algunas ocasiones puede haber más de una capa de este tipo. Este tipo de capa se encuentra completamente conectada a su capa adyacente y su objetivo es tomar las características finales calculadas por el resto de la red y producir puntuaciones o probabilidades correspondientes a las clases entrenadas. El resultado de estas capas representará los datos de

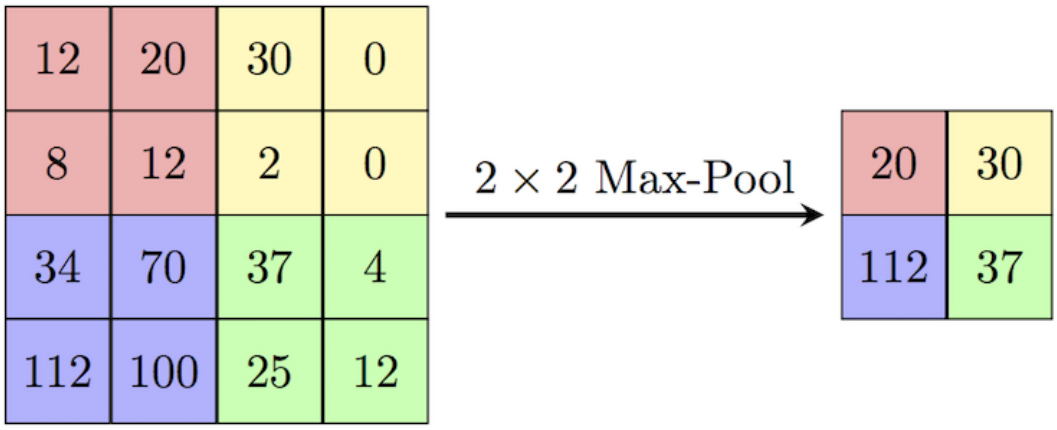


Figura 3.5 Funcionamiento de la capa de pooling [32].

salida, los cuales tendrán unas dimensiones completamente menores, siendo únicamente un vector con tantos elementos como clases hayan sido analizadas [31].

3.6 El método de aprendizaje Triplet Loss

El Triplet Loss es un método de aprendizaje supervisado que puede ser utilizado para realizar el entrenamiento de una CNN. Concretamente, este método fue introducido por F. Schroff *et al.* [33] para llevar a cabo el entrenamiento de su propio modelo de sistema de reconocimiento facial.

El fundamento de este método de entrenamiento recae en obtener lo que llaman un *embedding* $f(i)$ a partir de una imagen i, que servirá de representación de esta imagen en un espacio de características $\mathbb{R}^d$. Este espacio debe ser tal que las imágenes de una misma persona se encontrarán próximas, mientras que las imágenes entre personas distintas presentarán distancias mayores, independientemente de las condiciones de iluminación de las imágenes. Además, cada embedding calculado se encontrará en una hiperesfera, de modo que su módulo se corresponda con la unidad: $\|f(i)\| = 1$.

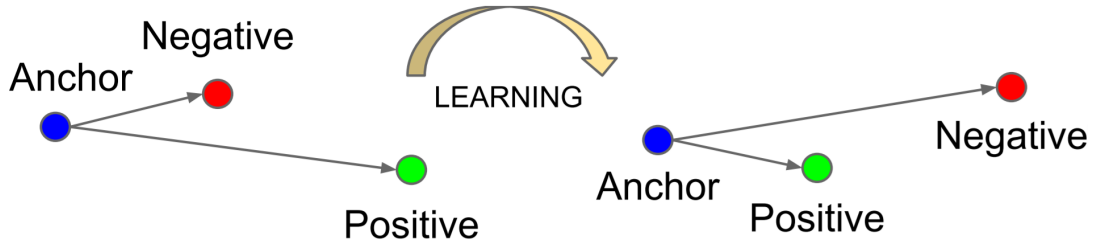


Figura 3.6 Ejemplo del funcionamiento de los triplets [33].

Por otro lado, se toma un *triplet*, que consistirá básicamente en un trío de imágenes. Cada triplet se corresponderá con la identidad de una persona concreta, de forma que habrá tres tipos de imágenes por persona: Una imagen *de referencia* a_i, una imagen *positiva* p_i y una imagen *negativa* n_i, como se muestra en la Figura 3.6. La imagen de referencia será una en la que aparezca el rostro de la persona en cuestión. La imagen positiva, será otra imagen distinta de la misma persona, de modo que, tras haber terminado todo el proceso, presente la menor distancia posible respecto a la imagen de referencia en el espacio de características. Finalmente, en la imagen negativa deberá aparecer el rostro de una persona distinta a las que aparecen en las otras dos imágenes. Esta imagen tendrá el objetivo de fijar distancias mayores respecto a la que podría presentar una imagen positiva con la de referencia, en el espacio de características. Esto evita que se produzcan futuros falsos positivos, es decir, que la red relacione a una persona con la identidad de otra diferente. Matemáticamente, se representa con la siguiente restricción:

$$\|a_i - p_i\|^2 + \alpha < \|a_i - n_i\|^2, \quad \forall (a_i, p_i, n_i) \in \tau \tag{3.12}$$

Donde α representa un margen entre los pares de imágenes positiva y negativa para evitar que la red no optimice hacia la igualdad, y τ es el conjunto de todos los N triplets posibles para entrenar.

Para realizar el entrenamiento de la CNN usando los triplets que optimicen los resultados se deberá calcular el mínimo valor de la llamada función Loss L que se representa en la Ecuación (3.13), en dependencia de los embeddings f de las tres imágenes del triplet.

$$L = \sum_{i}^{N} [\|f(a_i) - f(p_i)\|^2 - \|f(a_i) - f(n_i)\|^2 + \alpha] \tag{3.13}$$

Para elegir estos triplets de una forma adecuada, habrá que buscar una imagen positiva que, dada una imagen de referencia, maximice el primer término de la Ecuación (3.13) y una imagen negativa que minimice el segundo término de la misma, sin llegar a ser nulo. Como calcular uno a uno todos los triplets posibles no es factible por el tamaño de la base de datos, se calcularán los máximos y mínimos valores de los términos de la función Loss haciendo uso de una base de datos de tamaño menor, sacada a partir de la base de datos completa. Por ejemplo, F. Schroff *et al.* tomaron 40 imágenes positivas de cada persona y se añadieron un conjunto de ejemplos de imágenes negativas de forma aleatoria.

Luego, a partir de este conjunto, se deberá seleccionar las imágenes negativas más adecuadas teniendo en cuenta la restricción de la Ecuación (3.14), es decir, que la distancia entre la imagen positiva y la imagen de referencia sea menor que la distancia entre la negativa y la de referencia, lo que evita problemas de convergencia a la hora de entrenar el modelo.

$$\|f(a_i) - f(p_i)\|^2 < \|f(a_i) - f(n_i)\|^2 \tag{3.14}$$

Finalmente, cuando se hayan seleccionado los triplets para cada persona, bastará con presentarlos a la CNN para llevar a cabo el entrenamiento mediante el método de descenso de gradientes estocástico (SGD), con un funcionamiento muy similar al del método de backpropagation que ha sido explicado anteriormente para ajustar los pesos de la red.

4 Sistema propuesto

A lo largo de este capítulo se pretende llevar a cabo una revisión del sistema que se ha elegido para llevar a cabo el reconocimiento facial en este proyecto, así como los algoritmos que se han empleado a la hora de diseñarlo, después de haber hecho un estudio de las fases de reconocimiento facial, así como algunos de los algoritmos más utilizados.

Para esta labor, se va a dividir el capítulo en secciones, que coincidirán con los pasos que se deben de seguir para completar todo el proceso de reconocimiento facial, explicando cada uno de los métodos y algoritmos empleados. No obstante, antes de esto se va a realizar un repaso de las herramientas utilizadas para ello, como pueden ser el lenguaje de programación y las librerías utilizadas.

4.1 Herramientas de desarrollo

4.1.1 Lenguaje de programación

El lenguaje de programación que se va a utilizar a lo largo del desarrollo del proyecto es Python [34], ya que cuenta con una gran variedad de librerías que facilitan el desarrollo de aplicaciones de visión artificial, reconocimiento facial e inteligencia artificial, como se verá a continuación.

Concretamente, ha sido utilizada la versión 3.6, al ser compatible con las librerías utilizadas y ser la última versión estable de Python a fecha del inicio del desarrollo del sistema.

4.1.2 Entorno de desarrollo

El entorno de desarrollo utilizado ha sido Spyder, disponible bajo la distribución multiplataforma Anaconda [35], la cual es libre y abierta. La versión de Anaconda empleada ha sido la 5.1, también por ser la última estable a inicios del proyecto.

4.1.3 Librerías utilizadas

Para la implementación del sistema se han usado una serie de librerías que han facilitado el desarrollo gracias a una serie de funciones o clases que llevan implementan los algoritmos explicados durante este trabajo. Las principales librerías que se han utilizado son:

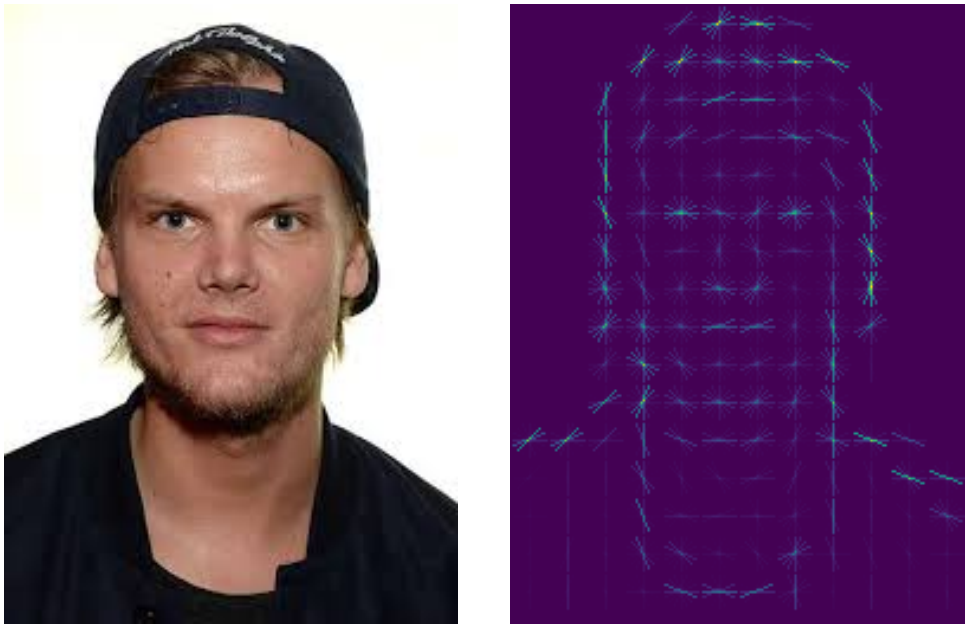
- **NumPy** [36] (versión 1.14.15): representa el paquete de matemáticas de Python más importante ya que añade funcionalidades de vectores y matrices.
- **OpenCV** [29] (versión 3.4.1): es una librería de código abierto que aporta funciones de visión artificial y machine learning.
- **Face Recognition** [37] (versión 1.2.2): se trata de una librería que aporta herramientas para reconocer y manipular caras de una forma muy sencilla.
- **Scikit-Learn** [38] (versión 0.19.1): es una librería que integra algoritmos de machine learning de forma eficiente como algoritmos de clasificación, regresión y clustering.

4.2 Algoritmos utilizados

Como se ha explicado en el Capítulo 2, el proceso de reconocimiento facial debe seguir una serie de pasos con diferentes algoritmos hasta obtener el resultado final. Para cada fase del sistema, se va a hacer un estudio de los algoritmos empleados.

4.2.1 Detección

El algoritmo de detección de rostros utilizado ha sido el de histogramas de gradientes orientes (HOG). Como se explicó en la Subsección 2.3.3, consiste en calcular los histogramas de pequeñas regiones de la imagen para obtener una llamada imagen HOG que representa la variación de la luminosidad en la propia imagen.



(a) Imagen original. **(b)** Imagen HOG.

Figura 4.1 Ejemplo del cálculo de una imagen HOG.

Además, para mejorar la detección de caras en diferentes escalas, se hace uso de una pirámide de imagen, que consiste en convertir una misma imagen a diferentes resoluciones, de modo que se obtiene una serie de imágenes de tamaños de mayor a menor, simulando una pirámide. A este conjunto de imágenes se le calcula la imagen HOG siguiendo los pasos explicados y se le aplica el detector HOG que representa un patrón genérico de una cara. Este detector es pasado a modo de filtro a lo largo de las imágenes y, al obtener cierta correlación entre la imagen y el modelo, significará que en esa zona se encuentra una cara. Este proceso es implementado en el sistema mediante la librería Face Recognition, que a su vez emplea herramientas proporcionadas por la librería Dlib [39].

4.2.2 Preprocesado

Para ajustar la imagen se hace uso de un algoritmo llamado *estimación de punto de referencia* que ha sido implementado en la librería Face Recognition. Concretamente, el algoritmo se basa en el trabajo de V. Kazemi y J. Sullivan [40], el cual consiste en estimar una serie de p puntos de referencia de la cara de una forma eficiente, haciendo uso de una cascada de regresores.

Estos puntos de referencia se denotan como $\mathbf{S} = (\mathbf{x}_1^\top, \mathbf{x}_2^\top, \ldots, \mathbf{x}_p^\top)^\top \in \mathbb{R}^{2p}$ que recoge todas las coordenadas $\mathbf{x_i} = (x_i, y_i) \in \mathbb{R}^2$ de cada punto de referencia de la cara en una imagen I.

Este vector tendrá que ser estimado mediante una serie de regresores r_t, que deben ser entrenados en cascada para ser capaces de acercarse cada vez más a los diferentes puntos del rostro. Estos regresores son entrenados a partir de un conjunto de datos de entrenamiento que contiene un número de imágenes de caras con sus respectivos puntos de referencia, haciendo uso de un algoritmo de aprendizaje automático llamado potenciación de gradiente [40].

Debido a que el entrenamiento de la cascada de regresores completa sería una tarea costosa tanto computacionalmente como en tiempo requerido, además de necesitar una considerable cantidad de imágenes para ello, la propia librería aporta el estimador ya entrenado para encontrar 68 puntos faciales como se muestra en la Figura 4.2.

Cuando los puntos de referencia hayan sido localizados en la imagen, será una tarea sencilla realizar las transformaciones convenientes explicadas anteriormente, como la rotación o la inclinación para ajustar la cara a las condiciones necesarias. Cuando se lleven a cabo estas transformaciones en función de los puntos de la cara, se recortará la región donde se encuentra la cara para dejarla preparada para la extracción de las características.

De esta forma, se muestra un ejemplo en la Figura 4.3, donde se observa una imagen original, otra con los puntos de referencia detectados y una última tras haber sido ajustada a las condiciones correctas.

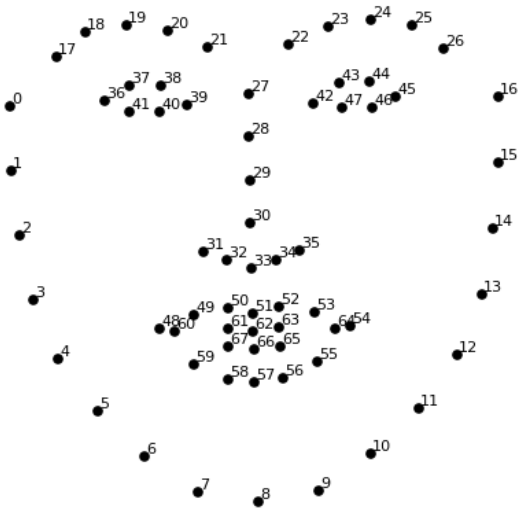
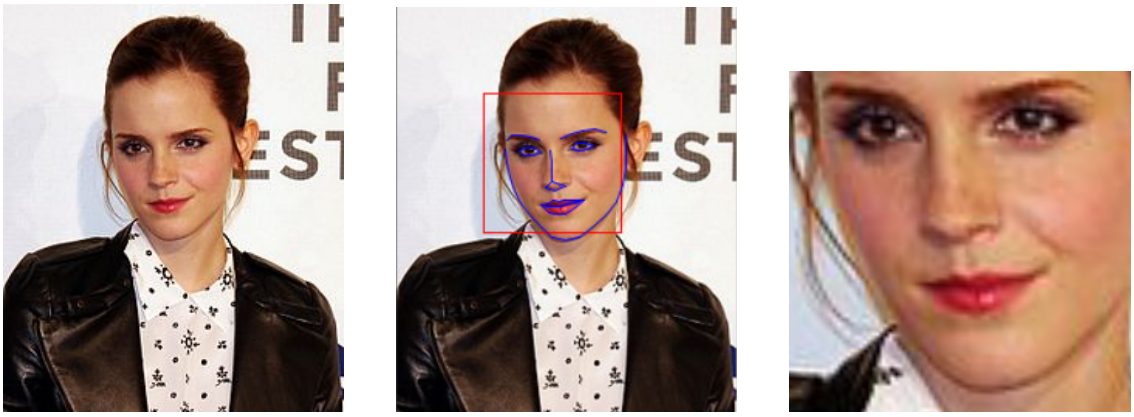


Figura 4.2 Los 68 puntos de referencia que serán buscados en cada imagen [41].



(a) Imagen original. **(b)** Puntos de referencia detectados. **(c)** Imagen preprocesada.

Figura 4.3 Ejemplo de preprocesamiento de una imagen.

4.2.3 Extracción de características

Como se ha estudiado anteriormente, una vez se tiene la imagen debidamente ajustada y centrada, el siguiente paso consiste en obtener las medidas de la cara, es decir, obtener unos datos que describan de forma adecuada la cara, reduciendo la dimensionalidad de los datos de la imagen. Para ello, se ha visto que existe una gran variedad de algoritmos y métodos y, para este proyecto, se va a utilizar una red neuronal convolucional profunda. Concretamente, esta CNN extraerá un vector de 128 valores a partir de cada imagen que se le muestre [41].

Evidentemente, aunque el entrenamiento de una CNN profunda solo sea necesario una vez, requiere un enorme número de iteraciones y una base de datos de imágenes lo suficientemente extensa como para poder llegar a generar mediciones de cualquier cara. Por este motivo, la red neuronal dedicada a la extracción de características que se va a usar en este proyecto será una proporcionada por OpenFace [42] que ya ha sido previamente entrenada con un conjunto de imágenes de rostros para ser capaz de obtener el vector de características y que, a su vez, ha sido implementada en la librería Face Recognition.

Aunque este proceso no se va a llevar a cabo en este proyecto, se sabe que esta red ha sido entrenada haciendo uso del método de los triplets anteriormente explicados, que están compuestos por dos imágenes de una misma persona y una tercera de otra persona distinta que ayude a distinguir y fijar ciertos márgenes entre estas.

Por tanto, haciendo uso de esta CNN profunda, bastará con presentarle la imagen previamente procesada y devolverá un vector de 128 valores que, realmente, no tienen un significado físico como podría ser la distancia entre los ojos por ejemplo, aunque en la práctica no es necesario conocer su significado, mientras que la representación sea correcta. Según el autor de la librería Face Recognition, este modelo presenta una precisión del 99,38 %, por lo que se puede decir que se perderá muy poca información antes de llegar a la última etapa, la comparación y clasificación de estos datos.

4.2.4 Comparación y clasificación

Para la última fase del sistema de reconocimiento facial será necesario comparar el vector de características que se corresponde con la cara en cuestión con las analizadas previamente para identificar si se trata de una persona conocida o si, por el contrario, se trata de una cara desconocida.

Para este proyecto, se ha decidido que esta tarea será realizada haciendo uso de un Perceptrón Multicapa (MLP) que, a diferencia de las redes neuronales usadas en los pasos anteriores, esta sí que va a ser configurada y entrenada personalmente. Esta tarea va a ser llevada a cabo gracias a la librería de Python Scikit-Learn, que cuenta con un módulo dedicado a las redes neuronales y, en concreto, funciones para simular clasificadores de tipo MLP.

Como las características que se deben tener en cuenta a la hora del diseño del MLP ya se han comentado en el Capítulo 3, este apartado se fijará en la descripción concreta del modelo utilizado.

En primer lugar, el MLP estará compuesto por un cierto número de neuronas en la capa oculta con la misma función de activación. Como se explicó en la Subsección 3.4.1, el algoritmo de aprendizaje hace uso de la derivada de la función de activación, por lo que esta debe ser derivable. En la práctica, es muy común utilizar funciones derivables con forma similar a una función escalón, entre las que se puede destacar la función tangente hiperbólica, la función logística o incluso la función rampa, que serán objeto de estudio en el Capítulo 5 para decidir cuál es la más adecuada.

Por otro lado, habrá que fijar las dimensiones del MLP, es decir, el número de capas que lo componen y, a su vez, el número de neuronas en cada una de ellas. Para empezar se va a fijar un valor por defecto de 100 neuronas por capa oculta, para hacer una serie de experimentos, donde se estudiará la función de activación y el número de capas ocultas del MLP, ya que a pesar de que existen numerosas aplicaciones que con una única capa se obtienen buenos resultados, se van a hacer una serie de pruebas para comprobarlo, como será la validación cruzada. Finalmente, se hará un último experimento para comprobar si el sistema presenta mejores o peores resultados variando el número de neuronas por capa y fijar este último parámetro.

4.3 Interfaz del sistema

El sistema de reconocimiento facial que será implementado en este proyecto consistirá en una interfaz de línea de comandos que ofrece varias opciones en el momento de ejecutarla.

Concretamente, este programa será estructurado en dos ficheros en formato Python, cuyo contenido se muestra en el Apéndice A. El primero se llamará `inicio.py` que será el que muestre las opciones al ser ejecutado. El segundo fichero se llamará `funciones.py` y estará compuesto por una serie de funciones declaradas para la implementación de los algoritmos explicados. El primero de los ficheros hará uso del segundo cuando sea necesario.

4.3.1 Comportamiento de la interfaz

Durante el diseño de la interfaz se ha pretendido que muestre en todo momento salidas sencillas y claras, por lo que solamente habrá que introducir los números de las opciones que se muestran por pantalla para

elegirlas.

Al ejecutar el fichero de inicio, se obtiene la respuesta mostrada en la Figura 4.4, donde se enumeran varias opciones: crear un modelo, ver los nombres disponibles y reconocimiento facial en imágenes y vídeo. Además, como último se informa de que si se introduce cualquier otra entrada, el programa se cancelará. En todas las pantallas de la interfaz donde se muestra un menú, el programa ha sido depurado para que en el caso de que no se introduzca una opción que tenga sentido, se detenga la ejecución o muestre un mensaje de error.

```
Elija una opción:
1) Crear modelo.
2) Ver nombres disponibles.
3) Reconocimiento facial en imágenes.
4) Reconocimiento facial en vídeo.
Otro para salir.
```

Figura 4.4 Introducción al menú de la interfaz.

En los siguientes apartados se hará un repaso de cada opción de la interfaz.

Crear modelo

Al seleccionar esta opción, el programa accederá al directorio que contiene todas las imágenes que conforman la base de datos. A medida que va analizando los directorios que almacenan las imágenes, se irá mostrando por pantalla tal y como se muestra en la Figura 4.5.

Cuando haya analizado cada directorio y obtenido los vectores de características de las caras contenidas en estas imágenes, se creará un modelo de clasificador de tipo MLP en el directorio “Model”. Cuando se complete el proceso de entrenamiento, se guardarán los parámetros de este modelo para poder cargarlos en el momento de analizar imágenes para evitar procesos de entrenamiento cada vez que se inicie el programa.

```
Analizando directorios:
1) Angelina Jolie
2) Avicii
3) Beyonce
4) Brad Pitt
5) Britney Spears
6) Cristiano Ronaldo
7) David Beckham
8) David Bisbal
9) David Guetta
10) Donald Trump
11) Emma Watson
12) Lebron James
13) Messi
14) Naomi Campbell
15) Nicki Minaj
16) Oprah Winfrey
17) Robin Williams
18) Sergio Ramos
19) Shakira
20) Usain Bolt
21) Will Smith
Clasificador MLP creado.
```

Figura 4.5 Comportamiento de la interfaz durante la creación de un modelo de MLP.

Ver nombres disponibles

Como su propio nombre indica, esta opción mostrará la lista de los nombres de las personas que el sistema es capaz de reconocer.

Esta opción es posible ya que en el mismo momento que es creado el clasificador MLP, también se guarda en un archivo la lista de nombres de las personas con las que se ha entrenado el modelo, debido a que serán necesarios cuando se quiera representar en las imágenes que sean analizadas. La forma en que se muestra la lista de nombres disponibles es muy similar a la mostrada a la hora de crear el MLP.

Reconocimiento facial en imágenes

Al seleccionar la tercera opción que muestra la interfaz, el sistema escanea todas las imágenes que se encuentran en un directorio llamado "Images", que será el directorio fuente de imágenes sin analizar, es decir, no usadas para entrenar el MLP. Al elegir esta opción se muestra por pantalla la lista de imágenes contenidas en dicho directorio (ver Figura 4.6).

Además de elegir las imágenes individualmente, existe la opción de analizar todas las imágenes del directorio de una vez si se introduce el comando que hace referencia a todas (`ALL`). Cuando se selecciona una opción correcta, el sistema hará uso de todos los algoritmos explicados y del MLP que haya sido creado previamente. Si no se hubiera creado dicho modelo, se generaría un mensaje de error. Si todo se lleva a cabo sin errores, se creará una nueva imagen que representará la imagen original con un recuadro marcando los rostros que aparezcan en ella, sobre el que se encontrará el nombre de la persona en caso de que la haya identificado o Unknown si no lo ha hecho. Esta imagen será almacenada en un directorio llamado "Images", perteneciente al directorio "Output".

```
Elija una opción:
1) 1.jpg
2) 2.jpg
3) cr7messi.jpg
4) guetta.png
5) Messi-Cristiano.jpg
6) shakira.jpg
Introduzca 'ALL' para todos.
```

Figura 4.6 Lista de imágenes mostradas con la tercera opción del menú.

Reconocimiento facial en vídeo

El funcionamiento de la última opción presenta un comportamiento similar al caso anterior. La principal diferencia es que al ser seleccionada, dará la opción de elegir que el origen del vídeo: que sea obtenido en tiempo real a través de una cámara web, o a partir de un archivo de vídeo ya existente.

```
Elija una opción:
1) Tomar de la cámara Web.
2) Tomar un vídeo existente.
```

Figura 4.7 Opciones de reconocimiento facial en vídeo.

Al seleccionar la primera opción, se mostrará por pantalla una ventana con la imagen captada por la cámara en tiempo real. Sin embargo, parece que el tiempo de procesamiento de las imágenes hace que el vídeo representado no sea totalmente fluido. Para detener la captura de imágenes y cerrar la ventana, bastará con presionar la barra espaciadora.

Si por el contrario se elige la segunda opción, el comportamiento del programa será muy similar al caso de reconocimiento facial en imágenes, variando la carpeta de la que tome la lista de vídeos, que se llamará "Videos". Al igual que para las imágenes, el resultado será almacenado en un nuevo archivo en una carpeta llamada "Videos" dentro del directorio "Output", cuya ruta será especificada cuando la creación del archivo haya sido completada.

5 Validación experimental

En este capítulo se entrará en detalle de cada experimento que se van a realizar con el sistema propuesto, así como la evaluación e interpretación de los resultados obtenidos.

5.1 Base de datos

Lo primero que se debe hacer para poder llevar a cabo el entrenamiento de una red neuronal, es obtener una base de datos. A partir de esta, se tomará la información que podrá ser utilizada para entrenamiento o para realizar una comprobación una vez la red neuronal haya sido preparada debidamente. En la actualidad, existen una gran cantidad de repositorios que disponen de diferentes conjuntos de datos y que ofrecen la oportunidad de descargarlos gratuitamente para su uso en sistemas de Machine Learning. Un ejemplo de ello podría ser el repositorio de Machine Learning UCI [43]. Además, existen librerías de Python como la propia Scikit-Learn que también ofrecen datasets para practicar cargándolos de forma fácil y rápida.

Sin embargo, para el caso que ocupa este proyecto, se ha decidido adquirir una serie de imágenes de diferentes personas famosas como actores, deportistas o músicos. Cada imagen ha sido tomada a través de Internet. Para ello, el único criterio que se ha tomado ha sido que la imagen contenga únicamente a la persona en cuestión y que se pueda apreciar su rostro de forma clara, aunque también se han añadido algunas imágenes en las que esa persona pueda aparecer con diferentes peinados o con complementos, como podrían ser sombreros o gafas. De esta forma, se han recopilado aproximadamente unas 30 imágenes de cada una de las 21 personas famosas que han sido elegidas para probar el sistema, en las que aparecen con diferentes expresiones faciales, como algunas sonriendo y otras con un rostro más serio. Además, entre las personas elegidas, hay diferentes colores de piel para comprobar si el sistema se comporta mejor o peor en ciertas condiciones.

Las imágenes de cada persona han sido numeradas con un orden aleatorio y almacenadas en un directorio con el nombre de dicha persona, el cual servirá además de fuente para que el programa pueda tomar un nombre concreto a la hora de identificar el rostro.

Finalmente, todos estos directorios se encontrarán en una carpeta con nombre “Source” y que deberá situarse en el mismo directorio el programa en lenguaje Python.

5.2 Método empleado

Para realizar experimentos con la red neuronal y evaluar sus resultados, se va a recurrir a la técnica denominada *validación cruzada* o cross-validation. Esta se trata de un método estadístico dedicado a evaluar y comparar algoritmos y consiste en tomar el conjunto de datos disponible y dividirlo en dos partes, que serán denominadas datos de entrenamiento y datos de validación o de prueba. Como sus propios nombres indican, el primer subconjunto será utilizado para llevar a cabo el entrenamiento de la red neuronal, y el segundo, generalmente de menor tamaño (aproximadamente un 20 % del conjunto total de datos) se destinará a realizar pruebas o análisis para comprobar los resultados del entrenamiento [44].

La técnica de la validación cruzada es comúnmente utilizada para estimar el comportamiento generalizado de un modelo de clasificador y, consecuentemente, comparar el comportamiento de varios algoritmos o de diferentes variantes de uno mismo. Por ello, sirve de utilidad para elegir un modelo que se adapte a las condiciones del problema en concreto.

Además, los algoritmos de validación cruzada permiten evitar un problema muy común a la hora de entrenar los modelos de clasificación. Si se lleva a cabo un entrenamiento con la totalidad de los datos disponibles, cuando se vuelvan a utilizar los mismos datos para obtener las predicciones, probablemente se obtendrán buenos resultados. Sin embargo, es posible que el clasificador pierda efectividad cuando en el futuro se prueben datos que no hayan sido vistos previamente. Este fenómeno se conoce como *overfitting* o sobreajuste [45].

Dentro de la validación cruzada existen diferentes métodos, como pueden ser el K-Fold, que consiste en dividir los datos en k pequeñas particiones y dejar una de ellas para la validación, repitiendo este proceso tantas veces como particiones se definan; o el LOOCV (siglas de Leave-One-Out Cross-Validation), que sería un caso particular del K-Fold, donde el valor de k coincide con el número total de datos disponibles [45].

Para el caso concreto de este proyecto, se ha optado por una variante de validación cruzada aleatoria. Este método consiste en tomar cierto porcentaje de los datos para dedicarlo al entrenamiento, y el resto para la validación, de modo que la división en ambos subconjuntos se lleve a cabo de forma aleatoria, como puede verse representado en la Figura 5.1.

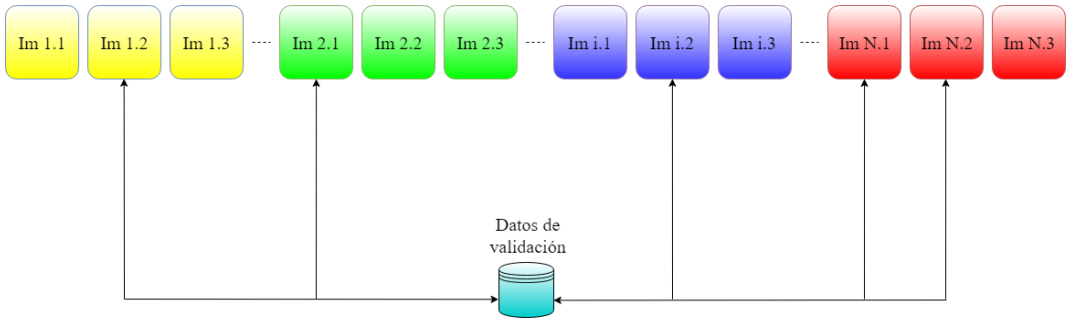


Figura 5.1 Validación Cruzada con selección aleatoria.

Este proceso de división de la base de datos, entrenamiento y validación tendrá que ser llevado a cabo de forma repetida, con el objetivo de obtener diferentes valores de error (o precisión) en cada iteración para luego calcular la media aritmética. Además, esto será realizado en paralelo para diferentes sistemas y, de esta forma, será posible obtener una buena aproximación de la precisión general de cada uno de ellos, con el fin de compararlos y elegir la topología de red neuronal más adecuada. De esta forma, se procede a explicar el procedimiento seguido para llevar a cabo dicha técnica.

En primer lugar, será necesario cargar la base de datos. Para ello, habrá que acceder a los directorios y, dentro de cada uno de estos, a las imágenes que se encuentran en ellos.

Una vez obtenido el vector de codificación de cada rostro de las imágenes y, a su vez, las etiquetas correspondientes a cada uno de estos rostros, se crearán dos archivos en los que se encontrarán almacenados estos parámetros, evitando así procesar toda la base de datos cada vez que se quisiera realizar estas pruebas.

El siguiente paso será la creación de los clasificadores de tipo Perceptrón Multicapa. Para llevar a cabo la validación experimental, se crearán varios clasificadores con distintas características, con el objetivo de comparar sus comportamientos y elegir el más adecuado.

En una primera instancia, se han creado tres clasificadores distintos. En primer lugar, se ha definido un Perceptrón Multicapa de una capa oculta, con 100 nodos, además de las capas de entrada con 128 nodos, uno para cada valor de los vectores que definen las características del rostro, y la de salida, que presenta una función de salida de tipo Softmax, definida en la Ecuación (5.1). Esta se encarga de reducir la dimensionalidad de un vector K-dimensional de entrada $\mathbf{x}$. Posteriormente, han sido creados otros dos clasificadores de tipo Perceptrón Muticapa, con dos y tres capas ocultas, del mismo número de nodos: cien en cada una de ellas. La estructura de estos será configurada de diferentes formas en los experimentos para llegar a una arquitectura final con la que se obtengan los mejores resultados.

$$\sigma(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \qquad j = 1, \ldots, K \tag{5.1}$$

Donde x_j es la entrada a la neurona j de la capa de salida y K es el número de clases o, en este caso, de personas de la base de datos, es decir, en este caso el valor de K es 21.

Con la base de datos disponible para poder ser manipulada adecuadamente y los clasificadores para realizar experimentos con ellos, se procede a hacer la división de los datos, para formar los dos subconjuntos: uno para utilizarlo a la hora de entrenar cada uno de los modelos de red neuronal, y otro para llevar a cabo la posterior validación de los mismos.

De esta forma, se entrena cada modelo con los datos destinados a ello y, tras esto, se van a realizar una serie de experimentos que se describen a continuación.

5.2.1 Experimento 1: Evaluación de función de activación

La primera prueba que será realizada estará destinada a elegir una función de activación para la capa o capas ocultas. Para llevarlo a cabo, se va a comparar el comportamiento de tres MLP con una capa oculta cada uno, pero con diferentes funciones de activación (ver Figura 5.2).

El primero presentará la función de activación rampa o ReLU, definida en la Ecuación (5.2).

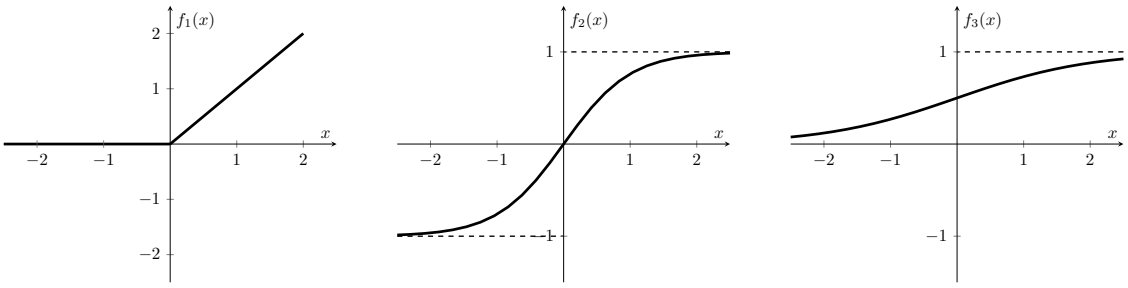
$$f_1(x) = \max(0,x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{5.2}$$

El segundo contará con la función tangente hiperbólica, definida en la Ecuación (5.3).

$$f_2(x) = \tanh(x) = \frac{\operatorname{senh}(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.3}$$

Finalmente, el tercero será implementado con la función logística, que se define en la Ecuación (5.4).

$$f_3(x) = \frac{1}{1+e^{-x}} \tag{5.4}$$



(a) Función ReLU. **(b)** Función tangente hiperbólica. **(c)** Función logística.

Figura 5.2 Funciones que serán evaluadas como función de activación del MLP.

Una vez están definidos los tres MLP, con un estado inicial similar para que los resultados del estudio sean lo más precisos posible, será hora de hacer la prueba. Como se ha explicado anteriormente, se va a dividir el conjunto de datos en uno de entrenamiento y otro de prueba. Tras entrenar los tres modelos con el primer grupo, se harán dos mediciones. En primer lugar, se medirá el coeficiente de determinación R^2 de la predicción y, a continuación, la media de la precisión obtenida para el caso correcto.

Coeficiente de determinación

Para hacer la primera medida, será necesario describir el coeficiente en que se basa. El coeficiente de determinación R^2 se puede definir como el estadístico que proporciona la medida de variabilidad de una variable Y, debida a la variabilidad de la variable explicativa X [46].

Matemáticamente, este parámetro se define en la Ecuación (5.5).

$$R^2 = 1 - \frac{\sigma_r^2}{\sigma^2} \tag{5.5}$$

Donde σ_r^2 y σ^2 representan la varianza residual y la varianza de la variable Y, respectivamente, y se definen en la Ecuación (5.6) y en la Ecuación (5.7).

$$\sigma_r^2 = \frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{N} \tag{5.6}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N}(y_i - \bar{y})^2}{N} \tag{5.7}$$

Donde y_i se corresponderá con el valor de la etiqueta correspondiente al rostro, siendo $\bar{y}$ su valor medio que se calcula según la Ecuación (5.8), $\tilde{y}_i$ representará el valor predicho por el modelo y N será el número total de datos.

$$\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N} \tag{5.8}$$

En este caso, la variable X representará a los datos descriptivos de cada rostro, y la variable Y representa la etiqueta numérica que identifica a la persona a quien pertenece dicho rostro. Haciendo uso de este parámetro, será posible medir la bondad de las predicciones que hagan los clasificadores. Si R^2 toma un valor igual a la unidad, indicará que el modelo se ajusta totalmente a la variabilidad de Y, por lo que no se producirá error alguno. Dicho de otra manera, cuanto más se acerque el valor de R^2 a 1, mayor porcentaje de acierto conseguirá el modelo en cuestión.

Para obtener una buena estimación de este parámetro, en este proyecto cada MLP será entrenado con los mismos datos y se calculará el coeficiente con el grupo de datos de prueba correspondiente. No obstante, este proceso será realizado hasta un total de 100 iteraciones para variar los datos con los que se entrena y se comprueba, siempre de forma aleatoria. De esta forma se obtendrá un total de 100 valores de R^2, a partir de los cuales se calculará su media aritmética para obtener un único valor promedio que represente los resultados del experimento.

Adicionalmente, cabe decir que el valor del coeficiente R^2 coincide con el porcentaje de aciertos del modelo, por lo que la interpretación de los resultados obtenidos resultará una sencilla tarea: cuanto mayor valor de R^2, más aciertos habrá conseguido el modelo en cuestión.

Promedio de la precisión

La otra prueba que será realizada en paralelo con el cálculo del coeficiente R^2 consistirá en medir el porcentaje de similitud que el clasificador presenta ante la etiqueta correcta, es decir, qué porcentaje devuelve el modelo para la persona correspondiente a esa imagen, independientemente de si existe una con más porcentaje (y por tanto el modelo falle).

El procedimiento consistirá en entrenar cada modelo con el subconjunto de entrenamiento, al igual que en el anterior, para que todos reciban exactamente los mismos datos.

Cuando hayan sido entrenados, se tomará cada uno de los datos del conjunto de validación y se le presentará a los diferentes modelos de MLP. Estos procesarán la información de entrada y calcularán el porcentaje de similitud de esta entrada para cada una de las clases que ha aprendido en el entrenamiento, es decir, el MLP devolverá el porcentaje de parecido entre los datos del rostro de entrada y las diferentes personas que ha aprendido a diferenciar con el entrenamiento. Este porcentaje es calculado haciendo uso de la función de pérdida de entropía cruzada (Cross-Entropy loss), que mide el rendimiento de clasificación de un modelo siguiendo la Ecuación (5.9).

$$H(p,q) = -\sum_{x} p(x)\ln(q(x)) \tag{5.9}$$

Donde $p(x)$ representa la probabilidad deseada para la clase x, y $q(x)$ representa la probabilidad real obtenida que, en este caso, será la salida de la capa Softmax del MLP. En la práctica, como $p(x)$ no suele ser conocida, se suele suponer que todas las N observaciones son equiprobables, es decir, $p(x) = 1/N$, por lo que la entropía cruzada será calculada como:

$$H(p,q) = -\frac{1}{N}\sum_{x} \ln(q(x)) \tag{5.10}$$

En este caso se habrán aprendido un total de 21 personas diferentes, por lo que cada MLP devolverá un vector de 21 elementos, uno por cada porcentaje de parecido entre el dato de entrada y cada rostro.

Luego, se extraerá de este vector el resultado obtenido para la que se conoce a priori que es la clase correcta, es decir, se tomará el porcentaje obtenido para la persona de la que en realidad se trata. Idealmente, este valor debería ser el mayor del vector devuelto por el MLP, pero en las ocasiones en las que la predicción de un modelo sea errónea, esto no ocurrirá.

Este proceso se hará para todos los datos de validación y, al igual que el proceso anterior, será realizado un total de 100 iteraciones para obtener un valor promedio de cada MLP, con el fin de compararlos y decidir cuál se comporta mejor.

El objetivo de este experimento no es únicamente que el modelo acierte, lo cual está contemplado en el anterior, sino que presente el máximo porcentaje de similitud posible.

5.2.2 Experimento 2: Determinación del número de capas ocultas

Una vez se haya realizado el primer experimento y se haya decidido la función de activación en función de sus resultados, se realizará el experimento que compare los 3 MLP con diferentes números de capas ocultas.

Tras haber revisado el experimento anterior, la explicación de este resultará mucho más sencilla. El procedimiento de este experimento será muy similar al seguido en el primero. Se realizarán dos pruebas en paralelo: el cálculo del coeficiente de determinación R^2 y la medida de la precisión obtenida para el resultado correcto. La principal diferencia con el anterior, será que este experimento será realizado con una función de activación ya fijada (la elegida a partir de los primeros resultados) y que los tres MLP ahora variarán en sus dimensiones, teniendo una, dos y tres capas ocultas, respectivamente, con 100 nodos o neuronas cada una.

5.2.3 Experimento 3: Variación del número de neuronas

El último experimento seguirá el mismo procedimiento que los dos anteriores. Sin embargo, en este caso ya se habrán fijado la función de activación y el número de capas ocultas del MLP. Con estos parámetros, se hará una comparación de los resultados obtenidos para el coeficiente de determinación R^2 y para la estimación que devuelve el MLP. En función de estos resultados se decidirá si el número de neuronas fijado al inicio puede ser reducido o, por el contrario, debe ser incrementado para conseguir mejores resultados.

5.3 Resultados e interpretación

En esta sección se hará un análisis de los resultados de los experimentos descritos, para tomar una decisión acerca de la topología o estructura más adecuada para el diseño del Perceptrón Multicapa.

5.3.1 Experimento 1

El objetivo del primer experimento es fijar una función de activación adecuada a este caso. Para ello se ha seguido el procedimiento explicado en la Subsección 5.2.1 y se han obtenido los resultados de la Tabla 5.1.

Tabla 5.1 Resultados del experimento 1.

Función $f(x)$	**Coeficiente de determinación** R^2 **(%)**	**Similitud (%)**
ReLU	99,2700	95,2438
Tangente hiperbólica	99,2700	96,5802
Logística	99,1970	95,2438

Haciendo un estudio los resultados obtenidos, se observa que la función logística ha presentado un valor de R^2 ligeramente menor al obtenido para las funciones ReLU y tangente hiperbólica. Por la parte de la segunda medida, la función ReLU y la función logística han presentado el mismo porcentaje de similitud, encontrándose por debajo del resultado para la tangente hiperbólica.

Con estos resultados como referencia, resulta evidente que la elección de la función de activación del MLP se corresponderá con una tangente hiperbólica ya que, aunque no se presenten diferencias abismales, ha quedado claro con este experimento que, para este caso concreto, la tangente hiperbólica es la que ha presentado un mejor comportamiento.

Por tanto, la función elegida es la tangente hiperbólica y, a partir de este resultado, se llevará a cabo el siguiente experimento.

5.3.2 Experimento 2

Una vez se ha fijado la función de activación como tangente hiperbólica, se procede a realizar el segundo experimento, en el que se va a llevar a cabo el mismo procedimiento de comparación, pero esta vez comparando

varios MLP con diferente número de capas ocultas. Los resultados de este experimento se muestran en la Tabla 5.2.

Tabla 5.2 Resultados del experimento 2.

Número de capas ocultas	Coeficiente de determinación R^2 (%)	Similitud (%)
1	99,2993	96,8010
2	99,0438	97,3120
3	98,9489	97,2883

Como se puede apreciar, se hace presente el hecho de que cada vez que se le añade una capa oculta al MLP, el valor promedio del coeficiente de determinación se reduce, lo que significa que el número de aciertos se hace cada vez menor. Por otro lado, el valor promedio de la similitud resultante con una capa es aproximadamente 0,5 % menor que con dos capas. Si se tuviera en cuenta únicamente esta prueba, sería muy posible que un usuario eligiera un Perceptrón Multicapa con una estructura definida por dos capas ocultas. Sin embargo, es necesario definir unas prioridades según la aplicación del modelo. Para el caso de este proyecto, en el ámbito del reconocimiento facial va a resultar prioritario el número medio de aciertos de un sistema, frente a la probabilidad media de parecido. Esto es fácil de entender con un ejemplo: si se dispone de un sistema de reconocimiento facial, es preferible que detecte de quién se trata una persona, aunque presente un menor porcentaje de parecido, a que cuando acierte aporte un mayor porcentaje de parecido, pero que en general falle un mayor número de veces el reconocimiento.

Por tanto, teniendo como referencia estos resultados, se toma la decisión de que el MLP contará con una única capa oculta, por lo que en este caso se confirma lo que se adelantaba en la Subsección 4.2.4, donde se decía que para una una gran cantidad de aplicaciones se obtienen buenos resultados con un MLP de una única capa oculta. Además, el hecho de que el MLP presente una única capa oculta no solo presenta los mejores resultados según este experimento, sino que además el coste computacional, sobre todo a la hora del entrenamiento, será menor para una capa que para más de una.

5.3.3 Experimento 3

Como última prueba para fijar la estructura final del MLP que se utilizará en el sistema de reconocimiento facial implementado en este proyecto, se va a comprobar cómo varían los resultados en función del número de neuronas que componen la capa oculta.

Para ello, se hace un primer ensayo con 50, 100 y 150 neuronas para cada MLP, respectivamente. Los resultados se representan en la Tabla 5.3, donde se observa una mejoría del comportamiento del MLP cuanto mayor número de neuronas compongan la capa oculta, tanto en la prueba del coeficiente de determinación como en la similitud obtenida. Por este motivo, se decide realizar un nuevo ensayo del experimento, esta vez cambiando el MLP de 50 neuronas, cuyos resultados son peores, por uno de 200 neuronas, para comprobar si los resultados siguen mejorando.

Tabla 5.3 Resultados del primer ensayo del experimento 3.

Número de neuronas	Coeficiente de determinación R^2 (%)	Similitud (%)
50	99,3869	96,3272
100	99,4818	96,8585
150	99,4891	97,0910

Tras realizar el segundo ensayo, se obtienen los resultados que han sido reflejados en la Tabla 5.4. En este caso, se ha obtenido que para una capa de 200 neuronas el coeficiente de determinación es mayor que para 100 neuronas, pero menor que el obtenido para el MLP de 150 neuronas. Por tanto, los resultados muestran la gran posibilidad de que se encuentre un pico alrededor de las 150 neuronas, a partir del cual el número de aciertos del MLP empezará a disminuir. Respecto a la similitud obtenida, se sigue obteniendo mayores porcentajes aumentando el número de neuronas. Sin embargo, siguiendo el mismo razonamiento que para el experimento anterior, se priorizará el caso que mayor número de aciertos presente.

Tabla 5.4 Resultados del segundo ensayo del experimento 3.

Número de neuronas	Coeficiente de determinación R^2 (%)	Similitud (%)
100	99,1015	96,8332
150	99,4161	97,0538
200	99,4088	97,1036

Por tanto, teniendo como referencia estos resultados, junto con los de los anteriores experimentos, se llega a la conclusión de que el Perceptrón Multicapa que será usado para este proyecto contará con una única capa oculta de 150 neuronas con la tangente hiperbólica como función de activación. Por otro lado, es posible que exista una solución en la que variando un poco más el número de neuronas se consiga mejorar el coeficiente de determinación R^2, pero debido a que la mejoría sería muy pequeña, se fijará el valor de 150 neuronas. Por este motivo, esta solución estará muy cerca de ser la óptima para este problema.

5.4 Representación gráfica de los resultados

A través de los resultados de los experimentos anteriores ha sido posible fijar los parámetros del MLP que se encargará de clasificar las caras detectadas. Además, estos han sido elegidos de forma que en las pruebas se ha obtenido aproximadamente un total del 99,40 % de aciertos, por lo que se podría afirmar que el sistema, en general, reconoce rostros con una considerable efectividad.

Para mostrar unos resultados más atractivos, se han tomado algunos ejemplos de imágenes que han sido analizadas por el sistema. Por supuesto, estas imágenes no han sido utilizadas para llevar a cabo el previo entrenamiento del MLP.

En el primer ejemplo representado en la Figura 5.3 se muestra una imagen analizada por el sistema en la que aparecen cuatro rostros. Concretamente, dos de ellos aparecen en primer plano y los otros dos se encuentran detrás y se ven un poco más borrosos debido al modo de enfoque.

Como se puede comprobar, el sistema ha conseguido detectar a la perfección los cuatro rostros. Además, la capacidad de reconocimiento en esta imagen supera todos los objetivos, debido a que todas las personas cuyas caras han formado parte del entrenamiento, han sido reconocidas con una importante similitud.

En primer lugar, las dos caras del primer plano han sido reconocidas con 99,93 % y 99,8 %, lo que significa que prácticamente con toda seguridad se tratan de las personas con las que han sido identificadas. Respecto a las dos caras del segundo plano, se diferencian dos casos. En primer lugar, a la derecha se ha obtenido un resultado del 99,48 % de parecido, a pesar de que la cara se muestra desenfocada en la imagen. En segundo lugar, a la izquierda, se encuentra una persona cuya cara no ha sido utilizada en el entrenamiento, por lo que el resultado coincide con lo esperado, ya que el sistema ha descrito esa cara como desconocida (Unknown).

En el segundo caso, mostrado en la Figura 5.4, se observan dos imágenes. En ambas se muestra a la misma persona con gafas de sol. Sin embargo, en la fotografía de la izquierda, el sistema logra reconocer a la persona con un 92,96 % de parecido. Sin embargo en la imagen de la derecha, el resultado ha sido erróneo. Este fenómeno puede haberse producido porque el sistema no sea efectivo cuando las personas llevan complementos tales como las gafas de sol. Sin embargo, aunque pueda ser cierto que el sistema funcione mejor cuando la cara está al descubierto, es muy posible que el error que se ve en este ejemplo haya sido producido porque haya sido necesario un mayor número de imágenes de esta persona con gafas a la hora de entrenar el MLP, ya que en la imagen de la izquierda sí que ha conseguido reconocerlo.

Para probar esta teoría, se decide hacer una nueva prueba con otra persona, de la cual han sido utilizadas para el entrenamiento 3 fotografías en las que aparecía con gafas de sol, además de imágenes con la cara descubierta. Para este caso, el resultado ha sido mucho más favorable ya que para una serie de imágenes de la misma persona con gafas de sol, el resultado siempre ha sido favorable puesto que siempre ha sido reconocida la persona correctamente. En la Figura 5.5 se representan dos ejemplos, en las que se observa claramente que el sistema ha sido capaz de identificar a la persona perfectamente teniendo gafas de sol o incluso sombrero como en el ejemplo de la derecha.

Con esta prueba, se llega a la conclusión de que la base de datos tendrá un papel muy importante a la hora de entrenar, en este caso, el MLP, ya que deberá incluir ejemplos que representen de manera adecuada los casos que se quiere ser capaz de distinguir con el sistema.

Por último, se ha querido probar hasta qué punto el sistema es capaz de detectar la cara, en función de las direcciones en las que mira una persona. Ya en la Figura 5.5 se observa que el sistema es capaz de detectar

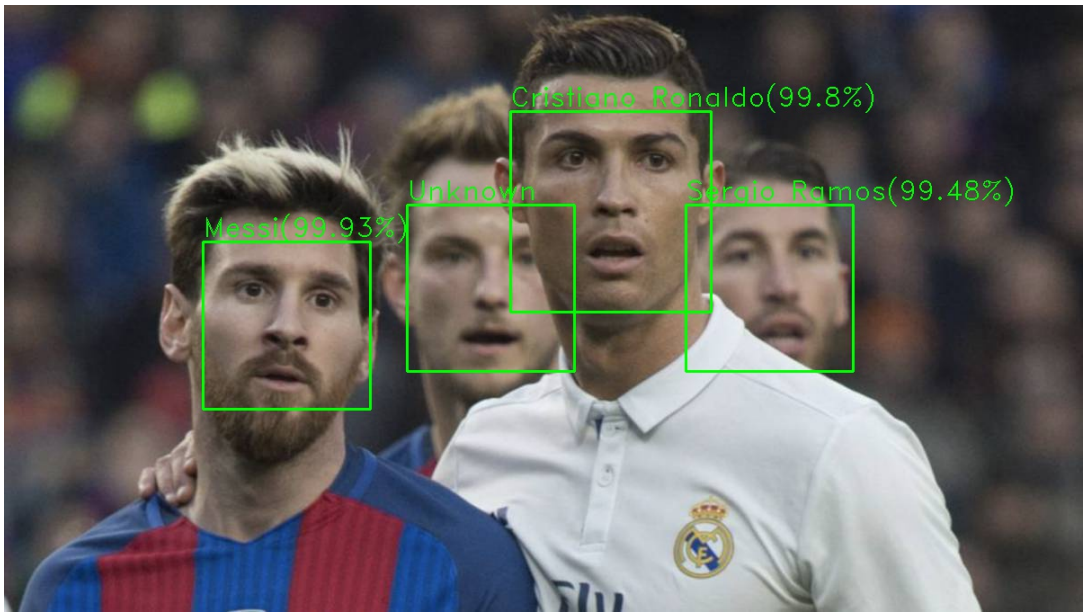
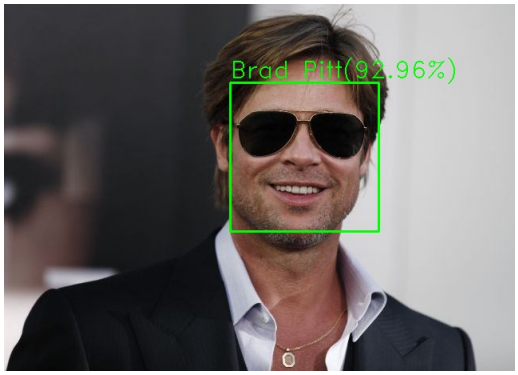
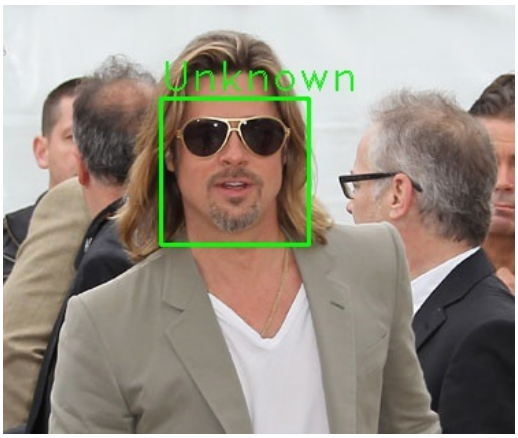


Figura 5.3 Imagen resultante tras ser procesada por el sistema.



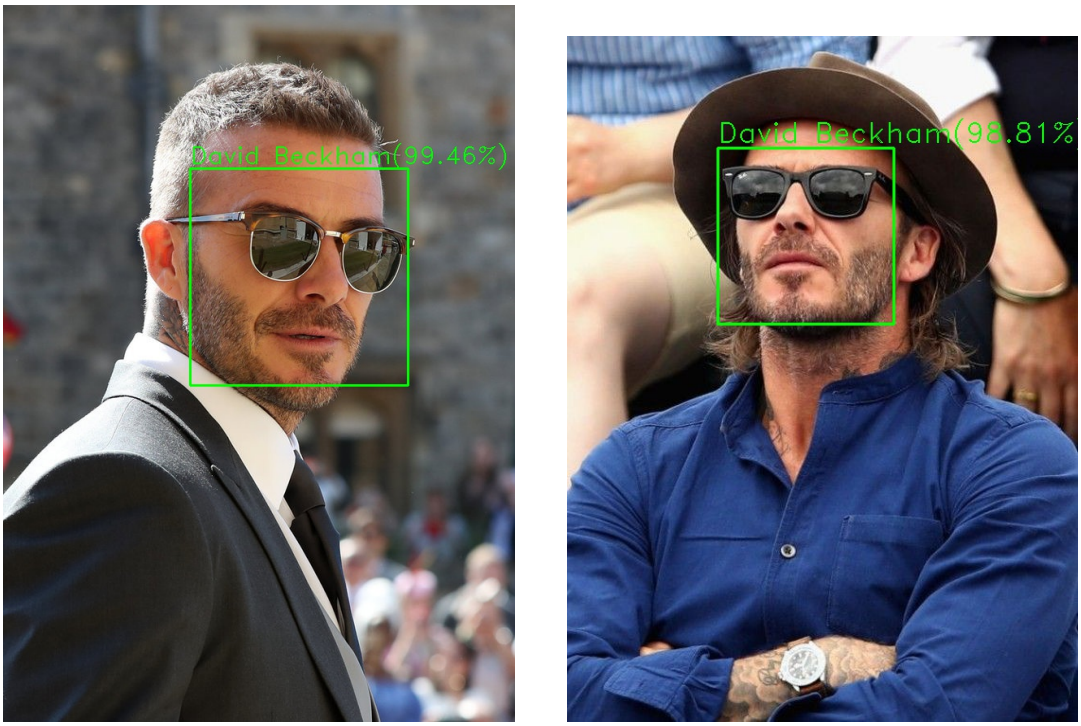
(a) Cara reconocida.



(b) Cara no reconocida.

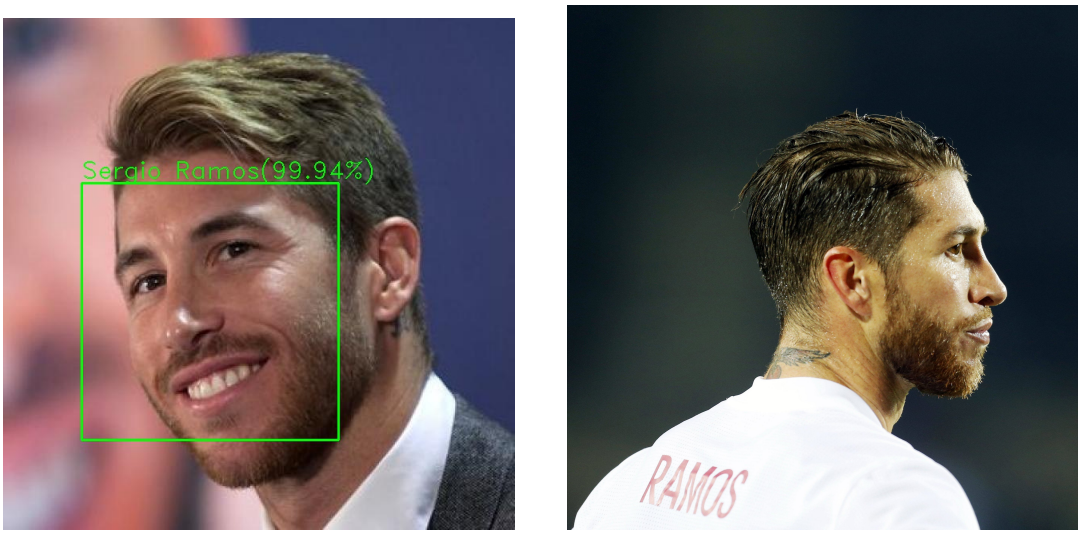
Figura 5.4 Ejemplo de imágenes con gafas de sol.

rostros con diferentes perspectivas, pero se ha decidido probar el comportamiento con una imagen donde aparece un rostro girado y otra en la que sale completamente de perfil. Esto se muestra en la Figura 5.6, donde aparece a la izquierda una imagen donde la persona ha sido detectada e identificada perfectamente a pesar de estar algo girada, gracias al preprocesado realizado para adaptar las caras antes de la extracción de características. Sin embargo, en la imagen de la derecha, se muestra a la misma persona totalmente de perfil, pero no ha sido detectada, puesto que a la hora de pasar la plantilla HOG con la forma de una cara genérica no se ha obtenido la correlación suficiente como para ser identificada como un rostro.



(a) (b)

Figura 5.5 Segundo ejemplo de fotografías con gafas de sol.



(a) Cara detectada e identificada. **(b)** Cara no detectada.

Figura 5.6 Comportamiento ante personas de perfil.

6 Conclusiones

6.1 Conclusión

Durante el desarrollo de este proyecto se ha llevado a cabo un estudio del funcionamiento de los sistemas de reconocimiento facial. Ha sido necesario realizar un análisis de algunos algoritmos específicos para las diferentes etapas que componen el proceso.

Por otro lado, se ha hecho una revisión del funcionamiento de las redes neuronales artificiales para el posterior empleo de un tipo concreto a la hora de implementar el sistema, el Perceptrón Multicapa.

Tras una serie de pruebas, destacando la técnica denominada validación cruzada, se ha conseguido definir una estructura para el MLP que hará la función de clasificador, de modo que se obtengan los mejores resultados posibles.

Finalmente, se ha comprobado el comportamiento definitivo del sistema, exponiendo algunos ejemplos y representándolos de forma gráfica, con imágenes resultantes del propio procedimiento del sistema. Como se ha demostrado, la capacidad de reconocimiento del sistema para personas que hayan sido correctamente entrenadas es bastante notable, tanto en imagen como en vídeo, aunque en este último caso no se pueda demostrar en este documento. No obstante, sí se reconoce que el tiempo de procesamiento a la hora de usar vídeos en tiempo real mediante una cámara provoca que la reproducción del vídeo no sea completamente fluida.

A pesar de este último punto, el proyecto ha conseguido cumplir sus objetivos principales, destacando sobre todo la implementación de un sistema capaz de reconocer rostros haciendo uso de redes neuronales con muy buenos resultados.

6.2 Líneas futuras

Con todos los aspectos relevantes de este trabajo revisados, se pretende terminar esta memoria con algunos conceptos que podrían ser tenidos en cuenta para mejorar en un futuro el sistema implementado.

En primer lugar, a pesar de que la interfaz por línea de comandos ha sido preparada para presentar un menú sencillo e intuitivo, siempre será más accesible para un usuario el uso de una interfaz gráfica. Por este motivo, se propone diseñar una interfaz gráfica en la que se ofrezcan unas opciones similares a un usuario de este sistema, además de que se le podrían añadir funcionalidades como seleccionar un grupo de imágenes concreto para analizar o incluso la visualización de los resultados de una manera rápida y sencilla una vez que haya sido procesada una imagen o un vídeo.

Por otro lado, se podrían hacer modificaciones en el sistema, que requerirían la elección de dos opciones distintas. En primer lugar, se podría investigar en sustituir algunos algoritmos para reducir el tiempo de procesamiento, con el objetivo de mejorar el comportamiento del sistema frente a flujos de vídeo en tiempo real como son los recogidos a través de cámaras web. Otra opción podría ser implementar un entrenamiento del sistema de tal forma que sea capaz de reconocer personas totalmente de perfil, como podría ser por ejemplo el empleo de modelos de imágenes HOG preparadas para tal misión. Sin embargo, el hecho de tener que detectar caras tanto de perfil como de frente, podría exceder el tiempo de computación, por lo que podría no ser compatible con la primera opción descrita.

Finalmente, también se plantea una tarea que podría no requerir la modificación del sistema. Se trata de llevar a cabo un entrenamiento más exhaustivo, utilizando bases de datos mayores. Esto podría mejorar, por

ejemplo, el problema de que en algunas ocasiones el sistema no sea capaz de reconocer una persona con gafas de sol, mientras que en otras situaciones sí que lo consigue.

Para terminar, tal y como se ha descrito a lo largo de este trabajo, el reconocimiento facial representa un campo en el que desde hace varias décadas se ha estado investigando y hasta la actualidad. Por tanto, es muy posible que en un futuro aparezcan algoritmos cada vez más potentes y, de la misma manera, formas de implementación de redes neuronales mucho más eficientes, por lo que será posible diseñar sistemas de reconocimiento facial mucho más rápidos, eficientes y eficaces.

Bibliografía

[1] N. Lydick, "A Brief Overview of Facial Recognition," 2007.

[2] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human faces," *J. Opt. Soc. Am. A*, vol. 4, no. 3, pp. 519–524, Mar 1987. [Online]. Available: http://josaa.osa.org/abstract.cfm?URI=josaa-4-3-519

[3] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of cognitive neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[4] R. Chellappa, C. L. Wilson, and S. Sirohey, "Human and machine recognition of faces: a survey," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 705–741, May 1995.

[5] A. K. Jain and S. Z. Li, *Handbook of face recognition*. Springer, 2011.

[6] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, Jan 2002.

[7] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001.

[8] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[9] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23–38, Jan 1998.

[10] S.-H. Lin, S.-Y. Kung, and L.-J. Lin, "Face recognition/detection by probabilistic decision-based neural network," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 114–132, Jan 1997.

[11] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[12] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 886–893.

[13] *The OpenCV Reference Manual*, 2nd ed., Itseez, April 2014.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (2nd Edition)*. Prentice Hall, 2002. [Online]. Available: https://www.amazon.com/Digital-Image-Processing-Rafael-Gonzalez/dp/0201180758?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201180758

[15] D. King, "Dlib 18.6 released: Make your own object detector!" Feb. 2014. [Online]. Available: http://blog.dlib.net/2014/02/dlib-186-released-make-your-own-object.html

[16] A. G. Marcos, F. M. de Pisón Ascacíbar, A. P. Espinoza, F. A. Elías, M. C. Limas, J. O. Meré, and E. V. González, "Técnicas y algoritmos básicos de visión artificial," *Universidad de la Rioja, España*, 2006.

[17] Z. Sufyanu, F. S. Mohamad, A. A. Yusuf, and A. N. Musa, "Feature extraction methods for face recognition," *International Review of Applied Engineering Research (IRAER)*, 2016.

[18] S. Mahto and Y. Yadav, "A Survey on Various Facial Expression Recognition Techniques," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 3, no. 11, pp. 13 028–13 031, Nov. 2014.

[19] M. Deriche, "Trends and challenges in mono and multi biometrics," in *2008 First Workshops on Image Processing Theory, Tools and Applications*, Nov 2008, pp. 1–9.

[20] I. T. Jolliffe, *Principal Component Analysis*. Springer-Verlag GmbH, 2002. [Online]. Available: https://www.ebook.de/de/product/2047838/i_t_jolliffe_principal_component_analysis.html

[21] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*. IEEE, 1991, pp. 586–591.

[22] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, July 1997.

[23] G. Ottado, "Reconocimiento de caras: Eigenfaces y fisherfaces."

[24] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face recognition with local binary patterns," in *European conference on computer vision*. Springer, 2004, pp. 469–481.

[25] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[26] Altman and N. S, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[27] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[28] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[29] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[30] J. R. Hilera González, *Redes neuronales artificiales. Fundamentos, modelos y Aplicaciones*. RA-MA S.A. Editorial y Publicaciones, 1994. [Online]. Available: https://www.ebook.de/de/product/8377686/jose_ramon_hilera_gonzalez_redes_neuronales_artificiales_fundamentos_modelos_y_aplicaciones.html

[31] A. Gibson and J. Patterson, *Deep Learning: The Definitive Guide*. O'Reilly UK Ltd., 2017. [Online]. Available: https://www.ebook.de/de/product/23640784/adam_gibson_josh_patterson_deep_learning_the_definitive_guide.html

[32] P. Veličković, "Deep learning for complete beginners: convolutional neural networks with keras," Mar. 2017. [Online]. Available: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[33] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[34] Python Software Foundation, "Python." [Online]. Available: http://www.python.org/

[35] Anaconda Inc., "Anaconda." [Online]. Available: https://anaconda.org/

[36] Travis Oliphant, “Numpy.” [Online]. Available: https://www.numpy.org/

[37] A. Geitgey, “Face recognition.” [Online]. Available: https://face-recognition.readthedocs.io/en/latest/

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[39] D. E. King, “Dlib-ml: A machine learning toolkit,” *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[40] V. Kazemi and J. Sullivan, “One millisecond face alignment with an ensemble of regression trees,” *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1867–1874, 2014.

[41] A. Geitgey, “Machine learning is fun! part 4: Modern face recognition with deep learning,” *Medium. https://medium. com/@ ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78*, 2016.

[42] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, “Openface: A general-purpose face recognition library with mobile applications,” CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[43] D. Dheeru and E. Karra Taniskidou, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[44] L. Pérez-Planells, J. Delegido, J. P. Rivera-Caicedo, and J. Verrelst, “Análisis de métodos de validación cruzada para la obtención robusta de parámetros biofísicos,” *Revista de Teledetección*, no. 44, p. 55, dec 2015.

[45] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-validation,” in *Encyclopedia of Database Systems*. Springer US, 2009, pp. 532–538.

[46] D. Ruiz Muñoz, *Manual de Estadística*, eumed.net, Ed. Universidad Pablo de Olavide, 2004.

Apéndice A
Códigos utilizados

En esta sección se recogen los códigos utilizados para implementar el sistema de reconocimiento facial del proyecto y para hacer los experimentos explicados.

A.1 Sistema de reconocimiento facial

A continuación se muestra el código en lenguaje Python correspondiente al sistema implementado en este proyecto. En el Código A.1 se muestra el fichero mediante el cual se inicia el programa. En el Código A.2 se muestra el fichero que recoge las funciones necesarias para cada opción del menú, yendo desde la creación de los clasificadores hasta el procesado de imágenes o vídeos seleccionados.

Código A.1 inicio.py.

```
Created on Thu Jun 7 17:05:18 2018

@author: Ildefonso Jiménez Silva
"""

#Se importan las funciones del fichero correspondiente
import funciones

seguir=True
while(seguir):
    #Se imprimen las opciones
    print()
    print("Elija una opción:")
    print("1) Crear modelo.")
    print("2) Ver nombres disponibles.")
    print("3) Reconocimiento facial en imágenes.")
    print("4) Reconocimiento facial en vídeo.")
    print("Otro para salir.")

    ini = input()

    #Utiliza una función dependiendo de la opción elegida
    if ini is "1":
        funciones.model_creator()
    elif ini is "2":
        funciones.print_names()
    elif ini is "3":
        funciones.select("Images/")
    elif ini is "4":
```

```
30          print("Elija una opción:")
31          print("1) Tomar de la cámara Web.")
32          print("2) Tomar un vídeo existente.")
33
34          ini = input()
35          if ini is "1":
36              funciones.video(0)
37          elif ini is "2":
38              funciones.select("Videos/")
39          else:
40              print("Opción incorrecta.")
41      else:
42          seguir=False
```

Código A.2 funciones.py.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jul 9 19:22:17 2018
4
5  @author: Ildefonso Jiménez Silva
6  """
7
8  import numpy as np
9  import sys
10 import face_recognition #Librería de reconocimiento facial
11 import os #Librería para interactuar con el entorno
12 import cv2 #Librería de visión artificial
13 from sklearn.neural_network import MLPClassifier #Clasificador MLP
14 from sklearn.externals import joblib #Para guardar/cargar clases en archivos
15
16 ################################Función 1#######################################
17
18 #Función que crea el modelo MLP
19 def model_creator():
20     #Por defecto, se tomará la carpeta Source
21     dirs=os.listdir(path='./Source/')
22
23     #Inicializando parámetros
24     encodings_final=[]
25     tags=[]
26     known_names=[]
27
28     print("Analizando directorios:")
29     for (directorio,index) in zip(dirs, range(len(dirs))):
30         #Muestra el directorio analizado
31         print(str(index+1) + ') ' + directorio)
32         load_dir(directorio, encodings_final, index, tags)
33         #Suponiendo que hay al menos un rostro correcto en cada directorio
34         known_names.append(directorio)
35
36     #Si se han obtenido algunas etiquetas (y por tanto encodings)
37     if tags:
38         #Crea y entrena el clasificador
39         clf = MLPClassifier(verbose=False, max_iter=1000, activation= 'tanh',
               hidden_layer_sizes = (100))
```

```
40          model = clf.fit(encodings_final, tags)
41
42          #Guarda el modelo de MLP entrenado y la lista de nombres
43          joblib.dump(model, './Model/MLPClassifier.pkl')
44          joblib.dump(known_names, './Model/Names.pkl')
45          print("Clasificador MLP creado.")
46      else:
47          print("No se obtuvieron rostros")
48
49
50  #Función que toma las imágenes de un directorio y obtiene la codificación sus
        rostros
51  def load_dir(directorio, encodings_final, index, tags):
52      path= './Source/' + directorio +'/'
53      #Lista con los nombres de las imágenes de la carpeta de la persona
54      file_list = os.listdir(path)
55
56      for imagen in file_list:
57          #Forma la ruta de la imagen
58          complete_path= path + imagen
59          #Carga la imagen y extrae su codificación (encodings)
60          imagen = face_recognition.load_image_file(complete_path)
61          encodings=face_recognition.face_encodings(imagen)
62          #Si se ha obtenido, se añade a la lista final de encodings y de
                etiquetas
63          if encodings:
64              encodings_final.append(encodings[0])
65              tags.append(index)
66
67
68  ################################Función 2#########################################
69
70  #Función que muestra los nombres disponibles
71  def print_names():
72
73      try:
74          #Carga el modelo y los nombres disponibles
75          model=joblib.load('./Model/MLPClassifier.pkl')
76          known_names=joblib.load('./Model/Names.pkl')
77      except Exception:
78          sys.exit("No se pudo cargar el clasificador.")
79
80      #Si ambos existen se imprime la lista de nombres
81      if known_names and model:
82          print("Los nombres disponibles son:")
83          for i in range(len(known_names)):
84              print(str(i+1) + ') ' + known_names[i])
85      else:
86          print("No hay nombres disponibles.")
87
88
89  ################################Función 3#########################################
90
91  #Función para seleccionar la imagen o vídeo que se va a analizar
92  def select(path):
93      #Comprueba si se ha elegido imágen o vídeo y se carga el directorio
94      correct_path=(path == "Images/" or path == "Videos/")
```

```
95      if(correct_path):
96          file_list=os.listdir(path)
97
98          if not file_list:
99              print("Sin archivos para analizar.")
100         else:
101             print("Elija una opción:")
102             #Muestra la lista de archivos del directorio
103             for name,i in zip(file_list,range(len(file_list))):
104                 print(str(i+1) + ') ' + name)
105
106             if path == "Images/":
107                 print("Introduzca 'ALL' para todos.")
108
109             #Lee por teclado
110             num = input()
111             is_all= (num == "ALL" or num =="all")
112
113             #Analiza las imágenes o imagen o vídeo seleccionado
114             if path == "Images/" and is_all:
115                 print("Se procesarán todas las imágenes.")
116                 for img in file_list:
117                     image(path+img)
118                 print("Completado.")
119             elif num.isdigit():
120                 if(int(num) > 0 and int(num) <= len(file_list) ):
121                     num=int(num)-1
122                     name=file_list[num]
123
124                     if path == "Images/":
125                         image(path+name)
126                     elif path == "Videos/":
127                         video(path+name)
128                         print("Completado.")
129                 else:
130                     print("Opción errónea.")
131             else:
132                 print("Opción errónea.")
133     else:
134         print("Error.")
135
136
137 #Función que analiza los rostros de una imagen
138 def image(image_input):
139     #Toma la ubicación y encoding de cada rostro de la imagen
140     image = face_recognition.load_image_file(image_input)
141     (detected_names, face_locations)=detect(image)
142
143     #Inserta la ventana de identificación en la imagen
144     insert_name(detected_names, face_locations, image)
145
146     #Pasa del formato de color de face_recognition (RGB) al de OpenCV (BGR)
147     image = image[:, :, ::-1]
148
149     #Guarda la imagen procesada
150     splited_name=os.path.splitext(image_input)
151     name_image="./Output/"+splited_name[0]+"_out"+splited_name[1]
```

```
152     cv2.imwrite(name_image, image)
153     print(name_image+" creada.")
154 
155 
156 #Función que detecta las caras de una imagen y devuelve sus nombres y
        localizaciones
157 def detect(rgb_frame):
158 
159     try:
160         #Carga el modelo
161         model=joblib.load('./Model/MLPClassifier.pkl')
162         known_names=joblib.load('./Model/Names.pkl')
163     except Exception:
164         sys.exit("No se pudo cargar el clasificador.")
165 
166     detected_names=[]
167     #Establece el umbral: mínimo parecido posible
168     prox_min=0.8
169 
170     # Encuentra todos los rostros del frame y obtiene sus encodings
171     face_locations = face_recognition.face_locations(rgb_frame)
172     face_encodings = face_recognition.face_encodings(rgb_frame)
173 
174     n_face=0
175     for face_encoding in face_encodings:
176         [proximities]=model.predict_proba(face_encoding.reshape(1,-1))
177 
178         prox_max=max(proximities)
179         if prox_max >=prox_min:
180             tag=np.argmax(proximities)
181             proximity = prox_max*100
182             name = known_names[tag] + '('+str(round(proximity,2))+'%)'
183             detected_names.append(name)
184         else:
185             name = "Unknown"
186             detected_names.append(name)
187         n_face+=1
188 
189     return detected_names, face_locations
190 
191 
192 #Función que añade a una imagen el marco con la identificación de cada rostro
        detectado
193 def insert_name(names, face_locations, frame):
194     for (top, right, bottom, left), name in zip(face_locations, names):
195         #Añade el marco y el texto a la imagen
196         cv2.rectangle(frame, (left, top), (right, bottom), (0,255,0), 2)
197         cv2.putText(frame, name, (left, top-6), cv2.FONT_HERSHEY_DUPLEX, 1,
                (0,255,0), 1)
198 
199 
200 ################################Función 4#######################################
201 
202 #Función que analiza los rostros de un vídeo
203 def video(video_input):
204     #Si video_input es 0, se usa la webcam y se muestra por pantalla
205 
```

```
206     #Abre el vídeo
207     video = cv2.VideoCapture(video_input)
208 
209     if (video.isOpened() == False):
210       print("No se pudo leer el video.")
211     else:
212         #Si se ha tomado un archivo de vídeo
213         if(video_input):
214             # Obtiene la resolución del video en formato entero.
215             frame_width = int(video.get(3))
216             frame_height = int(video.get(4))
217 
218             splited_name=os.path.splitext(video_input)
219 
220             video_name="./Output/"+splited_name[0]+"_out.avi"
221             # Crea el objeto que se encargará de guardar el vídeo destino
222             out = cv2.VideoWriter(video_name,cv2.VideoWriter_fourcc(*'XVID'),
                    video.get(cv2.CAP_PROP_FPS), (frame_width,frame_height))
223             print("Procesando video en: "+video_name+"...")
224         else:
225             print("Pulse ESPACIO para detener la grabación.")
226 
227         #Se procesará un fotograma sí y otro no
228         process_this_frame=False
229         while (video.isOpened()):
230             # Toma un frame y comprueba
231             ret, frame = video.read()
232             process_this_frame=not process_this_frame
233 
234             if ret==True:
235                 if(process_this_frame):
236                     #Convierte la imagen de color BGR (OpenCV) a RGB (
                            Face_Recognition)
237                     rgb_frame = frame[:, :, ::-1]
238                     (detected_names, face_locations)=detect(rgb_frame)
239 
240                 insert_name(detected_names, face_locations, frame)
241                 #Si se toma de la cámara, se muestra por pantalla
242                 if(not video_input):
243                     # Muestra la iamgen por pantalla
244                     cv2.imshow('Video', frame)
245                 else:
246                     out.write(frame)
247 
248                 # Se detiene si se presiona la barra espaciadora
249                 if cv2.waitKey(1) & 0xFF == ord(' '):
250                     video.release()
251                     break
252             else:
253                 break
254 
255     if(not video_input):
256         cv2.destroyAllWindows()
257     else:
258         out.release()
```

A.2 Experimentos

En este apartado se muestra el Código A.3, que recoge las funciones que han sido utilizadas para llevar a cabo los experimentos para fijar una estructura del MLP final.

Las líneas 49, 50 y 51 han sido modificadas en función del experimento que se quisiera ensayar. En el Código A.3 se muestra la función completa para el último experimento y en el Código A.4 se recogen las líneas correspondientes a cada experimento.

Código A.3 inicio.py.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jul 9 19:22:17 2018
4
5  @author: Ildefonso Jiménez Silva
6  """
7  import funciones
8  import os #Interactuar con el entorno
9  from sklearn.neural_network import MLPClassifier
10 from sklearn.externals import joblib #Para guardar/cargar clases en archivos
11 from sklearn.model_selection import train_test_split
12
13 #Función que obtiene las características extraídas de cada rostro de las imá
       genes que conforman la base de datos
14 def carga_datos():
15     #Por defecto, se tomará la carpeta Source
16     dirs=os.listdir(path='./Source/')
17
18     #Inicializando parámetros
19     encodings_final=[]
20     tags=[]
21
22     #Se extraen las características de las imágenes
23     print("Analizando directorios:")
24     for (directorio,index) in zip(dirs, range(len(dirs))):
25         #Muestra el directorio analizado
26         print(str(index+1) + ') ' + directorio)
27         funciones.load_dir(directorio, encodings_final, index, tags)
28
29     #Guarda los datos en dos ficheros para evitar repetir este proceso en los
           experimentos
30     joblib.dump(encodings_final, './Model/encodings.pkl')
31     joblib.dump(tags, './Model/tags.pkl')
32
33 #Función que realiza los experimentos
34 def experimentos(n_iter):
35     #Carga los vectores de características guardados de cada rostro y su
           correspondiente etiqueta para identificar de quién se trata
36     X=joblib.load('./Model/encodings.pkl')
37     Y=joblib.load('./Model/tags.pkl')
38
39     (aciertos_prc1_total, aciertos_prc2_total, aciertos_prc3_total)=(0,0,0)
40     (prob_total1, prob_total2, prob_total3)=(0,0,0)
41
42     #Se realizará el número de iteraciones que se elija al llamar a la función
43     for iteration in range(n_iter):
```

```
44          print("Iteración número:", iteration+1)
45          #Divide la base de datos en entrenamiento y validación
46          X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
47
48          #Crea y entrena los diferentes MLPs
49          clf1 = MLPClassifier(verbose=False, max_iter=1000, random_state=1,
                activation= 'tanh', hidden_layer_sizes = (100))
50          clf2 = MLPClassifier(verbose=False, max_iter=1000, random_state=1,
                activation= 'tanh', hidden_layer_sizes = (150))
51          clf3 = MLPClassifier(verbose=False, max_iter=1000, random_state=1,
                activation= 'tanh', hidden_layer_sizes = (200))
52
53          model1=clf1.fit(X_train, Y_train)
54          model2=clf2.fit(X_train, Y_train)
55          model3=clf3.fit(X_train, Y_train)
56
57          #Calcula el númmero de aciertos a partir de los datos de validación
58          aciertos_prc1_total+=model1.score(X_test, Y_test)*100
59          aciertos_prc2_total+=model2.score(X_test, Y_test)*100
60          aciertos_prc3_total+=model3.score(X_test, Y_test)*100
61
62          #Calcula las probabilidades obtenidas por los MLP y calcula su media
63          probs1=model1.predict_proba(X_test)
64          probs2=model2.predict_proba(X_test)
65          probs3=model3.predict_proba(X_test)
66
67          (probi1, probi2, probi3)=(0,0,0)
68
69          for prob1, prob2, prob3, label in zip(probs1, probs2, probs3, Y_test):
70              probi1+=prob1[label]
71              probi2+=prob2[label]
72              probi3+=prob3[label]
73
74          N=len(probs1)
75
76          prob_total1+=probi1/N
77          prob_total2+=probi2/N
78          prob_total3+=probi3/N
79
80      #Prepara los datos finales diviendo entre el número de iteraciones para
            calcular la media de cada parámetro
81      aciertos_prc1_total/=n_iter
82      aciertos_prc2_total/=n_iter
83      aciertos_prc3_total/=n_iter
84      aciertos = [aciertos_prc1_total, aciertos_prc2_total, aciertos_prc3_total]
85
86      prob_total1/=n_iter
87      prob_total2/=n_iter
88      prob_total3/=n_iter
89      probabilidades=[prob_total1*100, prob_total2*100, prob_total3*100]
90
91      return aciertos, probabilidades
```

Código A.4 inicio.py.

```
1  #Crea y entrena los diferentes MLPs del experimento 1
2  clf1 = MLPClassifier(verbose=False, max_iter=1000, random_state=1, activation=
       'relu', hidden_layer_sizes = (100))
3  clf2 = MLPClassifier(verbose=False, max_iter=1000, random_state=1, activation=
       'tanh', hidden_layer_sizes = (100))
4  clf3 = MLPClassifier(verbose=False, max_iter=1000, random_state=1, activation=
       'logistic', hidden_layer_sizes = (100))
5
6  #Crea y entrena los diferentes MLPs del experimento 2
7  clf1 = MLPClassifier(verbose=False, max_iter=1000, random_state=1, activation=
       'tanh', hidden_layer_sizes = (100))
8  clf2 = MLPClassifier(verbose=False, max_iter=1000, random_state=1, activation=
       'tanh', hidden_layer_sizes = (100,100))
9  clf3 = MLPClassifier(verbose=False, max_iter=1000, random_state=1, activation=
       'tanh', hidden_layer_sizes = (100,100,100))
10
11 #Crea y entrena los diferentes MLPs del experimento 3
12        clf1 = MLPClassifier(verbose=False, max_iter=1000, random_state=1,
              activation= 'tanh', hidden_layer_sizes = (50))
13        clf2 = MLPClassifier(verbose=False, max_iter=1000, random_state=1,
              activation= 'tanh', hidden_layer_sizes = (100))
14        clf3 = MLPClassifier(verbose=False, max_iter=1000, random_state=1,
              activation= 'tanh', hidden_layer_sizes = (150))
```

Índice de Figuras

Índice de Tablas

Glosario

AdaBoost Adaptative Boosting. 4

ADALINE ADAptative LINear Elements. 15

CNN Convolutional Neural Network. 5

HOG Histogram of Oriented Gradients. 6

IA Inteligencia artificial. 15

IEEE Institute of Electrical and Electronics Engineers. 15

INNS International Neural Network Society. 15

KNN K-Nearest Neighbors. 12

LBP Local Binary Patterns. 11

LDA Linear Discriminatn Analysis. 11

MLP Multilayer Perceptron. 18

PC Principal Component. 11

PCA Principal Component Analysis. 3

ReLU Rectified Linear Units. 20

RNA Red neuronal artificial. 15

SGD Stochastic Gradient Descent. 21

SVM Support Vector Machine. 12