
# Estimación de edad aparente utilizando información facial

**Güise Lorenzo Rodríguez Aguiar**

Grado en Ingeniería Informática
Trabajo de Fin de Grado
Junio de 2018
Las Palmas de Gran Canaria

**Tutores:**

José Javier Lorenzo Navarro
Pedro Antonio Marín Reyes

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

## SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE

D/Dª Güise Lorenzo Rodríguez Aguiar, autor del Trabajo de Fin de Título Estimación de edad aparente utilizando información facial, correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa/proyecto (indicar en su caso) _____

**S O L I C I T A**

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 1 de Junio de 2018.

Firmado por RODRIGUEZ AGUIAR GUISE LORENZO - 45350710T el día 01/06/2018 con un certificado emitido por AC FNMT

El estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[ X] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Firmado por LORENZO NAVARRO JOSE JAVIER - 42840825J el día 01/06/2018 con un

Firmado por MARIN REYES PEDRO ANTONIO -

Fdo.: _____

**DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA**

# Agradecimientos

*A mi familia, por estar siempre ahí y pretender que entienden mis problemas cuando no los entiendo ni yo.*

*A mis tutores, Jóse Javier Lorenzo Navarro y Pedro Antonio Marín Reyes, porque sin su ayuda este trabajo no hubiera sido posible.*

*Mención especial para Modesto Castrillón Santana, mi tutor no oficial favorito.*

*A mis amigos, especialmente a los que he conocido estos últimos años, porque sin ellos nada tendría sentido.*

*A todos los profesores que he conocido durante la carrera que se preocupaban por nuestro aprendizaje y futuro.*

# Resumen

Este Trabajo se plantea como un estudio de diferentes aproximaciones basadas en redes neuronales para obtener la edad estimada de una persona. Para ello se utiliza una base de datos con miles de imágenes de rostros de personas anotadas con sus edades respectivamente. Se utiliza una metodología basada en prototipos en la que se parte de diferentes modelos conocidos, utilizando además *transfer learning*.
Se especifica edad aparente debido a que diferentes personas con la misma edad pueden aparentar diferentes edades en función de sus rasgos faciales.
El resultado de este trabajo consiste en un sistema que clasifica en diferentes rangos de edad imágenes en la que se observe la cara de una persona, obteniendo una precisión en testeo del 73 %.

# Abstract

This Final Degree Project focuses on the study of different approaches to neural networks, with the goal of obtaining the estimated person age. For this purpose, a database containing thousands of images with human faces is used. Each face is labelled with its age. A methodology based on prototypes is used in this proyect, using different known models as starting point and also transfer learning.
The project title specifies apparent age due to the fact that different people with the same age can appear a different age because of their facial features. The result of this project consists of a system classifies images in which the human face can be seen across different age ranges with a 73 % testing accuracy.

# Índice de figuras

# Índice de tablas

# Índice general

# Capítulo 1

# Descripción del proyecto

## 1.1 Introducción

En este Trabajo de Fin de Grado se aplican los conocimientos aprendidos en la titulación de Ingeniería Informática, especialmente en las asignaturas de la mención en Computación. Con este bagaje, se pretende desarrollar en un sistema capaz de clasificar en diferentes rangos de edad a una persona utilizando una imagen en la que se observe su cara.

Dentro del campo del análisis facial, la estimación de edad ha sido históricamente uno de los problemas más desafiantes a los que la comunidad investigadora se ha enfrentado (Ramanathan et al. 2009, Fu et al. 2010, Han et al. 2013). Los primeros métodos computacionales para la estimación de edad fueron publicados hace más de 22 años y se basaban mayoritariamente en el cálculo de las proporciones entre diferentes medidas de las características faciales (Kwon et al. 1994) (Burt & Perrett 1995). Una vez se localizan los diferentes atributos faciales (nariz, barbilla, ojos, etc.), se calculan sus dimensiones y distancias entre ellos y se realiza el cómputo de las relaciones entre ellos para clasificar las caras en diferentes grupos de edad.

En el título del presente trabajo hablamos de edad aparente, que se refiere a la edad que los humanos infieren en función de la apariencia del individuo. La elección de este tipo de edad y no de la edad real se debe a los estudios que señalan que las personas tienen diferentes ritmos de envejecimiento (Guo, Fu, Dyer & Huang 2008, Guo, Fu, Huang & Dyer 2008). Este ritmo de envejecimiento no se determina únicamente por los genes de la persona, sino también por diversos factores como el estado de salud, el ambiente de trabajo, el estilo de vida y la sociabilidad (Berry et al. 1988). Otra causa im-

portante de signos avanzados de envejecimiento facial es la radiación de la luz ultravioleta. También el envejecimiento puede acelerarse mediante el consumo de tabaco, el estrés emocional, cambios drásticos de peso o la exposición a climas extremos.

Los factores comentados en el párrafo anterior añadirían una mayor dificultad a la estimación de la edad real por parte del sistema ya que, como se remarca en (Boissieux et al. 2000), el reto de modelar con precisión el envejecimiento de la piel es alto debido a los diferentes valores estéticos y puntos de atención que tienen las personas.

## 1.2 Motivación

Algunas de las razones por las que la estimación de la edad todavía se considera un problema desafiante son la fuerte especificidad de los rasgos personales de cada individuo, la alta varianza de las observaciones dentro del mismo rango de edad, la naturaleza incontrolable del proceso de envejecimiento y la dificultad para reunir los suficientes datos completos que permitan entrenar de forma precisa modelos (Escalera et al. 2015).

La edad juega un papel fundamental en las interacciones sociales junto al género de las personas. Por ejemplo, en muchos idiomas se utilizan diferentes reglas gramaticales, saludos y/o vocabulario dependiendo de estos dos atributos.

La estimación de edad es útil en aplicaciones donde no necesitamos conocer la identidad de una persona, lo que se busca es caracterizar débilmente a un individuo. Tal como podría ser en una aplicación que cuente cuántas personas con un determinado rango de edad entran en un centro comercial, con el objetivo de mejorar el modelo de negocio de este. En **biometría**, la estimación de la edad es considerada un tipo de biometría blanda (*soft biometrics*), ya que proporciona información auxiliar a la identidad de los usuarios (Jain et al. 2004). Se puede utilizar para complementar las características biométricas primarias (cara, iris, huella dactilar, etc.) o para mejorar el rendimiento de un sistema biométrico primario o duro (hard biometrics).

En **medicina**, la estimación de la edad se puede utilizar en sistemas médicos de monitorización. Aplicados a personas de avanzada edad en sus domicilios, contribuyendo a un diagnóstico médico al detectar un envejecimiento prematuro del paciente.

En **el campo de la seguridad**, un sistema de estimación de edad puede servir para evitar que menores de edad puedan acceder a locales donde se sirva alcohol, el uso de ciertas atracciones, el acceso a páginas de contenido adulto

o prohibirles la venta de tabaco en máquinas expendedoras sin necesitar la intervención humana o complementando esta.

En **el ámbito comercial**, el uso actual de las redes sociales y la disponibilidad de imágenes personales en estas ha conducido a la rápida integración del análisis facial por parte de las empresas. La estimación automática de la edad puede ayudar a realizar labores de segmentación de mercado de manera más eficiente, lo que permite detectar el perfil de los *target* (objetivo al que se dirige una acción) de la empresa y ayudar así a la toma de decisiones en campañas de publicidad o en otras estrategias de marketing. Por ejemplificar, los anuncios orientados a un determinado perfil comercial

Dentro del ámbito comercial también se encuentra la **gestión de relaciones electrónicas con clientes** o **ECRM** por sus siglas en inglés. ECRM es una estrategia de administración para el uso de las tecnologías de la información y las herramientas de interacción multimedia de forma que se gestionen de manera efectiva relaciones diferenciadas con todos los clientes y se realice la comunicación con ellos individualmente. Como estos grupos de clientes tienen diferentes hábitos de consumo, expectativas del cliente y preferencias, las empresas pueden aumentar sus beneficios sabiendo detectar estas diferencias y respondiendo únicamente a las necesidades específicas de los consumidores, proporcionando productos y servicios personalizados. Por ejemplo, una empresa de comida rápida podría conocer qué porcentaje de cada grupo de edad compra un tipo de hamburguesa o una empresa de publicidad podría conocer a qué grupo de edad le interesan más los anuncios que muestran en sus soportes de publicidad urbanos o modificar el anuncio exhibido para mostrar un producto o servicio con mayor probabilidad de consumo por parte del consumidor. Estos ejemplos de ECRM podrían llevarse a cabo utilizando un sistema de estimación de edad junto a una cámara que capture la cara del consumidor; etiquetando automáticamente a cada consumidor con su grupo de edad.

También en el campo del **entretenimiento** se puede hacer uso de un sistema de clasificación de edad. Por poner un ejemplo, se podría realizar una aplicación que gestione de forma automática las imágenes de un usuario utilizando la edad de las personas que aparecen en ellas (Das & Loui 2003).

## 1.3 Objetivos

El objetivo principal de este trabajo consiste en desarrollar un sistema capaz de clasificar en diferentes rangos de edad imágenes en las que se observe la cara de una persona.

Para cumplir con este propósito, en primer lugar se ha realizado el estudio de la información contenida en el conjunto de datos utilizado, después se han investigado los trabajos relacionados con el problema de la estimación de edad y las diferentes arquitecturas neuronales aplicadas. Posteriormente se ha realizado el estudio de las herramientas para el manejo de imágenes y se han implementado y analizado clasificadores basados tanto en la cara como en las imágenes completas (cabeza y hombros).

## 1.4 Planificación Temporal

Las 300 horas del Trabajo de Fin de Grado han sido distribuidas de la siguiente forma: sesenta horas para el estudio previo y el análisis, cien horas para el diseño, desarrollo e implementación, cien horas para la evaluación, validación y pruebas y 40 horas para las tareas de documentación y presentación. Las diferentes tareas están definidas en la Tabla 1.1.

La distribución temporal de las diferentes fases se ha mantenido sin cambios durante la realización del trabajo.

| **Fases** | **Duración Estimada (horas)** | **Tareas** |
|---|---|---|
| Estudio previo/Análisis | 60 | 1.1 Análisis y evaluación del conjunto de datos para la definición y balanceado de los distintos grupos de edad a considerar. |
| | | 1.2 Estudio de las librerías de visión por computador. |
| | | 1.3 Estudio de las redes neuronales convolucionales aplicadas a la estimación de edad. |
| | | 1.4 Familiarización con la librería Keras. |
| Diseño/ Desarrollo /Implementación | 100 | 2.1 Integración del detector de elementos faciales y obtención de las diferentes áreas de interés. |
| | | 2.2 Implementación de las diferentes redes neuronales. |
| | | 2.3 Implementación de un esquema de fusión basado en los estimadores de cada área de interés. |
| Evaluación/Validación/Prueba | 100 | 3.1 Evaluación de los resultados de la estimación en las diferentes áreas de interés consideradas. |
| | | 3.2 Evaluación de la fusión de los diferentes estimadores. |
| Documentación/Presentación | 40 | 4.1 Redacción de la memoria del TFG. |
| | | 4.2 Redacción de la presentación. |
| | | 4.3 Ensayos presentación. |

Tabla 1.1: Planificación prevista inicialmente

## 1.5 Justificación de la Competencia Específica Cubierta

En este trabajo ha sido cubierta la siguiente competencia específica de la mención de Computación del Grado de Ingeniería Informática:

***CP04:*** *Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.*

Esta competencia queda justificada a través de la realización de las tareas de investigación y estudio de las redes neuronales convolucionales (CNN) aplicadas a la estimación de edad. Una vez realizadas las anteriores tareas, se ha implementado un sistema capaz de clasificar en diferentes rangos de edad imágenes que contengan el rostro de una persona.

## 1.6 Legislación

En esta sección se analizan los aspectos legales aplicables al proyecto.

En el presente trabajo no se vulneran los derechos de imagen de las personas físicas que aparecen en las imágenes que se han utilizado.

El artículo 18, apartado 1, de la Constitución Española garantiza el derecho a la imagen, así como el derecho al honor y a la intimidad personal y familiar.

Según lo dispuesto por la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal, en el artículo 6, apartado 1: "El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la ley disponga otra cosa".

A estos efectos, la Ley Orgánica 1/1982, de 5 de mayo, de protección civil del derecho al honor, a la intimidad personal y familiar y a la propia imagen, en su artículo 7 expone aquellas circunstancias que se considerarán como intromisiones ilegítimas, estableciendo una excepción en el apartado 5, que nos remite al artículo 8, apartado 2.

Dicho precepto dispone que:

"En particular, el derecho a la propia imagen no impedirá:
a) Su captación, reproducción o publicación por cualquier medio cuando se trate de personas que ejerzan un cargo público o una profesión de notoriedad o proyección pública y la imagen se capte durante un acto público o en lugares abiertos al público.

b) La utilización de la caricatura de dichas personas, de acuerdo con el uso social.
c) La información gráfica sobre un suceso o acaecimiento público cuando la imagen de una persona determinada aparezca como meramente accesoria".

En este caso, y en base a los preceptos citados con anterioridad, al tratarse de personas físicas de notoriedad o proyección pública, no se estaría vulnerando el derecho a su imagen.

Por otro lado, la edad asociada a cada una de las imágenes ha sido recopilada utilizando imágenes públicas que incluyeran en su subtítulo dicha información, tal y como se describe en la sección 3.2.

## 1.7 Estructura del Documento

El objetivo de la presente memoria es documentar el proceso que se ha llevado a cabo en el Trabajo de Fin de Grado. Para ello se parte de las fases iniciales de búsqueda de información y análisis hasta los apartados de experimentación, análisis de los resultados y conclusiones. Con esta finalidad, la memoria se divide en el resumen, *abstract*, los índices de figuras y tablas, los diferentes seis capítulos, que serán descritos a continuación, y la bibliografía.

En el primer capítulo de la memoria se presenta una introducción del proyecto y del problema que se aborda. En el mismo se incluye la motivación del proyecto, los objetivos, la planificación temporal, la justificación de las competencias cubiertas y la estructura del documento.

En el segundo capítulo se detalla el marco teórico, fundamental para comprender los conceptos y técnicas que se utilizan en este proyecto; el estado del arte de la estimación de edad automatizada. Este capítulo incluye conceptos básicos sobre visión por computador, redes neuronales artificiales (ANN) y diferentes métodos de clasificación con el fin de comprender el estado del arte en el tópico a tratar.

El tercer capítulo se centra en el estudio previo y el análisis, incluyendo el planteamiento del problema, el análisis y la evaluación de conjuntos de datos que sirvan como entrada para el sistema clasificador, el estudio de las librerías de visión por computador y el estudio de las CNN que han sido utilizadas para abordar la estimación de edad de personas.

En el cuarto capítulo se detalla la metodología aplicada en el proyecto durante el proceso de desarrollo. También se incluye la descripción de las herramientas que han sido utilizadas. Además se describe cómo se ha realizado la implementación de los diferentes experimentos realizados.

Por otro lado, en el quinto capítulo se detallan los experimentos realizados

en las diferentes fases y sus resultados.

Por último, en el sexto capítulo se presentan las conclusiones extraídas durante la realización del proyecto; la valoración personal y posibles trabajos futuros.

# Capítulo 2

# Marco Teórico y Estado del Arte

## 2.1 Visión por Computador

La visión por computador es un campo interdisciplinar que combina biología e ingeniería y que se enfrenta al procesamiento e interpretación de imágenes y vídeos por parte de los ordenadores (Huang 1996). Desde el punto de vista biológico, la visión por computador tiene el propósito de obtener modelos computacionales del sistema de visión humana. Por otro lado, desde el punto de vista de la ingeniería, la visión por computador tiene como objetivo crear sistemas autónomos que puedan realizar algunas tareas del sistema de visión humana e incluso mejorar su rendimiento.

En 1963 el investigador del Instituto Tecnológico de Massachusetts Larry Roberts presentó su tesis doctoral. En la misma, estudiaba las posibilidades de extraer información geométrica tridimensional desde una perspectiva bidimensional de bloques, específicamente poliedros (Robers 1963). Después de su publicación, muchos investigadores del campo de la inteligencia artificial siguieron las líneas de investigación de su trabajo y comenzaron a estudiar la visión por computador desde un contexto de "mundo de los bloques" (*Blocks World*). Esta es la razón por la cual muchos consideran a Roberts como el padre de la visión por computador.

Posteriormente, los investigadores necesitaban un método que les permitiera poder tratar con imágenes reales para realizar tareas de bajo nivel como la detección de bordes. De esta forma se acogió el marco propuesto en (Marr

1982), que aplicaba a imágenes bidimensionales algoritmos de procesamiento de imagen de bajo nivel para obtener un "esquema principal" y acababa con la aplicación de técnicas de alto nivel para obtener un modelo tridimensional de los objetos en la escena.

Sin embargo, esta propuesta obligaba a realizar modelos 3D completos, algo innecesario en la mayoría de aplicaciones en las que se utilizaba la visión por computador. Por ejemplo, algunas aplicaciones requerían solo detectar si un objeto se estaba acercando o alejando al punto de observación y no conocer su movimiento exacto. Fue el propio Robers el que propuso un nuevo paradigma denominado por algunos "Visión deliberada" (*Purposive Vision*) que suplía esta desventaja del marco propuesto por Marr (Robers 1965).

La dificultad de la visión por computador se fundamenta principalmente en que es un problema inverso ya que se busca recuperar algunas incógnitas partiendo de información insuficiente para ofrecer una solución específica completa (Szeliski 2010).

Son múltiples las áreas de investigación de la visión por computador. Entre ellas se encuentran la clasificación, localización, detección de objetos y la segmentación (Khan et al. 2018). En este trabajo nos centramos en el uso en la visión por computador para detectar un tipo de biometría blanda, la edad de una persona.

En la literatura relativa a esta tarea, la detección de género junto a la estimación de la edad. Dentro de los métodos para el reconocimiento de género y la estimación de edad podemos dividir los métodos no convolucionales en tres tipos, siguiendo la categorización presentada en (Han et al. 2015):

La primera categoría recopila métodos basados en la forma (o en la antropometría). Se caracterizan por utilizar las relaciones de distancia entre marcas faciales para describir las diferencias topológicas entre caras de diferente edad. En la estimación de edad se suele considerar que las características antropométricas se deben al crecimiento craneofacial. Sin embargo, este factor solo suele ser generalmente útil cuando se intenta distinguir a los adultos de los niños, ya en la edad adulta la forma facial se vuelve bastante estable (Fu & Huang 2008). Además, en este enfoque se requiere una localización precisa de las marcas faciales, y en algunos casos también una anotación manual, lo cual limita su utilidad en los sistemas de estimación demográfica automatizados.

En segundo lugar, la categoría que agrupa métodos basados en la extración de características de las texturas a partir de imágenes. Estas texturas pueden ser, por ejemplo, arrugas o marcas faciales. Una forma de obtener la información de las textura de las imágenes suele ser utilizar directamente las intensidades de los píxeles. Características de textura como el *Local Gradient Gabor Pattern* (LGGP) o Gabor (Chen & Ross 2013), Patrones Binarios Lo-

cales (LBP) (Ojala et al. 2002) o las Características Biológicas Inspiradas (BIF) (Guo et al. 2009), entre otras, han sido utilizadas ampliamente para representar tanto las regiones frontales globales como locales. Este enfoque se considera efectivo tanto para la estimación de la edad como para el reconocimiento del género de las personas, además de la raza, pero es generalmente poco eficiente debido a la alta dimensionalidad de las características (Castrillón-Santana & Lorenzo-Navarro 2017, Castrillón-Santana et al. 2017).

Por último, la tercera categoría no convolucional se basa en la apariencia facial, es decir, utiliza tanto la información de las texturas como de las formas de la cara para diferenciar rostros entre diferentes grupos demográficos. El Modelo de Aspecto Activo (AMM) y sus variaciones son ampliamente utilizados para modelar la textura y la forma del rostro (Edwards et al. 1998) (Luu et al. 2011). Su desventaja, al igual que el enfoque basado en la antropometría, consiste en que requiere una localización precisa de las marcas faciales.

En la sección 2.4 de la presente memoria abordaremos los métodos convolucionales, donde además nos centraremos en el estado del arte para nuestra tarea.

## 2.2 Redes Neuronales Artificiales

### 2.2.1 Descripción del Modelo

Las Redes Neuronales Artificiales son sistemas de cómputo bioinspirados que constituyen uno de los paradigmas más importantes dentro de la inteligencia artificial actual.

Se inspiran en las neuronas de los mamíferos, cuyo diagrama esquematizado puede observarse en la Figura 2.1, aportando un sistema de cómputo altamente complejo, no lineal y paralelo que permite realizar múltiples operaciones de forma simultánea, en contraste con los sistemas de computación convencionales. Las redes neuronales se caracterizan por aprender a partir de ejemplos, generalizar a partir de las instancias de las que ha aprendido para reconocer nuevos datos de entrada y extraer de estos sus características principales.

Fue en 1943 cuando McCulloch y Pitts publicaron el primer trabajo de investigación en el que se explicaba en funcionamiento de estas neuronas, creando una red neuronal simple utilizando circuitos (McCulloch & Pitts 1943).

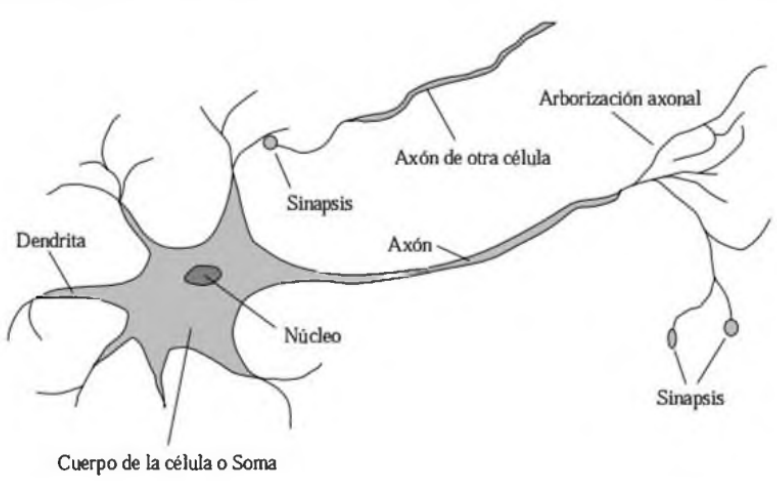



Figura 2.1: Partes de una neurona biológica. Sus elementos principales son el cuerpo de la neurona o soma, las entradas o dendritas y las salidas hacia otras neuronas denominado axón.

Fuente: Russell & Norvig (2004).

La arquitectura neuronal se puede resumir en los siguientes aspectos (López & Fernández 2008):

- Cada red está compuesta por un conjunto de unidades de procesamiento o neuronas.
- Cada una de las neuronas tiene un estado de activación, equivalente a la salida de la unidad.
- Existen conexiones entre las diferentes unidades de la red. Estas generalmente se encuentran definidas por el valor de un peso, que determina la influencia que tiene una neurona origen sobre la entrada de una neurona destino.
- Una regla de propagación o función de entrada que integra la información proveniente de las distintas neuronas artificiales y proporciona el valor del potencial postsináptico de la neuronales.
- Una función de activación o transferencia, que determina el estado de activación actual de la neurona.
- Una entrada externa denominada bias o umbral en cada unidad.
- Una función de salida.
- Una regla de aprendizaje.

Un diagrama simplificado de una neurona artificial se presenta en la Figura 2.2.

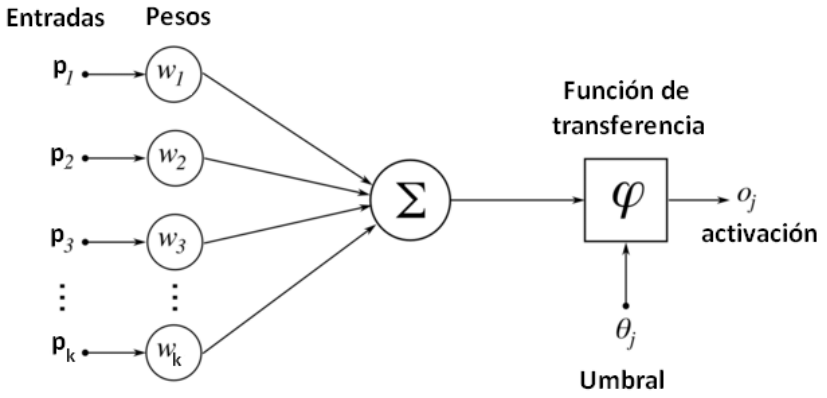



Figura 2.2: Diagrama de una neurona artificial simplificada.

Fuente: Ardila (2009).

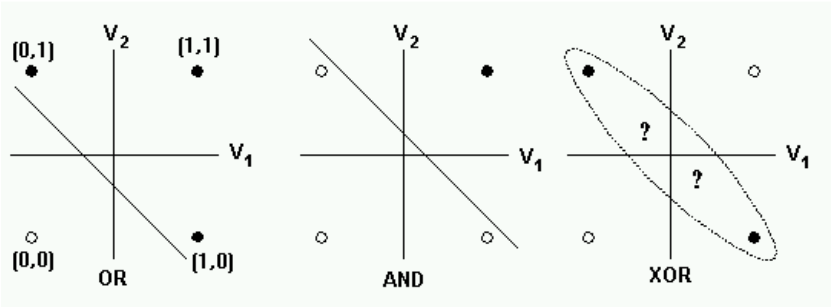



Figura 2.3: Funciones OR, AND y XOR separadas linealmente.

Fuente: (Beeman 2001).

Para lograr entender las ANN, es importante entender primero el funcionamiento de una de sus versiones más simplificadas, un perceptrón simple (Freund & Schapire 1999). Este modelo está compuesto por dos capas de neuronas, una de entrada y otra de salida.

Este arquitectura recibe una serie de entradas, que son ponderadas utilizando los pesos asociados a cada una de las conexiones. En caso de que la suma de los pesos (función de entrada) sea superior al valor del umbral o bias, la neurona se activa. La principal desventaja que presenta este modelo es que, generalmente, solo pueden representar funciones linealmente separables, como puede observarse en la Figura 2.3.

Para solventar el problema de los perceptrones simples surgen los perceptrones multicapas (MLP), que se diferencian con los anteriores en que al menos tienen una capa intermedia entre la capa de entrada y la capa de

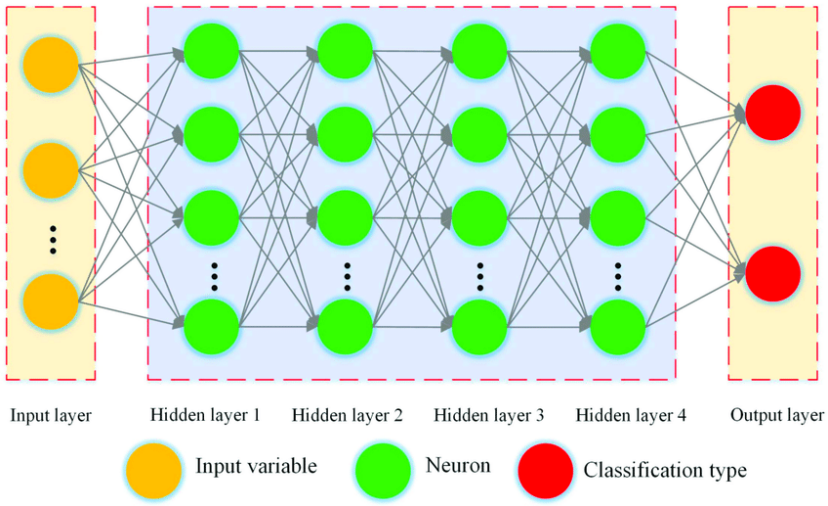



Figura 2.4: Perceptron Multicapa con 4 capas ocultas.

Fuente: (Jiang et al. 2018).

salida. Estas capas se se denominan "capas ocultas". La ventaja de añadir capas ocultas a nuestro modelo es que aumenta el espacio de hipótesis que puede representar la red (Russell & Norvig 2004), por lo tanto, puede tratar con problemas no lineales.

Los perceptrones multicapa, al igual que muchos otros modelos neuronales, utilizan una técnica de aprendizaje supervisado denominada *backpropagation* (Hecht-Nielsen 1992). Esta técnica nos permite modificar los pesos de todas las capas, no solo las de la capa de salida como ocurría en el perceptrón simple. Este algoritmo se utiliza junto a métodos de optimización, como puede ser el Descenso por Gradiente Estocástico (SGD). Una vez obtenida la salida de nuestra red, se compara el resultado real y el devuelto por la red con la función de coste, obteniendo el error de la red. Una vez obtenido este error, se propaga hacia atrás (backpropagation) hacia la capa de neuronas anterior, sirviendo para ajustar los pesos sinápticos de cada una de ellas, que generalmente se fijan aleatoriamente al empezar a entrenar. El error se sigue propagando hacia atrás hasta llegar a la capa de entrada. Este proceso se realiza con todas las instancias del conjunto de entrenamiento.

El aprendizaje de los pesos correspondientes en las capas ocultas permiten a las redes aprender nuevas características del conjunto de datos. Se denomina aprendizaje supervisado debido a que en el entrenamiento se utilizan las etiquetas, que determina a qué clase pertenecen las instancias, para poder calcular el error con el que se modifican los pesos para que la red aprenda.

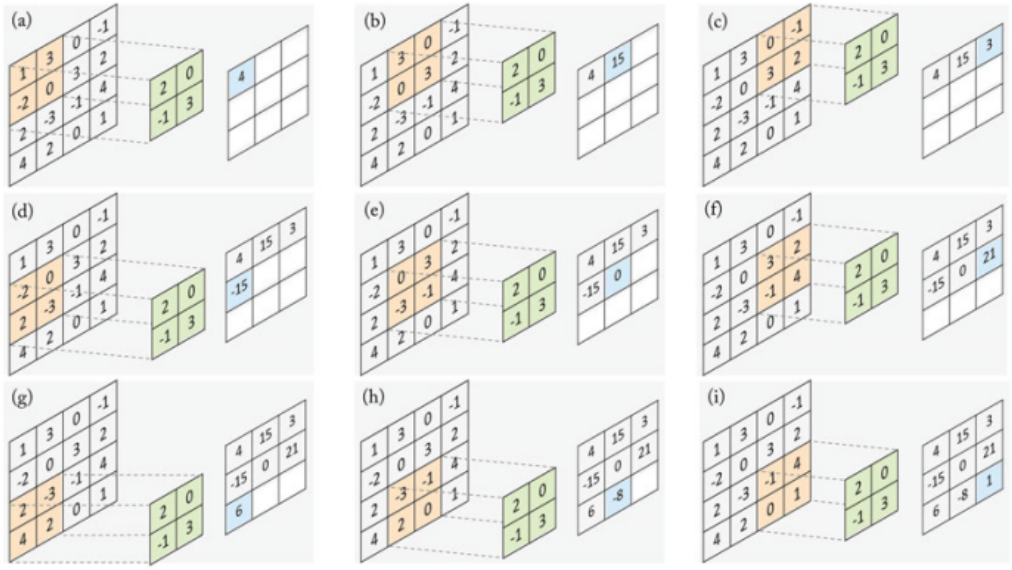



Figura 2.5: Operación de convolución ordenada paso a paso.

Fuente: (Khan et al. 2018).

### 2.2.2 Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales son un tipo de redes neuronales profundas que se utilizan especialmente cuando se tratan datos con una alta dimensionalidad, como ocurre en las imágenes o los vídeos. Es por ello por lo que se utilizan principalmente para clasificar imágenes, realizar tareas de clustering por similitud y reconocer objetos.

Los fundamentos de este tipo de redes se basan en un modelo denominado Neocognitron (Fukushima & Miyake 1982). Consistía en múltiples capas que aprendieron automáticamente una jerarquía de abstracciones de características para el reconocimiento de patrones. Este trabajo estaba motivado por la investigación realizada por (Hubel & Wiesel 1959) en la corteza primaria, que demostró que las neuronas del cerebro están organizadas en formas de capas. Estas capas aprenden a reconocer patrones visuales, extrayendo primero las características locales y, posteriormente, combinándolas para obtener representaciones a mayor nivel.

Las CNN están compuestas por una capa de entrada y una de salida junto a mútiples capas ocultas. Su diferencia principal con las redes neuronales artificiales convencionales se encuentra en las denominadas capas de convolución.

Una capa de convolución consta de un conjunto de filtros (también lla-

mados kernels convolucionales) que aplican una operación de convolución a una entrada determinada para generar un mapa de características como salida. Cada uno de los filtros es una matriz de números discretos. Cada uno de esos números constituyen los pesos del filtro, aprendidos durante la fase de entrenamiento de la CNN. Este proceso de aprendizaje se parte de una inicialización aleatoria de los pesos al principio del entrenamiento, usando por ejemplo la inicialización aleatoria gaussiana. Este tipo de inicialización, que también puede utilizarse para inicializar los pesos de los perceptrones multicapa o capas *fully connected*, obtiene estos valores de forma aleatoria utilizando una distribución de Gauss con media cero y desviación típica con un valor pequeño como 0.05. Posteriormente, utilizando los pares de entrada y salida, los pesos de los filtros se van ajustando durante el proceso de entrenamiento. De esta forma, se consigue que la capa de convolución sea capaz de detectar características abstractas y poco triviales de la imagen.

Anteriormente se comenta que estas capas efectuaban una operación de convolución, que se realiza entre la capa de entrada de la red y los filtros. Consideremos una convolución bidimensional como la de la Figura 2.5, que es el mismo tipo de convolución que se utilizará en los experimentos realizados en este trabajo, para entender el proceso que se lleva a cabo. Dado un mapa de características bidimensional de entrada y un filtro (cuadrado verde en la Figura 2.5) de $4 \times 4$ y $2 \times 2$ respectivamente, una capa de convolución multiplica el filtro del $2 \times 2$ con una porción de $2 \times 2$ resaltada (cuadrado naranja en la Figura 2.5) del mapa de características de entrada y suma todos los calores para generar un valor en el mapa de características de salida, resaltado en color azul en la Figura 2.5.

Como se puede observar en la Figura, el filtro se desliza de forma horizontal y vertical hasta que se ha abarcado todo el mapa de características de entrada. En este ejemplo, el fitro se mueve tanto en el eje horizontal como vertical de un valor a otro con un paso de diferencia. A este "paso" se le conoce como *stride* del filtro de convolución y puede tener un valor mayor a 1. En caso de que en el anterior ejemplo se hubiera fijado el stride a 2, se hubiera obtenido un mapa de características de salida menor al de la Figura 2.5. A esta reducción de la dimensión se la conoce como operación de submuestreo (*sub-sampling*). Cuanto mayor sea este valor menor tiempo y cómputo requerirá la capa de convolución. Sin embargo, si este valor es muy alto, supondrá una pérdida de información de la imagen.

Otra de las capas que se utilizan en las CNN son las denominadas *pooling layers*. Estas capas suelen utilizarse entre las capas de convolución y su función es reducir el tamaño espacial de la representación para reducir la cantidad de parámetros y el cálculo en la red (Karpathy 2015).

Las capas de pooling operan independientemente en cada nivel de profun-

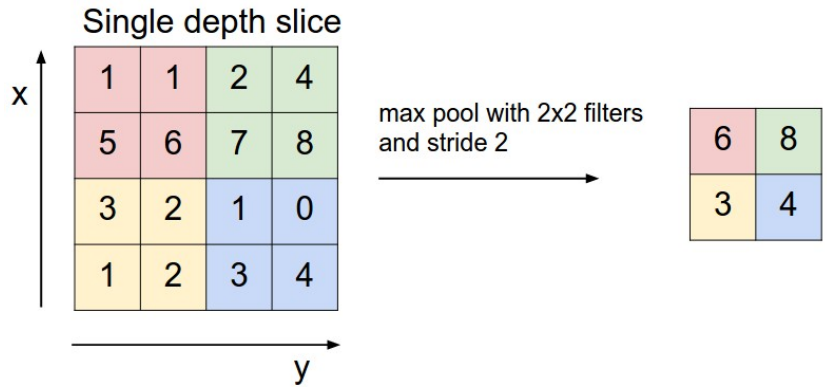



Figura 2.6: Ejemplo de capa de Maxpool con un filtro de $2 \times 2$ y un stride de tamaño 2.

Fuente: (Karpathy 2015).

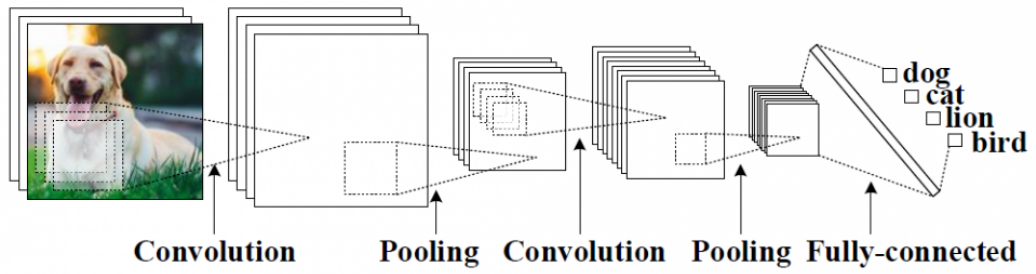



Figura 2.7: Estructura de una red neuronal convolutiva.

Fuente: (Li et al.).

didad de la entrada (que suele ser tridimensional debido al alto, ancho y numero de canales de la imagen) reduciendo su tamaño, utilizando comúnmente un filtro de máximo. La configuración más común de las capas de pooling utiliza filtros de tamaño $2 \times 2$ con un stride de 2, de forma que se descartan el 75 % de las activaciones, con el fin de conseguir una mejor regularización. En la Figura 2.6 se muestra un ejemplo de pooling.

En la Figura 2.7 se puede observar un ejemplo de CNN, donde la capa de entrada adquiere los datos de la imagen, en las capas ocultas se utilizan tanto capas convolucionales como Max-Poolings y al final del modelo se utiliza una capa fully-connected (un perceptrón multicapa como el de la Figura 2.4) que sirve como clasificador de las características encontradas por la CNN.

### 2.2.3 Algoritmos de Optimización

Los algoritmos de optimización son necesarios durante el entrenamiento de la red neuronal a minimizar o maximizar la función objetivo que se pretende aprender. La función objetivo es una función matemática dependiente de los parámetros de aprendizaje internos del modelo (entre los que se encuentran los pesos de la red o bias) que se utilizan para calcular los valores objetivos o salidas ($Y$) a partir del conjunto de predictores ($X$) utilizados en el modelo (Walia 2017).

## 2.3 Métodos de Clasificación

### Máquinas de Soporte Vectorial

Las Máquinas de Soporte vectorial (SVMs) son modelos de aprendizaje supervisado que tienen el objetivo de determinar la ubicación de la frontera de decisión que produce la separación óptimas de las clases (Vapnik 2013). En un problema de reconocimiento de patrones de dos clases donde las clases son separables linealmente, el SVM selecciona la frontera de decisión lineal que permite un mayor margen entre las dos clases. Este margen se calcula mediante la suma de las distancias del hiperplano desde los puntos más cercanos de las clases. Estos puntos de datos del hiperplano se denominan "vectores de soporte".

Si las dos clases no son separables linealmente, el SVM intenta encontrar un hiperplano que maximice de igual manera la distancia entre sus vectores de soporte y, al mismo tiempo, se intenta minimizar los posibles errores de clasificación. El equilibrio entre el margen y error de clasificación está controlado por un parámetro definido por el usuario denominado C (Cortes & Vapnik 1995). Las SVMs también pueden utilizarse en problemas con superficies de decisión no lineales. Existen métodos que proyectan los datos de entrada en un espacio de características con alta dimensionalidad a través de un mapeo no lineal y formular un problema de clasificación lineal en ese espacio de características (Boser et al. 1992). Esto ocurre en el ejemplo que se puede ver en la Figura 2.8. Para evitar el coste computacional de tratar con un espacio de características con alta dimensionalidad, se utilizan diferentes kernels como el kernel gaussiano radial basis function (RBF) o el kernel lineal que se utilizan en este trabajo.

Los SVMs fueron diseñados inicialmente para resolver problemas binarios. Cuando se tratan más clases, se utiliza un método apropiado multiclase.

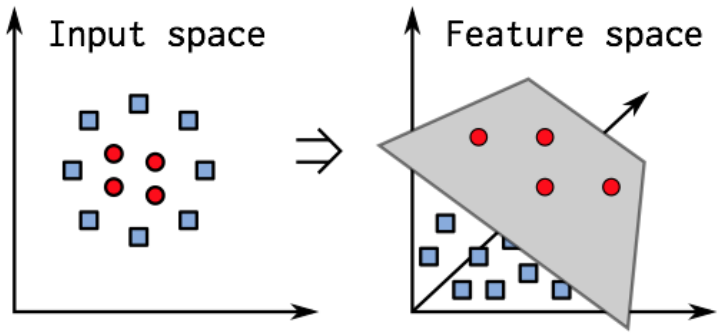



Figura 2.8: Separación lineal en el espacio de características con SVM.

Fuente: (Dima 2016).

Esquemas como *one against one* (uno contra uno), utilizado por ejemplo por el kernel RBF, o *one against the rest* (uno contra los demás), utilizado por el kernel lineal, son utilizados con frecuencia para resolver los problemas multiclase con SVM (Cristiani & Taylor 2000).

## Random Forest

Random forest es un algoritmo de aprendizaje supervisado que realiza las tareas de clasificación creando una combinación de árboles de decisión (Lior et al. 2014) donde cada clasificador se genera utilizando un vector aleatorio obtenido independientemente del vector de entrada. La clasificación se realiza asignando la clase que más elegida en los diferentes árboles de decisión ante un vector de entrada (Breiman 1999). Este clasificador se fundamenta en el uso de atributos al azar o una combinación de atributos en cada nodo para generar diferentes árboles de decisión. Los random forest también utilizan el método conocido como *Bagging*, que sirve para seleccionar aleatoriamente parte del conjunto de datos total para entrenar cada uno de los árboles de decisión que serán entrenados para cada atributo o combinación de atributos correspondiente (Breiman 1996).

El diseño de un árbol de decisión requiere la elección de un método para elegir los atributos y un método de poda. Hay muchos enfoques para realizar la elección de los atributos, caracterizándose la mayoría en asignar una medida de calidad a cada atributo. Las medidas de selección de atributos más frecuentes en la inducción de un árbol de decisión son el Ratio de Ganancia de Información (Information Gain Ratio) (Quinlan 2014) y el Coeficiente de Gini (Breiman et al. 1984). El random forest suele utilizar esta segunda me-

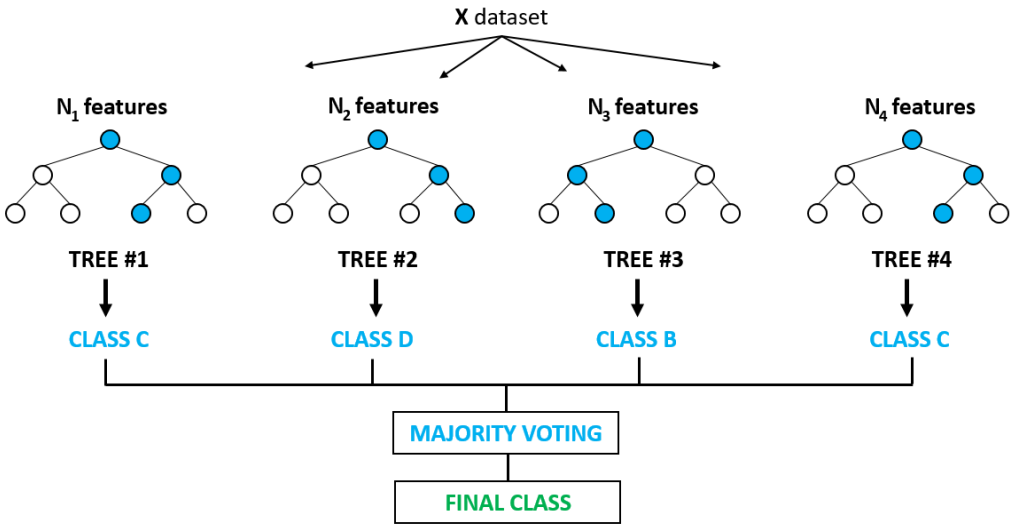



Figura 2.9: Ejemplo de Random Forest.

Fuente: (Holczer 2018).

dida para realizar la selección de atributos. Este criterio mide la impureza de un atributo con respecto a las clases. Una de las mayores ventajas de utilizar el random forest es que no es necesario podar los árboles de decisión cada vez que se supera la profundidad máxima. Diferentes estudios sugieren que los métodos de poda afectan al rendimiento de este tipo de clasificadores (Pal & Mather 2003) (Mingers 1989). También en (Breiman 1999) se sugiere que, a medida que aumenta el número de árboles, el error de generalización siempre converge, incluso sin podar el árbol. Además el sobreentrenamiento (ajuste del modelo a los datos del entrenamiento) no debería ocurrir debido a la Ley de los Grandes Números (Feller 1968).

## Análisis de Componentes Principales

El Análisis de Componentes Principales (PCA) es un procedimiento matemático multivariable que transforma un número de posibles variables correlacionadas en un grupo menor de variables no correlacionadas denominadas "componentes principales". La primera componente principal representa la mayor variabilidad posible en los datos mientras que el resto de las componentes principales siguientes representa la mayor variabilidad posible restante (Abdi & Williams 2010).

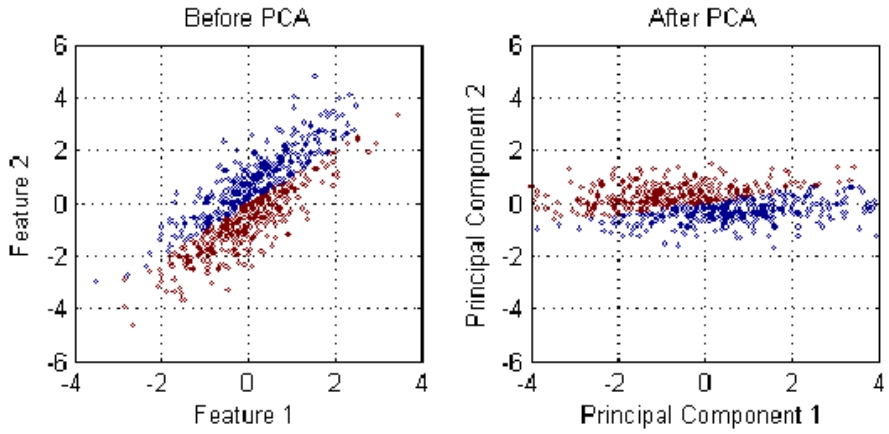



Figura 2.10: Ejemplo de Análisis de Componentes Principales.

Fuente: (Li et al. 2015).

Los objetivos del PCA son:

- Extraer la información más importante de los datos.
- Comprimir el tamaño del conjunto de datos manteniendo solo esta información importante.
- Simplificar la descripción del conjunto de datos.
- Analizar la estructura de las observaciones y la variable.

## 2.4 Estimación de Edad

Entre las diferentes fuentes de variación facial, las transformaciones no lineales son las más difíciles de modelar, ejemplo de ello son las expresiones faciales y la edad, entre otras. Algunos métodos no convolucionales, comentados en la sección 2.1, extraen características más simples en una jerarquía de diferentes niveles. Dado que las arquitecturas profundas como las CNN construyen no linealidades una encima de otra, son capaces de aprender transformaciones no lineales de forma más eficiente que los modelos simples y demuestran un mejor rendimiento en la mayoría de tareas, aunque son más vulnerables al sobreentrenamiento.

Las CNN han conseguido superar a los métodos clásicos en la mayoría de tareas, incluyendo la estimación de edad. El trabajo de (Yang et al. 2011) fue el primero en utilizar una CNN con cinco capas para estimar la edad. Este modelo cuenta con diferentes salidas para el género, raza y edad; aunque el rendimiento en la estimación de la edad era peor que los obtenidos por

métodos basados en las BIF. Para mejorar el rendimiento en la estimación de edad, (Yi et al. 2014) utiliza metodologías del análisis facial tradicional y utiliza un ensemble de 23 CNN. Cada una se aplica en diferentes partes faciales y el método consigue mejorar el rendimiento del estado del arte utilizando el conjunto de datos MORPH 2 (Ricanek & Tesafaye 2006).

En los últimos años, debido a la gran disponibilidad de datos y poder computacional, la popularidad de las arquitecturas más profundas y complejas han incrementado su popularidad entre los investigadores surgiendo modelos como la ResNET (He et al. 2016) o la DenseNet (Huang et al. 2017). Es el caso de (Xing et al. 2017), un modelo híbrido profundo que estima la edad, se consigue además reconocer géneros y razas entrenando cada estimador de edad diferente con cada una de las razas y géneros.

Otra arquitectura que hace uso de los datos débilmente etiquetados es la propuesta de (Hu et al. 2017). Este modelo realiza la estimación de edad con ayuda de información de diferencia de edad (AEAD). Esta información se obtiene utilizando imágenes de un mismo sujeto conociendo la diferencia de edad entre cada una de ellas.

Los métodos de clasificación profunda han conseguido un gran rendimiento en la estimación de edad. En (Niu et al. 2016) un regresor ordinal profundo se entrena utilizando el gran conjunto de datos AFAD. Este enfoque de clasificación utiliza múltiples salidas y supera a los algoritmos de clasificación clásicos al utilizar el conjunto de datos MORPH2.

De forma similar, el enfoque que utiliza múltiples clasificadores binarios profundos propuesto en (Chen et al. 2017) emplea un conjunto de CNN fusionadas con agregación. En ese mismo trabajo se prueba que la inconsistencia de las salidas binarias no afecta al rendimiento general ya que el error de la salida está limitado por el error máximo de los clasificadores binarios. Por lo tanto, siempre que se reduzca el error máximo, las etiquetas inconsistentes de los clasificadores no afectan al rendimiento general de la red.

En (Liu et al. 2017) se adopta un rango de edad diferente para aprender una serie de patrones de envejecimiento. La arquitectura Adeep CNN se emplea para minimizar la distancia de las caras dentro del mismo rango de edad mientras que se maximiza la distancia entre las caras de diferentes grupos. La predicción de edad se calcula posteriormente utilizando el algoritmo OHRank para cada grupo de edad.

Para maximizar de forma efectiva la distancia entre los diferentes grupos de edad, en (Liu et al. 2018) se introduce el *Label-Sensitive Deep Metric Learning* (LSDML). Este método optimiza el procedimiento de aprender conjuntamente una métrica discriminatoria y extraer pares opuestos. Con el fin de mitigar el efecto de los conjuntos de datos dispersos y desbalanceados, el método se extiende a LSDML de fuentes múltiples, que maximiza la correla-

ción de poblaciones cruzadas entre diferentes conjuntos de datos.

Una gran cantidad de métodos que utilizan modelos profundos se utilizan en la estimación de edad, siendo principalmente presentados en el desafío "Chalearn Looking at People" (LAP) (Escalera et al. 2015, 2016). En particular, en (Ranjan et al. 2015) se utiliza una red neuronal profunda consistente en 10 capas convolucionales, cinco capas de pooling y una capa fully-connected (Chen et al. 2016). Para realizar el cálculo de la edad, en primer lugar la imagen se clasifica en tres grupos de edad y, posteriormente, una ANN con tres capas estima la edad aparente. Cada red está entrenada utilizando conjunto de datos diferentes.

Los puestos más altos en el desafío LAP suelen ser trabajos basados en variaciones de arquitecturas profundas populares. Estas variaciones incluyen conjuntos de redes múltiples y de fusiones de diferentes modelos. Los más adoptados son el VGG-16 (Simonyan & Zisserman 2015) y el GoogLeNet (Szegedy et al. 2015) debido a su éxito en el desafío de reconocimiento de objetos en Imagenet (Russakovsky et al. 2015). Versiones modificadas del modelo VGG-16 se usan para realizar la estimación de la edad aparente en (Rothe et al. 2016, Antipov et al. 2016, Uricár et al. 2016, Kuang et al. 2015).

Los modelos comentados anteriormente son afinados y entrenados con diferentes conjuntos de datos para lograr una generalización mientras una variedad de técnicas de clasificación y regresión se emplean en las arquitecturas profundas. Particularmente, el clasificador basado en una SVM presentado en (Uricár et al. 2016) mejora los resultados del método de regresión utilizados en (Rothe et al. 2016). Por último, en (Liu et al. 2015) se utiliza una fusión de clasificador y regresor basado en GoogLeNet mientras que en (Ni et al. 2011) se emplea una agrupación de edad jerárquica con SVM seguida por una Regresión de Vectores Soporte (SVR) y random forest en las características profundas.

# Capítulo 3

# Estudio Previo y Análisis

## 3.1 Planteamiento del Problema

Para cumplir con los objetivos descritos en la sección 1.3, en primer lugar se han investigado diferentes conjuntos de datos y se han balanceado sus clases tanto por género como por grupo de edad. Posteriormente se han estudiado las librerías de visión por computador *Open Source Computer Vision Library* (OpenCV) (Bradski 2000) y Dlib (King 2009) para tratar las imágenes antes de su utilización en los modelos neuronales. Además del estudio de las CNN realizado en la sección 2.2, se han investigado modelos de este tipo que hayan sido aplicados a la estimación de edad.

## 3.2 Análisis y Evaluación del Conjunto de Datos

Para la realización de las diferentes pruebas se ha optado por el conjunto de datos o dataset denominado AgeDB (Moschoglou et al. 2017).

Esta decisión se justifica debido a que es el primer dataset conocido que ha recopilado de forma manual y no semiautomática 16488 imágenes *in-the-wild* (condiciones no controladas) de personas famosas, entre los que se encuentran escritores/as, científicos/as, políticos, actores/actrices, etc. Todas estas imágenes se encuentran etiquetadas con la edad del sujeto en ese año, género y su identidad. Estas características la hacen especialmente indicada

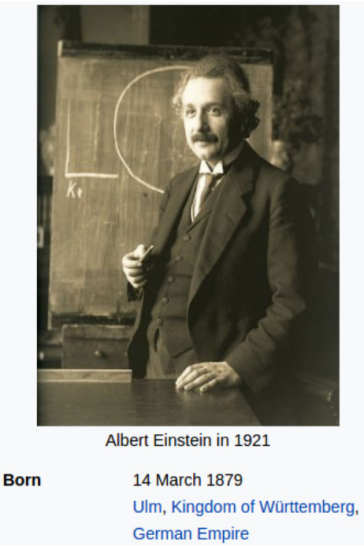



Figura 3.1: Ejemplo de imagen recopilada manualmente para AgeDB.

Fuente: (Moschoglou et al. 2017).

para realizar tareas de estimación de edad.

El proceso de recopilado de imágenes se realiza de forma manual y no utilizando sistemas semiautomáticos, como en otros conjuntos de datos, debido a que los resultados obtenidos por los rastreadores de dichos sistemas suelen contener etiquetas de edad "ruidosas", es decir, erróneas. Para conseguir etiquetas precisas, los autores de este dataset realizaron la búsqueda manual de las 16488 instancias en Google Imágenes, conservando únicamente aquellas en las que aparecía la edad exacta en el año de cada sujeto en el subtítulo que la acompaña. En la Figura 3.1 se puede observar una imagen del dataset, que contiene en el subtítulo el año en la que se tomó la imagen y la fecha de nacimiento del sujeto, por lo que la etiqueta de edad puede ser calculada.

Comos se ha comentado anteriormente, AgeDB es un dataset con imágenes *in-the-wild*. Eso significa que han sido tomadas bajo condiciones que no han sido controladas (diferentes poses, expresiones faciales, partes de la persona tapadas o recortadas, etc.).

La base de datos contiene datos de 568 sujetos distintos, siendo el número medio de imágenes por sujeto de 29. Por otro lado, la edad mínima y máxima de los sujetos es 1 a 101 años respectivamente, siendo la media de edad de $50,3$ años.

Existen otros conjuntos de datos públicos que han sido analizados pero que han sido descartados debido a que presentaban desventajas con respecto

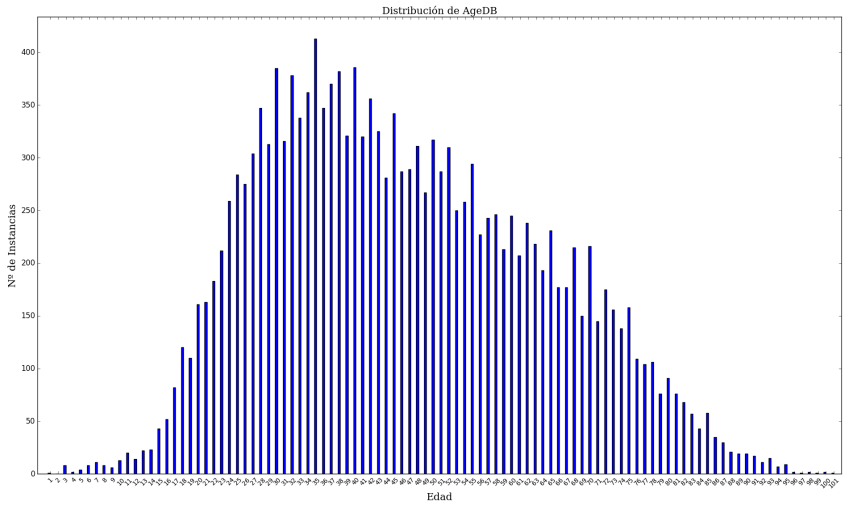



Figura 3.2: Histograma de edad en AgeDB.

a AgeDB. El primero de ellos es IMDB-WIKI (Rothe et al. 2018), que cuenta con 523051 imágenes que han sido obtenidas de Wikipedia e IMDB. De estas dos páginas también se obtuvieron los datos para sus etiquetas, incluyendo la edad. El inconveniente de este dataset es que ha sido recopilado utilizando un sistema semiautomático, por lo que sus etiquetas de edad pueden tener datos erróneos. También han sido analizados dos datasets creados pensando en la estimación de la edad aparente. Los dos son en concreto el (Escalera et al. 2015) y el APPA-REAL (Agustsson et al. 2017). El primero fue diseñado para la competición "Chalearn Looking at People" (LAP) y contiene 4691 imágenes mientras que el segundo contiene 7000. En ambos casos las edades aparentes se calcularon utilizando las estimaciones de los usuarios de la plataforma de voto online AgeGuess. APPA-REAL utilizó además trabajadores de Amazon Mechanical Turk (AMT) para que realizaran anotaciones sobre la edad que estimaban que tenían los sujetos. Estos dos conjuntos de datos los hemos descartado debido a las pocas instancias con las que cuentan cada uno en comparación a las 16488 imágenes de AgeDB. Como se ha podido observar en el histograma de la Figura 3.2, el número de imágenes disponibles en el dataset entre los menores de 21 son mucho menores que las del resto de edades. En concreto hay solo 708 imágenes. Así que teniendo en cuenta esta circunstancia y que, como se comentó en la sección 2.1, entre los niños la forma facial no es tan estable como entre los adultos debido al crecimiento craneofacial (Fu & Huang 2008), se decidió no tener en cuenta

| **Dataset** | **Año** | **Imágenes** | **Sujetos** | **Etiquetas de edad** | **Etiquetas sin ruido** | **Condiciones controladas** |
|---|---|---|---|---|---|---|
| AgeDB | 2017 | 16488 | 568 | Exactas en el año | Sí | No |
| IMDB-WIKI | 2016 | 523051 | 20284 | Exactas en el año | No | No |
| Escalera et al. | 2015 | 4691 | - | Exactas en el año y aparente | Sí | No |
| APPA-REAL | 2017 | +7000 | +7000 | Exactas en el año y aparente | Sí | No |

Tabla 3.1: Comparativa datasets con etiqueta de edad.

ese rango de edad en el trabajo. De esta forma se optó por formar seis grupos, como puede observarse en la Figura 3.3 dividido por género. El primer grupo está formado por las imágenes etiquetadas con una edad entre 21 y 30 años contando con 2725 instancias, el segundo está comprendido entre los 31 y los 40 años y cuenta con 3613 imágenes, el tercero abarca desde los 41 años hasta los 50 y cuenta con 3095 instancias, el cuarto comprende entre los 51 y los 60 años con 2573 imágenes, el quinto consta de 2022 imágenes que van desde los 61 hasta los 70 años y el último grupo comprende todos los valores mayores o iguales a 71, contando con 1752 instancias. En todos los casos los límites finitos incluidos dentro de cada grupo. Como puede observarse, se abarca un margen de diez años en cada grupo salvo en el último.

Se ha tenido en cuenta que una posible sobrerrepresentación de alguno de los dos géneros en los grupos puede afectar de forma negativa a las predicciones con el género minoritario. Es por ello por lo que se ha realizado un proceso de *undersampling* en cada grupo para igualar las instancias del género mayoritario a las del minoritario, quedando la distribución interna de los grupos como puede verse en la Figura 3.4. De esta forma, el número global de instancias pasa de 15780 a 12040.

Por último, una vez se han balanceado internamente los géneros en los seis grupos, se ha procedido ha realizar un proceso de *oversampling* para aumentar el número de instancias. Este proceso ha consistido en duplicar el número de imágenes creando para cada una su versión reflejada horizontalmente, lo que se conoce coloquialmente como "efecto espejo". Así se ha pasado de un dataset de 12040 instancias a 24080.

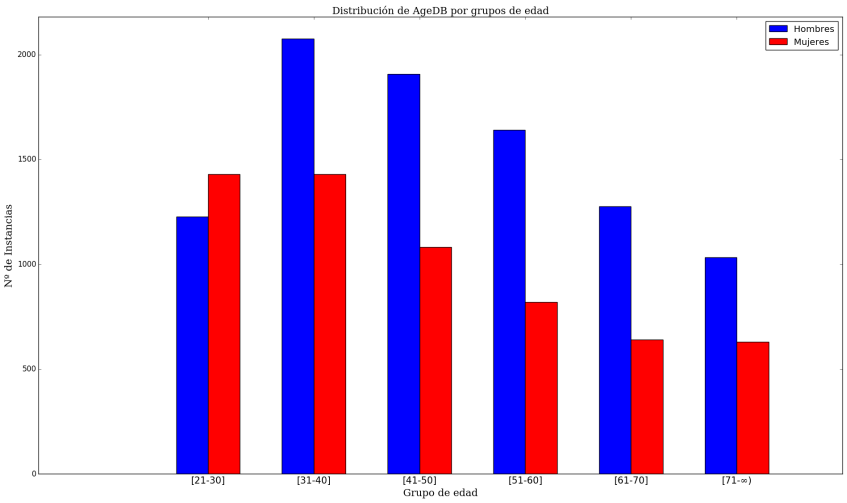



Figura 3.3: Histograma con los seis rangos de edad en AgeDB.

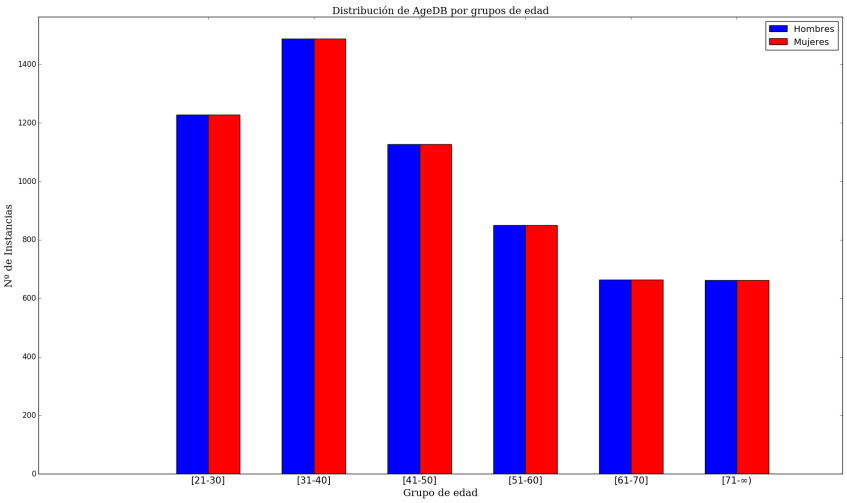



Figura 3.4: Histograma con los 6 rangos de edad balanceados en AgeDB.

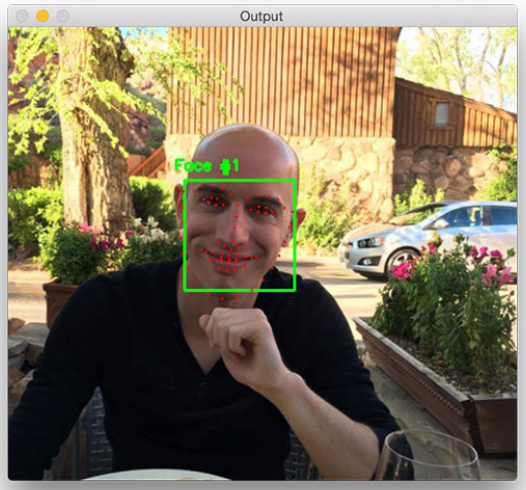



Figura 3.5: Ejemplo de detección de puntos faciales y de cara.

Fuente: (Rosebrock 2017).

## 3.3 Estudio de las Librerías de Visión por Computador.

Como se comentaba anteriormente, para tratar las imágenes antes de ser utilizadas en los modelos neuronales se han utilizado dos librerías especializadas.

La primera de ella se llama OpenCV y es una librería desarrollada originalmente por la empresa Intel en el año 2000. Está publicada bajo una licencia de BSD, por lo que puede ser utilizada libremente tanto en el mundo académico como comercial y cuenta con interfaces para C++, Java y Python. En este trabajo la hemos utilizado mediante su interfaz en Python para realizar la lectura de las imágenes del conjunto de datos AgeDB, redimensionar su tamaño para que sea compatible con el de las capas de entrada de las CNN utilizadas y normalizar sus valores para que se encontraran en un rango entre 0 y 1 en vez de entre 0 y 255. También se ha utilizado durante el proceso de oversampling, descrito en la sección anterior, para duplicar las instancias del conjunto de datos una vez ha sido balanceado. Para ello se ha utilizado una función que permite reflejar la imagen horizontal y/o verticalmente, eligiéndose únicamente la primera opción.

Por otra parte se ha utilizado Dlib, una librería de código abierto desarrollada desde 2002, siendo su principal autor Davis King. Está publicada bajo una licencia software de Boost, por lo que se puede utilizar libremente. Además cuenta con interfaces tanto para C++ como Python. En este trabajo la hemos utilizado para usar su detector de caras. Esto se debe a que en una de las fases de las pruebas sólo se emplea esta zona de las imágenes como entrada para las redes neuronales. Inicialmente se iba a realizar el recorte de la cara utilizando la posiciones de los ojos y calculando diferentes proporciones para conseguir su área pero posteriormente se decidió utilizar el detector facial (cuadrado verde en la Figura 3.5).

## 3.4 Estudio de las Redes Neuronales Convolucionales Aplicadas a la Estimación de Edad

Para la realización de este trabajo se utilizan dos arquitecturas de CNN diferentes. El primero de ellos se denomina VGG-Face (Parkhi et al. 2015) y consiste en una VGG-16 (Simonyan & Zisserman 2015) preentrenada para realizar una tarea de reconocimiento facial. Esta red obtiene un 97,2 % de precisión en el *benchmark LFW* (Huang et al. 2007).

La estructura de la VGG-Face utilizada es similar a la que se muestra en el diagrama de la Figura 3.6 pero el tamaño de salida de la capa de *maxpooling* en el quinto bloque es de $(7, 7, 512)$.

La capa de entrada de la imagen tiene un tamaño de $(224, 224, 3)$. Los dos primeros bloques de CNN cuentan con dos capas de convolución cada uno, teniendo las primeras 64 filtros y las segundas 128. Los tres siguientes bloques de CNN cuentan con tres capas de convolución cada uno, siendo 256 el número de filtros de las capas convolucionales del primero y 512 en los otros dos bloques. Todos los filtros de las capas convolucionales tienen un tamaño de $3 \times 3$ y un stride de 1. Además, estas capas de convolución utilizan una función de activación denominada *Rectified Linear Unit* (ReLU) (Nair & Hinton 2010).

Por otro lado, como se puede observar en la Figura 3.6, todos los bloques tienen en su final una capa de maxpooling, caracterizándose todas por un tamaño de pool de $2 \times 2$ y un stride de 2.

En el último bloque de la red se encuentran las capas que servirán para realizar la clasificación de las entradas de la red en las diferentes clases. Está

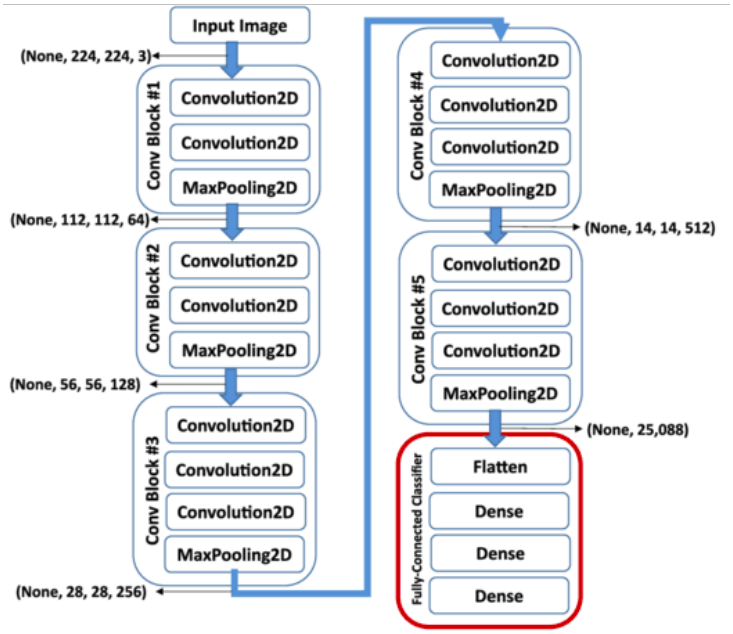



Figura 3.6: Estructura de la VGG-16.

Fuente: (Gopalakrishnan et al. 2017).

conformado, en primer lugar, por una capa de *Flatten*. Esta se caracteriza por transformar los datos de entrada a una única dimensión en su salida, de forma que puedan ser procesados por las capas fully connected. El bloque cuenta con tres capas de este tipo, contando las dos primeras con una capa de salida de 4096 neuronas, la función de activación $ReLU$ y sus respectivas capas de *dropout* (Srivastava et al. 2014) con una probabilidad del 50 %. Estas capas se utilizan para prevenir el sobreentrenamiento de las redes neuronales, desactivando las activaciones de un porcentaje de las neuronas de forma aleatoria durante cada actualización de los pesos en el entrenamiento.

La última capa de la red es una fully connected también con 2622 neuronas de salida, una por cada clase del sistema de clasificación. En esta capa se utiliza la función de activación de *softmax*, de forma que la salida de cada neurona es la probabilidad de que la entrada pertenezca a esa clase, donde la suma de todos los valores 1. (Nasrabadi 2007)

Se ha seleccionado este modelo debido a los buenos resultados obtenidos en la estimación de edad en (Antipov et al. 2016).

También ha sido utilizado en nuestras pruebas el modelo LeviHassner propuesto en (Levi & Hassner 2015). En este modelo, al contrario que en VGG-Face, no se utilizan los pesos precargados sino que se inicializan todas

las capas con valores aleatorios extraídos de una distribución normal con media cero y desviación típica de $0,01$.

La estructura de la LeviHassner cuenta con 15 capas, siendo significativamente menor al modelo VGG-Face, ya que cuenta únicamente con tres capas convolucionales frente a las trece del modelo anterior.

La capa de entrada del modelo tiene un tamaño de $(227, 227, 3)$. La primera capa de convolución cuenta con 96 filtros de $7 \times 7$ con un stride de 4 y está seguido por una función de activación ReLU, una capa de maxpooling con un tamaño de pool de $3 \times 3$ y un stride de 2 y una capa de *Local Response Normalization* (LRN) (Krizhevsky et al. 2012). Esta capa permite normalizar la entrada sobre las regiones de entrada locales que devuelve, en este caso la maxpooling de igual forma que se hace con los valores de las imágenes, antes de ser utilizadas como entrada de la red neuronal. La segunda capa de convolución cuenta con 256 filtros de $5 \times 5$ con un stride de 1 y a continuación cuenta con un ReLU, una capa de maxpooling y una LRN con las mismas características que las capas descritas anteriormente. La tercera capa convolucional cuenta con 384 filtros de $3 \times 3$ y un stride de 1 y a continuación cuenta con una ReLU y un maxpooling con las mismas características que los anteriormente descritos. En el último bloque de la red se encuentran las capas que servirán para realizar la clasificación de las diferentes entradas de la red en las diferentes clases. Está conformado, en primer lugar, por una capa de Flatten que transforma la salida de $(6, 6, 384)$ de la capa anterior en una entrada unidimensional de 13824. El bloque cuenta con tres capas de este tipo, contando las dos primeras con una capa de salida de 512 neuronas, la función de activación ReLU y sus respectivas capas de dropout con probabilidad del $50\,\%$.

La última capa de la red es también una fully connected con 8 neuronas de salida, una por cada clase del sistema de clasificación de edad. En esta capa se utiliza la función de activación de softmax, de forma que la salida de cada neurona es la probabilidad de que la entrada pertenezca a esa clase, donde la suma todos los valores 1. (Nasrabadi 2007)

Las diferentes capas que han sido descritas previamente pueden observarse en la Figura 3.7. Este modelo ha sido elegido para la realización de las pruebas junto a VGG-Face debido a los buenos resultados obtenidos en la estimación de edad en el propio trabajo (Levi & Hassner 2015).

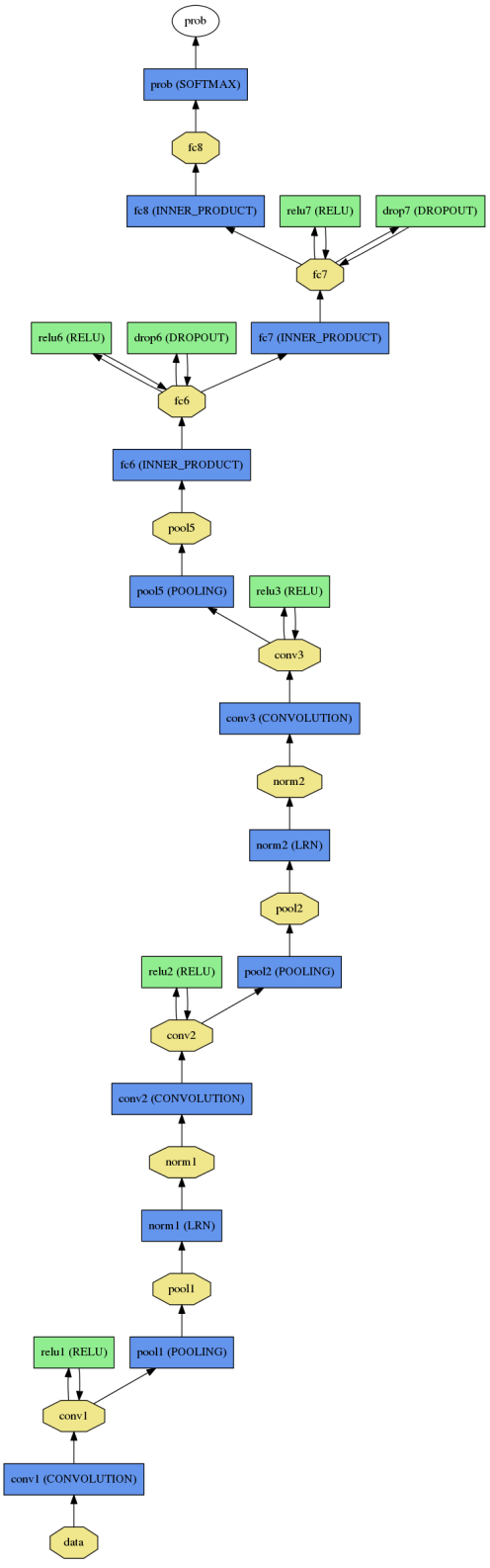



Figura 3.7: Estructura de la LeviHassner.

Fuente: (Levi & Hassner 2015).

# Capítulo 4

# Metodología e Implementación

## 4.1 Proceso de Desarrollo

Para el desarrollo de este proyecto se ha optado por una Metodología Basada en Prototipos debido a que cuenta con la ventaja de poder realizar diferentes modelos de forma rápida y sin consumir muchos recursos.

El proceso de creación de prototipos implica los siguientes pasos:

- **Identificar requisitos básicos:**
  Se determina que se va a utilizar un método convolucional para tratar las imágenes y realizar la estimación de edad. De esta forma se procede a realizar un revisión bibliográfica sobre el problema, buscando los mejores modelos.
- **Desarrollar el prototipo inicial:**
  En base a la documentación de los modelos seleccionados se realizan unas primeras pruebas con los modelos entrenados con los hiperparámetros utilizados por los autores.
- **Evaluación:**
  Se evalúa si los resultados del modelo con el dataset utilizado son los esperados o no y si los indicadores de rendimiento (precisión y pérdida tanto en entrenamiento, validación como testeo) que se han utilizado son los correctos.
- **Revisión y mejora del prototipo:**
  Se mejoran los modelos teniendo en cuenta los resultados obtenidos, de forma que este paso y el anterior se repiten hasta que un prototipo consiga los resultados esperados.

En la revisión y mejoras de prototipos se ha realizado la modificación de diferentes hiperparámetros, ejecución de diferentes tipos de entrenamiento, modificación de capas de los dos modelos seleccionados, uso de las redes neuronales como extractores de características de las imágenes; delegando la clasificación a otros métodos como Random Forest o SVM descritos anteriormente, etc. También se ha probado como entrada a las redes neuronales tanto hombro y cabeza de los individuos como solo el área de la cara, e incluso la combinación de ambas en la extracción de características. Todas estas modificaciones se encuentran explicadas en el capítulo 5 en sus respectivas fases.

## 4.2 Tecnologías

En este capítulo se describirán las diferentes herramientas utilizadas para la realización de este proyecto. Las librerías de visión por computador Dlib y OpenCV se encuentran comentadas en la sección 3.3.

### Python

Se trata de un lenguaje de programación de código abierto, interpretado y de tipado dinámico cuya sintaxis favorece un código legible. Actualmente es uno de los lenguajes más populares entre los científicos de datos debido a la gran cantidad de librerías estadísticas y numéricas con las que es compatible, motivo por el cual ha sido elegido para la realización de este proyecto.

### Keras

Es una librería de código abierto escrita en Python para el desarrollo de redes neuronales (Chollet et al. 2015). Fue diseñada para facilitar el desarrollo rápido de experimentos con redes neuronales profundas como las CNN y se caracteriza por ser modular, extensible y fácil de usar. Además sus programas pueden ser ejecutados en la GPU. Permite la utilización de múltiples *Backends* como Tensorflow (Abadi et al. 2015), el elegido en este proyecto, permitiéndonos crear cualquier implementación que pueda ser elaborada en ese lenguaje.

### Scikit-learn

Es una librería de aprendizaje automático de código abierto programada en Python que permite el uso algoritmos de clasificación, regresión y *clustering* (Pedregosa et al. 2011). En este proyecto se ha utilizado sus implementaciones de las SVM con kernel lineal y RBF en algunas de las pruebas.

### NumPy

Se trata de una librería de código abierto disponible en Python que permite la gestión de arrays y matrices multidimensionales extensas, además de disponer de numerosas funciones matemáticas de alto nivel para operar con ellas. En este proyecto hacemos uso de su clase *ndarray* para almacenar los valores de las imágenes con las que tratamos de forma eficiente en memoria (Walt et al. 2011).

### Google Colab

Se trata de una herramienta gratuita para el estudio e investigación de proyectos de aprendizaje automático en Python. Está basada en el código de software libre Jupyter y nos permite la ejecución de cuadernos utilizando los servidores de Google, pudiendo instalar la mayoría de librerías compatibles con Python en sus dos versiones. Cada usuario dispone de aproximadamente 12 gigas de RAM y puede hacer uso de aceleración hardware (GPU) si se necesita. Dispone además de una gran integración con el servicio de almacenamiento de la misma compañía, Google Drive, pudiendo programar la descarga de los ficheros de entrada de nuestros programas de este servicio y la subida de los resultados a nuestra carpeta personal.

### Pycharm

Se trata de un Entorno de Desarrollo Integrado (IDE) para el lenguaje de Python desarrollado por la compañía JetBrains. Ha sido utilizado para la ejecución de los programas elaborados en el trabajo junto a Google Colab.

### Weka

Es un programa que permite la ejecución de algoritmos de aprendizaje automático y minería de datos escrito en Java. En este proyecto se ha utilizado para la realización de ejecución de algoritmos de clasificación como

Random Forest o SVM, al igual que la ejecución de los respectivos PCA, previamente explicados.

#### LaTeX

Se trata de un sistema informático que permite la edición electrónica de documentos. Su elección frente a otros procesadores de texto como LibreOffice Writer o Microsoft Office se ha debido a que permite la elaboración cómoda de documentos estructurados como la presente memoria, dividida en diferentes capítulos, secciones, subsecciones, permitiendo el control en todo momento de la numeración de las mismas y las referencias cruzadas, además de su facilidad para construir índices o bibliografías de forma automática o editar los tamaños y tipos de letra según la parte del documento donde se encuentren.

#### Overleaf

Es un servicio web que permite la edición colaborativa de documentos en LaTeX. Se pueden iniciar proyectos a partir de plantillas de uso estándar (para artículos, tesis, informes, etc.), gestionar diferentes versiones de nuestros documentos, etc.

Ha sido utilizada para la realización de la presente memoria debido a que permite compartir los proyectos de LaTeX en línea para las revisiones de los tutores. Además de poder escribir este tipo de documentos sin necesidad de instalar ningún programa en nuestros equipos.

## 4.3 Implementación

### 4.3.1 Creación de un Modelo Neuronal para Fine-tuning

Como se adelantó en la sección 3.4, en este proyecto utilizamos tanto el modelo VGG-Face como LeviHassner como punto de inicio para nuestras pruebas.

El primero de ellos consiste en un modelo VGG-16 preentrenado para el reconocimiento facial de 2622 personas. Es por ello por lo que hemos tenido que hacer una serie de modificaciones al modelo, como puede observarse en la Figura 4.1. Después de crear el modelo utilizando la implementación de VGG-Face en Keras de (Malli 2016), se obtiene el modelo con los pesos del

```python
def crear_modelo():
    vgg_model = VGGFace(input_shape=(224, 224, 3))
    last_layer = vgg_model.get_layer('pool5').output
    x = Flatten(name='flatten')(last_layer)
    x = Dense(4096, name='fc6')(x)
    x = Dropout(0.5)(x)
    x = Activation('relu', name='fc6/relu')(x)
    x = Dense(4096, name='fc7')(x)
    x = Dropout(0.5)(x)
    x = Activation('relu', name='fc7/relu')(x)
    x = Dense(6, name='fc8')(x)
    out = Activation('softmax', name='fc8/softmax')(x)
    model = Model(vgg_model.input, out)

    #Entrenamos únicamente las FC
    for layer in model.layers[:19]: #Hasta el Pool5 no se realiza entrenamiento
        layer.trainable = False

    return model
```

Figura 4.1: Código en el que se crea el modelo VGG-Face para *fine-tuning*.

reconocimiento facial precargados desde la capa de entrada de la red hasta la capa de maxpooling del quinto bloque convolucional de la red. De esta forma podemos añadir posteriormente la capa de Flatten, las tres capas fully connected y las funciones de activación ReLU y softmax, además de dos capas de dropout que no se encontraban incluidas en la implementación utilizada. En este apartado también se realiza la modificación del número de neuronas de salida de la red al número de clases, en nuestro caso seis rangos de edad. Esta modificación también se realiza cuando se define el modelo LeviHassner, además de la modificación de la capa de entrada a un tamaño de $(224, 224, 3)$.

En el caso de VGG-Face, generalmente utilizamos un tipo de entrenamiento denominado *Fine-tuning*. Se basa en la utilización de redes preentrenadas *(transfer learning)* para ahorrar cómputo y tiempo de entrenamiento, además de prevenir posibles problemas de sobreentrenamiento debido a que el conjunto de datos sea limitado. En este caso ha sido utilizado además debido a que los pesos del reconocimiento facial han demostrado ayudar en la estimación de edad, como ya se ha descrito en la sección 3.4. Es por ello por lo que se "congelan" durante el entrenamiento las capas de los bloques convolucionales, modificando el valor de sus atributos "traineable" a *False*.

### 4.3.2 Inicialización de Capas y Cálculo del Número de Épocas en LeviHassner

Como ya se ha descrito en la sección 3.4, en este proyecto se utilizan tanto el modelo VGG-Face como el LeviHassner. Este último se caracteriza por inicializar los pesos de sus capas de forma aleatoria y no utilizar pesos precargados como VGG-Face.

```
def crear_modelo():

  img_input = Input(shape=(224, 224, 3))
  #conv1
  x = Conv2D(96, (7,7), strides=(4,4), padding='valid',
             use_bias=True,
             kernel_initializer=RandomNormal(mean=0.0, stddev=0.01), bias_initializer='zeros',
             name='conv1')(img_input)

  x = Activation('relu', name='relu1')(x)
  x = MaxPooling2D(pool_size=(3, 3), strides=2, padding='valid', name='pool1')(x)
  x = LocalResponseNormalization()(x)

  #conv2
  x = Conv2D(256, (5,5), strides=(1,1), padding='same',
             use_bias=True,
             kernel_initializer=RandomNormal(mean=0.0, stddev=0.01), bias_initializer='ones',
             name='conv2')(x)

  x = Activation('relu', name='relu2')(x)
  x = MaxPooling2D(pool_size=(3, 3), strides=2, padding='valid', name='pool2')(x)
  x = LocalResponseNormalization()(x)

  #conv3
  x = Conv2D(384, (3,3), strides=(1,1), padding='same',
             use_bias=True,
             kernel_initializer=RandomNormal(mean=0.0, stddev=0.01), bias_initializer='zeros',
             name='conv3')(x)

  x = Activation('relu', name='relu3')(x)
  x = MaxPooling2D(pool_size=(3, 3), strides=2, padding='valid', name='pool5')(x)

  #fc6
  x = Flatten(name='flatten')(x)
  x= Dense(512, activation=None, use_bias=True,
                        kernel_initializer=RandomNormal(mean=0.0, stddev=0.005),
                        bias_initializer='ones', name='fc6')(x)
  x = Activation('relu', name='relu6')(x)
  x = Dropout(0.5, name='dropout6')(x)

  #fc7
  x= Dense(512, activation=None, use_bias=True,
                        kernel_initializer=RandomNormal(mean=0.0, stddev=0.005),
                        bias_initializer='ones', name='fc7')(x)
  x = Activation('relu', name='relu7')(x)
  x = Dropout(0.5, name='dropout7')(x)

  #fc8
  x= Dense(6, activation=None, use_bias=True,
                        kernel_initializer=RandomNormal(mean=0.0, stddev=0.01),
                        bias_initializer='zeros', name='fc8')(x)
  out = Activation('softmax', name='fc8/softmax')(x)

  return Model(inputs=img_input, outputs=out)
```

Figura 4.2: Código en el que se crea el modelo LeviHassner siguiendo las características tanto del artículo asociado como de la implementación oficial.

```
class LocalResponseNormalization(Layer):

  def __init__(self, n=5, alpha=0.0001, beta=0.75, k=2, **kwargs):
    self.n = n
    self.alpha = alpha
    self.beta = beta
    self.k = k
    super(LocalResponseNormalization, self).__init__(**kwargs)

  def build(self, input_shape):
    self.shape = input_shape
    super(LocalResponseNormalization, self).build(input_shape)

  def call(self, x, mask=None):
    if K.image_dim_ordering == "th":
      _, f, r, c = self.shape
    else:
      _, r, c, f = self.shape
    squared = K.square(x)
    pooled = K.pool2d(squared, (self.n, self.n), strides=(1, 1),
      padding="same", pool_mode="avg")
    if K.image_dim_ordering == "th":
      summed = K.sum(pooled, axis=1, keepdims=True)
      averaged = self.alpha * K.repeat_elements(summed, f, axis=1)
    else:
      summed = K.sum(pooled, axis=3, keepdims=True)
      averaged = self.alpha * K.repeat_elements(summed, f, axis=3)
    denom = K.pow(self.k + averaged, self.beta)
    return x / denom

  def compute_output_shape(self, input_shape):
    return input_shape
```

Figura 4.3: Código en el se implementa la capa Local Response Normalization.

Basado en (Gulli & Pal 2017).

A pesar de que según el artículo (Levi & Hassner 2015) todas los valores aleatorios son extraídos de una distribución normal con media cero y desviación típica de $0,01$, en la implementación de los autores del modelo en Caffe (Framework para ANN profundas) (Jia et al. 2014), accesible a través de la web del trabajo que se encuentra en el propio artículo científico citado, observamos que no todas las capas de LeviHassner han sido inicializadas de esta forma. Para la implementación del modelo en Keras se ha seguido esta implementación, que puede observarse en la Figura 4.2.

Como la capa de LRN no se encuentra disponible en Keras, se ha creado una capa propia en la librería en base al ejemplo que se encuentra en (Gulli & Pal 2017), siendo corregida utilizando la implementación oficial en Caffe del modelo neuronal. Esta implementación se puede observar en la Figura 4.3.

También en (Levi & Hassner 2015) se hace referencia al número de iteraciones realizadas por ellos a la hora de entrenar la red, entendiéndose por iteraciones al número de veces que se actualizan los pesos de la red. Este número de iteraciones es 10000 y, en el caso de Keras, es el número de veces en el que se realiza un batch. Para calcular el número equivalente de batches en épocas, se ha realizado el siguiente cálculo

- 12714 instancias de entrenamiento.

- 50 instancias de tamaño de batch.
- $1271450 = 254{,}28 \simeq 255$ iteraciones/época.
- Objetivo: 10000 iteraciones
- 255 iteraciones/época
- $10000 \div 255 = 39{,}21 \simeq 40$ épocas

Realmente se realizan $40 \times 254{,}28$ iteraciones, es decir, $10171,2 \simeq 10171$.

### 4.3.3 Partición del Conjunto de Datos el Entrenamiento, Validación y Testeo

Como se detalló en la sección 3.2, el conjunto de datos utilizado cuenta con 24080 imágenes que han sido utilizadas en las fases de entrenamiento, validación y testeo. La fase de entrenamiento es aquella en la que la red modifica sus pesos observando las diferentes imágenes de entrada, es decir, es en la fase que aprende. En las fases de validación y testeo se le muestra a la red instancias diferentes a las del conjunto de entrenamiento para comprobar su capacidad de generalización. La fase de validación se realiza después de cada época, es decir, después de haberle enseñado a la red una vez todas las instancias de entrenamiento, de forma que se evite un sobreajuste a las imágenes de entrenamiento mientras que la fase de testeo se realiza una vez la red ha terminado de entrenar.

En el primer experimento, al tratarse de un problema muy sencillo, se ha hecho una división aleatoria de las 24080 instancias en un 60 % para entrenamiento (14445 imágenes) y un 40 % para testeo (9635 imágenes). Es decir, no se ha realizado ningún tipo de validación.

Por otro lado, en el resto de pruebas se ha utilizado una división dinámica de los datos. La denominamos dinámica debido a que esta no requiere que las instancias de las tres fases estén en subdirectorios distintos, como es sugerido en los ejemplos de la librería Keras, sino que hace uso de tres listas con las diferentes rutas relativas de todo el conjunto de datos. Para realizar la división del conjunto de datos se ha indexado, en primer lugar, todas las rutas relativas del conjunto de datos y se ha insertado en una lista. Posteriormente, se han distribuido aleatoriamente sus datos y se ha procedido a su división. Los primeros 66 % de los datos (15892 instancias) se han utilizado para el entrenamiento y la validación mientras que el 34 % restante para testeo (8188 instancias). Del 66 % de los datos, el 80 % (12714 instancias) se destinado al entrenamiento mientras que el 20 % restante (3178 instancias) se utilizan en la validación. Para comprobar que los cambios realizados durante los experimentos son los causantes de la mejora o empeoramiento de

```
def entrenar_modelo_guardando(epochs, bs, file, files_tr, files_val, directorio):
    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizers.Adadelta(lr=1e-1, rho=0.95,
                                                epsilon=None, decay=0.0),
                  metrics=['accuracy'])
    print('[INFO] Compilado el modelo.')

    print("[INFO] Empieza el entrenamiento")
    longitud = len(files_tr)
    longitud_validation = len(files_val)

    filepath="weights.best_" + file + '.h5'
    checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                                 save_best_only=True, save_weights_only=False,
                                 mode='auto')

    stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=5,
                             verbose=1, mode='auto')

    fileTrained =  file + '_trained.h5'

    callbacks_list = [checkpoint,stopping,
                      GuardarCadaN_Epochs(model, 10, fileTrained)]
    steps = longitud/bs if longitud % bs == 0 else int(longitud/bs)+1
    steps_val = longitud_validation/bs if longitud_validation % bs == 0 else int(longitud_validation/bs)+1
    hist = model.fit_generator(generate_arrays_from_file_selected(directorio + '/*',bs, files_tr),
                               max_queue_size=1, verbose=1,
                               steps_per_epoch=steps,
                               validation_data=generate_arrays_from_file_selected(directorio + '/*',bs, files_val),
                               validation_steps=steps_val,epochs=epochs,
                               shuffle=True, callbacks=callbacks_list)
    print(hist.history)
    print(hist.history.keys())

    copia = hist.history.copy()

    print("[INFO] Guardando modelo final entrenado en: " + fileTrained)
    model.save(fileTrained)
```

Figura 4.4: Código en el que se entrena y valida un modelo neuronal en Keras, utilizando además *Callbacks*.

los resultados, se ha utilizado siempre los mismos conjuntos de entrenamiento, validación y testeo, contando con la posibilidad de cambiarlos fácilmente gracias al sistema descrito.

### 4.3.4 Entrenamiento y Validación de un Modelo Neuronal Utilizando Keras

Para entrenar los diferentes modelos neuronales en Keras se ha optado por su función fit_generator. Esta función nos permite cargar las imágenes en memoria batch a batch. En nuestro caso, generalmente de 50 en 50 utilizando un generador. Los generadores permiten realizar la carga de los datos de forma paralela al entreno de la red, siendo por lo tanto más eficiente. Estos generadores son efectivos cuando se utilizan conjuntos de datos de gran tamaño como las imágenes, ya que la memoria RAM del equipo puede ser insuficiente para gestionar todos los datos cargados al mismo tiempo. Las fases de testeo y validación también puede realizarse utilizando generadores. Un ejemplo de función generador puede verse en la subsección 4.3.5.

En la Figura 4.4 se puede observar un ejemplo de entrenamiento y vali-

dación utilizando además los llamados *Callbacks* son funciones que pueden ejecutarse durante el entrenamiento. En el ejemplo observamos el uso de la clase "ModelCheckpoint" para guardar el mejor modelo, utilizando como medida el mayor valor de precisión en validación. También se usa la clase "EarlyStopping" para parar el entrenamiento de la red cuando la función objetivo no mejore en la validación durante cinco épocas seguidas. También se pueden crear clases que hereden de la clase "Callback", como es el caso de la clase "GuardarCadaN_Epochs", que guarda los pesos del modelo entrenado cada N épocas, en este caso 10.

En el momento en el que se compila el modelo neuronal antes entrenar, se debe establecer la función objetivo, el optimizador y las métricas que serán mostradas durante las tres fases. En el presente trabajo se han utilizado los siguientes algoritmos de optimización: SGD (Bottou 2010), RMSprop (Hinton et al. 2012), Adagrad (Duchi et al. 2011), Adadelta (Zeiler 2012), Adam (Kingma & Ba 2014), Adamax (Kingma & Ba 2014) y Nadam (Dozat 2016).

Como se puede observar también en la Figura 4.4, se debe indicar a la función fit_generator el número de "pasos" que debe realizar durante y después de cada época respectivamente para el generador del entrenamiento y de validación.

### 4.3.5 Lectura y Normalización de los Datos Utilizando OpenCV y NumPy

Como se ha comentado en la subsección 4.3.4, durante el entrenamiento se utiliza una función de tipo generador de Keras para cargar las imágenes en memoria de batch en batch.

En esta función, que puede observarse en la Figura 4.5, se inicializan las matrices tanto para las imágenes como para las etiquetas en las que se cargarán "batchsize" elementos, generalmente 50. También se realiza la normalización de los valores de las imágenes, de forma que se encuentren entre 0 y 1. Para la lectura de los ficheros se utiliza la librería OpenCV mientras que se utilizan las instancias de la clase ndarray de la librería NumPy para almacenarlos en variables de Python.

El dataset de entrada ya ha sido redimensionado para que sea compatible con las capas de entrada de los modelos utilizados, ambos de tamaño $(224, 224, 3)$. Este proceso de redimensionamiento se ha realizado junto al proceso de oversampling descrito en la sección 3.2.

```
def generate_arrays_from_file_selected(fpath, batchsize, files):
    nfiles = len(files)
    remainder = nfiles % batchsize
    while 1:
        n_entries = 0
        i = 0
        size = batchsize
        data = np.empty((size, 224, 224, 3), dtype=np.float64)
        labels = np.empty((size, 6), dtype=np.int64)
        for fpath in files:
            img = imread(fpath)
            data[i, ...] = img.astype("float") / 255.0
            labels[i, ...] = age_group(fpath)
            i += 1
            if i == size:
                n_entries += i
                i = 0
                yield (data,labels)
                if n_entries+size>nfiles:
                    size = remainder
                data = np.empty((size, 224, 224, 3), dtype=np.float64)
                labels = np.empty((size, 6), dtype=np.int64)
```

Figura 4.5: Código en el que se leen las imágenes y se normalizan sus valores.

### 4.3.6 Obtención salida de Capa Intermedia en Keras para Extracción de Características

La librería Keras permite crear un modelo nuevo a partir de uno ya existente conservando los pesos de sus conexiones. Para ello se carga en primer lugar el modelo que queramos utilizar y se crea uno nuevo indicando qué capa del modelo cargado queremos que sea la entrada del nuevo modelo y cuál la salida, tal y como se hace en la Figura 4.6.

Después se puede obtener por ejemplo las salidas de la capa final del nuevo modelo utilizando un generador y la función "fit". En este caso hemos ido guardando la predicción junto a su etiqueta en diferentes matrices creadas previamente. De forma que, al terminar de iterar todo el conjunto considerado, se concatenen las salidas de la capa y las etiquetas asociadas a las imágenes.

Este es el procedimiento seguido cuando se utiliza algún modelo entrenado de VGG-Face o LeviHassner para realizar la extracción de características, de forma que posteriormente las salidas concatenadas con sus correspondientes etiquetas se utilicen en otros métodos de clasificación como Random Forest o SVM.

```
face = load_model('VGG-Face_6Clases_v6_trained.h5')
face_intermedia = Model(inputs=face.input,
                        outputs=face.get_layer('fc6').output)

# Training set --> 12714
directorio = 'Dataset_withFlip'
bs = 1
nInstanciasTr = len(trainval_files)

datos_Entrenamiento = generate_arrays_from_file_selected(directorio + '/*',
                                                         bs, trainval_files)
entrada_train, etiqueta_train = next(datos_Entrenamiento)

predicciones = np.empty([nInstanciasTr, 4096])
etiquetas = np.empty([nInstanciasTr,6])

for i in range(nInstanciasTr):
  predicciones[i] = face_intermedia.predict(entrada_train, steps=1, verbose=1)
  etiquetas[i] = etiqueta_train
  entrada_train, etiqueta_train = next(datos_Entrenamiento)

resultados_train = np.concatenate((predicciones, etiquetas), axis=1)
```

Figura 4.6: Código en el se guardan las salidas de una capa intermedia junto a su etiqueta asociada.

### 4.3.7 Detección y Recorte del Área de la Cara Utilizando Dlib, OpenCV y NumPy

En determinados experimentos de este trabajo, el área de interés, que se le suministra en la capa de entrada a nuestros modelos, es únicamente el rostro. Como nuestro dataset contiene imágenes donde generalmente se observan los hombros y la cabeza, es necesario que creemos una versión de este conjunto de datos seleccionado con el área de la cara recortada. Un ejemplo de la estructura seguida se puede observar en la Figura 4.7.

Es necesario conocer que en caso de no detectar alguno de los extremos del área (caras giradas o imágenes borrosas), el detector podrá devolver números fuera del rango de la imagen. Es por ello por lo que se utilizan funciones de máximo y mínimo, de forma que si el detector no encuentra uno de los lados, se recorte el área de la cara desde el inicio de ese extremo.

El recorte ha sido realizado sobre el dataset AgeDB con las dimensiones de las imágenes originales y, una vez se han recortado las caras, se han redimensionado para hacerlas compatibles a los modelos utilizados.

```
folder = 'Dataset_caras'
face_detector = dlib.get_frontal_face_detector()

img = dlib.load_rgb_image('Dataset_withFlip/109_PaulAnka_72_m.png')
height = np.size(img, 0)
width = np.size(img, 1)
dets = face_detector(img, 1)


if len(dets) > 0: #Si alguna cara
  cara = dets[0]
  img = img[max(0,cara.top()):min(cara.bottom(),height),
            max(0,cara.left()):min(cara.right(),width)]

  name = name[name.rfind('/'):]
  file = name[:name.find(".")]
  cv2.imwrite(folder + file + '.png', img)
```

Figura 4.7: Código donde se recorta la cara de una imagen utilizando el detector de Dlib y se guarda.

### 4.3.8 Uso de Weka para la Aplicación de PCA y la Ejecución de un Método de Clasificación.

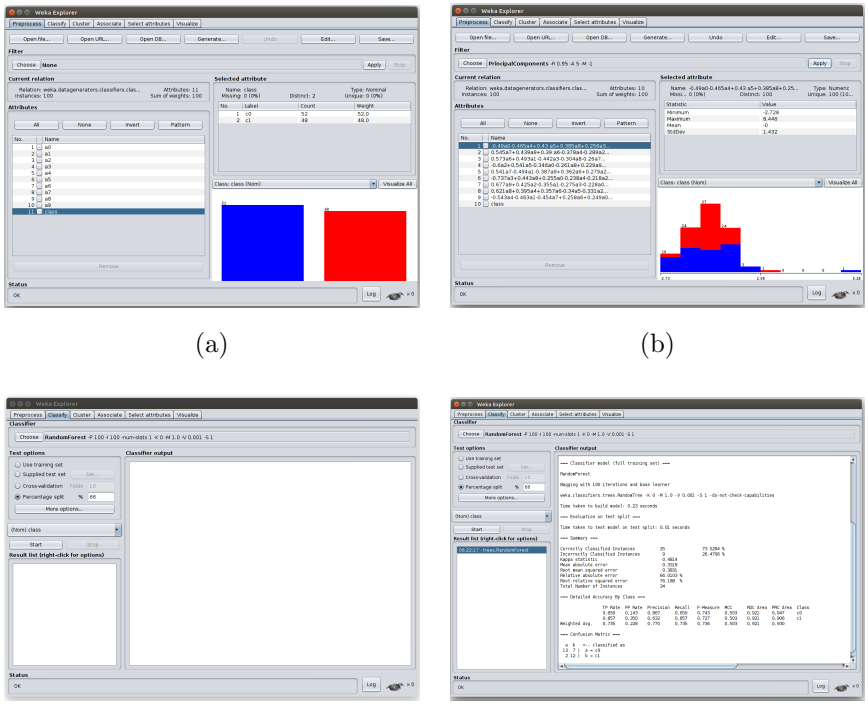
En algunas de las pruebas realizadas en el trabajo se plantea utilizar un PCA. Para realizarlo se ha utilizado la herramienta de Weka.

Para ello en la pestaña de "Preprocess" deben abrirse los datos utilizando las diferentes opciones que pueden verse en la Figura 4.8 (a). Posteriormente se elige en los filtros la opción "PrincipalComponents", que por defecto devuelve como salida los atributos que contengan como mínimo un 95 % de variabilidad. Seleccionamos "Apply" y se realiza el análisis. Una vez terminado se puede utilizar su salida en un método de clasificación, como puede ser Random Forest. Una vez seleccionada, indicado de dónde obtener el conjunto de testeo en "Test options", pulsamos el botón "Start" y se entrena y testea el modelo, muchos como la matriz de confusión en testeo, la tasa de acierto, la precisión, etc.

### 4.3.9 Entrenamiento y Testeo de Máquina de Soporte Vectorial en Scikit-learn

Al igual que se utiliza Weka para ejecutar los métodos de clasificación Random Forest y SVM, también se ha utilizado la librería Scikit-learn para realizar ejecuciones de algunas SVM.

En la Figura 4.9 podemos observar un ejemplo de cómo se realiza el



(a) (b)

(c) (d)

Figura 4.8: Ejemplo del uso de Weka para la Aplicación de un PCA y la Ejecución de un Método de Clasificación.

entrenamiento y testeo de un modelo SVM. En primer lugar se transforman las etiquetas, que se encontraban codificadas utilizando *One-hot* (vector con la misma longitud que el número de clases con todos los valores a cero salvo la posición de la clase a la que pertenece, donde se asigna el valor uno) en una codificación numérica. Después se crea una instancia de la clase *SVC*, es decir, un SVM con kernel RBF utilizando la implementación "libsvm". Después se entrena el objeto con el método "fit()" y se obtiene el valor de precisión en validación utilizando la función "score()".

```
clases = [1, 2, 3, 4, 5, 6]
etiquetas_numero = []
etiquetas_test_numero = []

for i in range(len(etiquetas)):
  etiquetas_numero.append(clases[np.argmax(etiquetas[i,:])])

for i in range(len(etiquetas_test)):
  etiquetas_test_numero.append(clases[np.argmax(etiquetas_test[i,:])])

clf = svm.SVC()
clf.fit(predicciones, etiquetas_numero)

precision = clf.score(predicciones_test,etiquetas_test_numero)
```

Figura 4.9: Código en el que se utiliza la implementación del algoritmo SVM con kernel RBF de Scikit-learn.

# Capítulo 5

# Experimentos y Resultados

En este capítulo se describen los diferentes experimentos realizados en el proyecto, estando estos agrupados en un total de diecinueve pruebas. En la mayoría de estas la entrada de las redes son las imágenes del dataset en las que se observan los dos hombros y la cabeza del sujeto. Una vez se ha encontrado el mejor modelo tanto en VGG-Face como en LeviHassner para esta entrada, se han usados sus configuraciones para entrenar utilizando como entrada las imágenes con las caras de los sujetos delimitadas.

## 5.1 Pruebas

En las diferentes tablas de este capítulo se muestran los resultados de las mejores épocas del entrenamiento de las redes, basándonos para ello en los valores de la tasa de acierto en validación. En caso de que dicho resultado sea prometedor, se realizará el testeo de la red.

La función objetivo utilizada en todas las pruebas es la denominada "categorical_crossentropy", recomendada para los problemas de clasificación multiclase.

### Prueba 1: VGG-Face Utilizando Diferentes Parámetros del Optimizador SGD

En esta primera prueba se realiza un entrenamiento de tipo fine-tuning en el modelo VGG-Face con 200 épocas máximas prefijadas y un batch size

de 50 instancias. Se utiliza el indicador de rendimiento de tasa de acierto en el entrenamiento en los callbacks tanto para guardar el mejor modelo con *ModelCheckpoint* como parar el entrenamiento con *EarlyStopping* si en 5 épocas seguidas no mejora un mínimo de 0,003. También se guardan los pesos cada 10 épocas con GuardarCadaN_Epochs.

Como se mencionó en la subsección 4.3.3, en esta prueba es en la única que se utiliza una partición inicial 60/40 del conjunto de datos para entrenamiento y testeo respectivamente, no realizándose testeo. Es por ello por lo que se tiene en cuenta la tasa de acierto de la mejor red en entrenamiento para considerar si es conveniente o no realizar la fase de testeo a la red.

| **ID** | **Épocas** | | **Entrenamiento** | | **Testeo** | | **Optimizador** |
|---|---|---|---|---|---|---|---|
| | **Mejor** | **Última** | **Pérdida** | **Tasa de Acierto** | **Pérdida** | **Tasa de Acierto** | |
| 1 | 61 | 62 | 0,045 | 0,996 | 1,486 | 0,552 | SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=True) |
| 2 | 46 | 47 | 1,134 | 0,503 | - | - | SGD(lr=1e-4, momentum=0.9) |

Tabla 5.1: Resultados de las mejores épocas de la Prueba 1.

Como se puede ver en la Tabla 5.1, se prueban arbitrariamente diferentes parámetros del optimizador SGD. En primer lugar el hiperparámetro LR. Este hiperparámetro controla cuánto se ajuntan los pesos de la red con respecto a la perdida del gradiente (Zulkifli 2018). También se ha utilizado el hiperparámetro *decay*, que permite modificar el learning rate cada $X$ épocas. También se prueba el hiperparámetro *momentum*, que acelera el optimizador en una dirección relevante y amortigua las posibles oscilaciones. Por último, también se prueba el argumento *nesterov*, que permite activar o no el momentum de Nesterov.

En esta prueba inicial se observa cómo el primer método, que había obtenido una tasa de acierto en el entrenamiento de $99,6\,\%$, está claramente sobreentrenado. De esta forma, con el conjunto de testeo esta cifra baja hasta un $55,2\,\%$. Esta variación de la tasa de acierto nos demuestra lo necesaria que es la validación del modelo, además del control que es necesario hacer para que los resultados con este conjunto de datos mejoren durante el entrenamiento, ya que el control único de la mejora de los indicadores de rendimiento del conjunto de entrenamiento pueden llevarnos a una situación de sobreentrenamiento de nuestra red.

Debido a que esta prueba ha sido realizada con una división de los datos diferente a las demás, donde se utilizan más imágenes para el entrenamiento

que en el resto de pruebas, esto resultados no son contemplados en el listado global de mejores puntuaciones. Sin embargo la misma configuración del optimizador se utiliza en la ejecución con SGD de la Prueba 2.

## Prueba 2: VGG-Face Probando Los Diferentes Optimizadores de Keras

En esta segunda prueba se emplean todos los optimizadores ofrecidos por Keras de forma nativa.

Con respecto a la prueba anterior, en esta se utiliza el conjunto de entrenamiento, validación y testeo que se mantiene en el resto de pruebas del proyecto y que se encuentra descrito en la sección 4.3.3. Además se modifica el indicador de rendimiento asociado a *EarlyStopping* al valor de la función de coste para el conjunto de validación, de forma que se detectan posibles sobreentrenamientos.

A excepción de optimizador SGD en el que utilizamos los parámetros que mejor resultado obtuvieron en la Prueba 1, en el resto de optimizadores se utilizan los parámetros por defecto, tal y como se recomienda en la mayoría de ellos en la propia documentación de la librería Keras.

| **ID** | **Épocas** | | **Entrenamiento** | | **Validación** | | **Testeo** | **Optimizador** |
|---|---|---|---|---|---|---|---|---|
| | **Mejor** | **Última** | **Pérdida** | **Tasa de Acierto** | **Pérdida** | **Tasa de Acierto** | **Tasa de Acierto** | |
| 1 | 21 | 21 | 0,818 | 0,677 | 1,132 | 0,502 | 0,517 | SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=True) |
| **2** | **10** | **10** | **0,315** | **0,884** | **1,671** | **0,537** | **0,548** | **Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)** |
| 3 | 7 | 7 | 0,649 | 0,742 | 1,271 | 0,519 | 0,521 | Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |
| 4 | 2 | 6 | 12,094 | 0,250 | 12,122 | 0,248 | 0,243 | Adagrad(lr=0.01, epsilon=None, decay=0.0) |
| 5 | 8 | 8 | 0,338 | 0,874 | 1,648 | 0,534 | 0,527 | Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 6 | 2 | 6 | 12,094 | 0,250 | 12,122 | 0,248 | 0,243 | Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, schedule_decay=0.004) |
| 7 | 1 | 6 | 12,053 | 0,249 | 12,122 | 0,248 | 0,243 | RMSprop(lr=1e-3, rho=0.9, epsilon=None, decay=0.0) |

Tabla 5.2: Resultados de las mejores épocas de la Prueba 2.

Como se puede observar en la Tabla 5.2, el modelo con el optimizador Adadelta consigue los mejores resultados en testeo. También consiguen re-

sultados cercanos los modelos con los optimizadores Adamax, Adam y SGD. Los tres modelos restantes mantienen el valor 0,248 en validación desde la primera época hasta la sexta, en la que terminan el entrenamiento debido a la condición del *EarlyStopping*.

## Prueba 3: VGG-Face, Influencia del Parámetro decay con Diferentes Optimizadores

En esta prueba se utilizan diferentes parámetros para los optimizadores Adam, Adamax y RMSprop. Los dos primeros fueron utilizados en la anterior prueba con los valores por defecto del artículo asociado pero, en estos casos, no se sugería no modificarlos.

De esta forma en Adam se observa cómo influye aplicarle decay y la variante AMSGrad del algoritmo (Reddi et al. 2018) mientras que en Adamax, con el LR a la mitad que el sugerido en el artículo asociado (de 0,002 a 0,001), se prueban los efectos del decay. Además, en el RMSprop también probamos los efectos del decay.

Esta prueba se diferencia de la anterior en que se guarda el mejor modelo en base al indicador de tasa de acierto en la validación, no en el entrenamiento como en las pruebas previas.

Como se puede ver en la Tabla 5.3, al aplicarle a la configuración original de Adam un decay, su valor en testeo desciende ligeramente mientras que si se aplica junto al amsgrad, su valor se mantiene sin cambios. Por otro lado, el Adamax mejora en una centésima en testeo con respecto al modelo original al modificar el LR a la mitad mientras que, si se le aplica también un decay, la tasa de acierto empeora ligeramente. Por último el RMSprop se mantiene en las 6 épocas de ambas versiones en el valor de la tasa de acierto en testeo de 0,243.

## Prueba 4: Entrenamiento Completo de VGG-Face con Pesos Aleatorios (VGG-16)

En esta cuarta prueba queremos comprobar si el entrenamiento con finetuning con los pesos de reconocimiento facial realmente contribuyen a obtener mejore resultados en la estimación de edad. Para ello seleccionamos los 4 mejores modelos globales de las pruebas anteriores (sin contar con la Prueba 1 debido a motivos ya explicados) y utilizamos su misma configuración

| ID | Épocas | | Entrenamiento | | Validación | | Testeo | Optimizador |
|---|---|---|---|---|---|---|---|---|
| | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| 1 | 7 | 7 | 0,649 | 0,742 | 1,271 | 0,519 | 0,521 | Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |
| 2 | 9 | 9 | 0,516 | 0,797 | 1,341 | 0,517 | 0,515 | Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=1e-6, amsgrad=False) |
| 3 | 7 | 8 | 0,645 | 0,736 | 1,218 | 0,519 | 0,521 | Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=1e-6, amsgrad=True) |
| 4 | 8 | 8 | 0,338 | 0,874 | 1,648 | 0,534 | 0,527 | Adamax(lr=0.002, beta_1=0.9, beta_2=0.999 , epsilon=None, decay=0.0) |
| **5** | **6** | **7** | **0,404** | **0,848** | **1,411** | **0,530** | **0,528** | **Adamax(lr=1e-3, beta_1=0.9, beta_2=0.999 , epsilon=None, decay=0.0)** |
| 6 | 4 | 7 | 0,743 | 0,693 | 1,195 | 0,522 | 0,518 | Adamax(lr=1e-3, beta_1=0.9, beta_2=0.999 , epsilon=None, decay=1e-6) |
| 7 | 1 | 6 | 12,053 | 0,249 | 12,122 | 0,248 | 0,243 | RMSprop(lr=1e-3, rho=0.9, epsilon=None, decay=0.0) |
| 8 | 1 | 6 | 12,054 | 0,249 | 12,122 | 0,248 | 0,243 | RMSprop(lr=1e-3, rho=0.9, epsilon=None, decay=1e-6) |

Tabla 5.3: Resultados de las mejores épocas de la Prueba 3.

configuración en la arquitectura VGG-Face con pesos aleatorios y entrenamiento completo, es decir, una VGG-16. En la Tabla 5.4 se puede observar los mejores modelos obtenidos.

Tal y como puede observarse en la Tabla 5.5, ninguno de los modelos llega a superar el 35 % de tasa de acierto en su mejor época de validación, es por eso por lo que se decide no realizar ni siquiera el testeo para estos modelos. Con esta prueba queda demostrado que los pesos del reconocimiento facial de VGG-Face contribuyen efectivamente a mejorar la exactitud en la tarea de estimación de edad.

## Prueba 5: Entrenamiento Completo de VGG-Face (con pesos de reconocimiento facial)

De la misma forma que en la Prueba 4 utilizábamos las cuatro mejores configuraciones de VGG-Face para realizar el entrenamiento completo de la red con pesos aleatorios, en esta Prueba 5 realizamos también el entrenamiento completo de la red pero manteniendo los pesos de reconocimiento facial precargados.

Como se puede ver en la Tabla 5.6, ningún modelo supera el 50 %, es

| ID | Épocas | | Testeo | Optimizador |
|---|---|---|---|---|
| | Mejor | Última | Tasa de Acierto | |
| **1** | **10** | **10** | **0,548** | **Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)** |
| 2 | 6 | 7 | 0,528 | Adamax(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 3 | 8 | 8 | 0,527 | Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 4 | 7 | 7 | 0,521 | Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |

Tabla 5.4: Resultados de las cuatro mejores configuraciones globales en validación.

por ello por lo que se le realiza el testeo a los modelos que superan el 40 %. De esta forma vuelve a quedar demostrada la validez del entrenamiento de tipo fine-tuning con los pesos de reconocimientos facial precargados para la estimación de edad.

## Prueba 6: Entrenamiento Completo de VGG-Face y VGG-16 con LR de 0,01

Desde esta prueba hasta la novena prueba utilizaremos los mejores modelos presentados previamente en la Tabla 5.4 modificando tanto su LR inicial y/o su hiperparámetro decay con un valor relacionado con el número de épocas máximas prefijadas y el LR inicial. Hay que tener en cuenta que en dicha tabla las dos configuraciones con el optimizador Adamax se diferencian únicamente en su LR, por lo que se descarta el asociado al ID 3, pasando a ser identificado con ese número la configuración con el optimizador Adam.

Además en las pruebas mencionadas se ha modificado el valor prefijado de épocas máximas de 200 a 50 y se realizará un entrenamiento completo tanto con pesos aleatorios (VGG-16) como con pesos de reconocimiento facial (VGG-Face).

| ID | Épocas | | Entrenamiento | | Validación | | Optimizador |
|---|---|---|---|---|---|---|---|
| | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | |
| 1 | 25 | 25 | 1,744 | 0,249 | 1,751 | 0,248 | Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0) |
| 2 | 11 | 13 | 1,173 | 0,519 | 1,528 | 0,335 | Adamax(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 3 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 4 | 37 | 37 | 1,744 | 0,249 | 1,751 | 0,248 | Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |

Tabla 5.5: Resultados de las cuatro mejores configuraciones globales en VGG-16.

En esta sexta prueba se ha modificado el LR inicial en las tres configuraciones a $1e-2$. Como puede observarse en la Tabla 5.7, solamente la configuración con el optimizador Adadelta y el modelo VGG-Face consigue superar el 50 % en validación, motivo por el cual se realiza solamente el testeo con esa configuración. La tasa de acierto en el testeo es de un $54,3$ %.

## Prueba 7: Entrenamiento Completo de VGG-Face y VGG-16 con LR de 0,01 y decay

En esta prueba se sigue el esquema establecido en la Prueba 6, incluyendo en las diferentes configuraciones un decay en cada época del valor del lr inicial 0,01 entre el número máximo de épocas prefijadas (50).

Nuevamente, la configuración con el optimizador Adadelta y el modelo VGG-Face es la única que consigue superar el 50 % en validación, motivo por el cual se realiza solamente el testeo con esa configuración. La tasa de acierto en el testeo es de un $54,2$ %. Los diferentes resultados pueden observarse en la Tabla 5.8.

| ID | Épocas | | Entrenamiento | | Validación | | Testeo | Optimizador |
|---|---|---|---|---|---|---|---|---|
| | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| 1 | 18 | 18 | 0,914 | 0,597 | 1,615 | 0,400 | 0,415 | Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0) |
| **2** | **12** | **12** | **0,703** | **0,693** | **1,843** | **0,427** | **0,426** | **Adamax(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)** |
| 3 | 37 | 37 | 1,743 | 0,249 | 1,751 | 0,248 | - | Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 4 | 19 | 19 | 1,743 | 0,249 | 1,752 | 0,248 | - | Adam(lr=1e-3, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |

Tabla 5.6: Resultados de las cuatro mejores configuraciones globales en VGG-Face entrenado de forma completa.

## Prueba 8: Entrenamiento Completo de VGG-Face y VGG-16 con LR de 0,1

En esta prueba se sigue el esquema establecido en la Prueba 6 pero aumentando el valor del LR inicial a 0,1 y sin decay.

Al igual que en las dos pruebas anteriores, la configuración con el optimizador Adadelta y el modelo VGG-Face es la única que consigue superar el 50 % en validación, motivo por el cual se realiza solamente el testeo con esa configuración. La tasa de acierto en el testeo es de un $51,9\,\%$, como puede verse en la Tabla 5.9.

## Prueba 9: Entrenamiento Completo de VGG-Face y VGG-16 con LR de 0,1 y decay

En esta prueba se sigue el esquema establecido en la Prueba 6, incluyendo en las diferentes configuraciones un decay en cada época del valor del LR inicial 0,1 entre el número máximo de épocas prefijadas (50).

Al igual que en las tres pruebas anteriores, la configuración con el optimizador Adadelta y el modelo VGG-Face es la única que consigue superar el 50 % en validación, motivo por el cual se realiza solamente el testeo con esa configuración. La tasa de acierto en el testeo es de un $53,5\,\%$, como puede verse en la Tabla 5.10.

| ID | Modelo | Épocas | | Entrenamiento | | Validación | | Testeo | Optimizador |
|---|---|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| **1** | **VGG-Face** | **9** | **10** | **0,668** | **0,730** | **1,250** | **0,535** | **0,543** | **Adadelta(lr=1e-2, rho=0.95, epsilon=None, decay=0.0)** |
| 2 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 3 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |
| 4 | VGG-16 | 26 | 31 | 1,509 | 0,340 | 1,530 | 0,311 | - | Adadelta(lr=1e-2, rho=0.95, epsilon=None, decay=0.0) |
| 5 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 6 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |

Tabla 5.7: Resultados de los tres mejores configuraciones globales entrenadas de forma completa en VGG-Face y VGG-16 con LR inicial de 0,01.

## Prueba 10: Entrenamiento con LeviHassner

Tal y como se describe en la sección 4.3.2, la inicialización de las capas de este modelo se realiza de forma aleatoria, por lo que en esta prueba se ha entrenado y validado cinco veces la misma configuración.

Se ha utilizado el optimizador SGD con el parámetro *momentum* con valor 0,9, un LR inicial y final de 0,001 y 0,0001 respectivamente, siguiendo los parámetros sugeridos en (Levi & Hassner 2015). También siguiendo estas recomendaciones no se utiliza *EarlyStopping* en estas configuraciones, ya que se desea entrenar las 40 épocas completas, equivalentes a las 10000 iteraciones sugeridas.

Para conseguir reducir el LR durante el entrenamiento se ha utilizado la clase *LearningRateScheduler*, a la que se le ha pasado una función en la que reducimos cada 5 épocas el LR utilizado, llegando al valor final en la época 40. Además, se sigue utilizando el indicador de rendimiento de la tasa de acierto para guardar el mejor modelo de la configuración en validación y se siguen guardando los pesos de la red cada 10 épocas.

De esta forma, como se puede observar en la Tabla 5.11, observamos como los resultados se mantienen constantes si realizamos la media y la desviación típica a los valores con tres decimales. También se observa como los resultados de la tasa de acierto validación se mantienen constantes en el valor 0,248, lo que se podría deber a que el modelo no es capaz de aprender con los valores de entrenamiento sugeridos en Levi & Hassner (2015) con nuestra cantidad de instancias. En las siguientes tres pruebas se realizan modifica-

| ID | Modelo | Épocas | | Entrenamiento | | Validación | | Testeo | Optimizador |
|---|---|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| **1** | **VGG-Face** | **11** | **11** | **0,596** | **0,763** | **1,254** | **0,532** | **0,542** | **Adadelta(lr=1e-2, rho=0.95, epsilon=None, decay=LR/épocas)** |
| 2 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas) |
| 3 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas, amsgrad=False) |
| 4 | VGG-16 | 45 | 50 | 1,484 | 0,357 | 1,531 | 0,322 | - | Adadelta(lr=1e-2, rho=0.95, epsilon=None, decay=LR/épocas) |
| 5 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas) |
| 6 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-2, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas, amsgrad=False) |

Tabla 5.8: Resultados de los tres mejores configuraciones globales entrenadas de forma completa en VGG-Face y VGG-16 con LR inicial de 0,01 y decay=LR/épocas.

ciones tanto al modelo como a los parámetros de entrenamiento para buscar una configuración que ofrezca al menos resultados similares a VGG-Face.

## Prueba 11: Entrenamiento con LeviHassner con Modificación de sus Capas

Debido a los pésimos resultados obtenidos en la anterior prueba y a la lectura en diferentes fuentes bibliográficas como (Khan et al. 2018) que indican que la capa LRN se utiliza cada vez menos en las nuevas arquitecturas debido a que es menos efectiva que otras como la *Batch Normalization* (BN) (Ioffe & Szegedy 2015), que permite normalizar los valores de entrada de las capas intermedias de forma similar a como hacemos con los valores de las imágenes de entrada a la red, o la dropout, se decide realizar la modificación de la LRN por estas capas. Se mantiene la misma configuración utilizada en la Prueba 10, cambiando únicamente las capas LRN por otras.

Cuando se realiza la sustitución en el modelo de las capas LRN por las BN, se observa como los valores de la tasa de acierto y la función objetivo mejoran considerablemente. Es por ello por lo que se repite la misma prueba una segunda vez, obteniendo resultados similares. En estas dos pruebas se observa cómo siempre en la época 40 los resultados mejoran, por lo que se decide comprobar si aumentando a 50 épocas los resultados de la tasa de

| ID | Modelo | Épocas | | Entrenamiento | | Validación | | Testeo | Optimizador |
|---|---|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| **1** | **VGG-Face** | **6** | **9** | **0,523** | **0,795** | **1,711** | **0,522** | **0,519** | **Adadelta(lr=1e-1, rho=0.95, epsilon=None, decay=0.0)** |
| 2 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 3 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |
| 4 | VGG-16 | 17 | 19 | 1,168 | 0,497 | 1,705 | 0,351 | - | Adadelta(lr=1e-1, rho=0.95, epsilon=None, decay=0.0) |
| 5 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) |
| 6 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False) |

Tabla 5.9: Resultados de los tres mejores configuraciones globales entrenadas de forma completa en VGG-Face y VGG-16 con LR inicial de 0,1

acierto mejorarían. Así lo hacen pero solo en dos centésimas más que nuestro mejor configuración con 20 épocas, por lo que se decide tomar como la mejor la configuración con ID 2 en la Tabla 5.12.

También se realizan las mismas pruebas con la capa dropout y también eliminando la capa de LRN del modelo. En estas pruebas se vuelve a mantener las 40 épocas calculadas en la sección 4.3.2.

Como se puede observar en la Tabla 5.13, ninguna de las dos consigue mejorar los pobres resultados obtenidos previamente con el modelo original.

En esta Prueba 11 concluimos que ofrece mejores resultados el modelo LeviHassner sustituyendo las LRN por BN que el modelo sugerido en (Levi & Hassner 2015).

## Prueba 12: Entrenamiento con LeviHassner con Batch Normalization

En esta prueba se utilizan los pesos iniciales del modelo LeviHassner con BN, específicamente el identificado con ID 2 en la Tabla 5.12, se aumenta a 200 el número máximo de épocas prefijado y como es lógico, se vuelve a utilizar el *EarlyStopping* comprobando cada 10 épocas mejoren los resultados.

En las cuatro primeras configuraciones, que pueden observarse en la Tabla 5.14, se prueban los efectos que tiene en el modelo aumentar el LR incial a 0,01 y 0,1 y realizar la reducción del LR cada cinco o diez épocas. En todos

| ID | Modelo | Épocas | | Entrenamiento | | Validación | | Testeo | Optimizador |
|---|---|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| **1** | **VGG-Face** | **6** | **7** | **0,322** | **0,879** | **1,912** | **0,531** | **0,535** | **Adadelta(lr=1e-1, rho=0.95, epsilon=None, decay=LR/épocas)** |
| 2 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas) |
| 3 | VGG-Face | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas, amsgrad=False) |
| 4 | VGG-16 | 29 | 31 | 1,407 | 0,381 | 1,490 | 0,330 | - | Adadelta(lr=1e-1, rho=0.95, epsilon=None, decay=LR/épocas) |
| 5 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adamax(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas) |
| 6 | VGG-16 | 6 | 6 | 12,099 | 0,249 | 12,122 | 0,248 | - | Adam(lr=1e-1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=LR/épocas, amsgrad=False) |

Tabla 5.10: Resultados de los tres mejores configuraciones globales entrenadas de forma completa en VGG-Face y VGG-16 con LR inicial de 0,1 y decay=LR/épocas.

| ID | Modelo | Épocas | | Entrenamiento | | Validación | | Optimizador |
|---|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | |
| 1-5 | LeviHassner | 40 | 40 | $1,744 \pm 4,899e-4$ | $0,250 \pm 0$ | $1,751 \pm 0$ | $0,248 \pm 0$ | SGD(momentum=0.9, nesterov=False) |

Tabla 5.11: Resultados de media y desviación típica para 5 ejecuciones aleatorias

estos casos la disminución en la función de callback *LearningRateSchedule* reduciendo en un 10 % el valor actual de LR.

Como puede observarse en la Tabla 5.14, solamente la quinta configuración consigue superar el 40 % de tasa de acierto en validación, es por ello por lo que se le realiza solo a esta el testeo, obteniendo un 39,4 % de tasa de acierto. Esta última configuración se diferencia de las demás en que utiliza el callback *ReduceLRonPlateau*. Esta función ha sido configurada para que reduzca en un 10 % el valor del LR si han pasado cinco épocas y no ha mejorado la función objetivo en validación, pudiendo llegar al valor mínimo de LR de 0,001.

| ID | Épocas | | | Entrenamiento | | Validación | | Testeo |
|---|---|---|---|---|---|---|---|---|
| | Prefijadas | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto |
| 1 | 40 | 40 | 40 | 0,586 | 0,758 | 2,097 | 0,414 | 0,403 |
| **2** | **40** | **40** | **40** | **0,567** | **0,761** | **2,053** | **0,414** | **0,413** |
| 3 | 50 | 50 | 50 | 0,439 | 0,830 | 2,369 | 0,424 | 0,415 |

Tabla 5.12: Resultados de las tres ejecuciones de LeviHassner con las capas BN sustituyendo las LRN.

| ID | Capa Sustitución LRN | Épocas | | Entrenamiento | | Validación | |
|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto |
| 1 | Dropout | 40 | 40 | 1,744 | 0,250 | 1,765 | 0,248 |
| 2 | Ninguna | 40 | 40 | 1,746 | 0,247 | 1,753 | 0,248 |

Tabla 5.13: Resultados de las tres ejecuciones de LeviHassner con las capas dropout sustituyendo las LRN o eliminándolas.

## Prueba 13: Entrenamiento con LeviHassner con aumento del Learning Rate

En esta prueba se pretende observar si aumentando el LR inicial en el modelo original de LeviHassner (con LRN) se mejoran los resultados. En todos se realiza cada diez épocas una reducción del LR en un 10 %, permitiendo un valor mínimo de 0,0001 utilizando el callback *LearningRateSchedule*.

Como se puede observar en los resultados de la Tabla 5.15, todas obtienen 24,8 % de tasa de acierto en validación, por lo que no se realiza el testeo a ninguna. En el caso de la identificada con ID 2 se mantiene el entrenamiento durante las 200 épocas prefijadas debido a que se produce una ligera mejora en la función objetivo en validación cada diez épocas.

| ID | Learning Rate | | Épocas | | Entrenamiento | | Validación | | Testeo | Callback Modificación LR |
|---|---|---|---|---|---|---|---|---|---|---|
| | Inicial | Decay (nº épocas) | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto | |
| 1 | 0,01 | 10 | 10 | 11 | 1,734 | 0,254 | 1,754 | 0,248 | - | LearningRateSchedule |
| 2 | 0,01 | 5 | 11 | 11 | 1,743 | 0,250 | 1,751 | 0,248 | - | LearningRateSchedule |
| 3 | 0,1 | 10 | 11 | 11 | 13,211 | 0,180 | 13,187 | 0,182 | - | LearningRateSchedule |
| 4 | 0,1 | 5 | 11 | 11 | 12,799 | 0,206 | 12,898 | 0,200 | - | LearningRateSchedule |
| **5** | **0,1** | **5 (si no mejora)** | **20** | **21** | **0,680** | **0,714** | **2,358** | **0,412** | **0,394** | **ReduceLROnPlateau** |

Tabla 5.14: Resultados de las diferentes configuraciones modificando el LR inicial y el decay en LeviHassner con BN.

| ID | LR Inicial | Épocas | | Entrenamiento | | Validación | |
|---|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto |
| 1 | 0,01 | 12 | 12 | 1,743 | 0,250 | 1,751 | 0,248 |
| 2 | 0,1 | 200 | 200 | 1,744 | 0,250 | 1,751 | 0,248 |
| 3 | 1 | 50 | 50 | 12,094 | 0,250 | 12,122 | 0,248 |

Tabla 5.15: Resultados de las diferentes configuraciones aumentando el valor de LR inicial en el modelo original de LeviHassner.

## Prueba 14: Entrenamiento completo y finetunning de VGG-Face con Batch Normalization

En la Prueba 11 se observa como añadiendo al modelo de LeviHassner las capas de BN el rendimiento de las redes mejoraba. Partiendo del mejor modelo de VGG-Face, identificado con el número 1 en la Tabla 5.4, que obtiene en testeo un 0,548 de tasa de acierto, se ha comprobado si añadiendo capas de BN mejoran los resultados tanto con el entrenamiento completo como el de tipo *Fine-tuning*.

Se han probado las siguientes modificaciones del modelo:

- Un bloque (cinco configuraciones).
- dos bloques seguidos (cuatro configuraciones).
- tres bloques seguidos (tres configuraciones).
- cuatro bloques seguidos (dos configuraciones).
- todos los bloques (una configuración).

En todos los casos la capa de BN se añade después de la capa de maxpooling

del bloque convolucional.

| ID | Batch Normalization | | Entrenamiento | Épocas | | Entrenamiento | | Validación | | Testeo |
|---|---|---|---|---|---|---|---|---|---|---|
| | Inicio | Nº Bloques Convolucionales | | Mejor | Última | Pérdida | Tasa de Acierto | Pérdida | Tasa de Acierto | Tasa de Acierto |
| 1 | 2º | 1 | Fine-Tuning | 12 | 12 | 0,530 | 0,791 | 5,477 | 0,512 | 0,520 |
| **2** | **4º** | **1** | **Fine-Tuning** | **12** | **13** | **0,278** | **0,902** | **2,468** | **0,523** | **0,529** |
| 3 | 3ª | 2 | Fine-Tuning | 13 | 13 | 0,285 | 0,897 | 2,441 | 0,500 | 0,502 |

Tabla 5.16: Mejores resultados de las diferentes configuraciones añadiendo a VGG-Face las capas de Batch Normalization.

En la Tabla 5.16 pueden observarse los únicos tres modelos de los treinta probados (15 para cada tipo de entrenamiento) que consiguieron superar el 50 % de tasa de acierto en validación. El mejor resultado lo obtiene el modelo identificado con el ID 2, en el que se añade una única capa de BN en el cuarto bloque convolucional de VGG-Face. Sin embargo, este resultado es peor que el que consigue el modelo original que se utilizó como punto de inicio para esta prueba.

En todas las configuraciones de esta prueba se ha entrenado con un LR inicial de 1,0, con la actualización de dicho valor utilizando la función "ReduceLROnPlateau" si no mejora la función objetivo en validación cada cinco épocas, siendo reducido un 10 %. También se utiliza ejecuta la función de "EarlyStopping" si no mejora cada diez épocas la función objetivo en validación.

## Prueba 15: Uso de Los Mejores Modelos de VGG-Face y LeviHassner como Extractores de Características

En esta prueba vamos a utilizar el mejor modelo de VGG-Face (ID 1 en la Tabla 5.4) y de LeviHassner (ID 2 en la Tabla 5.12) ya entrenados. De las tres capas de tipo fully connected para realizar la clasificación de los modelos, utilizando la función de activación softmax en la última, se obtendrá la salida de la primera en los dos modelos. En Weka se utilizarán los algoritmos de Random Forest y SVM (kernel RBF sin normalización) para clasificar las salidas de LeviHassner en las diferentes clases. Por otro lado, las salidas de VGG-Face se clasifican utilizando Random Forest en Weka y SVM (kernel RBF sin normalización) en Scikit-learn.

En Weka existe solo la posibilidad de establecer un conjunto de entrenamiento y de testeo. Debido a este motivo, se ha añadido a los conjuntos

de entrenamiento las salidas del modelo neuronal al tener como entrada el conjunto de imágenes usado para la validación.

| ID | Modelo | Épocas | | Testeo CNN | Testeo Weka (Tasa de Acierto) | |
|---|---|---|---|---|---|---|
| | | Mejor | Última | Tasa de Acierto | Random Forest | SVM |
| **1** | **VGG-Face** | **10** | **10** | **0,548** | **0,537** | **0,521** |
| 2 | LeviHassner | 40 | 40 | 0,413 | 0,415 | 0,418 |

Tabla 5.17: Resultados del mejor modelo de VGG-Face y LeviHassner utilizando algoritmos de clasificación en Weka.

Como se puede observar en la Tabla 5.16, en el modelo de LeviHassner mejora la tasa de acierto en clasificación ligeramente con respecto a la clasificación realizada en el modelo neuronal. Sin embargo, el mejor clasificador sigue siendo el del propio modelo VGG-Face en la configuración que se muestra en la tabla, empeorándose su tasa de acierto en testeo al utilizarse Random Forest o SVM.

## Prueba 16: Entrenamiento de Los Mejores Modelos de VGG-Face y LeviHassner Utilizando También El Conjunto de Validación

En la anterior prueba comentábamos que se habían unido los conjuntos de entrenamiento y validación debido a que los algoritmos de clasificación de Weka no contaban con validación. Sin embargo, se podría producir una disparidad en las salidas de la primera capa fully connected debido a que algunas instancias ya las conocía (conjunto de entrenamiento) mientras que otras no (conjunto de validación). Es por ello por ello por lo que en esta prueba realizamos los entrenamientos de los dos modelos utilizando las instancias de entrenamiento y validación. Teniendo en cuenta que ya conocemos los parámetros de entrenamiento que mejor resultado aportan tanto en VGG-Face como en LeviHassner, podemos no realizar la validación de estos dos modelos. De esta forma paramos en la mejor época de las anteriores configuraciones el entrenamiento.

De esta forma, como se puede observar en la Tabla 5.18, se consigue mejorar los resultados en el modelo de VGG-Face. Se pasa de un 54,8 % a un

| ID | Modelo | Épocas | | Entrenamiento | | Testeo |
|---|---|---|---|---|---|---|
| | | Mejor | Última | Pérdida | Tasa de Acierto | Tasa de Acierto |
| **1** | **VGG-Face** | **10** | **10** | **0,330** | **0,873** | **0,570** |
| 2 | LeviHassner | 40 | 40 | 0,554 | 0,769 | 0,349 |

Tabla 5.18: Resultados del mejor modelo de VGG-Face y LeviHassner utilizando el conjunto de validación para entrenar.

57 % de la tasa de acierto en testeo al incluir el conjunto de validación en el entrenamiento. Por otro lado, el resultado de LeviHassner empeora, bajando a un 34,9 %.

## Prueba 17: Uso de Los Mejores Modelos de VGG-Face y LeviHassner Entrenados Usando El Conjunto de Validación y Entrenamiento Como Extractores de Características

Utilizando los modelos comentados en la Prueba 16, se guardan las salidas de su primera capa fully connected y se utilizan algoritmos Random Forest y SVM, nuevamente para clasificar las salidas en las diferentes clases. En el caso del SVM en esta prueba utilizamos los kernels RBF y lineal, además la normalización de las salidas de los modelos neuronales por atributo y sin normalizarlos. El algoritmo de Random Forest lo ejecutamos en Weka mientras que para los diferentes SVM utilizamos Scikit-learn.

Como se puede ver en la Tabla 5.19, en el caso de VGG-Face, ninguno de los algoritmos de clasificación consigue mejorar los resultados; mientras que, en el caso de LeviHassner, todos consiguen mejorar la tasa de acierto del clasificador neuronal salvo el SVM con kernel RBF y normalización.

Además, se realizaron las mismas pruebas pero no se separaron las salidas de la primera capa fully connected de cada modelo en el conjunto de entrenamiento y testeo, sino que se fusionaron en un único conjunto. A estos dos conjuntos con los resultados para las 24080, uno por cada modelo, se le

| | | CNN | | | Testeo Weka | Testeo Scikit-learn | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Épocas | | Testeo | Random Forest | SVM (Tasa de Acierto) | | | |
| | | | | | | Con Normalización | | Sin Normalización | |
| ID | Modelo | Mejor | Última | Tasa de Acierto | Tasa de Acierto | Kernel Lineal | Kernel RBF | Kernel Lineal | Kernel RBF |
| **1** | **VGG-Face** | **10** | **10** | **0,570** | **0,542** | **0,479** | **0,243** | **0,447** | **0,526** |
| 2 | LeviHassner | 40 | 40 | 0,349 | 0,423 | 0,391 | 0,243 | 0,383 | 0,423 |

Tabla 5.19: Resultados del mejor modelo de VGG-Face y LeviHassner utilizando el conjunto de validación para entrenar como extractor de características.

aplicó en Weka un PCA con un 95 % de variabilidad. De esta manera, las salidas de VGG-Face pasaron de tener 4096 atributos a 625 mientras que las salidas de LeviHassner pasaron de 512 a 162 atributos.

Una vez aplicados los PCA en los dos conjuntos, se realizaron las mismas que aparecen en la Tabla 5.19 en Weka. Para ello primero el programa desordena aleatoriamente las instancias y las divide en un 66 % para el entrenamiento y lo restante para testeo.

| | | CNN | | | Testeo Weka | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Épocas | | Testeo | Random Forest | SVM (Tasa de Acierto) | | | |
| | | | | | | Con Normalización | | Sin Normalización | |
| ID | Modelo | Mejor | Última | Tasa de Acierto | Tasa de Acierto | Kernel Lineal | Kernel RBF | Kernel Lineal | Kernel RBF |
| 1 | VGG-Face | 10 | 10 | 0,570 | 0,330 | 0,578 | 0,242 | - | 0,468 |
| 2 | **LeviHassner** | **40** | **40** | **0,349** | **0,541** | **0,708** | **0,313** | **0,716** | **0,731** |

Tabla 5.20: Resultados del mejor modelo de VGG-Face y LeviHassner utilizando el conjunto de validación para entrenar y aplicando un PCA del 95 % como extractor de características.

Como se observa en la Tabla 5.20, el kernel lineal con normalización de los datos consigue mejorar ligeramente la tasa de acierto del clasificador neuronal de VGG-Face mientras el resto de clasificadores consiguen peores resultados. No se han añadido los resultados del clasificador lineal sin normalización debido a que su ejecución tuvo que ser interrumpida tras 72 horas, ya que el resto de clasificadores finalizaban antes de tres horas como máximo.

Por otro lado, los clasificadores de las salidas del modelo LeviHassner consiguen mayoritariamente superar no solo los resultados del clasificador neuronal, sino todos los clasificadores evaluados en este proyecto. Siendo el

clasificador SVM sin normalización en la entrada con el kernel RBF el que mejor tasa de acierto obtiene, con un 73,1 %.

Como se puede observar en la Figura 5.21, la matriz de confusión asociada a esta última configuración presenta la mayoría de sus valores en la diagonal principal. La mayoría de falsos positivos se encuentran en las clases adyacentes a la clase real. A medida que la clase real se aleja del resto de las clases, menores son las instancias mal clasificadas. Numéricamente podemos observarlo realizando el cálculo de la tasa de acierto permitiendo *one-off* o *Error of One Category* (AEO), es decir, aceptando como acierto las instancias que se encuentran adyacentemente (salvo los extremos entre sí). Si realizamos este cálculo con esta configuración, obtenemos una tasa de acierto del 95,5 %. Entre estas instancias probablemente se encuentren aquellas cuya edad es cercana al límite entre una clase y otra.

| | | **Predicción** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **[21-30]** | **[31-40]** | **[41-50]** | **[51-60]** | **[61-70]** | **[71-101)** |
| **Clase Real** | **[21-30]** | 1267 | 342 | 60 | 8 | 0 | 0 |
| | **[31-40]** | 155 | 1569 | 207 | 43 | 7 | 1 |
| | **[41-50]** | 21 | 277 | 1068 | 175 | 21 | 2 |
| | **[51-60]** | 9 | 49 | 185 | 772 | 163 | 14 |
| | **[61-70]** | 3 | 11 | 33 | 141 | 615 | 88 |
| | **[71-101)** | 1 | 6 | 15 | 31 | 135 | 693 |

Tabla 5.21: Matriz de confusión de la configuración con mayor tasa de acierto en testeo.

## Prueba 18: Entrenamiento con La Mejor Configuración de Los Modelos VGG-Face y LeviHassner con Las Caras Como Entrada

Esta es la primera prueba del proyecto en el que se utiliza como entrada los rostros de los sujetos y no la imagen completa (cabeza y hombros). Después de recortar el área de la cara en las instancias del conjunto de datos,

tal y como se ha explicado en la sección 4.3.7, se ha utilizado la mejor configuración tanto de VGG-Face como de LeviHassner cuando entrenan con las imágenes completas sumando el conjunto de validación. Los resultados de las pruebas citadas con la imagen completa puede verse en la Tabla 5.18. Los rostros utilizados se encuentran en el mismo conjunto de datos que la imagen completa del que fue extraído.

| **ID** | **Modelo** | **Épocas** | | **Entrenamiento** | | **Testeo** |
|---|---|---|---|---|---|---|
| | | **Mejor** | **Última** | **Pérdida** | **Tasa de Acierto** | **Tasa de Acierto** |
| **1** | **VGG-Face** | **10** | **10** | **0,532** | **0,791** | **0,455** |
| 2 | LeviHassner | 40 | 40 | 0,541 | 0,771 | 0,315 |

Tabla 5.22: Resultados del mejor modelo de VGG-Face y LeviHassner utilizando el conjunto de rostros para entrenar junto al conjunto de validación.

De esta forma obtenemos los resultados de la Tabla 5.22, en los que ninguno de los modelos utilizando los rostros mejoran los resultados ya obtenidos con las imágenes con el patrón cabeza y hombros.

### Prueba 19: Uso de los mejores modelos de VGG-Face y LeviHassner entrenados usando los conjuntos de entrenamiento y validación como extractores de características usando cara y cabeza y hombros

En esta prueba se realiza la evaluación de la fusión de los distintos estimadores. Para ello se utilizan los dos modelos de la Tabla 5.18 y se obtienen las salidas de la primera capa fully connected, tanto con las imágenes completas (cabeza y hombros) como para los rostros recortados.

Una vez se obtienen las diferentes salidas, se une la salida de la red para la imagen completa con la de la cara recortada por un lado en VGG-Face y por otro LeviHassner.

De esta forma se utiliza la misma técnica de extracción de características ya explicada en la segunda parte de la Prueba 17 pero concatenando al

| | | **CNN** | | **Testeo Weka** | | |
|---|---|---|---|---|---|---|
| | | **Épocas** | | **Random Forest** | **SVM (Tasa de Acierto)** | |
| | | | | | **Con Normalización** | **Sin Normalización** |
| ID | **Modelo** | **Mejor** | **Última** | **Tasa de Acierto** | **Kernel Lineal** | **Kernel RBF** |
| 1 | VGG-Face | 10 | 10 | 0,285 | 0,594 | - |
| 2 | **LeviHassner** | **40** | **40** | **0,384** | **-** | **0,727** |

Tabla 5.23: Resultados de los mejores modelos de VGG-Face y LeviHassner con el conjunto de validación utilizado en el entrenamiento como extractor de características y aplicando un PCA del 95 % de variabilidad a la concatenación de cara e imagen completa.

conjunto de datos las salidas para la cara recortada antes de realizar los dos PCA con el 95 % de variabilidad.

Así se obtienen los resultados de la Tabla 5.23, donde la tasa de acierto del modelo LeviHassner con Random Forest decrece de un 54,1 % a un 38,4 % mientras que usando un SVM con kernel RBF sin normalización desciende ligeramente de un 73,1 % hasta un 72,7 %.

Ocurre algo similar al comparar también los resultados de VGG-Face teniendo como entrada la imagen completa frente a la fusión de ambas. En el caso de Random Forest la tasa de acierto desciende de un 33 % hasta un 28,5 % mientras usando un SVM con kernel lineal normalizado se produce una mejora, pasando de un 57,8 % a un 59,4 %.

# Capítulo 6

# Conclusiones

En este trabajo se ha logrado la consecución de los diferentes objetivos planteados en su comienzo. Tal y como se ha descrito en la presente memoria, en primer lugar se ha realizado el estudio de la información contenida en el conjunto de datos AgeDB. Después se ha investigado en la literatura especializada los trabajos recientes relativos a la estimación de edad y se han analizado los métodos aplicados para ello. Una vez se han seleccionado las arquitecturas neuronales VGG-Face y LeviHassner, desde las que se comienza a desarrollar prototipos, y se ha realizado el estudio de las diferentes herramientas de manejo de imágenes, se han implementado y analizado diferentes clasificadores usando tanto las imágenes completas de AgeDB (cabeza y hombros) como los rostros de los individuos.

De esta forma se han cubierto los cinco objetivos planteados inicialmente, necesarios para realizar nuestro objetivo principal, proponer un sistema capaz de clasificar en diferentes rangos de edad aparente imágenes en las que se observe el rostro de una persona. El sistema con mayor tasa de acierto está basado en una modificación de las capas del modelo propuesto por LeviHassner. Esta arquitectura se utiliza como extractor de características, de esta formam a sus salidas se les aplica un PCA y se utiliza una SVM con kernel RBF, obteniendo una tasa de acierto de un 73,1 % y una tasa de acierto aceptando one-off de un 95,5 %.

Este tipo de sistemas aporta en el ámbito académico una propuesta de modelo para la clasificación de edad aparente usando técnicas novedosas basadas en redes profundas. En el ámbito socio-económico podría ser utilizado, por ejemplo, en el ámbito comercial para conocer la edad de los clientes que realizan alguna determinada acción en un entorno controlado por cámaras.

## 6.1 Valoración Personal

He elegido este proyecto debido a que me ofrecía la oportunidad utilizar los conocimientos aprendidos en el grado, principalmente las asignaturas de sistemas inteligentes, y aplicarlos a un sistema que tiene una utilidad real en un entorno académico y comercial, siendo el desarrollo de sistemas inteligentes uno de los campos que más me ha interesado de la titulación.

Tras la finalización de este trabajo me siento capacitado para desarrollar proyectos propios utilizando los conocimientos aprendidos en visión por computador y redes neuronales profundas, así como para investigar nuevos conceptos y arquitecturas que se publiquen en el futuro.

## 6.2 Trabajos Futuros

Entre los posibles trabajos futuros que pueden derivar de este proyecto se encuentra el estudio de otras arquitecturas y conjuntos de datos para realizar la estimación de la edad aparente. También se podrían realizar pruebas iniciales para la estimación de edad aparente usando rostros u otras áreas de interés sin partir de las configuraciones conocidas que obtienen una buena tasa de acierto para cabeza y hombros.

Otra opción sería estudiar si la biometría blanda relacionada con la indumentaria permitiría estimar la edad de los individuos, pudiendo incluso hacer la fusión de esta característica con los propuestos en este trabajo.

# Bibliografía


Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015), 'TensorFlow: Large-scale machine learning on heterogeneous systems'. Software available from tensorflow.org.
**URL:** *https://www.tensorflow.org/*

Abdi, H. & Williams, L. J. (2010), 'Principal component analysis', *Wiley interdisciplinary reviews: computational statistics* **2**(4), 433–459.

Agustsson, E., Timofte, R., Escalera, S., Baro, X., Guyon, I. & Rothe, R. (2017), Apparent and real age estimation in still images with deep residual regressors on appa-real database, *in* 'Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on', IEEE, pp. 87–94.

Antipov, G., Baccouche, M., Berrani, S.-A. & Dugelay, J.-L. (2016), Apparent age estimation from face images combining general and children-specialized deep learning models, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops', pp. 96–104.

Ardila, F. (2009), Predicción de niveles del río magdalena usando sistemas adaptativos basados en conocimiento, Master's thesis, Universidad Distrital Francisco José de Caldas.

Beeman, D. (2001), 'Multi-layer perceptrons (feed-forward nets), gradient descent, and back propagation'.
**URL:** *http://ecee.colorado.edu/ ecen4831/lectures/NNet3.html*

Berry, D., Zebrowitz-MeArthur, L. & Alley, T. (1988), 'Social and applied aspects of perceiving faces'.

Boissieux, L., Kiss, G., Thalmann, N. M. & Kalra, P. (2000), Simulation of skin aging and wrinkles with cosmetics insight, *in* 'Computer Animation and Simulation 2000', Springer, pp. 15–27.

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers, *in* 'Proceedings of the fifth annual workshop on Computational learning theory', ACM, pp. 144–152.

Bottou, L. (2010), Large-scale machine learning with stochastic gradient descent, *in* 'Proceedings of COMPSTAT'2010', Springer, pp. 177–186.

Bradski, G. (2000), 'The OpenCV Library', *Dr. Dobb's Journal of Software Tools* .

Breiman, L. (1996), 'Bagging predictors', *Machine learning* **24**(2), 123–140.

Breiman, L. (1999), 'Random forests', *UC Berkeley TR567* .

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984), 'Classification and regression trees. monterey, calif., usa: Wadsworth'.

Burt, D. M. & Perrett, D. I. (1995), 'Perception of age in adult caucasian male faces: Computer graphic manipulation of shape and colour information', *Proc. R. Soc. Lond. B* **259**(1355), 137–143.

Castrillón-Santana, M. & Lorenzo-Navarro, J. (2017), 'Soft biometric attributes in the wild: Case study on gender classification', *Human Recognition in Unconstrained Environments: Using Computer Vision, Pattern Recognition and Machine Learning Methods for Biometrics* p. 145.

Castrillón-Santana, M., Lorenzo-Navarro, J. & Ramón-Balmaseda, E. (2017), 'Descriptors and regions of interest fusion for in-and cross-database gender classification in the wild', *Image and Vision Computing* **57**, 15–24.

Chen, C. & Ross, A. (2013), Local gradient gabor pattern (lggp) with applications in face recognition, cross-spectral matching, and soft biometrics, *in* 'Biometric and Surveillance Technology for Human and Activity Identification X', Vol. 8712, International Society for Optics and Photonics, p. 87120R.

Chen, J.-C., Patel, V. M. & Chellappa, R. (2016), Unconstrained face verification using deep cnn features, *in* 'Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on', IEEE, pp. 1–9.

Chen, S., Zhang, C. & Dong, M. (2017), 'Deep age estimation: From classification to ranking', *IEEE Transactions on Multimedia* .

Chollet, F. et al. (2015), 'Keras'.

Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Machine learning* **20**(3), 273–297.

Cristiani, N. & Taylor, S. J. (2000), 'An introduction to support vector machines'.

Das, M. & Loui, A. C. (2003), Automatic face-based image grouping for albuming, *in* 'Systems, Man and Cybernetics, 2003. IEEE International Conference on', Vol. 4, IEEE, pp. 3726–3731.

Dima, C. (2016), 'Basics of support vector machines'.
**URL:** *http://www.cristiandima.com/basics-of-support-vector-machines/*

Dozat, T. (2016), 'Incorporating nesterov momentum into adam'.

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive subgradient methods for online learning and stochastic optimization', *Journal of Machine Learning Research* **12**(Jul), 2121–2159.

Edwards, G. J., Cootes, T. F. & Taylor, C. J. (1998), Face recognition using active appearance models, *in* 'European conference on computer vision', Springer, pp. 581–595.

Escalera, S., Fabian, J., Pardo, P., Baró, X., Gonzalez, J., Escalante, H. J., Misevic, D., Steiner, U. & Guyon, I. (2015), Chalearn looking at people 2015: Apparent age and cultural event recognition datasets and results, *in* 'Proceedings of the IEEE International Conference on Computer Vision Workshops', pp. 1–9.

Escalera, S., Torres Torres, M., Martinez, B., Baró, X., Jair Escalante, H., Guyon, I., Tzimiropoulos, G., Corneou, C., Oliu, M., Ali Bagheri, M. et al. (2016), Chalearn looking at people and faces of the world: Face analysis workshop and challenge 2016, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops', pp. 1–8.

Feller, W. (1968), *An introduction to probability theory and its applications*, Vol. 1, Wiley, New York.

Freund, Y. & Schapire, R. E. (1999), 'Large margin classification using the perceptron algorithm', *Machine learning* **37**(3), 277–296.

Fu, Y., Guo, G. & Huang, T. S. (2010), 'Age synthesis and estimation via faces: A survey', *IEEE transactions on pattern analysis and machine intelligence* **32**(11), 1955–1976.

Fu, Y. & Huang, T. S. (2008), 'Human age estimation with regression on discriminative aging manifold', *IEEE Transactions on Multimedia* **10**(4), 578–584.

Fukushima, K. & Miyake, S. (1982), Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, *in* 'Competition and cooperation in neural nets', Springer, pp. 267–285.

Gopalakrishnan, K., Khaitan, S. K., Choudhary, A. & Agrawal, A. (2017), 'Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection', *Construction and Building Materials* **157**, 322–330.

Gulli, A. & Pal, S. (2017), *Deep Learning with Keras*, Packt Publishing Ltd.

Guo, G., Fu, Y., Dyer, C. R. & Huang, T. S. (2008), 'Image-based human age estimation by manifold learning and locally adjusted robust regression', *IEEE Transactions on Image Processing* **17**(7), 1178–1188.

Guo, G., Fu, Y., Huang, T. S. & Dyer, C. R. (2008), Locally adjusted robust regression for human age estimation, *in* 'Applications of Computer Vision, 2008. WACV 2008. IEEE Workshop on', IEEE, pp. 1–6.

Guo, G., Mu, G., Fu, Y. & Huang, T. S. (2009), Human age estimation using bio-inspired features, *in* 'Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on', IEEE, pp. 112–119.

Han, H., Otto, C. & Jain, A. K. (2013), Age estimation from face images: Human vs. machine performance, *in* 'Biometrics (ICB), 2013 International Conference on', IEEE, pp. 1–8.

Han, H., Otto, C., Liu, X. & Jain, A. K. (2015), 'Demographic estimation from face images: Human vs. machine performance', *IEEE transactions on pattern analysis and machine intelligence* **37**(6), 1148–1161.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Hecht-Nielsen, R. (1992), Theory of the backpropagation neural network, *in* 'Neural networks for perception', Elsevier, pp. 65–93.

Hinton, G., Srivastava, N. & Swersky, K. (2012), 'Neural networks for machine learning lecture 6a overview of mini-batch gradient descent'.

Holczer, B. (2018), 'Random forest classifier – machine learning'.
**URL:** *http://www.globalsoftwaresupport.com/random-forest-classifier-bagging-machine-learning/*

Hu, Z., Wen, Y., Wang, J., Wang, M., Hong, R. & Yan, S. (2017), 'Facial age estimation with age difference', *IEEE Transactions on Image Processing* **26**(7), 3087–3097.

Huang, G. B., Ramesh, M., Berg, T. & Learned-Miller, E. (2007), Labeled faces in the wild: A database for studying face recognition in unconstrained environments, Technical report, Technical Report 07-49, University of Massachusetts, Amherst.

Huang, G., Liu, Z., Weinberger, K. Q. & van der Maaten, L. (2017), Densely connected convolutional networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', Vol. 1, p. 3.

Huang, T. (1996), 'Computer vision: Evolution and promise'.

Hubel, D. H. & Wiesel, T. N. (1959), 'Receptive fields of single neurones in the cat's striate cortex', *The Journal of physiology* **148**(3), 574–591.

Ioffe, S. & Szegedy, C. (2015), 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *arXiv preprint arXiv:1502.03167* .

Jain, A. K., Dass, S. C. & Nandakumar, K. (2004), Soft biometric traits for personal recognition systems, *in* 'Biometric Authentication', Springer, pp. 731–738.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. & Darrell, T. (2014), 'Caffe: Convolutional architecture for fast feature embedding', *arXiv preprint arXiv:1408.5093* .

Jiang, W., He, G., Long, T., Ni, Y., Liu, H., Peng, Y., Lv, K. & Wang, G. (2018), 'Multilayer perceptron neural network for surface water extraction in landsat 8 oli satellite images', *Remote Sensing* **10**.
**URL:** *http://www.mdpi.com/2072-4292/10/5/755*

Karpathy, A. (2015), 'Convolutional neural networks (cnns / convnets)'.
**URL:** *https://cs231n.github.io/convolutional-networks/*

Khan, S., Rahmani, H., Shah, S. A. A. & Bennamoun, M. (2018), 'A guide to convolutional neural networks for computer vision', *Synthesis Lectures on Computer Vision* **8**(1), 1–207.

King, D. E. (2009), 'Dlib-ml: A machine learning toolkit', *Journal of Machine Learning Research* **10**, 1755–1758.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

Kuang, Z., Huang, C. & Zhang, W. (2015), Deeply learned rich coding for cross-dataset facial age estimation, *in* 'Proceedings of the IEEE International Conference on Computer Vision Workshops', pp. 96–101.

Kwon, Y. H. et al. (1994), Age classification from facial images, *in* 'Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on', IEEE, pp. 762–767.

Levi, G. & Hassner, T. (2015), Age and gender classification using convolutional neural networks, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops', pp. 34–42.

Li, F., Tran, L., Thung, K.-H., Ji, S., Shen, D. & Li, J. (2015), 'A robust deep model for improved classification of ad/mci patients', *IEEE journal of biomedical and health informatics* **19**(5), 1610–1616.

Li, Y., Liu, Z., Xu, K., Yy, H. & Ren, F. (To Appear), 'A gpu-outperforming fpga accelerator architecture for binary convolutional neural networks', *ACM Journal on Emerging Technologies in Computing (JETC)* .

Lior, R. et al. (2014), *Data mining with decision trees: theory and applications*, Vol. 81, World scientific.

Liu, H., Lu, J., Feng, J. & Zhou, J. (2017), 'Group-aware deep feature learning for facial age estimation', *Pattern Recognition* **66**, 82–94.

Liu, H., Lu, J., Feng, J. & Zhou, J. (2018), 'Label-sensitive deep metric learning for facial age estimation', *IEEE Transactions on Information Forensics and Security* **13**(2), 292–305.

Liu, X., Li, S., Kan, M., Zhang, J., Wu, S., Liu, W., Han, H., Shan, S. & Chen, X. (2015), Agenet: Deeply learned regressor and classifier for robust apparent age estimation, *in* 'Proceedings of the IEEE International Conference on Computer Vision Workshops', pp. 16–24.

López, R. F. & Fernández, J. M. F. (2008), *Las redes neuronales artificiales*, Netbiblo.

Luu, K., Seshadri, K., Savvides, M., Bui, T. D. & Suen, C. Y. (2011), Contourlet appearance model for facial age estimation, *in* 'Biometrics (ijcb), 2011 international joint conference on', IEEE, pp. 1–8.

Malli, R. C. (2016), 'Vggface implementation with keras framework'.
**URL:** *https://github.com/rcmalli/keras-vggface*

Marr, D. (1982), *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, Henry Holt and Co., Inc., New York, NY, USA.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

Mingers, J. (1989), 'An empirical comparison of pruning methods for decision tree induction', *Machine learning* **4**(2), 227–243.

Moschoglou, S., Papaioannou, A., Sagonas, C., Deng, J., Kotsia, I. & Zafeiriou, S. (2017), Agedb: The first manually collected in-the-wild age database, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop', Vol. 2, p. 5.

Nair, V. & Hinton, G. E. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of the 27th international conference on machine learning (ICML-10)', pp. 807–814.

Nasrabadi, N. M. (2007), 'Pattern recognition and machine learning', *Journal of electronic imaging* **16**(4), 049901.

Ni, B., Song, Z. & Yan, S. (2011), 'Web image and video mining towards universal and robust age estimator', *IEEE Transactions on Multimedia* **13**(6), 1217–1229.

Niu, Z., Zhou, M., Wang, L., Gao, X. & Hua, G. (2016), Ordinal regression with multiple output cnn for age estimation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 4920–4928.

Ojala, T., Pietikainen, M. & Maenpaa, T. (2002), 'Multiresolution gray-scale and rotation invariant texture classification with local binary patterns', *IEEE Transactions on pattern analysis and machine intelligence* **24**(7), 971–987.

Pal, M. & Mather, P. M. (2003), 'An assessment of the effectiveness of decision tree methods for land cover classification', *Remote sensing of environment* **86**(4), 554–565.

Parkhi, O. M., Vedaldi, A., Zisserman, A. et al. (2015), Deep face recognition., *in* 'BMVC', Vol. 1, p. 6.

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

Quinlan, J. R. (2014), *C4. 5: programs for machine learning*, Elsevier.

Ramanathan, N., Chellappa, R. & Biswas, S. (2009), 'Computational methods for modeling facial aging: A survey', *Journal of Visual Languages & Computing* **20**(3), 131–144.

Ranjan, R., Zhou, S., Cheng Chen, J., Kumar, A., Alavi, A., Patel, V. M. & Chellappa, R. (2015), Unconstrained age estimation with deep convolutional neural networks, *in* 'proceedings of the ieee international conference on computer vision workshops', pp. 109–117.

Reddi, S. J., Kale, S. & Kumar, S. (2018), On the convergence of adam and beyond, *in* 'International Conference on Learning Representations'.

Ricanek, K. & Tesafaye, T. (2006), Morph: A longitudinal image database of normal adult age-progression, *in* 'Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on', IEEE, pp. 341–345.

Robers, L. (1963), Machine Perception of 3-D Solids, PhD thesis, Ph. D. Thesis, MIT.

Robers, L. (1965), 'Machine perception of three-dimensional solids', *Optical and Electro-Optical Information Processing* .

Rosebrock, A. (2017), 'Facial landmarks with dlib, opencv, and python'. **URL:** *https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/*

Rothe, R., Timofte, R. & Van Gool, L. (2016), 'Deep expectation of real and apparent age from a single image without facial landmarks'.

Rothe, R., Timofte, R. & Van Gool, L. (2018), 'Deep expectation of real and apparent age from a single image without facial landmarks', *International Journal of Computer Vision* **126**(2-4), 144–157.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al. (2015), 'Imagenet large scale visual recognition challenge', *International Journal of Computer Vision* **115**(3), 211–252.

Russell, S. J. & Norvig, P. (2004), *Inteligencia Artificial: un enfoque moderno*, number 04; Q335, R8y 2004.

Simonyan, K. & Zisserman, A. (2015), 'Very deep convolutional networks for large-scale image recognition. arxiv prepr arxiv14091556'.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: A simple way to prevent neural networks from overfitting', *The Journal of Machine Learning Research* **15**(1), 1929–1958.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. et al. (2015), Going deeper with convolutions, Cvpr.

Szeliski, R. (2010), *Computer vision: algorithms and applications*, Springer Science & Business Media.

Uricár, M., Timofte, R., Rothe, R., Matas, J. & Van Gool, L. (2016), Structured output svm prediction of apparent age, gender and smile from deep features, *in* 'Proceedings CVPRW 2016', pp. 25–33.

Vapnik, V. (2013), *The nature of statistical learning theory*, Springer science & business media.

Walia, A. S. (2017), 'Types of optimization algorithms used in neural networks and ways to optimize gradient descent'.
**URL:** *https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f*

Walt, S. v. d., Colbert, S. C. & Varoquaux, G. (2011), 'The numpy array: a structure for efficient numerical computation', *Computing in Science & Engineering* **13**(2), 22–30.

Xing, J., Li, K., Hu, W., Yuan, C. & Ling, H. (2017), 'Diagnosing deep learning models for high accuracy age estimation from a single image', *Pattern Recognition* **66**, 106–116.

Yang, M., Zhu, S., Lv, F. & Yu, K. (2011), Correspondence driven adaptation for human profile recognition, *in* 'Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on', IEEE, pp. 505–512.

Yi, D., Lei, Z. & Li, S. Z. (2014), Age estimation by multi-scale convolutional network, *in* 'Asian Conference on Computer Vision', Springer, pp. 144–158.

Zeiler, M. D. (2012), 'Adadelta: an adaptive learning rate method', *arXiv preprint arXiv:1212.5701* .

Zulkifli, H. (2018), 'Understanding learning rates and how it improves performance in deep learning'.
**URL:** *https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10*