

# Trabajo Fin de Grado

## Grado en Ingeniería de las Tecnologías de Telecomunicación

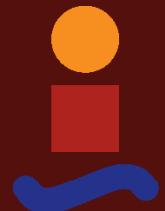
Detección, reconocimiento y seguimiento de rostros aplicando Redes Neuronales Convolucionales

Autor: Alejandro Bautista Gómez

Tutor: Rubén Martín Clemente

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

Sevilla, 2020





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

# **Detección, reconocimiento y seguimiento de rostros aplicando Redes Neuronales Convolucionales**

Autor:  
Alejandro Bautista Gómez

Tutor:  
Rubén Martín Clemente

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

Sevilla, 2020



Trabajo Fin de Grado: Detección, reconocimiento y seguimiento de rostros aplicando Redes Neuronales Convolucionales

Autor: Alejandro Bautista Gómez  
Tutor: Rubén Martín Clemente

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

---

El agradecimiento de este trabajo va dedicado a todos aquellos gigantes que, de forma amable y desinteresada, se prestaron a subirme sobre sus hombros. Gracias, pues me habéis regalado la capacidad de contemplar todo aquello más allá de lo que nunca podría haber observado en una vida.

A mi familia, y en particular a mis padres: Francisco Manuel Bautista Cubero y M<sup>a</sup> del Mar Gómez García; por su amor, por nunca haber dejado de creer en mi y por apoyarme con todo lo que tenían, incluso en los tiempos en los que no había.

A aquellos profesores, que durante años han invertido su pasión, compasión y humildad a la docencia. Por dedicar largas y extensas sesiones de tutoría, de las que algunas incluso han llegado a ser impartidas fuera de horario lectivo.

Mención especial para Rubén Martín Clemente, estimado tutor de este trabajo, por su guía y paciencia. También al profesor Marcos Calle Suárez, que en primero de carrera me enseñó el valor más importante en esta y cualquier otra profesión: la humildad.

A mis amigos de la carrera, por su confianza y camaradería; forjada a lo largo de años de entregas de prácticas, trabajos y noches de estudio. Gracias en especial a mi amigo Dunai Fuentes Hitos, gigante de corazón dorado cuyo consejo y guía han sido determinantes en la realización de este trabajo.

Finalmente, a las personas que generan contenido basado en el conocimiento y lo divultan de forma gratuita a través de la red. Muchas gracias, pues también habéis sido mis maestros.

*Alejandro Bautista Gómez  
Sevilla, 2020*



# Resumen

---

Las tecnologías de reconocimiento facial pertenecen a uno de los numerosos grandes hitos en la historia del desarrollo tecnológico. Mediante la aplicación de técnicas de detección, identificación y clasificación de rostros, se abre la posibilidad al desarrollo de numerosos tipos de aplicaciones. Algunas de estas, por ejemplo, irían desde sistemas de autenticación biométricos orientados al control de acceso, hasta la clasificación de rostros por grupos de edad para la emisión de anuncios personalizados o el análisis de las expresiones para el reconocimiento de emociones.

Sin embargo, todo este desarrollo de aplicaciones jamás podría haber llegado hasta este punto de la década del siglo XXI, sin la aparición y avances de un revolucionario modelo de computación: el Machine Learning (ML).

Para este proyecto, se presenta un sistema de detección, identificación y seguimiento de rostros en imágenes y video. Para ello en primer lugar se realizará un estudio de los distintos tipos de técnicas empleadas de forma clásica que se han podido emplear para la resolución de este tipo de problemas. Despues se presentará una introducción a varios modelos de clasificación y algoritmos empleados basados en ML. Finalmente, se harán un conjunto de pruebas para poner a prueba dicho sistema.



# **Abstract**

---

**F**acial recognition technologies belong to one of the many great milestones in the history of technological development. The application of techniques for the detection, identification and classification of faces opens up the possibility for the development of many types of applications. Some of these, for example, would go from biometric authentication systems oriented to access control, to the classification of faces by age groups for the emission of personalized announcements or the analysis of expressions for the recognition of emotions.

However, all this application development could never have reached this point in the decade of the 21st century, without the appearance and advances of a revolutionary computing model: Machine Learning (ML).

For this project, a system for the detection, identification and tracking of faces on video is presented. To do this, first a study of the different types of techniques used in a classical way that have been used to solve this type of problem will be carried out. Then an introduction to various classification models and algorithms used based on ML will be presented. Finally, a set of tests will be made to test the system.



# Índice

---

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Notación</i>	XI
<i>Acrónimos</i>	XII
<b>1 Introducción</b>	<b>1</b>
1.1 Motivación del proyecto	1
1.2 Objetivos	2
1.3 Organización de la memoria	2
<b>2 Tecnologías de reconocimiento facial</b>	<b>3</b>
2.1 Historia del reconocimiento facial	3
2.2 Fases del sistema de reconocimiento	4
2.2.1 Detección	4
Algoritmo de Viola-Jones	5
Método de los Histogramas de Gradientes Orientados	5
Redes Neuronales	6
2.2.2 Preprocesado	7
2.2.3 Extracción de características	7
Local Binary Patterns	9
Principal Component Analysis	9
2.2.4 Comparación y clasificación	10
Distancia Euclídea	10
Random Forest	10
K-Nearest Neighbours	10
Support Vector Machines	11
Redes Neuronales	11
<b>3 Redes Neuronales</b>	<b>13</b>
3.1 Introducción	13
3.1.1 Inteligencia Artificial	13
3.1.2 Machine Learning	13
3.1.3 Deep Learning	14
3.2 Arquitectura	14
3.2.1 Neuronas	15
La función de activación	16

3.3	Redes Neuronales Convolucionales	16
3.3.1	Capa de convolución	17
3.3.2	Capa de pooling	18
3.4	Entrenamiento	18
3.4.1	Algoritmo de backpropagation	19
3.4.2	Resultados y datos de entrenamiento	20
<b>4</b>	<b>Técnicas de seguimiento</b>	<b>23</b>
4.1	Introducción y desafíos al seguimiento de rostros	23
4.2	Escenario de seguimiento y soluciones propuestas	24
4.2.1	Filtros de Kalman	24
4.2.2	Filtros de Partículas	25
4.2.3	Seguimiento por detecciones ( <i>Tracking by detections</i> )	25
	Métrica: Inserción sobre la Unión	26
	Métrica: Distancias Euclídeas	26
	Asignación: <i>Greedy Assignment Algorithm</i>	26
<b>5</b>	<b>Sistema Propuesto</b>	<b>29</b>
5.1	Entorno de desarrollo	29
5.1.1	Herramientas de programación	29
	Lenguaje	29
	Framework	29
	Hardware	29
	Librerías	30
5.2	Estructura del sistema	30
5.2.1	Detección, preprocesado y extracción de características	30
5.2.2	Clasificación y comparación	31
5.2.3	Algoritmo de seguimiento	31
5.3	Interfaz del Sistema	32
5.3.1	Instalación y orden de los directorios	32
5.3.2	Inicialización y ejecución	33
	Entrenamiento - Clasificación de modelos	33
	Video-Tracking	34
<b>6</b>	<b>Experimentos y Resultados</b>	<b>35</b>
6.1	Experimento 1: Determinación de clasificadores óptimos	35
6.1.1	Clasificador MLP	36
6.1.2	Clasificador KNN	37
6.1.3	Clasificador RF	37
6.1.4	Clasificador SVM	37
6.1.5	Resultados y conclusión del experimento	38
6.2	Experimento 2: Determinación de umbral de confianza óptimo	38
6.2.1	Resultados y conclusión del experimento	39
6.3	Experimento 3: Fiabilidad del sistema de seguimiento	39
6.3.1	Resultados y conclusión del experimento	41
<b>7</b>	<b>Conclusiones y líneas futuras</b>	<b>43</b>
7.1	Conclusiones	43
7.2	Líneas futuras	43

<b>Apéndice A Códigos empleados</b>	<b>45</b>
<i>Índice de Figuras</i>	61
<i>Índice de Tablas</i>	63
<i>Índice de Códigos</i>	65
<i>Bibliografía</i>	67



# Notación

---

$\lfloor \quad \rfloor$	Función suelo
$\nabla$	Operador gradiente
$\sum_{j=1}^n$	Sumatorio desde 1 a n
$\sqrt{\quad}$	Raiz cuadrada
e	número e
$ELU$	Función Exponencial Lineal Unidad
$\max(0, x)$	Función max
$\tanh$	Función Tangente Hiperbólica
$ReLU$	Función Rectilinea Lineal Uniforme
$Sigmoid$	Función Sigmoide
$\frac{\partial y}{\partial x}$	Derivada parcial de y respecto a x
$\omega_i^L$	Parámetro de un peso i-esimo a la entrada de una neurona de capa L
$b^L$	Parámetro de sesgo a la entrada de una neurona de la capa L
$a^L()$	Función de activación a la salida de una neurona de la capa L
$z^L()$	Función suma ponderada de parámetros $\omega$ y $b$ de una neurona de la capa L
$C()$	Función de error de costes a la salida de una red neuronal
$\delta$	Valor de error imputado a una neurona

## Acrónimos

<b>AI</b>	Artificial Intelligence
<b>ARN</b>	Artificial Neural Network
<b>CCTV</b>	Círculo Cerrado de Televisión
<b>CNN</b>	Convolutional Neural Network
<b>CPU</b>	Central Processing Unit
<b>CUDA</b>	Compute Unified Device Architecture
<b>DL</b>	Deep Learning
<b>DNN</b>	Deep Neural Network
<b>EKF</b>	Extended Kalman Filter
<b>FPS</b>	Frames per Second
<b>GOTURN</b>	Generic Object Tracking Using Regression Networks
<b>GPU</b>	Graphic Processing Unit
<b>HOG</b>	Histogram Oriented Gradients
<b>IA</b>	Inteligencia Artificial
<b>IoU</b>	Inserction over Union
<b>ILSVRC</b>	ImageNet Large Scale Visual Recognition Competition
<b>KF</b>	Kalman Filter
<b>KNN</b>	K-Nearest-Neighbours
<b>LBP</b>	Local Binary Patterns
<b>LFW</b>	Labeled Faces in the Wild
<b>MC</b>	Monte Carlo
<b>ML</b>	Machine Learning
<b>PCA</b>	Principal Component Analysis
<b>PF</b>	Particle Filter
<b>ResNet</b>	Residual Network
<b>RF</b>	Random Forest
<b>RGB</b>	Red Green Blue
<b>SGD</b>	Stochastic Gradient Descent
<b>SVM</b>	Support Vector Machines
<b>TPU</b>	Tensor Processing Unit
<b>UKF</b>	Unscent Kalman Filter
<b>USD</b>	United States Dollar
<b>YOLO</b>	You Only Look Once

# 1 Introducción

---

## 1.1 Motivación del proyecto

Las tecnologías de análisis biométrico<sup>1</sup> son una de las claves en el desarrollo para las próximas décadas del siglo XXI. Dentro de este campo tecnológico, las tecnologías relacionadas al análisis y procesado de rostros han experimentado un continuo crecimiento anual de su valor de mercado. Tanto es así que la consultora tecnológica Global Market Insights tasó el valor de estas tecnologías en más de tres mil millones USD en el año 2019, proyectando crecimientos de hasta cuatro veces su valor para el año 2024 y superando así la barrera de los doce mil millones USD [13].

El impacto de sus aplicaciones en la sociedad es amplio y profundo. Tenemos desde sistemas de seguridad que ofrecen autenticación de usuarios para el acceso a perímetros, dispositivos electrónicos y aplicaciones, así sistemas de detección y seguimiento. También existen sistemas que permiten extraer y clasificar características como puedan ser la edad o el sexo. Así, por ejemplo, compañías y organizaciones pueden estudiar el perfil demográfico de su audiencia a la hora de establecer estrategias de marketing y ventas. Otra clase de uso más alejado de los anteriormente citados residiría en la medicina, como es el caso del desarrollo de software de análisis facial para la detección temprana de enfermedades como el síndrome de DiGeorge [12].

Sin embargo, todo este avance tecnológico no está exento de dilemas éticos y sociales. La utilización de software basado en reconocimiento facial para la identificación de ciudadanos ya es una realidad en algunos países, como es el caso de China. El país asiático, emplea su infraestructura de cámaras de CCTV en sus ciudades junto a avanzados centros de procesado de datos para monitorizar a su población. Así como consecuencia, el estado dispone de una herramienta de represión sobre la ciudadanía disidente con el régimen. Otro, es el famoso caso de la empresa estadounidense Clearview AI, la cual recolecta sin permiso fotografías de perfiles de usuarios de redes sociales como Facebook e Instagram para entrenar modelos de reconocimiento de personas. El objetivo de esta empresa sería vender dichos modelos de reconocimiento a las fuerzas del orden de los diferentes países, facilitando así las tareas de reconocimiento de criminales, aunque no especifican para qué clase de delitos según que país.

Además esta clase de modelos no son infalibles. Ya aparecieron noticias sobre la aparición de falsos positivos, como fue el caso de la estudiante de Brown, Amara K. Majeed acusada erróneamente por las autoridades de Sri Lanka de perpetrar un atentado terrorista tras una errónea identificación por parte de un software de reconocimiento facial.

---

<sup>1</sup> La biometría es el estudio de las características físicas únicas e intransferibles que permiten distinguir a un ser vivo de otro igual, como por ejemplo patrones faciales, huellas dactilares, voz, etc ...

Es por tanto a raíz de todas estas causas, que en mi ha crecido una profunda curiosidad por conocer como funcionan esta clase de sistemas, y el interés por desarrollar un sistema de detección y seguimiento que minimice la posibilidad de falsos positivos.

## 1.2 Objetivos

El objetivo de este trabajo será el de añadir mejoras sobre un sistema de identificación de rostros previamente implementado en el trabajo final de Ildefonso Jiménez Silva [17]. Para ello se le dotará de un nuevo entorno de desarrollo gratuito en la nube, el cual facilitará la ejecución de los programas y dotará al usuario de una GPU virtual para realizar el procesado de las imágenes y videos.

Además, se agregarán nuevos algoritmos de clasificación y la implementación de un algoritmo de tracking para dotar de mayor robustez al sistema a la hora de realizar identificaciones y no perder referencias.

## 1.3 Organización de la memoria

Esta memoria se dividirá en los siguientes capítulos:

- 1. Introducción:** El capítulo actual, en el que se exponen la causas que han motivado a la realización de este proyecto, así como los objetivos a perseguir.
- 2. Tecnologías de reconocimiento facial:** Este capítulo recoge los orígenes de las tecnologías de reconocimiento facial, así como las fases del proceso de reconocimiento y los distintos tipos de algoritmos que se suelen emplear.
- 3. Redes Neuronales:** En este capítulo se explican las bases del modelo matemático y computacional en el que se basan las redes neuronales.
- 4. Técnicas de seguimiento:** En este capítulo se recoge los diversos algoritmos empleados para las tareas de seguimiento, así como se discuten algunas de sus ventajas e inconvenientes a la hora de ser implementados.
- 5. Sistema propuesto:** Aquí se detallarán los pasos que se han seguido para la implementación del sistema. De la misma forma se presentará al entorno de trabajo empleado para realizar dicha implementación.
- 6. Experimentos:** Los experimentos que se llevan a cabo para evaluar el funcionamiento y rendimiento del sistema. Aquí se realizan experimentos basados en la identificación y experimentos basados en el seguimiento.
- 7. Conclusiones y líneas futuras:** Capítulo final en donde se resume el trabajo realizado y en el que se extraen las conclusiones más importantes. A su vez, se proponen líneas de investigación futura para añadir funcionalidades al proyecto.

## 2 Tecnologías de reconocimiento facial

---

### 2.1 Historia del reconocimiento facial

El ser humano al igual que el resto de animales, se vale de sus sentidos para reconocer y diferenciar a los miembros de su especie. Su sentido más desarrollado y por ende de mayor complejidad es la visión, que le permite percibir las formas, proporciones y tonos de aquellos elementos que hacen único al rostro de cada individuo. Las tecnologías de reconocimiento facial al igual que los seres humanos, se valen de esta información a la hora de identificar y clasificar rostros, acudiendo a mediciones de elementos como ojos, boca y nariz, a la relación entre sus proporciones o a su textura.

La aparición de esta tecnología es relativamente nueva, ya que empezó a ser investigada en la década de los sesenta por W.W Bledsoe y su equipo [6]. Estos desarrollaron una técnica llamada “reconocimiento hombre-máquina”, que era un sistema semiautomático que requería de un administrador que debía indicar en fotografías la posición de los rasgos faciales de una persona e introducirlos en una base de datos. Luego estos rasgos pasarían a ser comparados a los de otras fotografías en busca de la mínima distancia euclídea.

El siguiente paso fue en el año 1988, cuando L.Sirobich y M.Kirby [36] implementaron la técnica de álgebra lineal conocida como *análisis de componentes principales* (PCA) para el reconocimiento de rostros. El hito de este avance yace en la capacidad de cifrar en menos de 100 componentes, aquellos que eran necesarios para codificar una cara alineada y normalizada.

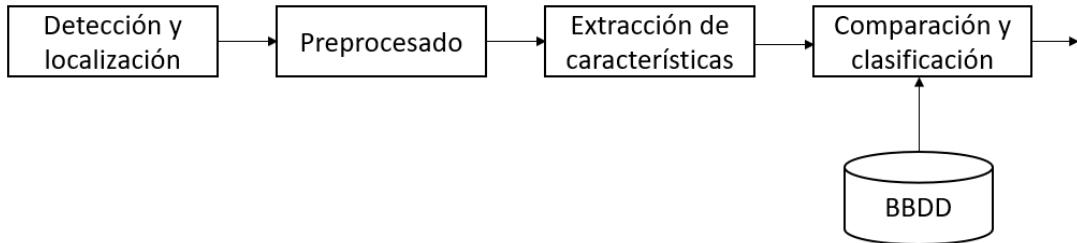
Posteriormente en el año 1991, M. Turk y A. Pentland [38] presentan *Eigenfaces*, un sistema que permite detectar rostros en imágenes, logrando así automatizar la tarea del reconocimiento facial. Gracias a este avance se podría empezar a operar, en teoría, con sistemas en tiempo real.

Sin embargo, estos sistemas son sensibles a las condiciones del entorno y dependen de elementos como la iluminación, la expresión de la cara o la ausencia de complementos sobre esta. Es por ello que durante las dos siguientes décadas se incidiría investigar en esta cuestión y aparecerían nuevos trabajos de investigación como la detección de rostros empleando filtros de Garbor [26], que conseguirían alrededor del 70 % de aciertos en el dataset *Labeled Faces in the Wild* (LFW).

Finalmente no fue hasta el año 2012 que AlexNet [3], una *red neuronal profunda* (DNN), se declaró como la ganadora de la *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) [40], donde puntuó con un 85.4 % de aciertos. Este hito incentivó la investigación en nuevos modelos de *aprendizaje profundo* (DL), los cuales darían paso a la aparición en 2014 de DeepFace [21], una DNN que fue capaz de obtener un porcentaje de aciertos del 97.53 % en el dataset LFW.

## 2.2 Fases del sistema de reconocimiento

Cualquier sistema de reconocimiento facial puede ser por lo general dividido en cuatro fases: detección, preprocesado, extracción de características y, comparación y clasificación.



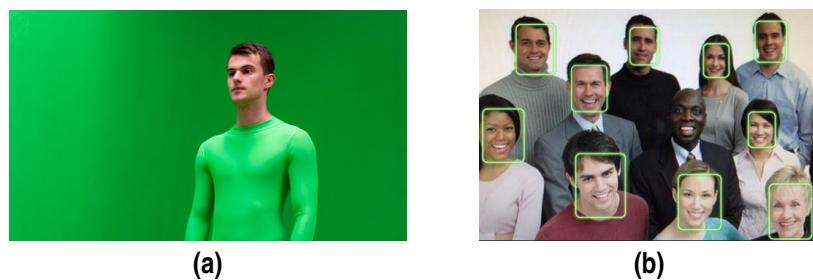
**Figura 2.1** Diagrama de un sistema de reconocimiento.

En la primera fase se detectan y localizan los rostros en la imagen. Después se pasa a la fase de preprocesado, en la que se normalizan los rostros mediante operaciones de alineación, escalado, recorte y ecualización. A continuación, se realiza la extracción de características, de la cual se obtiene un vector de información que permiten codificar a cada rostro. Finalmente, en la fase comparación y clasificación, el vector de características es comparado con los otros almacenados en la base de datos y se obtiene un valor de confianza, que permite clasificar o no al rostro como perteneciente al conjunto entrante.

A continuación se presentan las secciones donde se explica con mayor detalle los algoritmos y operaciones que se realizan en cada una de las fases anteriormente mencionadas mencionadas.

### 2.2.1 Detección

La fase de detección es aquella en la que el sistema localiza áreas de una imagen o fotograma que contenga caras para después aislarlas. El escenario más simple al que se puede enfrentar este problema es aquel en el que tenemos control sobre el color del fondo y el cuerpo, como es el caso de las producción de efectos especiales en películas, donde solo necesitan filtrar dicho color para obtener el rostro del actor o actriz. Otra forma simple de resolver el problema sería si de forma anticipada, conocíramos el tono de la piel a detectar, aunque esto podría dar lugar a detectores con sesgo racial.



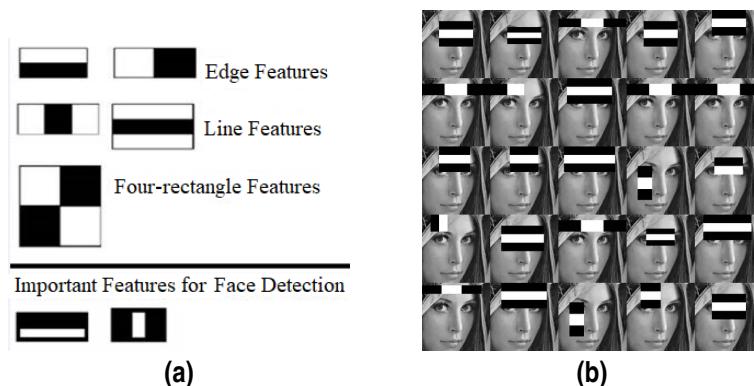
**Figura 2.2** Ejemplo de un escenario con aislamiento simple y un detector con sesgo racial.

El escenario al que se enfrenta este trabajo no contempla ningún tipo control sobre las condiciones del entorno ni de la persona, por lo tanto, serán requeridos sistemas de detección más robustos. Por ello se plantea a continuación los fundamentos de tres métodos, de los cuales se escogerá aquel basado en redes neuronales convolucionales, ya que ofrece la mayor robustez y precisión [5].

### Algoritmo de Viola-Jones

En el año 2001, Paul Viola Y Michael Jones [39] proponen un método de detección de rostros que ofrece la suficiente rapidez como para operar con imágenes de video a 15 fps<sup>1</sup>. Aunque no sea el más robusto de la lista, a día de hoy este algoritmo sigue siendo ampliamente utilizado por numerosas aplicaciones de detección de rostros debido a su bajo requisito computacional.

El algoritmo de Viola-Jones se basa en el uso de una serie de clasificadores débiles en cascada denominados *Haar-like features*. El cálculo de estos clasificadores se obtiene realizando el producto escalar entre la imagen y patrones con forma rectangular que indican la diferencia de intensidad de luces entre regiones adyacentes tal y como se indica en la Figura 2.3



**Figura 2.3** Haar-Cascade feautres aplicadas a una imagen.

A partir de estos productos escalares, obtenemos imágenes positivas (imágenes que contienen caras) e imágenes negativas (imágenes que no contienen caras), las cuales son empleadas como datos de entrenamiento para *AdaBoost*, un algoritmo de Machine Learning que obtiene como resultado, clasificadores fuertes a partir de una serie de clasificadores débiles.

### Método de los Histogramas de Gradientes Orientados

Ya para el año 2005, Navneet Dalal y Bill Triggs [10] presentan un algoritmo de detección basado en la información obtenida a partir de la variación del gradiente de luminosidad de los píxeles de una imagen. Aunque su primer uso estuvo orientado a la detección de peatones, fue implementado con relativa rapidez a la detección de rostros, ya que con una menor fase de entrenamiento, el algoritmo obtenía mayor porcentaje de detecciones y menos falsos positivos respecto al algoritmo de Viola-Jones[30]

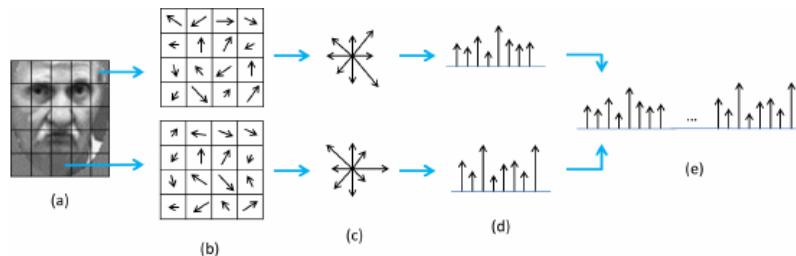
El método de los Histogramas de Gradientes orientados (HOG) se compone de cinco pasos:

- **Transformación a escala de grises:** el primer paso es transformar la imagen a una escala de grises, así el algoritmo operará sobre el valor de luminosidad de los píxeles.
- **Cálculo de gradientes:** a continuación, se calcula el operador gradiente para cada uno de los píxeles en la imagen. Para ello, hay que aplicar la definición de operador gradiente a las direcciones horizontal y vertical. Esta operación refleja la variación en la luminosidad de los píxeles situados de izquierda a derecha, y de arriba a abajo.
- **División en bloques:** una vez calculados los gradientes, se define un rango de orientaciones por subregiones que irán desde 180 grados (gradiente sin signo) hasta 360 grados (gradiente con signo). Se realizan divisiones por celdas sobre la imagen de tamaño 6 x 6, 8 x 8 o 16 x 16 píxeles. Por ejemplo, para un total de 9 subregiones de un gradiente sin signo [10], se

<sup>1</sup> Los cuadros por segundo o *frames per second* (FPS) son la unidad de medida que indican la velocidad de refresco de imágenes en un vídeo

agrupan a los gradientes de cada celda en histogramas, para contar que cantidad de estos se haya en cada una de las subregiones que irán desde  $0^\circ$  a  $20^\circ$ ,  $20^\circ$  a  $40^\circ$ , etc . . .

- **Normalización de histogramas:** las imágenes contienen áreas donde luces y sombras resultan predominantes, dando como resultado variaciones indeseadas de los niveles de los histogramas. Para reducir este efecto se toman dichos histogramas de las celdas del paso anterior y se agrupan por bloques. Para cada bloque, se concatenan los histogramas de forma que se obtiene un vector y se calcula su norma.

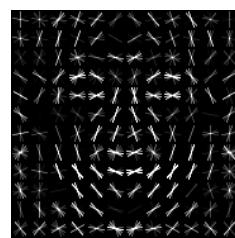


**Figura 2.4** Representación gráfica de los pasos realizados para obtener una imagen HOG.



**Figura 2.5** Ejemplo de una imagen descrita a partir de sus HOG features.

- **Localización:** finalmente para realizar la localización se aplica una plantilla de un rostro genérico previamente modelado de una imagen HOG sobre la imagen.



**Figura 2.6** Plantilla de un rostro genérico HOG.

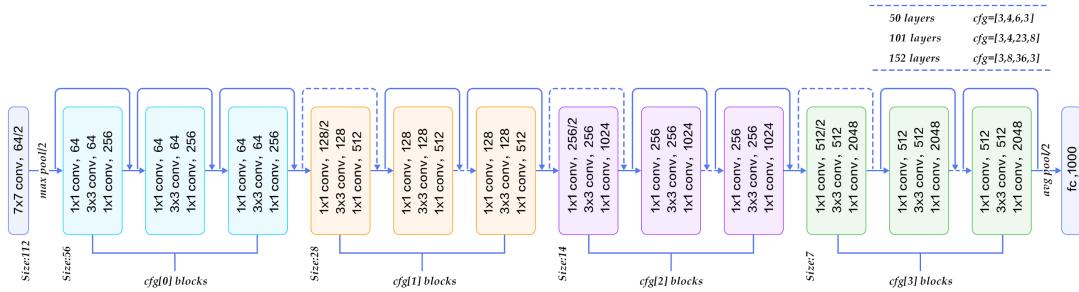
### Redes Neuronales

El método más preciso y robusto, dado que es capaz hasta de detectar rostros de perfil. A su vez es el de mayor coste computacional, ya que no pudo ser implementado de manera ágil hasta que en el año 2009 se presentó el trabajo de investigación de Rajat, Madhavan y Andrew [31], en el, que se demostraba que el uso de tarjetas gráficas (GPU) aceleraba las fases de entrenamiento de modelos de aprendizaje basados en Machine Learning hasta 70 veces, en comparación con las clásicas CPU multinúcleos. Gracias a este avance, comenzó una revolución en el desarrollo de modelos de redes

neuronales, ya que lo que anteriormente hubiese tomado semanas de entrenamiento e incluso meses, ahora requería solo de horas para poder entrenar a un modelo.

Para este trabajo se ha empleado una red neuronal ya pre-entrenada del tipo convolucional con arquitectura ResNet [15]. Este tipo de arquitectura se caracteriza por sumar a la salida de sus capas de convolución el resultado de las anteriores. Esta característica ofrece como ventaja una disminución en los tiempos de entrenamiento y de la función de error, así como una solución al problema del desvanecimiento del gradiente [16].

Los conocimientos y conceptos relacionados a la teoría sobre este tipo de arquitecturas serán desarrollados en profundidad en los capítulos 3 y 5 de este trabajo.



**Figura 2.7** Arquitectura de una red neuronal tipo ResNet.

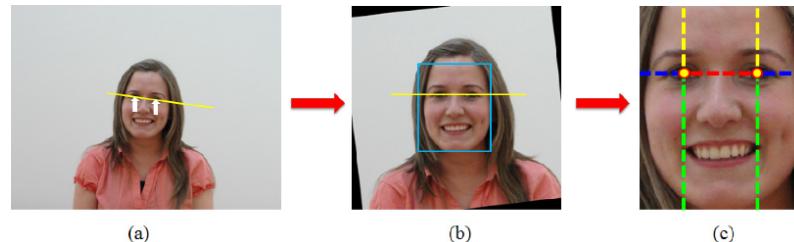
## 2.2.2 Preprocesado

La fase preprocesado sirve para normalizar y alinear los rostros obtenidos durante la detección. Realizando transformaciones geométricas sobre la imagen, el objetivo de esta fase es el de preparar a los rostros detectados para una correcta extracción de características. El preprocesado consta de las siguientes operaciones:

- **Rotación:** Normalmente los rostros contienen cierta inclinación en la imagen, por lo que es necesario rotarlos para alienarlos y facilitar la tarea del clasificador. Para realizar esta operación, se puede utilizar como método de alineación la referencia de la línea de los ojos.
- **Escalado:** Los algoritmos de escalado permiten reducir o aumentar el tamaño de la imagen, así como realizar zoom a determinadas partes. Para conseguir que todos los rostros tengan las mismas proporciones, se vuelve a utilizar la distancia entre ojos para calcular la ratio de aumento o disminución del tamaño de la imagen.
- **Recorte:** La operación de recorte sirve para tomar la imagen de la región en la que se haya un rostro. Dado que la imagen se trata de una matriz de píxeles, para seleccionar la región a recortar solo se deberá de seleccionar a la submatriz de píxeles que delimitan al rostro.
- **Ecualización del histograma:** Las imágenes pueden presentar variaciones de luminosidad y contraste, lo que puede dar lugar a que imágenes similares presenten diferentes niveles de iluminación en sus píxeles. Aplicando ecualización a los histogramas, se consigue que, imágenes que concentren la mayor parte de sus píxeles en una región del histograma, pasen a extenderse por todo su rango. Como resultado, se obtienen imágenes con mayor contraste y mejor diferenciación de los rasgos

## 2.2.3 Extracción de características

La extracción de características consiste en la obtención de información relevante de un rostro mediante la reducción de redundancias y características irrelevantes [37]. Existe una amplia variedad

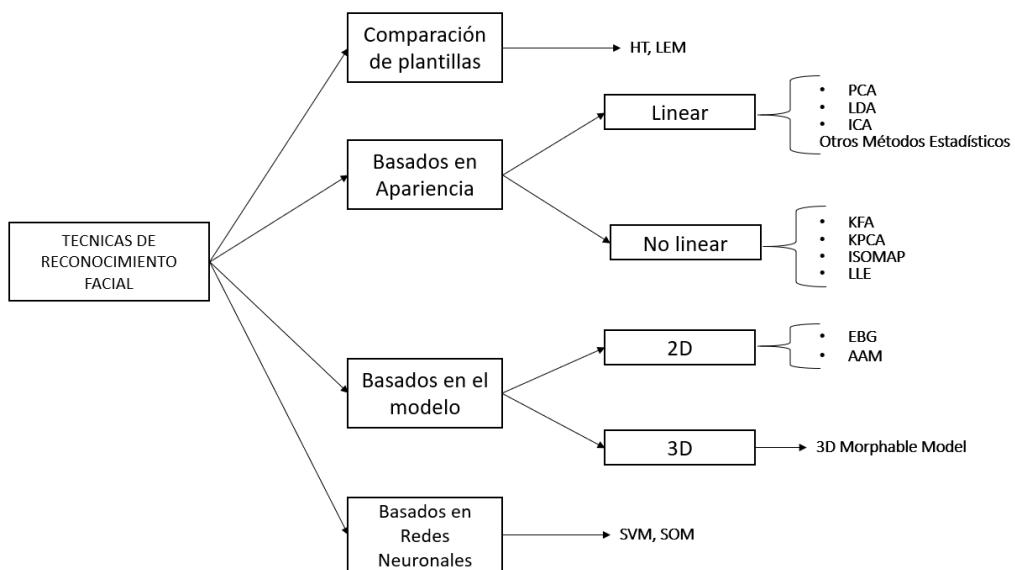


**Figura 2.8** Operación de alineación y recorte de un rostro utilizando como referencia la linea de los ojos.

de algoritmos para llevar a cabo la extracción de características pero en su mayoría, pueden ser clasificados a través de dos grandes aproximaciones.

- **Basados en los rasgos geométricos:** En este método se emplean conjuntos de rasgos biométricos como la distancias entre ojos, tamaño de la boca, anchura de la nariz, etc.... La ventaja de este método es su robustez frente a los cambios de expresión y orientación del rostro, así como se ve menos afectado por los cambios en la iluminación. Sin embargo, esto es a expensas de requerir un conocimiento previo de la imagen, así como ser más lentos y complejos en comparación con los métodos basados en apariencia.
- **Holísticos o basados en la apariencia:** En los métodos holísticos o basados en la apariencia se consideran las propiedades globales del rostro. Esto es así pues este método aproxima el problema del reconocimiento como un problema estadístico en lugar de analítico. La principal ventaja de este método es su rapidez y aplicabilidad a imágenes de baja resolución. Sin embargo, entre sus inconvenientes se haya la necesidad de tomar un mayor número de muestras en comparación con los métodos basado en rasgos geométricos. Además son sensibles a las expresiones y a la iluminación del rostro.

Además de estos métodos, se podrían expresar dos más: aquellos basados en comparación de plantillas y aquellos basados en redes neuronales. De esta manera, se puede presentar el mapa conceptual de la Figura 2.9 [37].



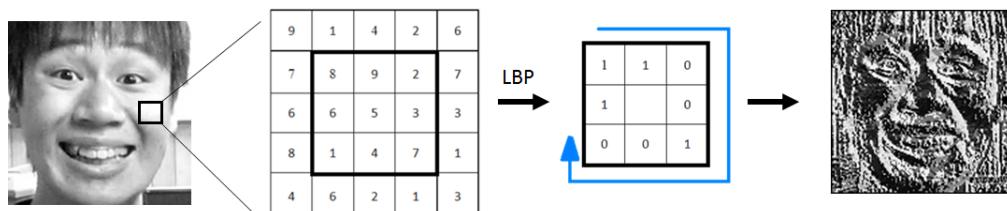
**Figura 2.9** Clasificación de Técnicas de Extracción de Características.

A modo de introducción, se presentarán dos técnicas de extracción de características: una basada en rasgos geométricos y una basada en apariencia.

### Local Binary Patterns

El método de Patrones Locales Binarios (LBP) es un algoritmo utilizado a modo de descriptor de texturas a nivel local. Es sencillo, rápido de implementar y además ofrece gran robustez frente a los cambios de iluminación en la imagen.

Para implementar este método, primero se define una submatriz de píxeles en una imagen sobre una escala de grises. Despues, se calcula la diferencia de intensidad del píxel central respecto de los vecinos de la submatriz, asignando 0 si la intensidad es menor, o 1 si es mayor. Una submatriz, de por ejemplo, tamaño 3x3 estará por tanto formada por ocho valores de ceros y unos, los cuales concatenados formarán un byte de información. Este valor podría ser por ejemplo: 1000100 (68 en decimal). El valor de esta operación se almacena en un histograma local y el proceso se vuelve repetir para cada submatriz de la imagen.

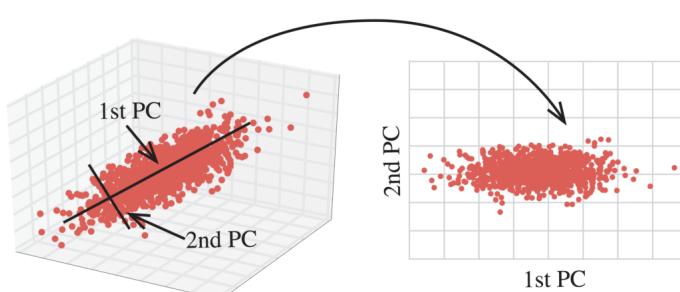


**Figura 2.10** Extracción de texturas mediante LBP.

Una vez obtenidos todos los histogramas locales, estos se suman y se obtiene un histograma final. De él, se podrá extraer el número de características totales con las que formar un vector de N dimensiones, donde N es el número de valores que conforman al histograma. Finalmente, este vector se compara con los de otras clases almacenadas y se determina una clase.

### Principal Component Analysis

La técnica de Análisis de Componentes Principales (PCA) en la que se basa *Eigenfaces*, es uno de los métodos más populares empleados para la extracción de características de un rostro. En este caso, las características extraídas por el sistema son mucho más genéricas, lo cual ofrece como resultado un vector de características de un elevado numero de dimensiones. Para poder operar de forma eficiente con este, se le hace reducir a sus componentes principales (PC), que son una serie de transformaciones que permiten eliminar componentes que aportan poco peso a la hora de generar agrupaciones.



**Figura 2.11** Ejemplo de reducción de un espacio de tres dimensiones a dos.

## 2.2.4 Comparación y clasificación

Los métodos hasta ahora presentados en este trabajo han permitido localizar rostros en imágenes, normalizarlos y codificarlos para representar sus características como un conjunto de componentes de un espacio vectorial. El objetivo de esta fase final es comparar dicho vector de características con aquellos almacenados en la base de datos del sistema, buscando similitudes y diferencias para establecer si el rostro en análisis pertenece al conjunto de rostros ya previamente analizados, o si por el contrario se trata de un rostro nuevo desconocido. En este apartado se exponen algunas de las diferentes técnicas empleadas en el problema de clasificación de imágenes.

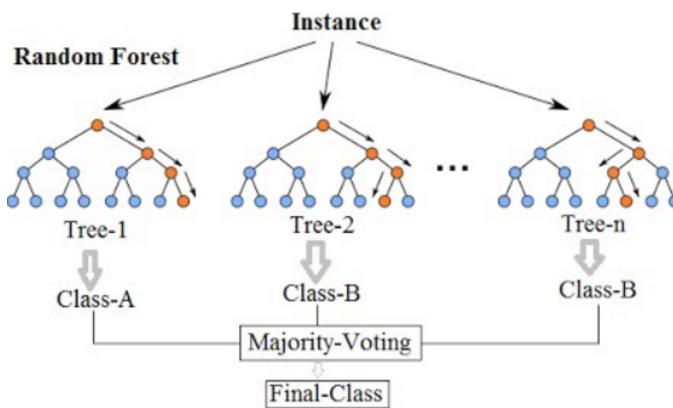
### Distancia Euclídea

Si bien la distancia euclídea no es un método de clasificación en sí mismo, es la herramienta básica de la que dependemos para determinar las similitud entre muestras. Haciendo uso del Teorema de Pitágoras se puede generalizar que la distancia euclídea de dos vectores  $X = [x_1, x_2, \dots, x_N]$  e  $Y = [y_1, y_2, \dots, y_N]$  pertenecientes a un espacio de  $N$  dimensiones es:

$$d(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_N - y_N)^2} \quad (2.1)$$

### Random Forest

Los Bosques Aleatorios o *Random Forest* (RF) es una técnica de clasificación basada en Machine Learning que determina la probabilidad de pertenencia a una clase en función al número de votos obtenidos a la salida de un conjunto de árboles de decisión. Para construir estos árboles, primero se crea un dataset con datos de entrenamiento remuestreados (*bootstrapping*) y después se empiezan a generar árboles de decisión a partir de pares de las variables de dimensión que forman nuestro vector de características. Finalmente, una vez generados los árboles, utilizamos los datos que han quedado fuera de los datos de entrenamiento para validar el modelo (*out-of-bag-dataset*).



**Figura 2.12** Diagrama Random Forest.

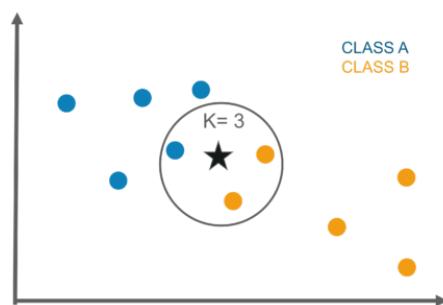
Pese a que en la teoría este método debería de funcionar en todos los problemas de clasificación, no se recomienda su uso en problemas de clasificación de imágenes. Sin embargo, como en este caso la extracción de características de la imagen se realizará mediante una Red Neuronal Convolucional, se experimentará con este método.

### K-Nearest Neighbours

El algoritmo de los K vecinos más cercanos (KNN) [4] es una técnica de clasificación no paramétrica que organiza a las muestras según su distancia respecto a otras muestras ya observadas y clasificadas. El funcionamiento del algoritmo es el siguiente:

1. Selecciona un número de vecinos  $K$ .
2. Calcula la distancia euclídea del vector de características a la entrada respecto del conjunto de todas las muestras ya observadas y clasificadas por el sistema.
3. Selecciona las  $K$  muestras de menor distancia y contabiliza a que clase pertenecen.
4. Asigna al vector la clase con mayor número de vecinos.

Este algoritmo presenta importantes ventajas, como su sencillez, su robustez frente a datos de entrenamiento ruidosos o el incremento de su efectividad ante conjuntos de entrenamiento de gran magnitud. Sin embargo también presenta problemas, como la determinación del número óptimo de vecinos  $K$  como ocurre en el ejemplo de la figura 2.13. Realizar estos cálculos conlleva un elevado coste computacional ya que cada muestra a la entrada, a de compararse respecto a cada muestra del conjunto de datos observados.



**Figura 2.13** Para  $K=3$ , la muestra se clasifica como clase B, pero para  $K=5$  como clase A.

### Support Vector Machines

Las Máquinas de Soporte Vectorial (SVM) [9] son modelos de aprendizaje usados para clasificación y regresión a partir de un conjunto de datos previamente etiquetados. Para ello, este método representa al conjunto de datos mediante un espacio de mayor número de dimensiones respecto del conjunto original. Esta operación se realiza, por ejemplo, creando una nueva variable cuyo valor es el resultado de elevar al cuadrado el valor de los datos de una dimensión.

Una vez establecido el nuevo conjunto de datos, el algoritmo traza un hiperplano que separa al conjunto de puntos en clases siguiendo un criterio de máxima distancia.

### Redes Neuronales

Las Redes Neuronales (RN) son el modelo computacional que mejores resultados presentan y a su vez el que mayor complejidad ofrece. Su modelo se basa en la interacción entre neuronas, unidades de procesamiento que realizan sumas ponderadas que definen un problema de regresión lineal, cuya salida pasa a la entrada una función de activación. El objetivo de dicha función es eliminar linealidad para evitar que el sistema colapse. Este punto será explicado en mayor profundidad en el capítulo 3.

Gracias a este tipo de modelo, las redes neuronales no solo resultan útiles para los problemas de clasificación, si no que como se ha podido observar en el apartado detección, también resultan útiles a la hora hallar patrones.



# 3 Redes Neuronales

---

## 3.1 Introducción

Las Redes Neuronales Artificiales (ARN) son una familia de algoritmos pertenecientes al paradigma del aprendizaje automático, también denominado como *Machine Learning* (ML), que a su vez se haya dentro del campo de estudio de la Inteligencia Artificial (IA).

Es importante antes de continuar, explicar bien la diferencia entre estos dos términos y el ya citado Deep Learning, dado que sus significados se tienden a entremezclar y a confundir la parte por el todo.

### 3.1.1 Inteligencia Artificial

Según una primera definición del año 1955, Inteligencia Artificial es: *la subdisciplina de las ciencias de la computación que estudia la creación de máquinas que imitan comportamientos inteligentes* [25]. Programar un brazo robótico para que este siga un conjunto de instrucciones de movimiento o convertir texto a voz, son comportamientos aparentemente inteligentes, pero que sin embargo carecen de las capacidades cognitivas para reprogramarse en caso de que se necesite adaptar para resolver un problema similar.

Por ejemplo, si se entrena a la IA de un robot bípedo para que este aprenda a caminar, la IA no será capaz de transferir su conocimiento para desplazar las extremidades a la tarea "dar una patada"

### 3.1.2 Machine Learning

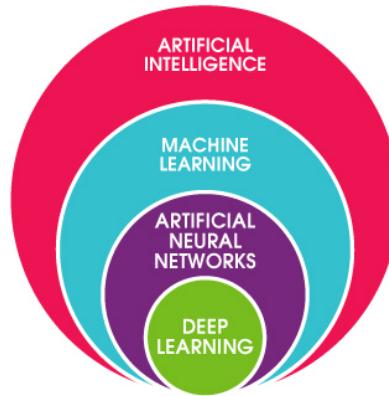
Arthur Samuel define en 1959 Machine Learning [35] como: *la rama del campo de la Inteligencia Artificial, que estudia como dotar a las máquinas de la capacidad de aprendizaje, entendido este como la generalización del conocimiento a partir de un conjunto de experiencias*. De manera parecida a como aprendemos los seres humanos, los algoritmos de Machine Learning desarrollan modelos generalizados de clasificación basados en la búsqueda de patrones o tendencias, a base de ser expuestos a un elevado número de datos de entrenamiento.

Para entrenar un sistema mediante Machine Learning existen principalmente dos métodos de aprendizaje:

- **Aprendizaje supervisado:** Este método parte de un conjunto de datos etiquetados que indican la solución deseada al sistema. Algunos de estos métodos ya han sido vistos en este trabajo como son por ejemplo: *Random Forest, Support Vector Machines y Redes Neuronales*.
- **Aprendizaje no supervisado:** En este método los datos carecen de etiquetas y es el propio algoritmo el que los clasifica. En esta categoría se encuentran algoritmos como *PCA* y *KNN*.

### 3.1.3 Deep Learning

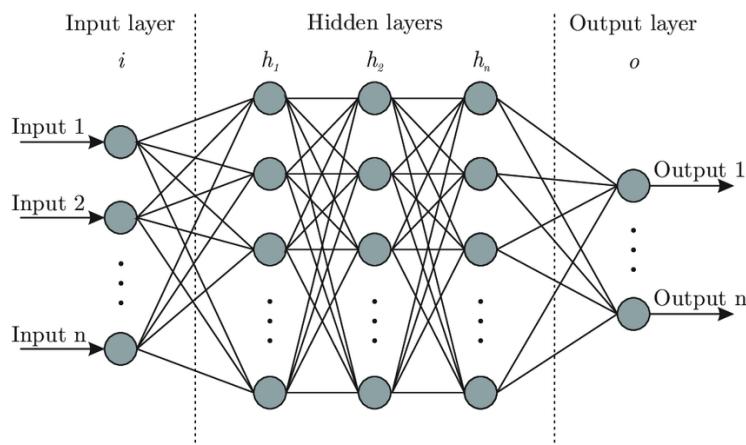
Los modelos de computación basados en aprendizaje profundo o *Deep Learning* son aquellos que se componen de múltiples capas de procesamiento que se dedican a aprender representaciones de datos a múltiples niveles de abstracción [22]. Utilizan arquitecturas basadas en *Redes Neuronales*, ya que estas permiten realizar el aprendizaje de forma jerarquizada a través de diferentes capas de neuronas. Por ejemplo para el caso de una cara, en las primeras capas una red puede aprender los elementos individuales que conforman la cara (ojos, boca, nariz ...), en las posteriores que estos poseen texturas, que se localizan en una determinada región del rostro, y que existen distancias y proporciones entre estos. Finalmente abstrae todos estos conocimientos, aprendiendo a identificar un rostro y a diferenciarlo de otros.



**Figura 3.1** Diagrama de jerarquías del aprendizaje maquina .

## 3.2 Arquitectura

La arquitectura de una red neuronal se puede definir como un grafo cuyos nodos (*neuronas*) se organizan por niveles (*capas*). Las neuronas de cada capa se hayan conectadas por su entrada, a la salida de las neuronas de la capa anterior y estas a su vez, conectan su salida a las entradas de las neuronas de la capa posterior, tal y como se representa en el ejemplo de la figura 3.2.



**Figura 3.2** Arquitectura básica de una red neuronal.

La capa de entrada (*input layer*) recibe los datos de entrada y la capa de salida (*output layer*) devuelve la predicción realizada por la red. Las capas intermedias se denominan capas ocultas

(*hidden layers*), y aunque en el ejemplo de la figura 3.2 aparezcan solo tres, se pueden añadir a la red tantas capas con diferentes números de neuronas como se desee, dotándola así de mayor complejidad y profundidad (*Deep Neural Network*).

### 3.2.1 Neuronas

La neurona es la unidad básica de procesamiento de una red neuronal. Esta representa el conjunto de soluciones de un problema de regresión lineal, el cual viene descrito por la suma de los valores de sus conexiones de entrada  $x_j$ , ponderadas por una serie de pesos  $\omega_j$  y sumadas a un parámetro de sesgo  $b$ .

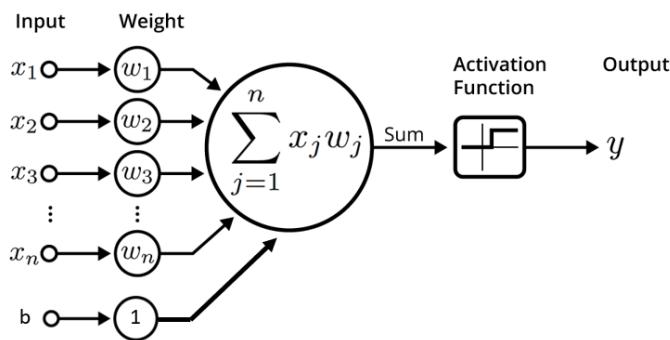


Figura 3.3 Arquitectura de una neurona.

Este es el modelo en el que se basa la idea más básica de neurona: *el perceptrón* [33]. La solución del perceptrón, presentada en 1957 por Frank Rosenblatt, define un hiperplano que establece una región de decisión para el problema de clasificación. Sin embargo, este modelo presenta serias limitaciones ya que solo puede resolver problemas lineales. Esto significa, que el modelo nos limita únicamente a problemas de clasificación que podemos resolver empleando únicamente una recta, tal y como se muestra en la figura 3.4.

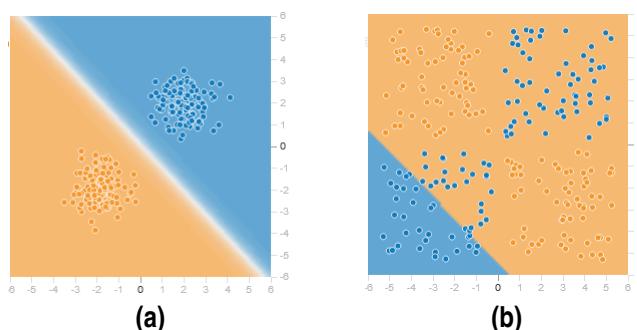


Figura 3.4 Soluciones a problemas de clasificación utilizando un perceptrón.

En la figura 3.4a podemos resolver el problema de clasificación de las clases naranja y azul empleando un único perceptrón, sin embargo, en la figura 3.4b no. Para resolver este problema se podría plantear el uso de varios perceptrones para modelar la geometría de la región, sin embargo, resultaría inútil dado que el resultado de emplear grupos de perceptrones en serie y paralelo acaban por colapsar dando como resultado un único perceptrón. Para ello hace falta la introducción de un nuevo elemento matemático: **la función de activación**

### La función de activación

El objetivo de la función de activación es transformar el valor a la salida de una neurona a través de una función  $a(z)$ , para que esta pase de ser lineal a no lineal y desarrollar así sistemas de mayor complejidad geométrica. Para ello se propone el uso de un conjunto de funciones denominadas *de activación* que aportan esta serie de no linealidades a la salida de la suma ponderada de la neurona.

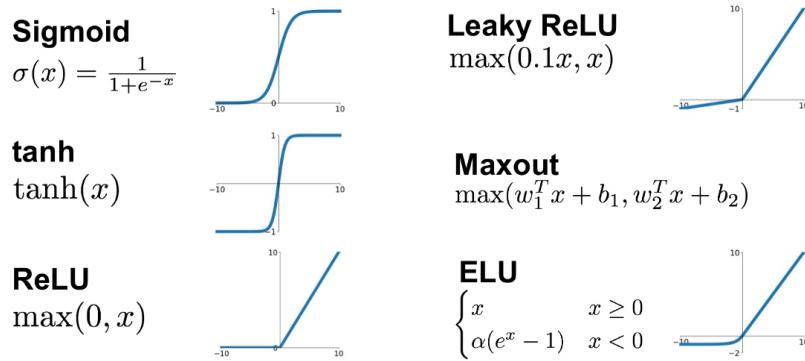


Figura 3.5 Funciones de activación.

En el libro *Perceptrons* [24] del año 1968, Marvin y Minsky explicaban la limitación del modelo del perceptrón proponiendo como ejemplo, la imposibilidad de modelar una puerta lógica XOR. Así para resolver el ejemplo de la figura 3.4b, se podrían asignar a las clases azul y naranja los 0s y 1s de la salida del problema de la puerta lógica. Utilizando así para resolver este problema una única capa oculta con tres neuronas y una función de activación ReLU, se puede obtener la siguiente solución al problema de clasificación tal y como se presenta en la figura 3.6.

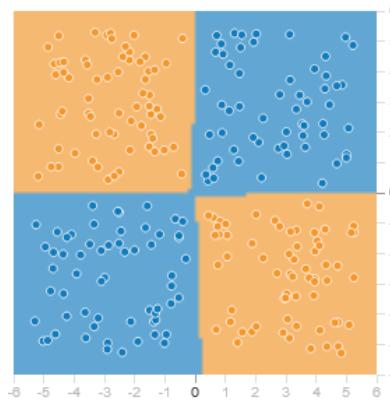


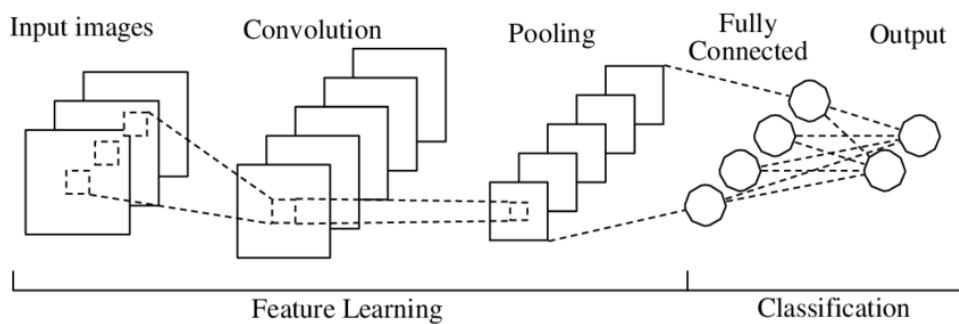
Figura 3.6 Solución al problema de clasificación de la puerta lógica XOR.

### 3.3 Redes Neuronales Convolucionales

Hasta ahora, la arquitectura de las redes neuronales se ha presentado como un conjunto de capas de neuronas, en donde cada neurona de cada capa se haya conectada a cada una las neuronas de las capas anterior y posterior. Este modelo de capas interconectadas, que es denominado como denso (*dense layers*), no es eficiente a la hora de trabajar con imágenes. Esto es así porque al estar las imágenes compuestas de píxeles codificados a partir de canales de color, hace falta una neurona para cada píxel de la imagen, dando como resultado un elevado número neuronas y parámetros que incrementarían de forma seria los costes de computación de la red.

Para exponer cuan costosa sería esta arquitectura, deberíamos de saber que solo para una neurona de la primera capa oculta se requeriría de un número de parámetros igual a **Píxeles Altura x Píxeles Anchura x Canales de color + 1**, siendo el +1 el parámetro sesgo de la neurona. Es decir, que para una imagen de por ejemplo, 300 x 300 píxeles codificada con canales RGB (rojo, verde y azul), harían falta 270.001 parámetros para cada neurona de la primera capa.

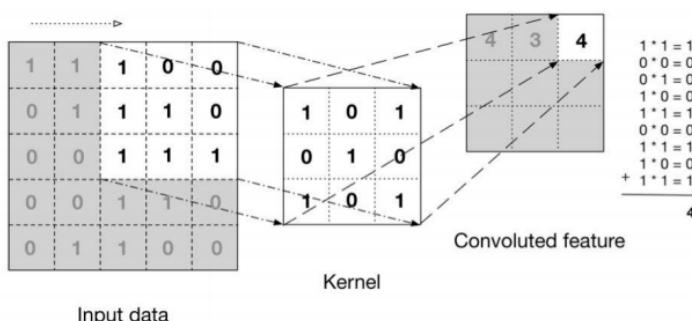
Las redes neuronales convolucionales (CNN) son una arquitectura particular de las redes neuronales artificiales, que funcionan de manera excelente en tareas de reconocimiento de patrones en imágenes. Para ello se valen de dos nuevos tipos de capas denominadas de **convolución** y de **pooling**, donde se realizan las operaciones para la extracción de características geométricas de la imagen. Así en las primeras capas ocultas la red detecta formas simples como curvas y aristas, que se propagan a lo largo de las siguientes capas de convolución, donde se jerarquiza el aprendizaje desde los elementos más simples a aquellos de mayor complejidad, hasta llegar finalmente a las capas densas donde se realizará la clasificación.



**Figura 3.7** Diagrama de una red neuronal convolucional.

### 3.3.1 Capa de convolución

La capa de convolución extrae los patrones de la matriz de datos de entrada. Para ello se vale de los *kernels*, matrices de tamaño  $N \times N$  que almacenan los valores de los pesos  $\omega$  de las neuronas, y sirven para definir los filtros. El kernel opera desplazándose a lo largo de la matriz de datos de entrada, realizando series de productos y sumas de filas por columnas, cuyo resultado es almacenado a su salida en una nueva matriz denominada como *mapa de características* o *mapa de activación*.



**Figura 3.8** Ejemplo de la operación de convolución de un kernel de tamaño 3x3.

A consecuencia de esta operación, se reduce el tamaño del mapa de características respecto a la matriz de datos de entrada. Para evitar esto se realiza el *padding*, una operación que consiste en agregar ceros alrededor de los bordes de la matriz de entrada. Además del padding, existe un parámetro adicional denominado *stride*, que indica el número de celdas sobre las que se desplaza el

kernel para cada paso de operación realizado sobre la matriz de entrada. Así el tamaño del mapa de características a la salida se puede calcular a partir de la siguiente formula:

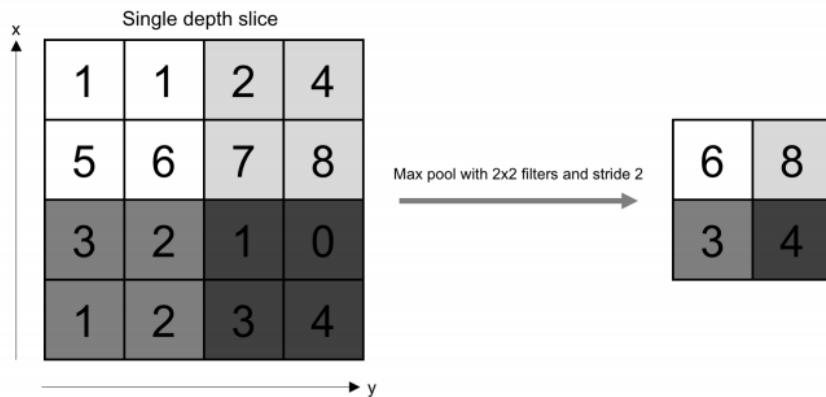
$$F = \left\lceil \frac{N + 2P - F}{S} + 1 \right\rceil \quad (3.1)$$

Donde:

- $F$  = Tamaño de la matriz de salida.
- $N$  = Tamaño de la matriz de la matriz de entrada.
- $P$  = Parámetro de padding, por defecto vale 0.
- $S$  = Parámetro de stride, por defecto vale 1.

### 3.3.2 Capa de pooling

Las capa de pooling se emplea para simplificar aquellos valores que se hayan semánticamente próximos dentro del mapa de características, para así evitar el problemas durante la fase de entrenamiento como el *overfitting* [29]. La función más empleada es **Max-pooling**, que define una submatriz de tamaño  $M \times M$  con stride  $M$  sobre el mapa de características, y que da a su salida una matriz que contiene los mayores valores del mapa de características.



**Figura 3.9** Ejemplo de una operación de Max-pooling con una matriz de tamaño 2x2.

## 3.4 Entrenamiento

En el año 1968, Marvin y Minsky expusieron en su libro *Perceptrons* [24] el principal problema que limitaba el aprendizaje de las redes neuronales. Hasta ese momento para calcular los valores de los pesos de las neuronas, se empleaba un algoritmo de fuerza bruta que si bien resultaba útil para el cálculo de parámetros en el perceptrón, no era posible de escalar sobre arquitecturas de sistemas con mayor números de capas y neuronas. Este hecho dio lugar el llamado *invierno de la Inteligencia Artificial* (AI Winter), un periodo de más de quince años por el cual, la investigación de sistemas basados en aprendizaje automático se vio detenida.

Sin embargo, en el año 1986 Rumelhart, Hinton y Williams [34] presentaron un trabajo de investigación que proponía un nuevo método para auto-ajustar los parámetros de la red neuronal a través del cálculo de las derivadas de sus errores y su propagación. Hablamos del algoritmo de *backpropagation*.

### 3.4.1 Algoritmo de backpropagation

El algoritmo de backpropagation está basado en la teoría del conocido *algoritmo del descenso del gradiente* (SGD). La intuición que se esconde tras este algoritmo es la de calcular el valor del error a la salida de un modelo matemático en un punto de su función, calculando en el proceso sus derivadas parciales. Mediante estos cálculos obtenemos un vector gradiente  $\nabla f$  que nos indica la dirección de la pendiente de la función error de nuestro modelo, es decir: en qué dirección se incrementa el error de la predicción. Tomando de nuevo un punto en el sentido opuesto a la dirección de crecimiento, volvemos a operar  $\nabla f$  sobre la función de error de forma iterativa, hasta llegar finalmente al óptimo global.

Cuando inicializamos una red neuronal, los valores de los parámetros  $\omega$  y  $b$  de las neuronas serán valores aleatorios, y en consecuencia darán como resultado predicciones erróneas a la salida de la red. Las funciones y parámetros que definen las relaciones matemáticas dentro una red neuronal son:

- $C(a^L)$  como la función de coste que determina el error a la salida de la red y cuyo parámetro de entrada es la función  $a^L(z^L)$ .
- $a^L(z^L)$  como el valor a la salida de la función de activación de una neurona en la última capa y cuyo parámetro de entrada es la función  $z^L(w^L, b^L)$ .
- $z^L(w^L, b^L)$  como el valor a la salida de la suma ponderada de los pesos  $w^L$  y el sesgo  $b^L$  de una neurona en la última capa. Su ecuación es:

$$z^L = w^L \cdot x^L + b^L \quad (3.2)$$

Para actualizar los valores de los parámetros de una neurona, estudiamos la variación del error a la salida de la red como la derivada parcial de la función de coste  $C$ , respecto de los pesos y sesgo  $\omega^L$  y  $b^L$ :

$$\frac{\partial C}{\partial \omega^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial \omega^L} \quad (3.3)$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L} \quad (3.4)$$

Esta expresión que es únicamente válida para una neurona de la última capa, es el resultado de aplicar la regla de la cadena sobre la derivada parcial de la función de coste  $C$ , donde cada derivada expresa cada uno de los siguientes significados:

- $\frac{\partial C}{\partial \omega^L}$  como la variación del valor de la función de coste, respecto al peso  $\omega^L$ .
- $\frac{\partial C}{\partial b^L}$  como la variación del valor de la función de coste, respecto al sesgo  $b^L$ .
- $\frac{\partial C}{\partial a^L}$  como la variación del valor de la función de coste, respecto a la función de activación.
- $\frac{\partial a^L}{\partial z^L}$  como la variación del valor a la salida de la función de activación, respecto al valor de la suma ponderada.
- $\frac{\partial z^L}{\partial \omega^L}$  como es la variación del valor a la salida de la suma ponderada, respecto al peso  $\omega^L$ .
- $\frac{\partial z^L}{\partial b^L}$  como la variación del valor de la suma ponderada, respecto al sesgo  $b^L$ .

De las expresiones 3.3 y 3.4, podemos simplificar el término  $\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$  por uno nuevo denominado  $\delta^L$ . A este valor lo denominaremos como *error imputado a la neurona*, y nos indicará como cambia el valor del error a la salida de la red frente a un pequeño cambio en el valor de la suma ponderada  $z^L$  de una neurona de la última capa. La derivada parcial  $\frac{\partial z^L}{\partial b^L}$  es 1, ya que es la derivada parcial

respecto al término independiente y  $\frac{\partial z^L}{\partial \omega^L}$  es igual a  $a_i^{L-1}$ , que es el valor a la salida de la función de activación de una neurona de la capa previa. De esta forma las ecuaciones se simplifican y quedan como:

$$\delta^L = \frac{\partial C}{\partial z^L} \quad (3.5)$$

$$\frac{\partial C}{\partial \omega^L} = \delta^L \cdot a_i^{L-1} \quad (3.6)$$

$$\frac{\partial C}{\partial b^L} = \delta^L \quad (3.7)$$

Sin embargo, estas ecuaciones solo determinan como se ve afectado el error a la salida, frente a variaciones de los parámetros  $\omega$  y  $b$  de las neuronas en la última capa. Si aplicamos el mismo razonamiento que en 3.3 y 3.4, para las neuronas de la capa L-1 y aplicando la regla de la cadena se obtiene la siguiente expresión:

$$\frac{\partial C}{\partial \omega^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial \omega^{L-1}} \quad (3.8)$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial b^{L-1}} \quad (3.9)$$

Estas ecuaciones obtenidas se pueden simplificar de nuevo de manera fácil, ya que  $\frac{\partial a^{L-1}}{\partial z^{L-1}}$  es igual a la derivada de la función de activación  $a^{L-1}$ ,  $\frac{\partial z^{L-1}}{\partial \omega^{L-1}}$  es igual a la función de activación  $a^{L-2}$  y  $\frac{\partial z^{L-1}}{\partial b^{L-1}}$  es igual a 1. Nos queda la derivada  $\frac{\partial z^L}{\partial a^{L-1}}$ , que una vez calculada resulta la matriz de parámetros  $W^L$  que conectan ambas capas, lo que nos hace propagar el error de la capa posterior a la anterior.

Finalmente el algoritmo de backpropagation se presenta resumidamente como una iteración en tres pasos a realizar para cada una de las neuronas de nuestro sistema:

1. Calcular el error de computo de la última capa:

$$\delta^L = \frac{\partial C}{\partial z^L} \quad (3.10)$$

2. Retropropagar el error a la capa anterior:

$$\delta^{l-1} = W^l \delta^l \cdot \frac{\partial a^{l-1}}{\partial z^{l-1}} \quad (3.11)$$

3. Calcular las derivadas de la capa usando el error anterior:

$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1} \quad (3.12)$$

$$\frac{\partial C}{\partial \omega^{l-1}} = \delta^{l-1} a^{l-2} \quad (3.13)$$

### 3.4.2 Resultados y datos de entrenamiento

El proceso de entrenamiento de una red neuronal es largo y complejo. Para empezar se parte de un conjunto de datos (*dataset*) que debe de contener un número elevado de muestras y al mismo tiempo, estas deben de ser lo suficientemente representativas y libres sesgos para que la red pueda

realizar predicciones de la manera esperada. Así, a la hora de entrenar una red se pueden esperar dos posibles resultados:

- **No convergencia:** Tras aplicar SGD mediante backpropagation, no se llega a un punto donde el valor de la derivada de la función de error se mantenga estable alrededor de la región del cero. Es decir, no se haya solución estable al problema.
- **Convergencia:** Tras aplicar SGD mediante backpropagation, la red alcanza un punto de equilibrio donde la derivada de la función de error y los pesos se mantienen próximos a cero. Aunque sin embargo, que el error se mantenga estable no es sinónimo de que la red haga predicciones de forma correcta, ya que esta se puede hallar en alguno de los siguientes tres escenarios.
  - **Subajuste o Underfitting:** Este caso ocurre cuando el conjunto de datos empleados para entrenar la red resulta insuficientemente representativo como para realizar una extracción generalizada de conocimiento. Como consecuencia, la red realiza predicciones erroneas.
  - **Sobreajuste o Overfitting:** Este caso ocurre cuando la complejidad de la red es tan elevada que se acaba por sobreespecializar en el conjunto de datos de entrenamiento. El resultado de esto es que realizan predicciones correctas con los datos de entrenamiento, pero sin embargo es incapaz de extraer el conocimiento para aplicarlo a nuevos datos de entrada.
  - **Entrenamiento Correcto:** el caso deseado, en donde la red ha sido capaz de extraer el conocimiento y realizar predicciones con un elevado porcentaje de aciertos.

Para garantizar un correcto entrenamiento de nuestra red, es por tanto necesario disponer de una base de datos de entrenamiento amplia y variada. Para ello existen técnicas como el *Data Augmentation* [23], a través de la cual obtenemos nuevos datos de entrenamiento a partir de los datos originales ya etiquetados (*aprendizaje supervisado*). En el caso de las imágenes esta técnica se aplica agregando ruido, realizando rotaciones, traslaciones, aumentos, etc . . .



**Figura 3.10** Ejemplo de operaciones de Data Augmentation para imagen de la BBDD.

Finalmente para validar nuestro modelo hace falta el uso de *datos de validación*, los cuales podemos obtener de nuestra base de datos de entrenamiento realizando previamente una división entre dichos datos, asignando un valor de entre el 70%-80% como datos para entrenar a la red y el 30%-20% para validar la red.



# 4 Técnicas de seguimiento

---

## 4.1 Introducción y desafíos al seguimiento de rostros

Las técnicas de seguimiento de objetos (*object tracking*) consisten en el conjunto de operaciones para la detección y seguimiento de objetos (*agentes*) en secuencias de imágenes de vídeo, en las que se conserva la identidad y localización de dicho agente en todo momento.

La aproximación clásica al problema de seguimiento se basa en la utilización de filtros para la predicción de trayectorias junto a la implementación de una función de error. Esta función permite comparar los resultados obtenidos por la predicción de dicho filtro frente al valor medido a partir del detector, los cuales pasan a ser procesados a través de un bucle de retroalimentación en el que se actualizará el próximo valor volcado por el filtro predictor.

Esta solución, aunque correcta, no es sencilla de implementar en escenarios carentes de restricciones. En el caso de los rostros, la perdida de la referencia de la cara en una secuencia de vídeo provocará la desactualización de los valores predichos por el filtro y en consecuencia, actualizaciones abruptas de la función de error que harán que el sistema de seguimiento sea poco preciso. Algunas de las causas tras las que se esconden la razón para que se produzca dicha perdida de la referencia son, por ejemplo:

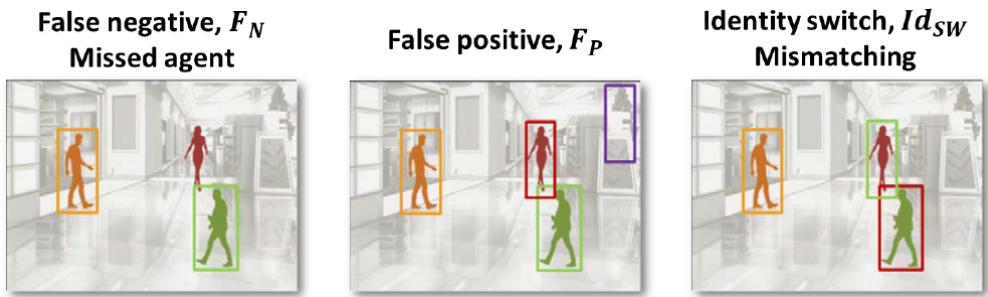
- Que el rostro esté rotado en un cierto ángulo que impida su detección.
- Condiciones de iluminación adversas.
- Oclusiones parciales o totales del rostro debido a obstáculos.

Es por ello que para este trabajo, se propone una aproximación diferente al seguimiento predictivo: *el seguimiento por detecciones*. Una nueva aproximación al problema de seguimiento que tomará en su lugar la información referente a los N agentes detectados en un frame L-1, que pasarán a ser comparados con los M agentes detectados en un frame L, mediante la utilización de una función de costes y un algoritmo de reasignación de identidades.

En este nuevo paradigma, tres son los principales errores que podremos encontrar en nuestro sistema:

- **Falsos negativos:** producidos por fallos en la detección de los agentes.
- **Falsos positivos:** causados por la asociación de un objeto distinto al agente a detectar.
- **Intercambios en la identidad** debido a una asociaciones equivocadas entre agentes.

Estos errores (especialmente los falsos positivos y negativos) se pueden ver incrementados drásticamente debido a la falta de precisión por parte del detector, razón que motiva a que se requieran sistemas de seguimiento robustos.



**Figura 4.1** Representación de los tres tipos de errores que podemos hallar en el sistema de seguimiento por detecciones [18].

Además del problema de seguimiento, en una secuencia de vídeo, podemos encontrar de manera adicional *el problema de la reidentificación*. Este problema se da especialmente en sistemas de visión que emplean múltiples ángulos de cámara, como es el caso de la video vigilancia. Aquí la transferencia de información del agente resultará una tarea especialmente difícil, ya que el sistema no dispone de imágenes previas de entrenamiento para reconocer los rostros y por lo tanto deberá de obtener modelos dinámicos de reconocimiento basados únicamente y exclusivamente en las imágenes con las que trabaje.

## 4.2 Escenario de seguimiento y soluciones propuestas

El problema de seguimiento se puede abordar a través de múltiples ópticas, siendo algunas más precisas y eficientes dependiendo de las condiciones del entorno de monitorización. En este caso, el entorno propuesto será un medio no lineal en donde habrá ocasiones en las que se perderá la referencia del agente por razones desconocidas. Se proponen tres soluciones a este escenario, de las cuales se presentarán sus fundamentos, ventajas y desventajas.

### 4.2.1 Filtros de Kalman

Los *filtros de Kalman* (KF) [19] son un método empleado comúnmente para la eliminación de ruido en aplicaciones relacionadas al procesado de señales e imágenes. También se emplean en aplicaciones relacionadas al seguimiento de objetos, dado que su capacidad para filtrado es aplicada al ruido inherente en las mediciones realizadas por los sensores. La utilidad de este filtro radica en su capacidad para realizar estimaciones dinámicas del estado de los agentes (predice su próximo estado).

Un filtro de Kalman convencional parte de una hipótesis de incertidumbre, donde el sistema dinámico a observar se supone lineal y con distribución de ruido gaussiana. Este tipo de filtros son fácilmente implementables ya que requieren de una baja carga computacional, aunque sin embargo se limitan únicamente a la resolución de problemas lineales, lo cual restringe su uso a la hora de ser implementados en sistemas reales donde se produzcan no linealidades. Es debido a esta restricción, que los filtros de Kalman emplean módulos adicionales que crean filtros mixtos, como los *Extended Kalman Filter* y *Unscented Kalman Filter* (EKF y UKF) [32]. Sin embargo, esta nueva clase de filtros requieren de un conocimiento previo de las condiciones del entorno de monitorización, así como de un afinado manual de sus parámetros de control.

El filtro de Kalman se pueden describir a partir de la interacción de las siguientes ecuaciones:

$$x_k = Ax_{k-1} + Bu_k + w_k \quad (4.1)$$

$$z_k = Cx_k + v_k \quad (4.2)$$

Donde la ecuación 4.1 da información sobre el estado (proceso) del sistema. El estado actual  $x_k$  depende del estado previo  $x_{k-1}$ , de la señal de control  $u_k$  y del ruido de proceso  $w_k$ . La ecuación 4.2 se compone del valor medido  $z_k$ , que depende del estado actual  $x_k$  y del ruido de medición  $v_k$ . Los coeficientes A, B y C son parámetros de control de valor numérico constante, y  $w_k$  y  $v_k$  representan al ruido blanco gaussiano.

#### 4.2.2 Filtros de Partículas

Los *filtros de partículas* (PF), son uno de los métodos más implementados a la hora de modelar sistemas no lineales con distribución de ruido no gaussiana. Esta clase de filtros emplean un algoritmo basado en el análisis estadístico de *Monte Carlo* (MC), que realiza muestreos aleatorios para crear funciones de distribución de probabilidad posterior del espacio de estado de forma iterativa. Gracias a este tipo de aproximación, los filtros de partículas tienen la capacidad para operar a partir de múltiples hipótesis [41].

Un filtro de partículas puede ser descrito de manera similar a Kalman, a partir del siguiente par de ecuaciones:

$$x_k = h_k(x_{k-1}, w_{k-1}) \quad (4.3)$$

$$z_k = h_k(x_k, v_k) \quad (4.4)$$

Donde  $x_k$ ,  $x_{k-1}$ ,  $w_{k-1}$ ,  $z_k$  y  $v_k$ , conservan los mismos significados que en el KF, y  $h_k$  es  $h_k : RxR \rightarrow R$  una posible función no lineal, que da a la salida un conjunto de partículas ponderadas por pesos en función de su cercanía respecto a la medida del estado predicho [11].

Si bien es cierto que el PF nos permite trabajar con sistemas no lineales y realizar el seguimiento de objetivos que puedan maniobrar súbitamente de manera brusca, esta clase de filtros tampoco terminan por ser la mejor solución. Esto se debe que al contrario que pasa en los KF, los PF operan de manera considerablemente peor en escenarios donde el comportamiento del objetivo se puede modelar como un problema lineal.

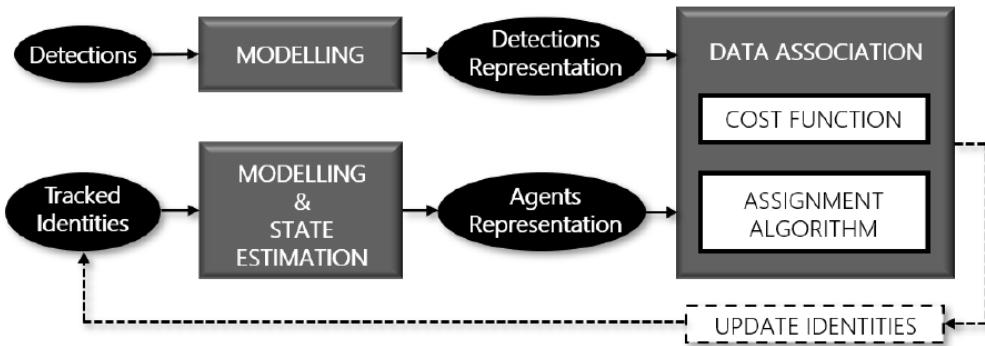
#### 4.2.3 Seguimiento por detecciones (*Tracking by detections*)

El seguimiento por detecciones (*tracking-by-detections*) es la técnica más expandida y con mejores resultados a la hora resolver problemas de monitorización a través de videovigilancia. En lugar de localizar a los agentes a partir de la predicción de la posición, se emplean modelos basados en el paradigma del aprendizaje máquina. Este tipo de sistemas se basan en la selección de un sistema de detección y una fase de asignación de datos en un bucle de retroalimentación, tal y como se expone en a figura 4.3.

En un primer paso, el sistema de detección ofrece potenciales localizaciones de objetos a partir de un conjunto de *bounding boxes*<sup>1</sup>. A continuación, el sistema de seguimiento se encarga de realizar la asociación entre los modelos de los objetos detectados en el frame actual, frente a modelos de objetos ya previamente identificados en frames anteriores. Para ello se emplea una función de costes que asigna métricas entre los diferentes objetos, y un algoritmo optimización para la minimización de costes en las métricas de asignación.

Esta técnica pese a ser una de las más efectivas en la tarea de seguimiento, como contrapunto hay que resaltar que es una de las más costosas a nivel computacional. Sin embargo será la aproximación que se empleará para este trabajo, dado que esta última restricción no resultará en si un problema a tener en cuenta. Para este método de seguimiento, dos son principales enfoques empleados a la hora de la asignación de métricas de costes:

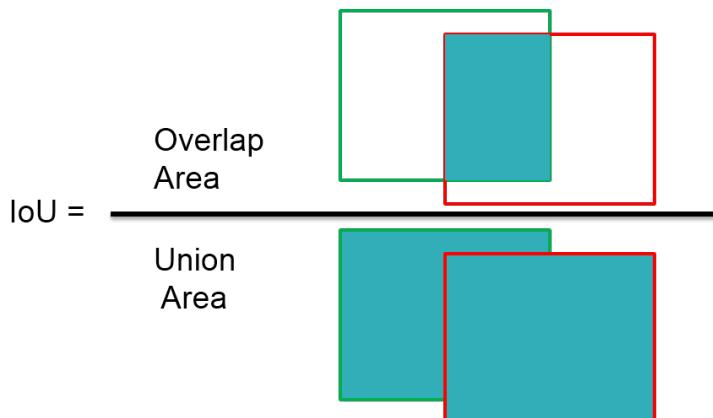
<sup>1</sup> En español *cuadro delimitador*, es aquel elemento que en análisis de imagen delimita gráficamente una región de interés.



**Figura 4.2** Esquema de detección de un algoritmo de seguimiento mediante detecciones [18].

#### Métrica: Inserción sobre la Unión

Inserción sobre la unión o *Inserction-Over-Union* (IoU) es una métrica que expresa la similitud entre dos *bounding boxes*. Para esta aplicación, se escogen dos *bounding boxes* pertenecientes a las detecciones del frame L y L-1. Despues se calcula el coeficiente que relaciona al área de la conjunción de ambas regiones frente al área de la unión. Finalmente el resultado se compara con un valor de umbral (*threshold*) que determina la similitud entre ambas detecciones.



**Figura 4.3** Aplicación gráfica de *Inserction-Over-Union*.

Cabe destacar que el uso de esta técnica se limita a problemas de seguimiento en donde existe una continuidad de plano, y que por lo tanto es sensible ante cambios repentinos de perspectiva en la secuencia de vídeo.

#### Métrica: Distancias Euclídeas

Esta métrica ya ha sido previamente explicada en la sección 2.2.4. En este caso, se emplea el vector de características del objeto que ha sido detectado frente a los modelos obtenidos en frames anteriores. A continuación se calculan las distancias euclídeas entre ellos y se escoge aquel cuya distancia sea la menor respecto de un umbral de asignación. Finalmente se actualiza el modelo detectado por la medida actual, para garantizar la distancia mínima en la siguiente medida que se realice.

#### Asignación: *Greedy Assignment Algorithm*

El objetivo de un algoritmo es el de hallar el valor óptimo a la solución de un problema de optimización. En el caso del algoritmo *greedy*, la búsqueda de este óptimo pasa por la selección de máximos locales en cada iteración del algoritmo, esperando alcanzar así el máximo global de la función.

Un ejemplo para visualizar el algoritmo se puede hallar en la figura 4.4, donde la ruta naranja representa al óptimo de un problema de suma máxima de nodos frente a la ruta azul, que representa la ruta escogida por el algoritmo *greedy* en cada iteración.

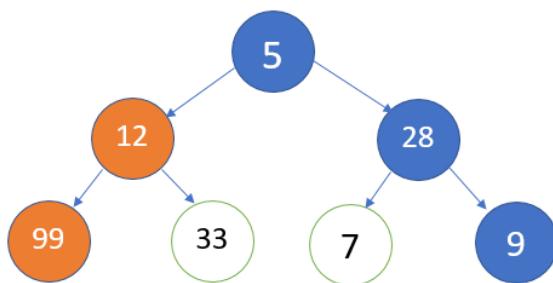


Figura 4.4

Pese a no garantizar la obtención del resultado óptimo, este algoritmo ofrece simplicidad a la hora de ser implementado y gran rapidez en tiempo de ejecución.



# 5 Sistema Propuesto

---

## 5.1 Entorno de desarrollo

Este proyecto ha apostado principalmente para su realización, por la facilidad y simplicidad de acceso a un entorno de desarrollo y experimentación de elevado nivel de potencia computacional. Para ello, este sistema ha sido diseñado para ser desplegado en un entorno virtual y gratuito alojado en la nube, para que cualquier persona que lo deseé pueda realizar experimentos sobre los diferentes aspectos relacionados al procesado de rostros en imágenes y vídeo.

Se podrá acceder a este proyecto a través del siguiente repositorio: [https://github.com/Tsuru-ai/TFG\\_python\\_face\\_recognition](https://github.com/Tsuru-ai/TFG_python_face_recognition)

### 5.1.1 Herramientas de programación

#### Lenguaje

El lenguaje empleado para este trabajo será Python [2] en su versión 3.6.9. La razón de su uso es el amplio número de librerías para el desarrollo de aplicaciones basadas en visión artificial, reconocimiento facial y algoritmos de Machine Learning. Además, es el lenguaje empleado por el framework<sup>1</sup> en el que se ha desarrollado este proyecto.

#### Framework

El framework empleado para este sistema es *Google Colaboratory* [1], también conocido como *Google Colab*. Este framework proporcionado por Google se caracteriza por tener un diseño orientado al desarrollo de aplicaciones basadas en Machine Learning y Big Data<sup>2</sup> con fines educativos. Se basa en un servicio alojado en la nube que emplea *Jupyter Notebooks* ya preconfigurados y que además, ofrecen acceso a GPUs y TPUs<sup>3</sup> de forma gratuita.

#### Hardware

El hardware empleado para esta aplicación será una GPU virtual ofrecida por el entorno de Google Colaboratory. El conjunto de GPUs ofrecidas por la plataforma son todas pertenecientes al fabricante de tarjetas gráficas Nvidia, en sus modelos K80, T4, P4 y P100, que nos permitirá trabajar con sus módulos de aceleración CUDA [31]. Al iniciar una sesión de Colab, el sistema nos asignará una

---

<sup>1</sup> Entorno de desarrollo de programación

<sup>2</sup> Big Data o *macrodatos* es un campo de estudio que investiga la recolección, tratamiento y análisis del conjunto de datos tan grande y complejo que precisa de aplicaciones informáticas no tradicionales de procesamiento de datos para tratarlos de manera adecuada.

<sup>3</sup> Unidad de Procesamiento Tensorial o *Tensor Processing Unit* es un circuito integrado de aplicación específica para aceleración de IA elaborado por Google y desarrollado para aplicaciones de aprendizaje automático basadas en redes neuronales artificiales. Están específicamente optimizadas para utilizar con la librería *TensorFlow*.

GPU de este conjunto de tarjetas según su disponibilidad, por lo que no será posible seleccionar un modelo concreto con el que operar.

### Librerías

El sistema hace uso de las siguientes librerías

- **Numpy** [27]: (versión 1.18.5) es la librería de Python empleada para realizar operaciones con matrices y vectores.
- **OpenCV** [7]: (versión 4.1.2) es la librería con funciones empleadas en aplicaciones de visión artificial.
- **Face\_Recognition** [14]: (versión 1.3.0) es la librería utilizada para la detección, preprocesado y extracción de características de rostros en imágenes. Por defecto utiliza el método HOG [10] para la detección, sin embargo para este trabajo se utilizará la CNN ya que es más eficiente [5].
- **dlib** [20]: (versión 19.18.0) esta librería escrita en C++ y adaptada para Python, incluye multitud de funcionalidades basadas en machine learning, entre las que se incluyen la CNN a emplear en este trabajo.
- **Scikit-Learn** [8]: (versión 0.22.2.post1) es la librería a través de la que se implementarán los diversos algoritmos de clasificación estudiados en este trabajo.
- **Funciones decoradoras**: los decoradores o "*decorator functions*", son una utilidad de metaprogramación disponible en Python. Gracias ellas podemos tomar funciones ya existentes y agregar nuevas funcionalidades. En este trabajo se han implementado decoradores de las librerías *functools*, *time* y *typing* para crear funciones decoradoras que nos permitan comprobar los tiempos de ejecución de las diferentes procesos relacionados a la ejecución de los algoritmos y sus subprocessos.

## 5.2 Estructura del sistema

En esta sección, se expondrán los diferentes métodos empleados para la implementación del sistema de detección, reconocimiento y seguimiento de rostros propuestos para este trabajo.

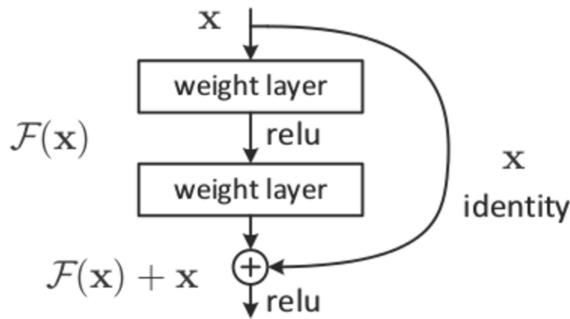
### 5.2.1 Detección, preprocesado y extracción de características

El método empleado para este sistema será la CNN de arquitectura ResNet [15] ofrecida por la librería Face Recognition. El fin de esta arquitectura es el de implementar una serie de conexiones denominadas de salto o "*skip*" en las primeras capas de la CNN para mejorar su fase de entrenamiento y precisión en las predicciones.

La razón de introducir estos saltos, se debe a la aparición del problema del desvanecimiento del gradiente o *vanishing gradient* [16], problema común en redes neuronales con elevados números de capas. Este problema se relaciona junto al ya presentado algoritmo de backpropagation y se debe a que al aplicar dicho algoritmo, los valores de los gradientes resultados de ajustar los pesos de las neuronas de una capa L, se van viendo reducidos a medida que dicha capa está más alejada de la última capa de la red (primeras capas).

Analíticamente este fenómeno ocurre ya que al aplicar la regla de cadena sobre las derivadas de las funciones de coste de cada capa, la acumulación de productos de las derivadas parciales acabarán convergiendo alrededor del cero (cambios aplicados sobre los pesos de las neuronas imperceptibles).

Una vez la cara haya sido detectada por la red, esta pasará a ser preprocesada y se extraerá su vector de características, codificandolo como un vector de 128 dimensiones. Dichas componentes



**Figura 5.1** Diagrama de una conexión de salto (*skip*) sobre una capa de una red tipo ResNet.

carecen de un significado físico como tal, por lo que únicamente caben a ser interpretadas por la red.

Esta red está basada en el modelo de la librería dlib *cnn\_face\_detection\_model\_v1*, y ha sido entrenada a partir de más de tres millones de imágenes de rostros procedentes de los datasets face scrub[28] y VGG [42], siendo esta red validada mediante el dataset LFW con un 99.38 % aciertos según su creador.

### 5.2.2 Clasificación y comparación

En esta fase, el sistema de reconocimiento crea los modelos de clasificación a partir de los vectores de características de 128 dimensiones extraídos en la fase anterior. Para ello, se dará la posibilidad de emplear la librería de Python sklearn, para implementar los diferentes algoritmos de clasificación expuestos en este trabajo. Los parámetros que se recomiendan modificar para cada uno de los siguientes clasificadores a la hora de experimentar son:

- **Multilayer Perceptron:** Este clasificador se basa en el modelo ya expuesto de la sección 3.2.1. Los valores de los parámetros que se recomiendan modificar son número de capas (*activation*), número de neuronas y función de activación (*hidden\_layer\_sizes*).
- **K-Nearest-Neighbours:** El parámetro recomendado a modificar para este clasificador será el número de vecinos (*n\_neighbours*), que por defecto será 5.
- **Random Forest:** El parámetro recomendado a modificar para este clasificador será el número de estimadores empleados (*n\_estimators*), que por defecto será 100.
- **State Vector Machine:** El parámetro recomendado a modificar para este clasificador será el tipo de kernel empleado, que por defecto será una función de base radial (*kernel='rbf'*)

### 5.2.3 Algoritmo de seguimiento

El algoritmo de seguimiento empleado en este trabajo será el método de seguimiento por detecciones ya expuesto en la sección 4.2.3 de este trabajo. Debido a que el enfoque de este sistema pretende ser lo más generalista, rápido y preciso posible, se ha optado por el uso de métricas de seguimiento basadas en la comparación de distancias euclídeas entre los rostros detectados y el uso del algoritmo de asignación greedy.

El algoritmo comienza detectando un rostro en la secuencia de video, creando un objeto a través de la clase *Object()* y asignando a este una serie de características identificadorias (localización, encodings, tiempo desaparecido en la secuencia, etc...). Una vez creado el objeto este pasa a ser asignado a una lista de objetos activos en seguimiento a través de la clase *FaceTracker()*, la cual realizará el siguiente conjunto de acciones cada vez que se aplique el método de actualización *update()*:

1. Asignar el rostro a una lista de objetos activos en seguimiento.
2. Asignar las caras detectadas en el frame anterior a la caras detectadas en el frame actual, aplicando la distancia euclídea como métrica de comparación y el algoritmo greedy como método de asignación.
3. En el caso de que un rostro que estuviera en activo no haya sido detectado, iniciar un contador. Dicho contador es reiniciado si el rostro se vuelve a detectar dentro de un periodo de tiempo determinado.
4. En caso de que un rostro no vuelva aparecer en el periodo de tiempo establecido, borrar el objeto asociado a dicho rostro.
5. Asignar el rostro nuevo detectado a un objeto de la clase FaceTracker().

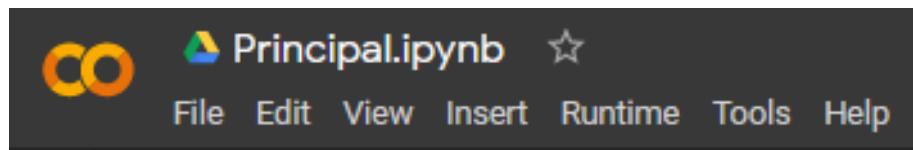
### 5.3 Interfaz del Sistema

#### 5.3.1 Instalación y orden de los directorios

La interfaz del sistema será el ya citado entorno Google Colaboratory. Para ello primero deberemos abrir un navegador web y acceder a nuestra cuenta de Google Drive. Desde ahí accedemos a la aplicación de *Google Workspace Marketplace* y vinculamos nuestra cuenta de Drive a la aplicación de Colaboratory. Una vez vinculada la aplicación, habrá que descargar el archivo comprimido del repositorio indicado al principio del capítulo, descomprimir dicho archivo y subir su contenido al directorio raíz del sistema de almacenamiento de *Google Drive*.

El directorio */Programa* se divide a su vez en los siguientes subdirectorios y ejecutables:

- **Principal.ipynb:** Es el archivo principal del programa en formato ".ipynb". La primera vez que se abra será necesario configurarlo para que procese mediante GPU. Para ello hay que acceder a la pestaña Runtime → *Change runtime type* → *hardware accelerator* → *GPU*.



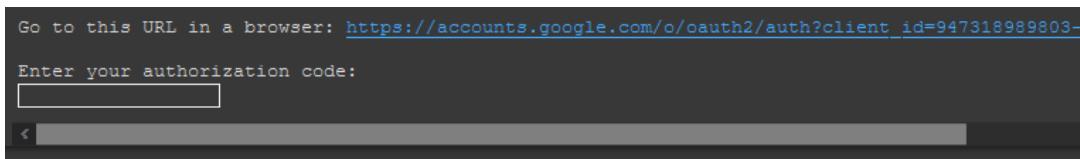
**Figura 5.2** Menú del archivo Principal.ipynb.

- **Personas:** aquí se almacenan las imágenes de entrenamiento para generar los archivos .pkl con los que entrenar a los clasificadores. Este directorio se divide a su vez en subdirectorios que contendrán el conjunto de imágenes de entrenamiento del modelo de cada persona. Se recomienda que en dichas imágenes se halle única y exclusivamente la persona que se deseé reconocer, en la mayor variedad de estilos y poses. Además se recomienda que el número mínimo de imágenes por directorio sean 30.
- **Model:** aquí se almacenan los archivos en formato .pkl de los rostros que han sido codificados para entrenar a los clasificadores del sistema. Existen dos clases de archivos, "*name.pkl*" que contiene los vectores codificados y etiquetas de cada rostro del directorio */Personas* y "*name\_clasifier.pkl*", que contiene el modelo generado por el tipo de clasificador.
- **Imágenes:** en este directorio se almacenan las imágenes en formato .jpg o .png que se deseen testear mediante el sistema.
- **Vídeos:** en este directorio se almacenan los vídeos en formato .mp4 o .avi que se deseen testear mediante el sistema.

### 5.3.2 Inicialización y ejecución

El archivo "Principal.ipynb" está dividido en diferentes grupos de celdas. Cada celda se puede ejecutar pulsando el botón "play" situado en la región superior izquierda.

La primera celda siempre se debe ejecutar para iniciar el programa, pues sincroniza nuestro directorio Drive con la aplicación, a la vez que instala e importa las librerías a emplear. A la salida de su ejecución, nos pedirá que accedamos a un enlace a través del cual obtendremos una clave de sincronización de directorios. Accedemos a enlace, copiamos la clave facilitada en el portapapeles y la introducimos en la ventana.



```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-
Enter your authorization code:

```

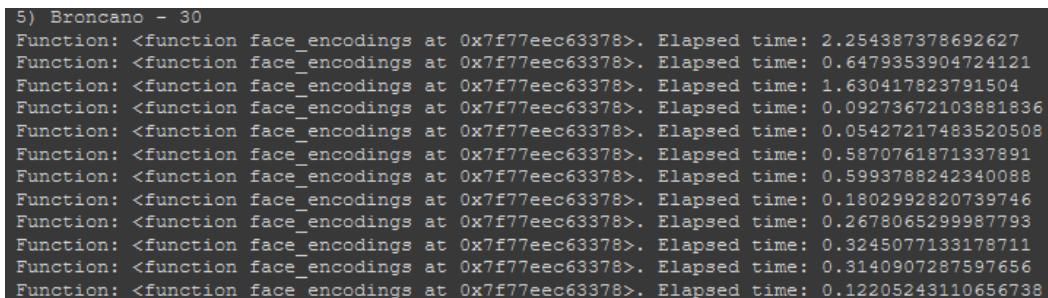
**Figura 5.3** Ventana de sincronización de directorios.

Una vez ejecutada exitosamente la primera celda, se podrán ejecutar las siguientes secciones del programa.

#### Entrenamiento - Clasificación de modelos

En esta sección se ejecutan las celdas mediante las que obtenemos los archivos .pkl:

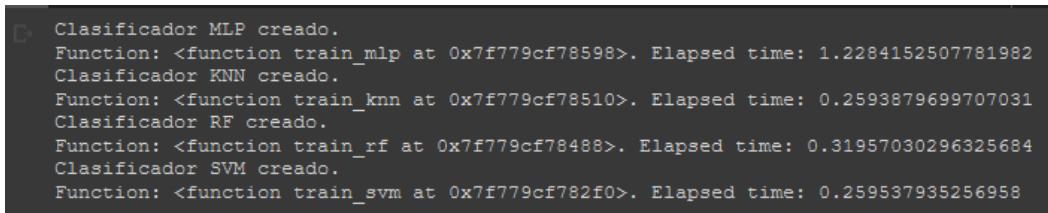
- La primera celda ejecuta las funciones para crear el fichero "Names.pkl" que contiene las etiquetas y vectores extraídos tras el análisis de las imágenes del directorio /Personas.



```
5) Broncano - 30
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 2.254387378692627
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.6479353904724121
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 1.630417823791504
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.09273672103881836
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.05427217483520508
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.5870761871337891
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.5993788242340088
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.1802992820739746
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.2678065299987793
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.3245077133178711
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.3140907287597656
Function: <function face_encodings at 0x7f77eec63378>. Elapsed time: 0.12205243110656738
```

**Figura 5.4** Resultado en tiempo de ejecución de extracción de características de un individuo.

- La segunda celda ejecuta las funciones que crean los ficheros en formato .pkl de los clasificadores a entrenar. Cada función estará inicializada con un determinado valor en sus parámetros, aunque se anima a experimentar a variar sus valores según las indicaciones de la sección 5.2.2.



```
Clasificador MLP creado.
Function: <function train_mlp at 0x7f779cf78598>. Elapsed time: 1.2284152507781982
Clasificador KNN creado.
Function: <function train_knn at 0x7f779cf78510>. Elapsed time: 0.2593879699707031
Clasificador RF creado.
Function: <function train_rf at 0x7f779cf78488>. Elapsed time: 0.31957030296325684
Clasificador SVM creado.
Function: <function train_svm at 0x7f779cf782f0>. Elapsed time: 0.259537935256958
```

**Figura 5.5** Resultado en tiempo de ejecución de los modelos de clasificación.

- La tercera celda define clases orientadas al procesado de imágenes de entrada y umbral de similitud para ser considerado como correcto (se suele recomendar un valor de confianza mayor o igual al 80 %). También se establece que clasificador se empleará por parte del sistema de reconocimiento.

Finalmente se dispone de un apartado de experimentos, donde se podrán realizar diversas pruebas para determinar la robustez del sistema de reconocimiento de rostros en imágenes.



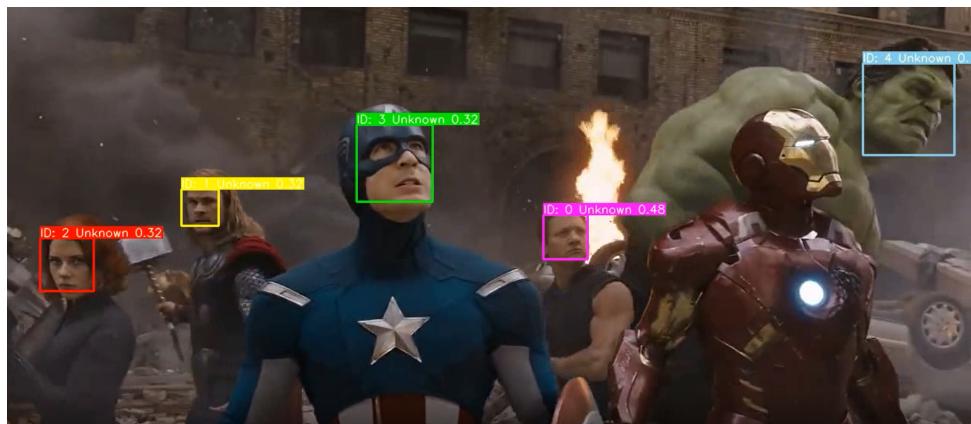
**Figura 5.6** Identificación de tres rostros entrenados en una imagen aplicando un clasificador SVM.

### Video-Tracking

En esta sección se ejecutan las celdas para poner en marcha el sistema de procesado de vídeo:

- La primera y segunda celda sirven tanto para inicializar las clases que realizan el procesado de los frames de los vídeos, como para generar objetos que identifiquen a los rostros seguidos. La clase *FaceTracker()* es la encargada de aplicar las métricas de seguimiento expuestas para este sistema. Además se incluye una variación del sistema de *bounding boxes* para asignar un color diferente a cada rostro y así hacer más sencillo el seguimiento visual.

Una vez ejecutadas estas celdas se podrá proceder a las celdas de experimentación, donde se podrá poner a prueba el sistema de seguimiento mediante secuencias de vídeo.



**Figura 5.7** Ejemplo de un frame al que se le ha realizado video-tracking.

# 6 Experimentos y Resultados

---

**E**n este capítulo se recogen los diferentes experimentos que han sido considerados como de interés para determinar tanto la fiabilidad, como la robustez del sistema propuesto para este trabajo. Para ello se proponen tres experimentos: uno para la determinación de un clasificador óptimo, otro para determinar un umbral de confianza óptimo y un último relacionado a la robustez del sistema de seguimiento.

## 6.1 Experimento 1: Determinación de clasificadores óptimos

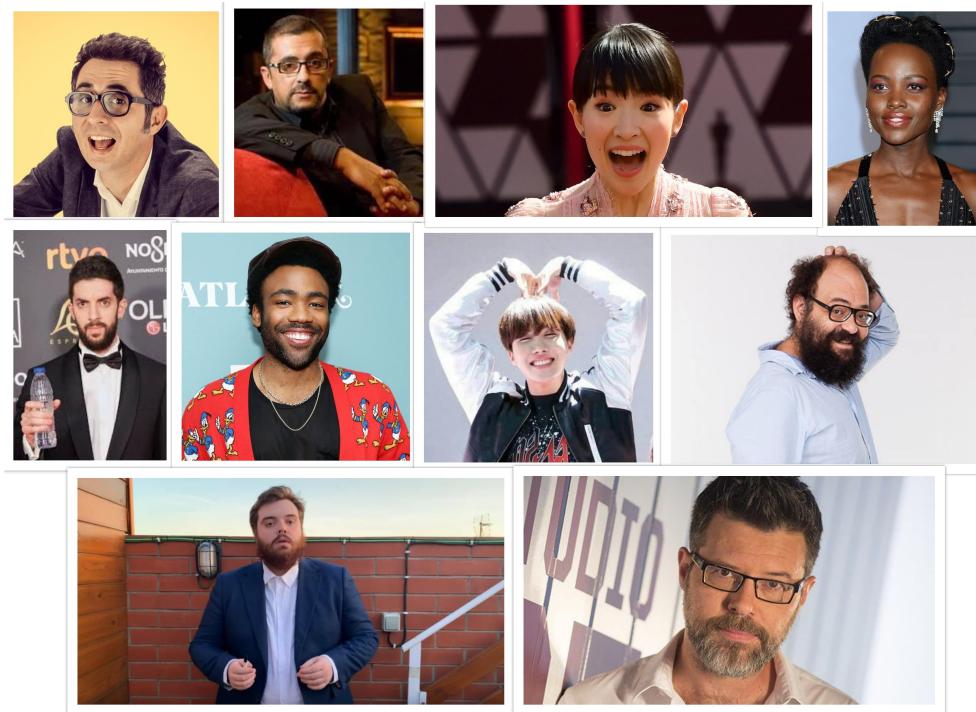
Para este experimento se ha decidido comparar la eficiencia de los cuatro algoritmos de clasificación de rostros presentados en este trabajo con el fin de determinar un clasificador óptimo. Dichos clasificadores son el *Multi Layer Perceptron*, *K-Nearest Neighbours*, *Random Forest* y *State Vector Machines*.

Cada clasificador opera de forma parecida: primero toma un vector de *encodings* de 128 dimensiones a la entrada, lo procesa y devuelve a la salida una etiqueta con la predicción de la persona identificada y un porcentaje de confianza con el que la realiza. La etiqueta puede contener o bien el nombre de una las personas con las que se ha entrenado el clasificador, o bien el valor "*Unknown*" en caso de que la confianza del sistema no haya superado el valor de umbral requerido para hacer una asignación.

Partiendo de esta premisa, se propone un experimento en el que estos clasificadores en lugar de tomar como entrada los valores codificados del vector de características de una cara, toman en su lugar vectores construidos a partir de valores aleatorios. Es decir, que a través de un elevado número de muestras aleatorias, se puede determinar que clasificador ofrece la *confianza media* más baja y en consecuencia el menor número de *falsos positivos*.

Para ello primero se ha construido una base de datos con imágenes de personas que serán empleadas para construir los clasificadores. Dichas personas de esta base de datos se tratan en total de diez personajes públicos y famosos, de los que se han tomado treinta imágenes de cada uno a través de internet. El criterio para seleccionar estas imágenes de entrenamiento es que la persona en cuestión se halle exclusivamente en la imagen y tenga la cara despejada, aunque también se admiten algunos complementos como gafas de ver, de sol o auriculares. También otro criterio que ha sido seguido para seleccionar a dichas personas ha sido la variedad de tonos de piel, raza y género, con el fin de así determinar si el sistema se comporta mejor o peor ante diferentes individuos.

Una vez generada la base de datos, toca generar los diferentes clasificadores realizando modificaciones sobre sus parámetros de entrenamiento y a continuación, introducir una matriz de 1000 vectores aleatorios de 128 dimensiones. En los siguientes apartados se pueden comprobar los resultados obtenidos de dichos experimentos para cada clasificador.



**Figura 6.1** Imágenes de la base de datos de entrenamiento.

### 6.1.1 Clasificador MLP

Para este clasificador el parámetro sobre el que se ha operado ha sido el número de capas y el número de neuronas por cada capa, dejando como constante la función de activación *tangente hiperbólica*. Este clasificador se considera de los menos eficientes a la hora de realizar predicciones en imágenes, dado que su estructura de capas densas le hace sobre especializarse y producir *overfitting*. Es decir, que tenderá a ofrecer con mayor probabilidad una predicción basada en los datos de entrenamiento aunque el vector de entrada no se corresponda con ninguno de la base ya entrenada.

**Tabla 6.1** Resultados entrenamiento clasificador Multi Layer Perceptron.

Nº de capas ocultas	Neuronas en cada capa	Nº de Falsos Positivos	Confianza Media
1	150	755	88.49 %
1	200	758	88.41 %
2	150/50	622	81.47 %
2	200/50	663	83.76 %
3	200/100/50	411	72.01 %

De estos resultados se escoge el clasificador con menor número de falsos positivos y menor confianza media, que sería el de 3 capas ocultas con 200, 100 y 50 neuronas en cada capa respectivamente.

### 6.1.2 Clasificador KNN

Para este clasificador los parámetros sobre los que se ha operado son tanto el número de vecinos a tener en cuenta para la predicción, como la representatividad del peso de estos en función a su distancia respecto de la muestra, siendo el parámetro *weights="uniform"* para la igualdad entre pesos y *weights="distance"* para que estos sean inversamente proporcionales.

**Tabla 6.2** Resultados entrenamiento clasificador K-Nearest Neighbours con pesos uniformes.

Nº de vecinos	Nº de Falsos Positivos	Confianza Media
5	186	62.05 %
7	188	60.25 %
9	82	57.85 %
11	120	55.95 %
13	67	54.01 %

**Tabla 6.3** Resultados entrenamiento clasificador K-Nearest Neighbours con pesos según el inverso de la distancia.

Nº de vecinos	Nº de Falsos Positivos	Confianza Media
5	343	62.28 %
7	157	59.58 %
9	92	57.85 %
11	124	56.03 %
13	73	54.15 %

De estos resultados se escoge el clasificador con menor número de falsos positivos y menor confianza media, que sería el de pesos uniformes y 13 vecinos.

### 6.1.3 Clasificador RF

Para este clasificador el parámetro sobre el que se ha operado ha sido el número de estimadores a considerar para cada predicción.

**Tabla 6.4** Resultados entrenamiento clasificador Random Forest.

Nº de estimadores	Nº de Falsos Positivos	Confianza Media
25	0	24.15 %
50	0	27.69 %
100	0	24.03 %
200	0	24.78 %

De estos resultados se escoge el clasificador con menor número de falsos positivos y menor confianza media, que sería el de 25 estimadores.

### 6.1.4 Clasificador SVM

Para este clasificador el parámetro sobre el que se ha operado, ha sido el tipo de función empleada por el kernel.

**Tabla 6.5** Resultados entrenamiento clasificador SVM.

Función empleada por el kernel	Nº de Falsos Positivos	Confianza Media
poly	2	27.55 %
linear	552	76.38 %
rbf	0	17.26 %
sigmoid	744	87.34 %

De estos resultados se escoge el clasificador que menor número de falsos positivos y menor confianza media, que sería de kernel con función de base radial.

#### 6.1.5 Resultados y conclusión del experimento

Escogiendo cada uno de los clasificadores óptimos de las tablas anteriores, se puede obtener la siguiente tabla comparativa:

**Tabla 6.6** Comparación de resultados entre clasificadores óptimos entrenados.

Tipo de clasificador	Nº de Falsos Positivos	Confianza Media
Multi Layer Perceptron	411	72.01 %
K-Nearest Neighbours	67	54.01 %
Random Forest	0	24.15 %
SVM	0	17.26 %

Por lo que se concluye a partir de este experimento que **el clasificador óptimo es el clasificador SVM con función de kernel de base radial**.

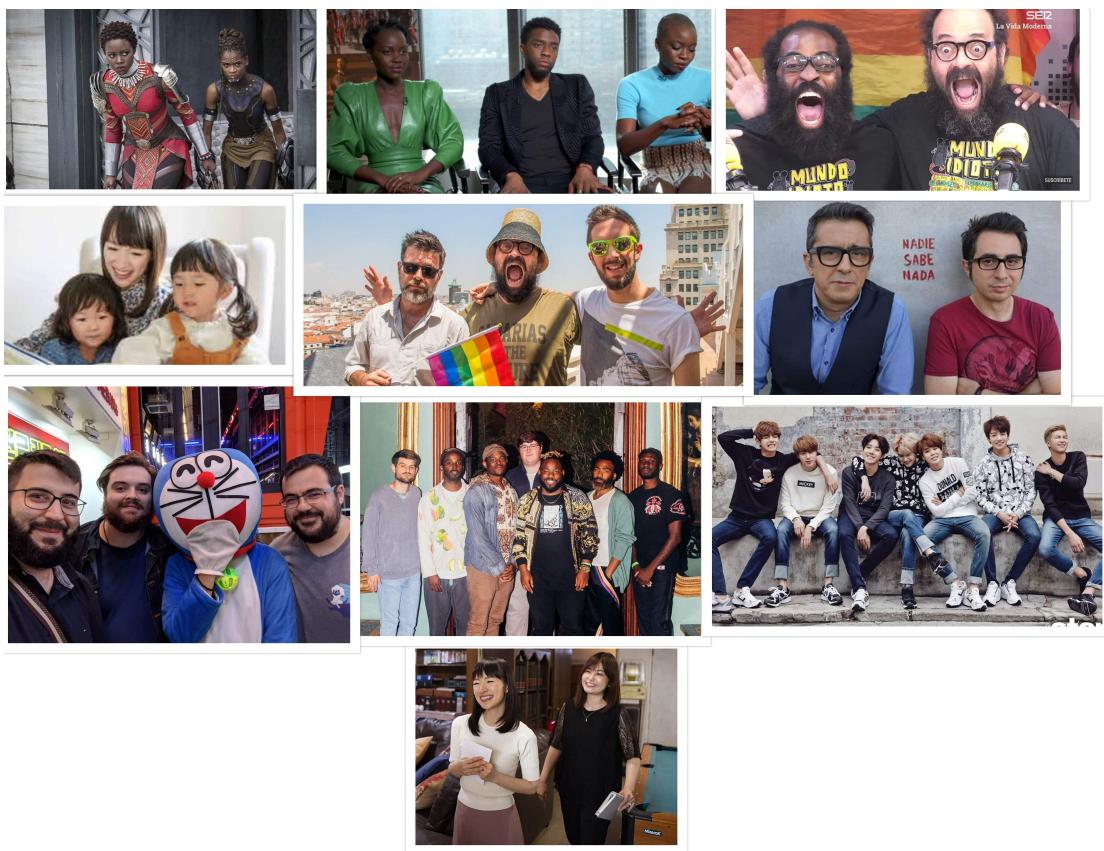
## 6.2 Experimento 2: Determinación de umbral de confianza óptimo

Una vez hemos determinado el clasificador óptimo a emplear en el sistema de reconocimiento de rostros, el siguiente paso es determinar el umbral de confianza mínimo con el que realizar las predicciones.

Para ello se han tomado diez imágenes en donde aparecen las personas entrenadas por el sistema de reconocimiento. Dichas imágenes de prueba se denominan *datos de validación*, y no pueden pertenecer a los datos de entrenamiento con los que se ha generado el modelo de clasificación. Además en dichas imágenes, las personas a identificar aparecen junto a otras personas que no han sido entrenadas por el sistema, a fin de determinar un umbral óptimo de sensibilidad a la confianza.

Para comprobar la validez de estas detecciones suponemos tres casos:

- **Asignación correcta:** el clasificador ha asignado correctamente su predicción, siendo o bien sobre persona con su nombre, o bien asignado el valor "*Unknown*" a un rostro desconocido.
- **Asignación incorrecta (*identidad no reconocida*):** este caso se produce cuando la predicción de la persona se haya por debajo del valor del umbral de confianza y da como resultado "*Unknown*".
- **Asignación incorrecta (*identidad equivoca*):** este caso se produce cuando el resultado de la predicción a una persona desconocida es una de las personas pertenecientes a la base de datos de entrenamiento.



**Figura 6.2** Imágenes de la base de datos de validación.

Una vez determinados los supuestos de la detección queda registrar los datos de experimentación en la siguiente tabla comparativa.

**Tabla 6.7** Comparación de resultados ante sensibilidad de umbral de confianza.

Valor umbral	Asignaciones correctas	Identidades no reconocidas	Identidades equivocadas
0.8	28	6	0
0.7	32	2	0
0.6	31	1	2

### 6.2.1 Resultados y conclusión del experimento

A menor sea el umbral de confianza del experimento, mayor es el número de asignaciones realizadas. Sin embargo, esto es a costa de sacrificar certeza en la predicción, por lo que es preferible ante valores semejantes de asignaciones correctas, escoger aquel umbral con el menor número de identidades equivocadas.

Por lo tanto el **valor óptimo de umbral de confianza** para este clasificador queda determinado por el valor **0.7**.

## 6.3 Experimento 3: Fiabilidad del sistema de seguimiento

Para este último experimento se pretende poner a prueba el algoritmo de seguimiento de rostros propuesto para este trabajo. Para ello se han utilizado tres secuencias de vídeo con las siguientes

características.

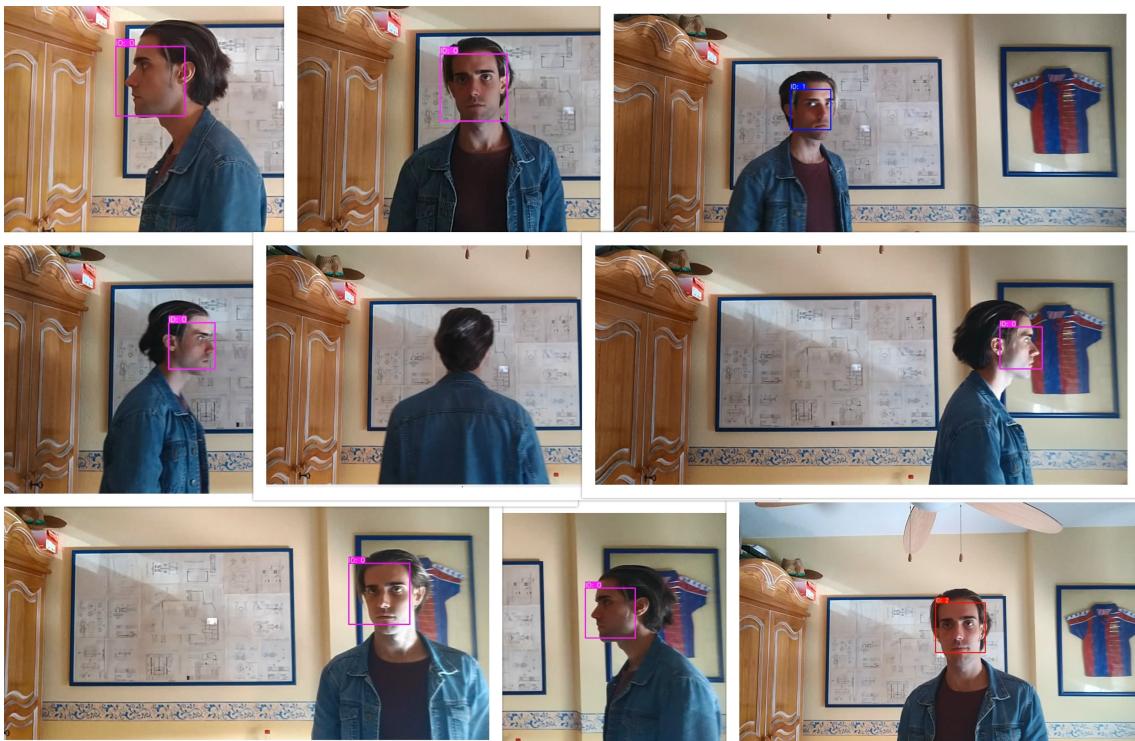
1. Secuencia de vídeo con cámara fija y rostro entrenado.
2. Secuencia de vídeo con cámara fija y rostro no entrenado.
3. Secuencia de vídeo con cámara en movimiento y rostros no entrenados.

En la primera secuencia se pone a prueba si tanto el sistema de reconocimiento como el de seguimiento trabajan correctamente de manera conjunta. Para ello se pretende demostrar si, ante variaciones en la posición del rostro y occlusiones parciales, el sistema es capaz de conservar la identidad del sujeto identificado.



**Figura 6.3** Frames de resultado pertenecientes a la primera secuencia de vídeo.

En la segunda secuencia se pone a prueba al sistema enfrentándolo a una persona desconocida que cambiará su posición y se pondrá de espaldas, haciéndole perder así la referencia, para finalmente volver a aparecer.



**Figura 6.4** Frames de resultado pertenecientes a la segunda secuencia.

En la última secuencia aparecen varios rostros desconocidos los cuales, se hayan en movimiento a la vez que lo hace la cámara y en ocasiones se producirán occlusiones parciales.



**Figura 6.5** Frames de resultado pertenecientes a la tercera secuencia.

### 6.3.1 Resultados y conclusión del experimento

De la primera secuencia podemos observar que en el primer frame, el sistema de reconocimiento facial no es capaz de identificar a uno de los sujetos de la base de datos. Sin embargo en frames posteriores lo hace y además es capaz de mantener su identidad incluso en situaciones complejas como es teniendo a la persona de perfil u ocluida parcialmente por el micrófono.

En la segunda secuencia podemos comprobar que en la mayoría de frames se mantiene la identidad del sujeto, aunque sin embargo existen dos en los que la identidad de este es cambiada. En el primero la duración de esta identificación errónea con cuadrado azul apenas dura un segundo hasta que se vuelve a reasignar con la *bounding box* correcta de color magenta, aunque la segunda con el identificador rojo se acaba manteniendo durante más tiempo.

Esto demuestra que si bien el sistema es capaz de conservar identidades con relativa fiabilidad, existe la posibilidad de que se produzcan falsos positivos y que incluso algunos se mantengan en el tiempo.

Finalmente en la tercera secuencia no se produce ningún error y todos los sujetos son identificados de manera correcta.



# 7 Conclusiones y líneas futuras

---

## 7.1 Conclusiones

Este trabajo es el resultado de un proyecto ya existente [17], al que se le ha dotado de un framework distinto de su original, que facilita el acceso al desarrollo de aplicaciones y experimentos relacionados con el análisis de rostros en imágenes y vídeo. Se ha otorgado acceso a recursos de elevado nivel computacional, mejorado algunas de las funcionalidades ya existentes relacionadas a la detección de rostros e implementando técnicas de mayor precisión para la clasificación de los mismos.

Respecto a las técnicas empleadas, se ha demostrado de forma experimental que un sistema de detección y clasificación de rostros de alto rendimiento puede ser obtenido a partir de la implementación de Redes Neuronales Convolucionales, junto con el uso de clasificadores SVM. Sin embargo, también se ha demostrado que existen otros clasificadores como el RF, que si bien no suele ser empleado en problemas de clasificación de imágenes, ha demostrado ofrecer también buenos resultados.

Se ha comprobado que en algunos casos, durante la fase de experimentación, determinados tipos de rostros ofrecían mayor valor de confianza a la hora de ser clasificados de manera errónea. Dichos rostros pertenecían a hombres de razas pertenecientes a la región oeste de Asia y a mujeres de razas pertenecientes a las regiones del sur de África.

Se ha incluido además una nueva funcionalidad: *el seguimiento de rostros a partir de encodings*, una técnica basada en el paradigma de Machine Learning. Esta técnica ha permitido realizar una aproximación distinta a las soluciones clásicas ofrecidas por el problema de seguimiento, que si bien es cierto requiere de un mayor nivel de coste en recursos de computación, ofrece a su vez una de las soluciones más estables y precisas ante sistemas sin restricciones ante la detección.

Sin embargo este sistema no está exento de fallos, ya que se ha demostrado que en algunas ocasiones se pueden producir errores en la identificación cuando el sujeto se trata de una persona no entrenada previamente por la red. Este fallo en la identificación puede propagarse al resto de la tarea de seguimiento dando como resultado la aparición de identidades múltiples sobre una misma persona.

## 7.2 Líneas futuras

Con el objetivo de mejorar el sistema presentado, se proponen a continuación algunas posibles líneas de desarrollo.

En primer lugar y en relación al sistema de seguimiento, se podría enfocar el problema de la identificación a más allá del rostro y emplear otras CNNs. Por ejemplo para detectar personas se podría emplear *YOLO (You Only Look Once)*, una CNN orientada a la detección e identificación de

objetos, como personas, vehículos, etc . . . Así se podría disponer de información adicional de la persona a monitorizar, como por ejemplo la forma del cuerpo o los colores de la ropa que lleva, con la que solucionar así el problema de la perdida de la referencia incluso cuando el rostro está de espaldas a la cámara.

Otra solución podría implementar *GOTURN* (*Generic Object Tracking Using Regression Networks*), una solución que emplea *Redes Neuronales Recurrentes* (RNN) y permite realizar el seguimiento de objetos genéricos a partir de unas coordenadas iniciales de referencia.

# Apéndice A

## Códigos empleados

---

### Código A.1 Montar entorno de Google Colab.

```
# Trabajo Final de Grado de Alejandro Bautista Gómez

"""
En esta celda utilizamos los comandos para montar nuestro entorno de
trabajo en
Google Drive, así como instalar los siguientes módulos
#
# face_recognition: la red neuronal con la que detectamos y realizamos
operaciones con caras
#
"""

from google.colab import drive
drive.mount('/content/drive')
%cd drive/'My Drive'/Programa
!pip install -q face_recognition

"""
En esta celda importaremos algunos de los módulos que se emplearán en
este
trabajo

Nota: Timing es un wrapper, una función decoradora que nos permitirá
      calcular
los tiempos de ejecución de los distintos procesos
"""

import numpy as np
import sys
import face_recognition #Librería de reconocimiento facial
import os #Librería para interactuar con el entorno
import cv2 #Librería de visión artificial
from sklearn.neural_network import MLPClassifier #Clasificador MLP
```

```

from sklearn.neighbors import KNeighborsClassifier #Clasificador KNN
from sklearn.ensemble import RandomForestClassifier #Clasificador RF
from sklearn.svm import SVC #Clasificador SVM
from sklearn.externals import joblib #Para guardar/cargar clases en
    archivos
import pandas as pd

from functools import wraps
from time import time, sleep
from typing import List
import dlib

def timing(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        start = time()
        result = f(*args, **kwargs)
        end = time()
        print('Function: {}. Elapsed time: {}'.format(f, end-start))
        return result

    return wrapper

face_recognition.face_encodings = timing(face_recognition.face_encodings
    )
face_recognition.face_locations = timing(face_recognition.face_locations
    )

dlib.DLIB_USE_CUDA #Devuelve True en caso de que se estén utilizando GPU

```

---

### Código A.2 Carga de ruta y datos de entrenamiento..

---

"""

En esta celda se definen:

load\_dir: función que carga la ruta del directorio donde se almacenan los datos de entrenamiento

load\_data: función que carga los datos de entrenamiento

get\_data: función que llama a load\_data. Si no existen ficheros .pkl ejecuta load\_data, si existen, introducir True o False para indicar si se desea entrenar de nuevo la red

La ruta será /My Drive/programa/Personas

En ella estarán los directorios con nombre e imágenes de la persona que se quiera identificar en formato .jpg o .png.

Se recomienda un mínimo de [30] imágenes de la persona que se quiera identificar para entrenar al clasificador.

"""

```
@timing
def load_dir(directorio, encodings_final, index, tags):
    path= './Personas/' + directorio +'/'
    #Lista con los nombres de las imágenes de la carpeta de la persona
    file_list = os.listdir(path)
    for imagen in file_list:
        #Forma la ruta de la imagen
        complete_path= path + imagen
        #Carga la imagen y extrae su codificación (encodings)
        imagen = face_recognition.load_image_file(complete_path)
        encodings = face_recognition.face_encodings(imagen)
        #Si se ha obtenido, se añade a la lista final de encodings y de
        #etiquetas
        if encodings:
            encodings_final.append(encodings[0])
            tags.append(index)

def load_data():
    #Por defecto, se tomará la carpeta Source
    dirs=os.listdir(path='./Personas')

    #Inicializando parámetros
    encodings_final=[]
    tags=[]
    known_names=[]
    n_samples = 0

    print("Analizando directorios:")
    for (directorio,index) in zip(dirs, range(len(dirs))):
        #Muestra el directorio analizado
        load_dir(directorio, encodings_final, index, tags)
        print(str(index+1) + ' ' + directorio + ' - ' + str(len(tags)) -
              n_samples)
        #Suponiendo que hay al menos un rostro correcto en cada
        #directorío
        known_names.append(directorio)
        n_samples = len(tags)

    return encodings_final, known_names, tags

def get_data(select):
    if ((not os.path.exists('Model/Names.pkl')) or (not os.path.exists('
        Model/Encodings_final.pkl')) or (not os.path.exists('Model/Tags.
        pkl')) ):
```

```

encodings_final, known_names, tags = load_data()
joblib.dump(known_names, './Model/Names.pkl')
joblib.dump(encodings_final, './Model/Encodings_final.pkl')
joblib.dump(tags, './Model/Tags.pkl')
if not tags:
    raise "No images found! Need some images to train a classifier"
else:
    if (select == True):
        encodings_final, known_names, tags = load_data()
        joblib.dump(known_names, './Model/Names.pkl')
        joblib.dump(encodings_final, './Model/Encodings_final.pkl')
        joblib.dump(tags, './Model/Tags.pkl')
    else:
        known_names = joblib.load('Model/Names.pkl')
        encodings_final = joblib.load('Model/Encodings_final.pkl')
        tags = joblib.load('Model/Tags.pkl')

"""
Extrae caracteristicas en caso de no existir los ficheros: Names.pkl,
Encodings_final.pkl o Tags.pkl
True: sobreescribe los archivos actuales y carga los ficheros.pkl en los
objetos
False: carga los ficheros.pkl en los objetos
"""
[known_names, encodings_final, tags] = get_data(False)

```

---

### Código A.3 Entrenamiento de clasificadores.

---

```

"""
Función de entrenamiento de modelo Multi Layer Perceptron
"""

@timing
def train_mlp(encodings_final, tags):
    #Si se han obtenido algunas etiquetas (y por tanto encodings)
    #Crea y entrena el clasificador
    clf = MLPClassifier(max_iter=1000, activation= 'tanh',
                         hidden_layer_sizes = (200,100,50))
    model = clf.fit(encodings_final, tags)
    #Guarda el modelo de MLP entrenado y la lista de nombres
    joblib.dump(model, './Model/MLPClassifier.pkl')
    print("Clasificador MLP creado.")

"""
Función de entrenamiento de modelo con K-Nearest Neighbours
"""

@timing
def train_knn(encodings_final, tags):
    #Si se han obtenido algunas etiquetas (y por tanto encodings)
    #Crea y entrena el clasificador

```

```

clf = KNeighborsClassifier(n_neighbors=13, weights='uniform')
model = clf.fit(encodings_final, tags)
#Guarda el modelo de KNN entrenado y la lista de nombres
joblib.dump(model, './Model/KNNClassifier.pkl')
print("Clasificador KNN creado.")

"""
Función de entrenamiento
"""

@timing
def train_rf(encodings_final, tags):
    #Si se han obtenido algunas etiquetas (y por tanto encodings)
    #Crea y entrena el clasificador
    clf = RandomForestClassifier(n_estimators=25)
    model = clf.fit(encodings_final, tags)
    #Guarda el modelo de RF entrenado y la lista de nombres
    joblib.dump(model, './Model/RFClassifier.pkl')
    print("Clasificador RF creado.")

"""
Función de entrenamiento SVM
"""

@timing
def train_svm(encodings_final, tags):
    #Si se han obtenido algunas etiquetas (y por tanto encodings)
    #Crea y entrena el clasificador
    clf = SVC(probability=True, kernel='rbf')
    model = clf.fit(encodings_final, tags)
    #Guarda el modelo de SVM entrenado y la lista de nombres
    joblib.dump(model, './Model/SVMClassifier.pkl')
    print("Clasificador SVM creado.")

#encodings_final, known_names, tags = load_data()

#if (encodings_final == 'None'):
#  encodings_final, known_names, tags = load_data()

train_mlp(encodings_final, tags)
train_knn(encodings_final, tags)
train_rf(encodings_final, tags)
train_svm(encodings_final, tags)

```

---

**Código A.4** Pipeline de clasificación.

```

from google.colab.patches import cv2_imshow
class SkleanClassifier():

    def __init__(self, model_path, names_path):
        self.model = joblib.load(model_path)

```

```

    self.names = joblib.load(names_path)

@timing
def __call__(self, face_encodings, conf_thr=0.8):
    if len(face_encodings) == 0:
        return []

    preds = self.model.predict_proba(face_encodings) # get the
    predictions

    predictions = []
    for pred in preds: # map predictions to names and confidence
        conf = max(pred)
        if conf < conf_thr:
            name = "Unknown"
        else:
            tag = np.argmax(pred)
            name = self.names[tag]
        predictions.append((name, conf))

    return predictions


class Face():
    def __init__(self, face_location, face_encoding, name, conf,
                 frame_idx=None):
        self.frame_idx = frame_idx
        self.location = face_location
        self.y1, self.x2, self.y2, self.x1 = self.location
        self.encoding = face_encoding
        self.name = name
        self.conf = conf
        self.label = name + f" {self.conf:.2f}"
        self.color = (255, 0, 0) # blue by default

    def draw_on_img(self, img):
        c1, c2 = (self.x1, self.y1), (self.x2, self.y2)
        cv2.rectangle(img, c1, c2, (255, 0, 0), 2)
        tf = 1 # font thickness
        tl = 0.5
        t_size = cv2.getTextSize(self.label, 0, fontScale=tl, thickness=
                               tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, self.color, -1) # filled
        cv2.putText(
            img,
            self.label,
            (c1[0], c1[1] - 2),
            0,
            tl,

```

```

        [225, 255, 255],
        thickness=tf,
        lineType=cv2.LINE_AA,
    )

def todict(self):
    return {
        "x1": self.x1,
        "y1": self.y1,
        "x2": self.x2,
        "y2": self.y2,
        "name": self.name,
        "conf": self.conf,
        "frame_idx": self.frame_idx,
    }

class ProcessingPipeline():

    def __init__(self, classifier):
        self.classifier = classifier

    @timing
    def __call__(self, img, frame_idx=None) -> List[Face]:
        rgb_frame = img[:, :, ::-1] # BGR to RGB
        face_locations = face_recognition.face_locations(rgb_frame,
            model="cnn")
        face_encodings = face_recognition.face_encodings(rgb_frame,
            face_locations)
        predictions = self.classifier(face_encodings)

        faces = []
        for fl, fe, (name, conf) in zip(face_locations, face_encodings,
            predictions):
            faces.append(Face(fl, fe, name, conf, frame_idx))
        return faces

    names_path = './Model/Names.pkl'
    # Test MLP
    model_path = './Model/MLPClassifier.pkl'
    mlp_classifier = SkleanClassifier(model_path, names_path)

    # Test KNN
    model_path = './Model/KNNClassifier.pkl'
    knn_classifier = SkleanClassifier(model_path, names_path)

    # Test RF
    model_path = './Model/RFClassifier.pkl'
    rf_classifier = SkleanClassifier(model_path, names_path)

```

```

# Test SVM
model_path = './Model/SVMClassifier.pkl'
svm_classifier = SkleanClassifier(model_path, names_path)

""" IMPORTANTE:
    En la siguiente linea hay que introducir uno de los 3 clasificadores
    entrenados para ProccesingPipeline:
    mlp_classifier: clasificador Multi Layer Perceptron
    knn_classifier: clasificador K-Nearest Neighbours
    rf_classifier : clasificador Random Forest
    svm_classifier : clasificador State Vector Machine
"""
processing_pipe = ProcessingPipeline(svm_classifier)

```

---

#### Código A.5 Experimento 1.

---

```

"""
Banco de pruebas 1: Validación de clasificador. El objetivo de este banco
de pruebas es el de validar a los modelos de clasificación
entrenados. Para ello,
utilizamos dichos modelos comparados con vectores aleatorios de 128
dimensiones. La forma de comparar qué modelo es más eficiente será
comparando el número de falsos positivos detectados por
el sistema en forma de "personas identificadas" y el valor medio de
confianza otorgado por este
"""

@timing
def false_positives_on_classifier(predictions):
    false_positives=0
    avg_conf = 0
    for i in predictions:
        if (i[1] >=0.8):
            false_positives = false_positives + 1
    avg_conf = avg_conf + i[1]
    avg_conf = avg_conf/len(predictions)
    print("Número de predicciones erróneas",false_positives,"\\n"+
          "Confianza media", avg_conf)

random_vectors = np.random.rand(1000, 128)
print('Clasificador MLP')
false_positives_on_classifier(mlp_classifier(random_vectors))
print('Clasificador KNN')
false_positives_on_classifier(knn_classifier(random_vectors))
print('Clasificador RF')
false_positives_on_classifier(rf_classifier(random_vectors))
print('Clasificador SVM')
false_positives_on_classifier(svm_classifier(random_vectors))

```

---

---

**Código A.6** Experimento 2.

```

"""
Banco de pruebas 2: Validación del clasificador. El objetivo de este banco
de pruebas es validar los modelos de clasificación
entrenados. Para ello, utilizamos los modelos ya entrenados
con imágenes de personas alojadas en el directorio raíz y comprobamos si
dichas personas son clasificadas de manera correcta.
"""

input_img = 'Imagenes/1.jpg'
img = cv2.imread(input_img) #Esta linea indica el nombre del archivo y
                           el formato
faces = processing_pipe(img)
for face in faces:
    print(face.todict())
    crop_img = img[face.y1:face.y2, face.x1:face.x2]
    cv2_imshow(crop_img)
    face.draw_on_img(img)
cv2_imshow(img)

```

---

**Código A.7** Clases para procesado de video.

```

class VideoLoader():
    """
    This class controls how frames are taken from the camera
    """

    def __init__(self, video_path, downsampling_factor=1):
        self.downsampling_factor = downsampling_factor
        self.source = video_path
        self.cap = cv2.VideoCapture(self.source)
        self.shape = self.cap.read()[1].shape
        self.cap = cv2.VideoCapture(self.source)
        self.total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
        self.fps = self.cap.get(cv2.CAP_PROP_FPS)
        self.latency = 1.0 / self.fps
        self.taken_fps = self.fps / self.downsampling_factor
        self.frame_counter = 0

    def read(self):
        """
        Read one Frame from video. Return None if none left.
        """
        while self.frame_counter % self.downsampling_factor: # Throw away
            _, img = self.cap.read()
            self.frame_counter += 1
        # read frame from video
        _, img = self.cap.read()

```

```

# condition to finish iterating
if img is None:
    print("[VideoLoader]: Reached end of video")
    return None

return img

def release(self):
    self.cap.release()

def __iter__(self):
    return self

def __next__(self):
    frame = self.read()
    if frame is None:
        raise StopIteration()
    else:
        return frame


class VideoWriter():
    """
    This class, active in debug mode, offers an simple interface to
    writting
    visualizations over input frames.
    """

    def __init__(self, save_path, shape, fps):
        self.fourcc = cv2.VideoWriter_fourcc(*"mp4v")
        self.fps = fps
        self.shape = shape

        self.writer = cv2.VideoWriter(
            save_path,
            self.fourcc,
            self.fps,
            self.shape[:2][::-1],
        )

    def release(self):
        self.writer.release()

    def write(self, img):
        self.writer.write(img)

```

---

**Código A.8** Clases procesado de seguimiento.

---

```

from itertools import cycle

class IDGenerator():
    def __init__(self, max_id=10e9):
        max_id = int(max_id)
        self.iterator = cycle(range(max_id))

    def __next__(self):
        return next(self.iterator)

color_dict = {
    "purple": (255, 0, 255),
    "blue": (255, 0, 0),
    # "yellow": (0, 255, 255),
    "red": (0, 0, 255),
    "green": (0, 255, 0), # reserved for the detector
    "skyblue": (235, 206, 135),
    "navyblue": (128, 0, 0),
    "azure": (255, 255, 240),
    "slate": (255, 0, 127),
    "choco": (30, 105, 210),
    "olive": (112, 255, 202),
    "orange": (0, 140, 255),
    "orchid": (255, 102, 224),
}
colors = list(color_dict.values())

class Object():
    """
    Tracked Object.
    """

    def __init__(self, face, given_id, fps):
        self.id = given_id
        self.fps = int(fps)
        self.n_observations = 1
        self.unseen_ticks = 0
        self.unseen_secs = 0
        self.location = face.location
        self.encoding = face.encoding
        self.y1, self.x2, self.y2, self.x1 = self.location
        self.color = colors[given_id % len(color_dict)]
        self.name = face.name
        self.conf = face.conf
        self.lock_name = False
        if (self.name != 'Unknown'):
            self.label = self.name
            self.lock_name = True

```

```

        else:
            self.label = self.get_label()

def unseen(self):
    self.unseen_ticks += 1
    self.unseen_secs = self.unseen_ticks / self.fps

def update(self, face):
    self.unseen_ticks = 0
    self.n_observations += 1
    self.location = face.location
    # Optionally you could do spatial smoothing
    # self.location = np.mean(np.array([self.location, face.location
    # ]), axis=0).astype(np.int32).tolist()
    # Averaging the encodings with past ones makes it closer to the
    # centroid of the class
    self.encoding = np.mean(np.array([self.encoding, face.encoding]),
                           axis=0).tolist()
    self.y1, self.x2, self.y2, self.x1 = self.location
    self.conf = face.conf
    if (self.lock_name == False):
        self.name = face.name
        self.label = self.get_label()

        if (self.name != 'Unknown'):
            self.lock_name = True

    else:
        self.label = self.name

def get_label(self):
    return "ID: " + str(self.id)

def draw_on_img(self, img):
    c1, c2 = (self.x1, self.y1), (self.x2, self.y2)
    cv2.rectangle(img, c1, c2, self.color, 2)
    tf = 1 # font thickness
    tl = 0.5
    t_size = cv2.getTextSize(self.label, 0, fontScale=tl, thickness=
                           tf)[0]
    c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
    cv2.rectangle(img, c1, c2, self.color, -1) # filled
    cv2.putText(
        img,
        self.label,
        (c1[0], c1[1] - 2),
        0,

```

```

        tl,
        [225, 255, 255],
        thickness=tf,
        lineType=cv2.LINE_AA,
    )

class FaceTracker():
    def __init__(self, fps, max_unseen_secs=5, dist_thr=0.7):
        self.fps = fps
        self.max_unseen_secs = max_unseen_secs # Seconds until track is
            lost
        self.dist_thr = dist_thr

        self.id_generator = IDGenerator()
        self.active_objects = []

    def pair_faces_to_objects(self, faces: List[Face], objects: List[
        Object]):
        all_pairs = []
        for i, face in enumerate(faces):
            for ii, obj in enumerate(objects):
                # using only encoding info. TODO: add location to help
                    assignation
                dist = np.linalg.norm(face.encoding - obj.encoding) # see:
                    https://face-recognition.readthedocs.io/en/latest/
                        _modules/face_recognition/api.html#face_distance
                if dist < self.dist_thr: # We want pairs that are close
                    all_pairs.append([dist, i, ii])

        # Greedy assignation
        all_pairs = sorted(all_pairs) # sorts by distance
        f_idx_list, o_idx_list = [], []
        for pair in all_pairs:
            f_idx, o_idx = pair[1], pair[2]
            if (f_idx in f_idx_list) or (o_idx in o_idx_list):
                pass # if they are already assigned, pass
            else:
                f_idx_list.append(f_idx)
                o_idx_list.append(o_idx)

        return list(zip(f_idx_list, o_idx_list))

@timing
def update(self, faces: List[Face]):
    # Init in first iteration
    if not self.active_objects:
        self.active_objects = [Object(face, next(self.id_generator),
            self.fps) for face in faces]
    return

```

```

# 1. Pair new faces to objects
pairs = self.pair_faces_to_objects(faces, self.active_objects)
for (f_idx, o_idx) in pairs:
    face = faces[f_idx]
    obj = self.active_objects[o_idx]
    obj.update(face) # Propagate ID

# 2. Push unseen counter of not paired objects
assigned_obj_idxs = [pair[1] for pair in pairs]
unassigned_objs = [
    obj
    for i, obj in enumerate(self.active_objects)
    if i not in assigned_obj_idxs
]
[obj.unseen() for obj in unassigned_objs]

# 3. Delete objects if not seen for a while
marked_for_del = []
for i, obj in enumerate(self.active_objects):
    if obj.unseen_secs > self.max_unseen_secs:
        marked_for_del.append(i)

self.active_objects = [
    obj for i, obj in enumerate(self.active_objects) if i not in
    marked_for_del
]

# 4. Create new active_objects from unassigned detections
assigned_face_idxs = [pair[0] for pair in pairs]
unassigned_faces = [
    face
    for i, face in enumerate(faces)
    if i not in assigned_face_idxs
]
for face in unassigned_faces:
    self.active_objects.append(Object(face, next(self.
        id_generator), self.fps))

```

---

#### Código A.9 Experimento 3.

---

```

input_video = 'Videos/prueba_track.mp4'
save_path = 'Videos/prueba_track_output.mp4'

@timing
def process_video_tracking(input_video, save_path):
    video_loader = VideoLoader(input_video, downsampling_factor=1)
    video_writer = VideoWriter(save_path, video_loader.shape,
        video_loader.taken_fps)

```

```
face_tracker = FaceTracker(video_loader.taken_fps, max_unseen_secs  
=10, dist_thr=0.6)  
  
for img in video_loader:  
    faces = processing_pipe(img)  
    print(faces)  
    face_tracker.update(faces)  
    [obj.draw_on_img(img) for obj in face_tracker.active_objects if  
     obj.unseen_ticks == 0] # Draw debugging  
    video_writer.write(img)  
  
video_loader.release()  
video_writer.release()  
  
process_video_tracking(input_video, save_path)
```



# Índice de Figuras

---

2.1	Diagrama de un sistema de reconocimiento	4
2.2	Ejemplo de un escenario con aislamiento simple y un detector con sesgo racial	4
2.3	Haar-Cascade feautres aplicadas a una imagen	5
2.4	Representación gráfica de los pasos realizados para obtener una imagen HOG	6
2.5	Ejemplo de una imágén descrita a partir de sus HOG features	6
2.6	Plantilla de un rostro genérico HOG	6
2.7	Arquitectura de una red neuronal tipo ResNet	7
2.8	Operación de alineación y recorte de un rostro utilizando como referencia la linea de los ojos	8
2.9	Clasificación de Técnicas de Extracción de Características	8
2.10	Extracción de texturas mediante LBP	9
2.11	Ejemplo de reducción de un espacio de tres dimensiones a dos	9
2.12	Diagrama Random Forest	10
2.13	Para K=3, la muestra se clasifica como clase B, pero para K=5 como clase A	11
3.1	Diagrama de jerarquías del aprendizaje maquina	14
3.2	Arquitectura básica de una red neuronal	14
3.3	Arquitectura de una neurona	15
3.4	Soluciones a problemas de clasificación utilizando un perceptrón	15
3.5	Funciones de activación	16
3.6	Solución al problema de clasificación de la puerta lógica XOR	16
3.7	Diagrama de una red neuronal convolucional	17
3.8	Ejemplo de la operación de convolución de un kernel de tamaño 3x3	17
3.9	Ejemplo de una operación de Max-pooling con una matriz de tamaño 2x2	18
3.10	Ejemplo de operaciones de Data Augmentation para imagen de la BBDD	21
4.1	Representación de los tres tipos de errores que podemos hallar en en el sistema de seguimiento por detecciones [18]	24
4.2	Esquema de detección de un algoritmo de seguimiento mediante detecciones [18]	26
4.3	Aplicación gráfica de <i>Inserction-Over-Union</i>	26
4.4		27
5.1	Diagrama de una conexión de salto ( <i>skip</i> ) sobre una capa de una red tipo ResNet	31
5.2	Menú del archivo Principal.ipynb	32
5.3	Ventana de sincronización de directorios	33
5.4	Resultado en tiempo de ejecución de extracción de características de un individuo	33
5.5	Resultado en tiempo de ejecución de los modelos de clasificación	33
5.6	Identificación de tres rostros entrenados en una imagen aplicando un clasificador SVM	34

5.7	Ejemplo de un frame al que se le ha realizado video-tracking	34
6.1	Imágenes de la base de datos de entrenamiento	36
6.2	Imágenes de la base de datos de validación	39
6.3	Frames de resultado pertenecientes a la primera secuencia de vídeo	40
6.4	Frames de resultado pertenecientes a la segunda secuencia	40
6.5	Frames de resultado pertenecientes a la tercera secuencia	41

# Índice de Tablas

---

6.1	Resultados entrenamiento clasificador Multi Layer Perceptron	36
6.2	Resultados entrenamiento clasificador K-Nearest Neighbours con pesos uniformes	37
6.3	Resultados entrenamiento clasificador K-Nearest Neighbours con pesos según el inverso de la distancia	37
6.4	Resultados entrenamiento clasificador Random Forest	37
6.5	Resultados entrenamiento clasificador SVM	38
6.6	Comparación de resultados entre clasificadores óptimos entrenados	38
6.7	Comparación de resultados ante sensibilidad de umbral de confianza	39



# Índice de Códigos

---

A.1	Montar entorno de Google Colab	45
A.2	Carga de ruta y datos de entrenamiento.	46
A.3	Entrenamiento de clasificadores	48
A.4	Pipeline de clasificación	49
A.5	Experimento 1	52
A.6	Experimento 2	53
A.7	Clases para procesado de video	53
A.8	Clases procesado de seguimiento	54
A.9	Experimento 3	58



# Bibliografía

---

- [1] Google colaboratory, <https://colab.research.google.com/>.
- [2] Python, <https://www.python.org/>.
- [3] Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esen, Abdul A S. Awwal, and Vijayan K. Asari, *The history began from alexnet: A comprehensive survey on deep learning approaches*, arXiv preprint arXiv:1803.01164 (2018).
- [4] N. S. Altman, *An introduction to kernel and nearest-neighbor nonparametric regression*, American Statistician **46** (1992), no. 3, 175–185 (English (US)).
- [5] Muhammet Fatih Aslan, Akif Durdu, Kadir Sabancı, and Meryem Afife Mutluer, *Cnn and hog based comparison study for complete occlusion handling in human tracking*, Measurement **158** (2020), 107704.
- [6] W. Bledsoe, *The model method in facial recognition*, (1964).
- [7] G Bradski, OpenCV, <https://opencv.org/>.
- [8] David Copeornau, Scikit-learn, <https://scikit-learn.org/>.
- [9] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Mach. Learn. **20** (1995), no. 3, 273–297.
- [10] N. Dalal and B. Triggs, *Histograms of oriented gradients for human detection*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), vol. 1, 2005, pp. 886–893 vol. 1.
- [11] Erol Duymaz, Abdullah Ersan Oğuz, and Hakan Temeltaş, *Eş zamanlı konum belirleme ve haritalama probleminde yeni bir durum tahmin yöntemi olarak parçacık akış filtresi*, Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi **32** (2017), 1255 – 1270.
- [12] Kruszka et al, *22q11.2 deletion syndrome in diverse populations*, American Journal of Medical Genetics Part A **173** (2017), no. 4, 879–888.
- [13] Preeti Wadhwani & Saloni Gankar, *Facial recognition market size by component (software [2d facial recognition, 3d facial recognition, facial analytics], service), by application (criminal investigation, homeland security, id management, attendance tracking monitoring, intelligent signage, photo indexing & sorting, physical security), by end-use (aerospace & defense, automotive, bfsi, education, retail & e-commerce, healthcare), industry analysis report, regional outlook, growth potential, competitive market share & forecast, 2020 – 2026*, Jul 2020.

- [14] A Geitgey, *Face recognition*, <https://face-recognition.readthedocs.io/en/latest/>.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.
- [16] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München*, 1991.
- [17] Ildefonso Jiménez Silva, *Reconocimiento facial basado en redes neuronales convolucionales*, (2018), 77.
- [18] María José and Gómez Silva, *Appearance Similarity Learning for Multi-Person Tracking and Re-Identification*, (2019), no. November.
- [19] R. E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, Journal of Basic Engineering **82** (1960), no. 1, 35–45.
- [20] Davis E. King, *dlib*, <http://dlib.net/>.
- [21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *Imagenet classification with deep convolutional neural networks*, Communications of The ACM **60** (2017), no. 6, 84–90.
- [22] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton, *Deep learning*, **521** (2015), no. 7553, 436–444.
- [23] A. Mikołajczyk and M. Góchowski, *Data augmentation for improving deep learning in image classification problem*, (2018), 117–122.
- [24] Marvin Minsky and Seymour A. Papert, *Perceptrons: An introduction to computational geometry*, The MIT Press, 1969.
- [25] Marvin L. Minsky, Nathaniel Rochester, Claude E. Shannon, and John McCarthy, *A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955*, **27** (2006), no. 4, 12.
- [26] Proceedings Of, T H E Romanian, A Series, and Tudor Barbu, *Gabor filter-based face recognition technique*, **11** (2010), no. 3, 277–283.
- [27] Travis Oliphant, *Numpy*, <https://numpy.org/>.
- [28] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman, *Deep face recognition*, Proceedings of the British Machine Vision Conference (BMVC) (Mark W. Jones Xianghua Xie and Gary K. L. Tam, eds.), BMVA Press, September 2015, pp. 41.1–41.12.
- [29] Josh Patterson and Adam Gibson, *Deep learning: A practitioner’s approach*, 1st ed., O’Reilly Media, Inc., 2017.
- [30] Cahya Rahmad, Rosa Andrie, D Putra, I Dharma, H Darmono, and I Muhiqquin, *Comparison of viola-jones haar cascade classifier and histogram of oriented gradients (hog) for face detection*, IOP Conference Series: Materials Science and Engineering **732** (2020), 012038.
- [31] Rajat Raina, Anand Madhavan, and Andrew Y. Ng, *Large-scale deep unsupervised learning using graphics processors*, ICML ’09, Association for Computing Machinery, 2009, p. 873–880.

- [32] Matti Raitoharju, Robert Piché, and Henri Nurminen, *A systematic approach for kalman-type filtering with non-gaussian noises*, (2016).
- [33] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton : project para, cornell aeronautical laboratory report*, 1957.
- [34] D.E. Rumelhart, G.E. Hintont, and R.J. Williams, *Learning representations by back-propagating errors*, Nature **323** (1986), no. 6088, 533–536.
- [35] Arthur L. Samuel, *Some studies in machine learning using the game of checkers*, IBM JOURNAL OF RESEARCH AND DEVELOPMENT (1959), 71–105.
- [36] L. Sirovich and M. Kirby, *Low-dimensional procedure for the characterization of human faces*, Journal of the Optical Society of America A **4** (1987), no. 3, 519.
- [37] Zahraddeen Sufyanu, Fatma Mohamad, Abdulganiyu Yusuf, and Abdulbasit Nuhu, *Feature extraction methods for face recognition*, International journal of applied engineering research (IRAER) **5** (2016), 5658–5668.
- [38] M. A. Turk and A. P. Pentland, *Face recognition using eigenfaces*, Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1991.
- [39] P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features*, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, vol. 1, 2001, pp. I–I.
- [40] Mei Wang and Weihong Deng, *Deep Face Recognition : A Survey*, 1–31.
- [41] Qi Wei, Zhang Xiong, Chao Li, Yuanxin Ouyang, and Hao Sheng, *A robust approach for multiple vehicles tracking using layered particle filter*, AEU - International Journal of Electronics and Communications **65** (2011), no. 7, 609 – 618.
- [42] Hong wei Ng and Stefan Winkler, *A data-driven approach to cleaning large face datasets*.