Algoritmo de Pre-Procesamiento de Imágenes para la Estimación de la Edad

– –

Image Pre-Processing Algorithm for Age Estimation

TRABAJO FIN DE GRADO
DOBLE GRADO DE MATEMÁTICA E INFORMÁTICA
CURSO 2020–2021

Miguel Franqueira Varela

Directores
Luis Javier García Villalba
Esteban Alejandro Armas Vega

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2021

Agradecimientos

Gracias a los directores por su dedicación, a la UCM por permitirme la realización de varios cursos gratis. A Irene por el apoyo constante, permitiéndome estar de buen humor incluso en los malos días. A mis compañeros de titulación, sobre todo, personas como Jorge que ayuda en todo lo que puede y es de valorar. Por último agradecer a mis amigos y familiares por hacerme ser quien soy.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Acrónimos

AAM	Active Appearance Model
AGES	Aging Pattern Subspace
AI	*Artificial Intelligence*
ANNs	*Artificial Neural Networks*
API	Application Programming Interface
BCE	Binary Cross Entropy
CA	*Cluster analysis*
CE	Cross Entropy
CNNs	*Convolutional Neural Networks*
CPU	Central Processing Unit
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
CV	*Computer Vision*
DA	*Data Analysis*
DL	*Deep Learning*
DNNs	*Deep Neural Networks*
DT	Decision Trees

GMM Gaussian Mixture Modelling

GPR Gaussian Process Regression

GPU Graphics Proccessing Unit

HL Hinge Loss

k-NN k-Nearest Neighbors

LBP Local Binary Patterns

LDA Linear Discriminant Analysis

LSTM *Long Short Term Memory*

MAE Mean Absolute Error

ML *Machine Learning*

MSE Mean Squared Error

PCA Principal Componnent Analysis

ReLU Rectified Linear Unit

RF Random Forests

RL *Reinforcement Learning*

RNNs *Recurrent Neural Networks*

SFP Spatially Flexible Patch

SGD Stochastic Gradient Descent

SL *Supervised Learning*

SSD *Single Shot Detector*

SVM Support Vector Machines

SVR Support Vector Regression

SWL *Swallow Learning*

TL Transfer Learning

TPU Tensor Processing Unit

UL *Unsupervised Learning*

Abstract

Nowadays, neural networks are currently being used to develop many intelligent applications. Facial recognition systems have gained great importance in society in recent years, being incorporated into people's daily lives, for example, to unlock cell phones. However, there are many other tasks to be solved, one of the most important ones is the age recognition using an image. The more important issue encountered to carry out this task is the small number of databases dedicated to the subject and the poor quality of the images in the existing ones.w Therefore, in this work we try to create different models and implement a pre-processing of the images, so that the training of the models is performed more accurately. To achieve this goal, extensive research has been carried out on the concepts encompassing the field of age estimation, on previous work related to the subject as well as the most widely used and best performing techniques used to preprocess images. Finally, the knowledge acquired has been tested by means of different experiments to measure the influence of image pre-processing on the different models created.

Key words: Artificial Intelligence, Deep Learning, Neural Networks, Preprocessing, Images, Estimation, Recognition, Age.

Resumen

Hoy en día, el campo de las redes neuronales se encuentra en plena expansión y son muchos los avances que se realizan. Los sistemas de reconocimiento facial han ganado un gran peso en la sociedad en los últimos años, incorporándose en el día a día de las personas, por ejemplo, para desbloquear el teléfono móvil. Sin embargo, existen otras muchas tareas por realizar, siendo una de las más importantes el reconocimiento de la edad a través de una imagen. El principal problema encontrado para realizar esta tarea es la poca cantidad de bases de datos dedicadas al tema y la poca calidad de las imágenes en las existentes. En este trabajo se intenta crear diferentes modelos y realizar un pre-procesamiento de las imágenes, para que el entrenamiento de los modelos se realice de manera más precisa. Para cumplir este objetivo, se ha investigado acerca de los conceptos relativos al campo de la estimación de la edad en los trabajos previos relacionados con el tema y en las técnicas más utilizadas y con mejores resultados a la hora de preprocesar imágenes. Finalmente se han probado los conocimientos adquiridos mediante diferentes experimentos para medir la influencia del pre-procesamiento de imágenes en los diferentes modelos creados.

Palabras clave: Inteligencia Artificial, Aprendizaje Profundo, Redes Neuronales, Preprocesamiento, Imágenes, Estimación, Reconocimiento, Edad.

Capítulo 1

Introducción

1.1. Motivación

Desde el comienzo de la informática en la segunda mitad del siglo XX, ha tenido un crecimiento exponencial que la ha llevado a ser una de las ramas científicas más en auge en la actualidad. Desde los primeros sistemas informáticos, su objetivo ha estado claro: recibir unos datos de entrada que se quieren procesar y devolver una salida que contenga la información pedida de los datos. Los algoritmos han jugado un papel clave en el desarrollo de la humanidad y el surgimiento de la programación y los lenguajes de programación le han dado un impulso notable a nuestra sociedad actual.

Numerosas tareas de hoy en día serían irrealizables de no ser por la informática, los sistemas informáticos y los algoritmos. La informática está involucrada en la vida de todas y cada una de las personas y mejora la calidad de vida, permitiendo ver a los familiares con los que no se convive en época de pandemia sin ningún riesgo, enterarse de las noticias al momento y un sinfín de cosas.

Después de décadas de desarrollo de la informática surge la rama de la inteligencia artificial (del inglés *Artificial Intelligence* (AI)) que cambia completamente el paradigma de programación clásico y permite realizar tareas que con la computación clásica serían irrealizables. En concreto, en el año 1958 aparecen las llamadas redes neuronales que surgen con la intención de imitar la inteligencia humana. Desde entonces empezaron a crecer a gran velocidad impulsadas por el avance del hardware, permitiendo crear máquinas más avanzadas que soportan sistemas más complejos y tienen mucho más poder de cómputo. La gran ventaja de las redes neuronales es que permiten realizar una tarea muy costosa computacionalmente de manera muy rápida una vez que las entrenas previamente para realizar esa tarea.

Por otro lado en el ámbito audiovisual, hasta no hace tanto, se requería una cámara especializada para poder grabar un vídeo y para su reproducción. Además, la calidad de

las imágenes y vídeos obtenidos era mucho menor que la actual. Sin embargo, a la vez que avanzaba la tecnología y con ello la informática se empezó a mejorar la calidad de las cámaras fotográficas. Con la creación de las cámaras digitales se dio un primer paso a la representación de las fotos en un sistema informático. Ya en 1998, se creó el famoso código RGB (siglas en inglés de *red, green, blue*) el cual permite representar cada píxel de una imagen como una composición del color en términos de la intensidad de los colores primarios de la luz (rojo, verde y azul). La representación de las fotografías, y por tanto, de la realidad y la visión, permite afrontar una multitud de problemas hasta entonces imposibles.

Uno de los problemas más útiles relacionados con la identificación de objetos es el reconocimiento facial, dando lugar a numerosas aplicaciones. Este sistema sirve, tanto para ayudar en la identificación en aeropuertos, como para facilitar las investigaciones policiales, siendo incluso utilizado para detectar síntomas de enfermedades y dar un diagnóstico médico.

Otro problema que ha surgido en los últimos años, dado el éxito de los sistemas de reconocimiento facial, es la identificación de la edad. La mayoría de estos métodos se basan en la cara de una persona, por lo que es necesario utilizar primero un sistema de reconocimiento facial en la imagen para ubicar la cara. En el mundo actual que está tan digitalizado, existen numerosas fuentes de pruebas fotográficas que son utilizadas habitualmente por las fuerzas policiales para identificar a los sospechosos y a las víctimas de delitos tanto en línea como fuera de ella. Las características humanas, como la edad, la altura, el peso, el sexo y el color del pelo suelen ser utilizadas por los agentes de policía y los testigos en su descripción de los sospechosos no identificados. En determinadas circunstancias, la edad de la víctima puede determinar la clasificación del delito, por ejemplo, en las investigaciones de abusos a menores. La estimación de la edad también puede utilizarse para el acceso a locales de ocio para adultos, la compra de productos con restricciones de edad, como el alcohol y el tabaco, la publicidad dirigida a la edad y, recientemente, servicios como *how-old.net* se han se han utilizado para el reconocimiento de los niños refugiados de Siria.

Por ello, se propone abordar este problema que es bastante actual, con el objetivo de profundizar el conocimiento sobre el tema, haciendo un estudio a fondo del estado del arte y de las técnicas utilizadas para resolver el problema, incidiendo en los modelos basados en redes neuronales. Una vez entendido eso, se propone hacer pruebas que permitan ver como se comportan varios modelos y como influyen los datos de entrada a la hora de entrenar el modelo. Esta búsqueda de conocimiento sobre un tema tan útil e interesante es el motor que impulsa la realización de este trabajo.

1.2. Contexto

Este Trabajo Fin de Grado ha sido realizado dentro del Grupo de Análisis, Seguridad y Sistemas (Grupo GASS, https://gass.ucm.es/, Grupo 910623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación THEIA (Techniques for Integrity and Authentication of Multimedia Files of Mobile Devices) con referencia FEI-EU-19-04.

1.3. Objeto de la Investigación

La visión humana permite reconocer muchas características presentes en las imágenes como puede ser el color, la forma y la textura de los objetos. Reconocer las múltiples características permite, de manera conjunta, identificar los diferentes objetos de una imagen y por tanto extraer conclusiones como saber los animales que aparecen en la misma o incluso saber el número de rostros que hay en ella. Sin embargo, con el avance de la tecnología y, sobre todo, el avance de los sistemas de AI, existe la posibilidad de entrenarlos para realizar una identificación de las características de las imágenes, llegando a reconocer detalles no perceptibles al ojo humano. Estos pequeños detalles son de vital importancia cuando se está tratando de realizar la identificación de algún objeto o alguna tarea basada en el análisis de una imagen.

El principal objetivo de este trabajo es probar distintas técnicas de pre-procesamiento midiendo la efectividad de las mismas en el proceso de entrenamiento de diferentes modelos basados en Redes Neuronales Convolucionales (del inglés *Convolutional Neural Networks* (CNNs)) que realizan una estimación de la edad en imágenes.

1.4. Plan de Trabajo

La realización de este proyecto se divide en cuatro fases:

1. **Investigación**: Al comienzo, se llevó a cabo un proceso de aprendizaje para adquirir los conocimientos necesarios para comenzar con el posterior desarrollo. Para adquirir conocimientos específicos sobre el Aprendizaje Automático (del inglés *Machine Learning* (ML)), sobre la librería de *Python* llamada *Tensorflow* y sobre el tratamiento de imágenes, se realizaron 8 cursos diferentes sumando un total de 115 horas de formación. Estos cursos se realizaron en la página web de *Coursera* gracias a la ayuda de la Universidad Complutense de Madrid, ya que se tiene un convenio que proporciona acceso a muchísimos cursos de manera gratuita. Además, gracias al trabajo de los directores de este trabajo se aprendieron diversas plataformas y

herramientas como *Google Scholar*, la cual se usaría posteriormente para buscar artículos científicos en donde se investiga acerca de la estimación de la edad. Una vez entendido las diferentes maneras de tratar el estudio y habiendo concluido cómo se quería abordar, se comenzó el desarrollo de la propuesta.

2. **Desarrollo**: Una vez se entendió cómo se quería abordar el proyecto, se comenzó a programar. Sin embargo, la fase de investigación no se cesó, pero si que dejó de ser el foco principal del trabajo. Durante esta fase se aplicaron los conceptos sobre el lenguaje de programación *Python* y de librerías como *Tensorflow* y *Pandas* aprendidos en los cursos realizados en la fase anterior. Durante esta fase también se procesaron diversas bases de datos para entrenar el modelo, encontradas gracias a los artículos científicos investigados en la etapa anterior.

3. **Experimentación**: En esta fase se evaluaron los prototipos desarrollados y se analizaron los resultados obtenidos. Estos resultados hicieron necesario la modificación y el desarrollo de nuevos modelos, por lo que esta fase transcurrió de manera paralela a las etapas de investigación y desarrollo. Una vez vistos los primeros resultados y una vez se crearon nuevos modelos, también se comenzó a experimentar el pre-procesamiento de las imágenes y se analizó su influencia en la mejora del proceso de entrenamiento.

4. **Documentación**: Una vez comenzado el desarrollo, se comenzó la fase de escritura y revisión de la memoria final del proyecto. Esta etapa comenzó con la escritura de los conocimientos teóricos necesarios para entender el proyecto y con la recopilación de los artículos del campo de estudio para su explicación. Una vez desarrollado y experimentado los modelos, se realizó una escritura iterativa, buscando erratas o posibles apartados que mejorar.

La Tabla 1.1 muestra las diferentes actividades realizadas para desarrollar el proyecto.

Tabla 1.1: Actividades del Plan de Trabajo

Mes	**Actividad**
Octubre	Aprendizaje de conceptos, herramientas e investigación de artículos académicos.
Noviembre	
Diciembre	
Enero	Creación de los modelos más básicos y procesamiento básico de las bases de datos.
Febrero	
Marzo	- Experimentación y optimización de los modelos, preprocesamiento - Realización de la memoria de trabajo.
Abril	
Mayo	
Junio	

1.5. Estructura del Trabajo

El resto del trabajo está organizado en 8 capítulos:

En el Capítulo 2 se explican los conceptos básicos de la AI, para luego centrarse en algoritmos de clasificación que puedan ser utilizados a la hora de tratar la estimación de la edad a partir de rostros.

En el Capítulo 3 se explica de manera detallada el funcionamiento de una Red Neuronal Artificial, explicando cada una de sus componentes. También se explican algunos de sus tipos y cómo se realiza el entrenamiento de las mismas.

En el Capítulo 4 se muestran los distintos modelos y técnicas propuestos para realizar la estimación de la edad en imágenes. Antes de explicar las técnicas que se utilizan en los diversos trabajos, se muestran algunos de los factores que influyen en el reconocimiento de la edad. Los trabajos luego se dividen en función del tipo de técnica que utilizan para crear el modelo, si se basan en extraer características para luego pasarlas por un clasificador o si se basan en Aprendizaje Profundo (del inglés *Deep Learning* (DL)). Por último, se realiza una comparación de varios de los modelos previamente explicados en el capítulo.

En el Capítulo 5 se describen las técnicas de pre-procesamiento utilizadas para mejorar la calidad de las imágenes para entrenar nuestros modelos. A su vez, se explican los modelos utilizados para probar la eficacia de dichas técnicas. Por último, se hace una breve explicación de las tecnologías utilizadas para poder realizar tanto los modelos como el pre-procesamiento de las imágenes.

En el Capítulo 6 se detallan los experimentos realizados para evaluar la efectividad de los diferentes modelos creados, a su vez detalla la creación de estos modelos y como surgen los modelos explicados en el Capítulo 5 en función de los experimentos con los modelos anteriores. También se describen los experimentos con las diferentes técnicas de pre-procesamiento de imágenes para mejorar la calidad de las mismas y así conseguir un mejor entrenamiento de la Red Neuronal. Todos estos experimentos están acompañadas con gráficos donde se muestran los resultados de los experimentos, terminando con una sección comparativa con los diferentes modelos realizados.

En el Capítulo 7 se muestran las principales conclusiones de este proyecto, así como diferentes mejoras posibles sobre el trabajo realizado para investigar más acerca de esta temática.

Por último, en los Capítulos 8 y 9 se muestran las traducciones al inglés de la Introducción y las Conclusiones.

Capítulo 2

Algoritmos de clasificación

En este capítulo se describen los conceptos y técnicas relativas al ML, centrándose en la explicación de los algoritmos de clasificación. Se explican conceptos previos como inteligencia artificial y de aprendizaje automático en las Secciones 2.1 y 2.2. Finalmente, en la Sección 2.3 se presentan las diferentes técnicas utilizadas para el aprendizaje automático y se describen los diferentes algoritmos de clasificación.

2.1. Inteligencia Artificial

La inteligencia es una facultad de los seres humanos que permite aprender, entender, razonar y tomar decisiones ante la realidad. El concepto de AI surge de la pretensión de dotar a las máquinas con esa misma capacidad, permitiendo resolver problemas que no tienen una fácil solución algorítmica [RN09]. Esta es la definición dada en la AI moderna, pero no es la única definición que se ha dado a lo largo de la historia. Alan Touring, el considerado como precursor de la AI, formuló su famoso *Test de Touring* en 1950, en el cual se pretendía comprobar si un ordenador es inteligente o no, utilizando conversaciones en lenguaje natural entre un humano y una máquina [Tur50]. En este test solo se mide el comportamiento inteligente, pero no cómo se ha llegado a él. En la práctica esto es realmente lo que nos interesa medir, ya que nos interesa que la máquina se comporte de manera inteligente y no que realmente sea inteligente.

El propósito de la AI es analizar los mecanismos que dan lugar a conductas inteligentes para reproducir dichas conductas en máquinas. Estas conductas inteligentes necesitan diferentes capacidades de percepción, razonamiento y aprendizaje. En general, este capítulo se centrará más en la capacidad de aprendizaje con el ML.

Otro aspecto importante de la AI es que está en constante desarrollo y cada año permite resolver problemas que anteriormente pertenecían a la ciencia ficción. Del mismo modo, la informática clásica también avanza, haciendo que problemas antes considerados

de AI dejen de serlo, ya que se pueden resolver con programación tradicional. Un ejemplo de problema que en el pasado pertenecía a la ciencia ficción es el del primer ordenador que ganó a un campeón mundial de ajedrez en 1997, donde el ordenador *Deep Blue* de IBM ganó al campeón del mundo de ajedrez Gary Kasparov, tras un encuentro de 6 partidas [FH99]. Posteriormente, en 2015, el programa *AlphaGo* de Google ganó a un campeón mundial de Go [Che16], juego conocido por su complejidad y factor de ramificación, haciendo imposible resolver el problema mediante programación tradicional.

En la actualidad, existen muchos campos de la AI que están en plena investigación como el *Big Data*, utilizado para recoger una cantidad inmensa de datos para procesarlos y extraer conocimiento de ellos. En general, es una rama de la informática que está en plena expansión y cada vez las empresas dedican más dinero a su implementación en sus diferentes sistemas. A parte de los ejemplos citados, existen aplicaciones mucho más prácticas como el reconocimiento facial, el procesamiento del lenguaje natural y un sinfín de aplicaciones que han llevado a la AI a ser una de las áreas más importantes y influyentes de la informática en el siglo XXI.

2.2. Aprendizaje Automático

El ML es una subrama de la AI la cual consiste en la aplicación de técnicas y algoritmos capaces de aprender a partir de distintas fuentes de información, construyendo algoritmos que mejoren de forma autónoma con la experiencia [Gon17]. Es muy importante destacar la idea de que los algoritmos aprenden de forma autónoma, ya que no es necesario intervenir directamente en este aprendizaje. Siguiendo la idea del *test de Touring*, lo importante es centrarse en si las computadoras pueden hacer lo que los seres humanos podemos hacer y no en si realmente pueden pensar. Sin entrar en detalle de en cómo se realiza esta tarea, la idea es que el ordenador aprenda automáticamente a través de la experiencia. Para ello, habrá que tener un *modelo* que sea *entrenado* para realizar cierta predicción o tarea. Para realizar este entrenamiento al modelo se le pasan datos, conocidos como datos de entrenamiento.

Para que el modelo funcione correctamente es necesario que los datos sean fiables y haya gran cantidad de ellos. Cuanto mejor sea la calidad de los datos, mayor será la precisión del modelo a la hora de realizar su tarea o predicción. Por ello, una de las fases más importantes de cualquier proyecto de ML es la del análisis de datos (del inglés *Data Analysis* (DA)) [MRB+18], que normalmente suele llevar más tiempo que la propia elaboración del modelo.

Existen numerosas aproximaciones que se pueden utilizar para abordar un problema de ML, pero normalmente se dividen en los siguientes paradigmas básicos [Ayo10].

- **Aprendizaje Supervisado**: El Aprendizaje Supervisado (del inglés *Supervised Learning* (SL)), agrupa aquellos algoritmos en la que los datos de entrada contienen la salida esperada. Estos datos son etiquetados normalmente por el humano y el objetivo es aprender una regla general que, dada una entrada, le asigne una salida. Esta regla general se consigue construyendo un modelo matemático que se entrena con unos ciertos datos de entrenamiento [Ver20]. Un esquema de cómo funciona el aprendizaje supervisado se puede ver en la Figura 2.1.

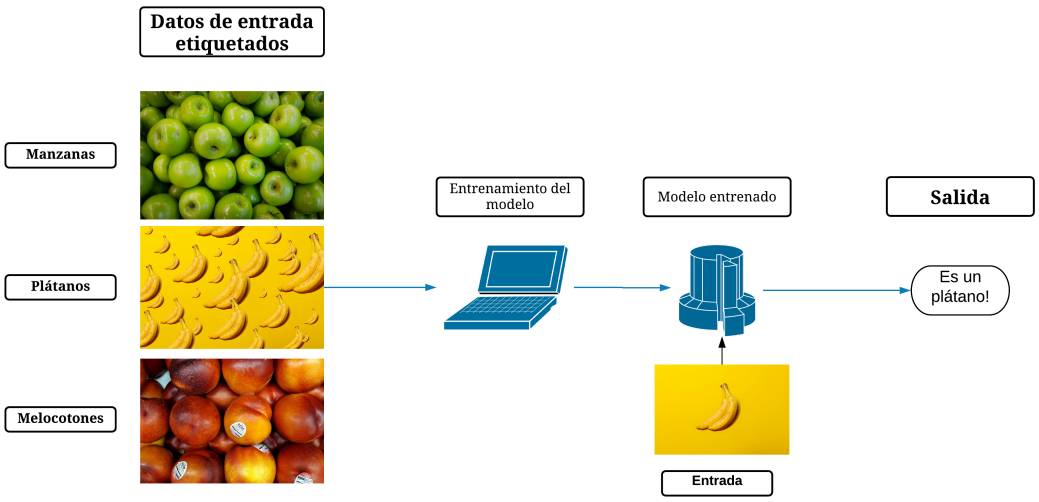


Figura 2.1: Esquema de funcionamiento del aprendizaje supervisado

Un ejemplo de tipos de algoritmos que pertenecen a este grupo son los de clasificación, utilizados cuando la salida está restringida a una cantidad limitada de valores, conocidos como *clases*. En el caso de la estimación de la edad, cada clase corresponderá con una edad, siendo el límite de valores la edad de la persona más mayor a analizar.

- **Aprendizaje No Supervisado**: El Aprendizaje No Supervisado (del inglés *Unsupervised Learning* (UL)), agrupa aquellos algoritmos en la que los datos de entrada no contienen la salida. Funcionan buscando patrones en los datos de entrenamiento haciendo agrupaciones de los mismos, para posteriormente agrupar en base a la presencia o la ausencia de puntos en común en cada nuevo dato [MP18]. Un esquema de este tipo de modelo se puede ver en la Figura 2.2.

  Una aplicación de este tipo de algoritmos es el del análisis de grupos (del inglés *Cluster analysis* (CA)), donde se trata de agrupar los datos en grupos o conjuntos en el que los datos comparten uno o más criterios predeterminados.

- **Aprendizaje Por Refuerzo**: El Aprendizaje Por Refuerzo (del inglés *Reinforcement Learning* (RL)), agrupa aquellos algoritmos en los que en vez de partir de unos datos de entrada dados, utilizan la información que reciben del

entorno que les rodea. Para que el *agente* aprenda, es necesario introducir premios y recompensas en base a las decisiones tomadas en función del entorno [SB18]. Dado su generalidad, es un campo estudiado en numerosas disciplinas siendo sus métodos principales la programación dinámica y los métodos de Monte Carlo. Su esquema de funcionamiento se puede ver en la Figura 2.3, donde se ve que el agente selecciona la mejor acción y el entorno le devuelve un premio o recompensa en función de lo buena que sea la acción. Un ejemplo de implementación de algoritmo siguiendo este tipo de razonamiento es el *AlphaGo*, algoritmo mencionado anteriormente por ganar a un campeón del mundo de Go.

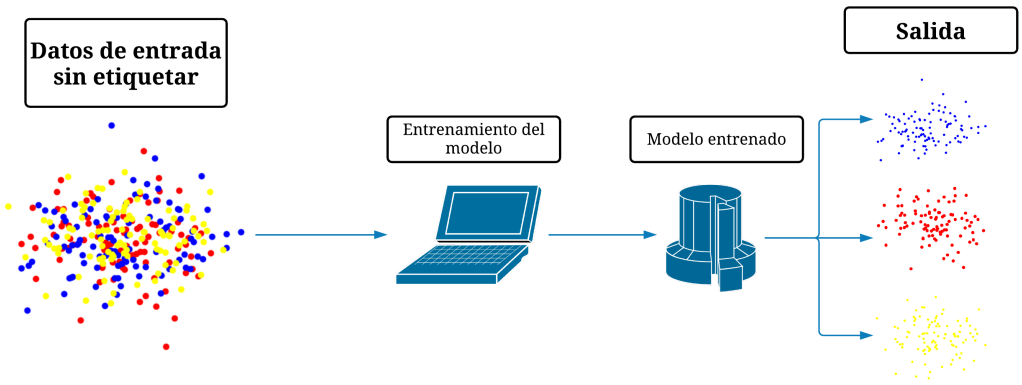


Figura 2.2: Esquema de funcionamiento del aprendizaje no supervisado

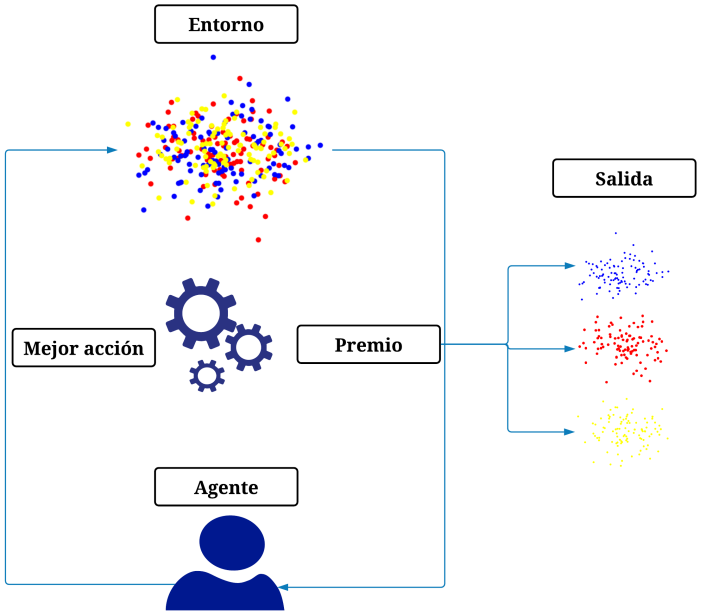


Figura 2.3: Esquema de funcionamiento del aprendizaje por refuerzo

Este trabajo se centra en los algoritmos de aprendizaje supervisado, en particular aquellos que son de clasificación, ya que el reconocimiento de la edad consiste en hallar la clase (edad) a la que pertenece la persona.

Uno de los subcampos más importantes del ML que se explicará más adelante es el DL, basado en las *Artificial Neural Networks* (ANNs) explicadas más adelante en la Sección 3.1. Es importante entender la relación que hay entre los conceptos de AI, ML y DL mostrada en la Figura 2.4.

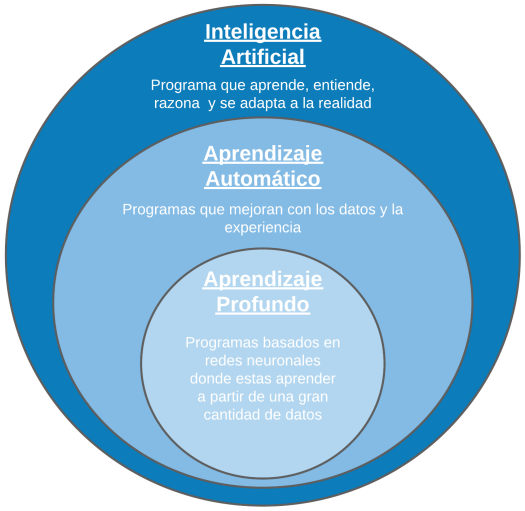


Figura 2.4: Relación entre Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo

2.3. Algoritmos de Clasificación

Los algoritmos de clasificación son utilizados para modelizar la probabilidad de una determinada clase y pertenecen al enfoque del SL. Es importante entender que muchos de los algoritmos utilizados para realizar tareas de clasificación son a su vez algoritmos que pueden ser utilizados para regresión. Existen numerosos tipos de algoritmos de clasificación pero veamos a continuación los más importantes.

2.3.1. Regresión Local

El modelo más básico que se puede utilizar y el primero en surgir. Se basa en la presunción de que la variable de entrada x y la variable de salida y están relacionadas de manera lineal [KZP07]. Esta presunción se puede comprobar mediante la representación de la variable de entrada x con la variable de salida y en un gráfico. El objetivo de la regresión local es medir esta relación y establecer una ecuación matemática que la modele. Una vez establecida la

relación se puede usar para predecir los valores de salida para nuevos datos. Esta relación se mide mediante la ecuación de una recta vista como:

$$y = \beta_0 + \beta_1 x \tag{2.1}$$

Donde $\beta_0 \in \mathbb{R}$ representa el valor sobre el eje de ordenadas y $\beta_1 \in \mathbb{R}$ la pendiente de la recta. El objetivo de entrenar este modelo es averiguar los valores óptimos β_0 y β_1.

Como se ve, la Ecuación 2.1 no ayuda a predecir clases ya que si la x toma valores continuos, la y también los tomará. Sin embargo, se puede utilizar en el caso de problemas de clasificación donde a cada clase se le puede asignar un valor numérico. En este caso, la clase de salida Y será la correspondiente con el valor más cercano al valor de salida y.

La regresión local también se puede aplicar en el caso de variables multidimensionales, pero dado su simplicidad y su poca usabilidad en problemas complejos no se explicará. Lo que si se explicará es la llamada **regresión logística**, un modelo estadístico que utiliza una función logística para modelar problemas de clasificación. La forma más básica es la llamada regresión logística binaria que como su propio nombre indica se utiliza para problemas de clasificación binaria. Cuando se tienen más de dos clases, utilizamos la llamada regresión logística multinomial.

Por ahora se verá la regresión logística binaria, la cual utiliza la llamada función *sigmoide* que será explicada posteriormente en la Sección 3.1.2, la cual recibe como entrada cualquier valor real y devuelve como salida un valor entre 0 y 1. La ecuación estándar de la regresión local se puede escribir como:

$$P(x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}} \tag{2.2}$$

En este caso se asignará a una clase el valor 0 y a la otra el valor 1 y el resultado dependerá de cuál de los dos valores esté más próximo el resultado. En el caso de introducir más clases habría que asignar a cada clase una probabilidad entre 0 y 1 siendo la suma de todas las posibles clases 1, pero este caso está fuera del alcance de este trabajo.

2.3.2. Árboles de Decisión

El segundo ejemplo a comentar es el de los Árboles de Decisión (del inglés Decision Trees (DT)), los cuales pueden ser utilizados tanto para problemas de regresión como de clasificación. Como su propio nombre indica, se construyen en forma de árbol dividiendo los datos en subconjuntos cada vez más y más pequeños mientras que incrementamos el tamaño del árbol de decisión creado [KS08]. El resultado final es un árbol con *nodos de decisión* y *nodos hojas*. Cada nodo de decisión tiene dos o más ramas, cada una

representando valores de un atributo evaluado. Un nodo hoja representa una decisión del valor del resultado objetivo. El primero de los nodos de decisión es llamado *nodo raíz*. Un esquema del funcionamiento del árbol de decisión se puede ver en la Figura 2.5

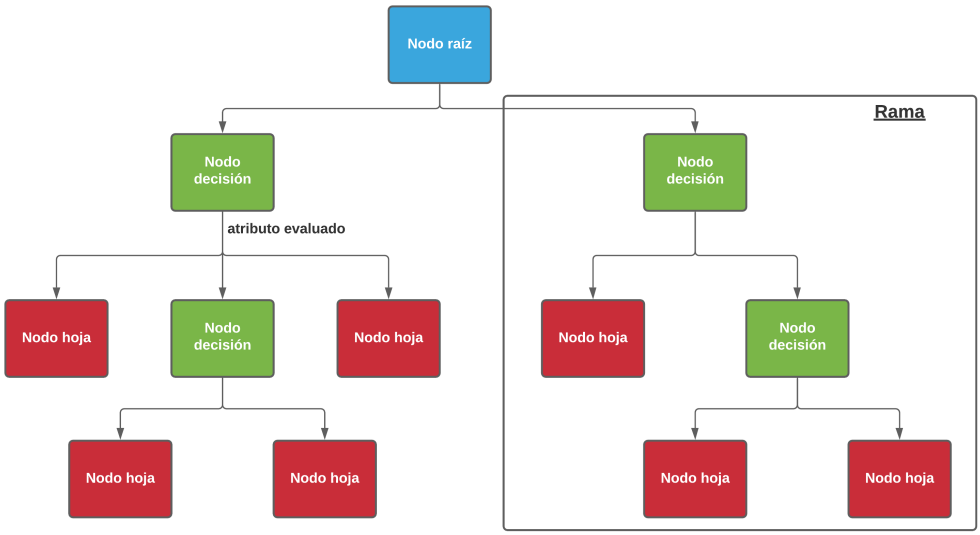


Figura 2.5: Esquema de un árbol de decisión

Un ejemplo muy sencillo de la aplicación de los árboles de decisión es el siguiente: supongamos que queremos saber si un dato de entrada es un hámster, una cabra, un elefante, un pingüino, un calamar y una araña. Para ello realizamos un árbol de decisión como en la Figura 2.6.

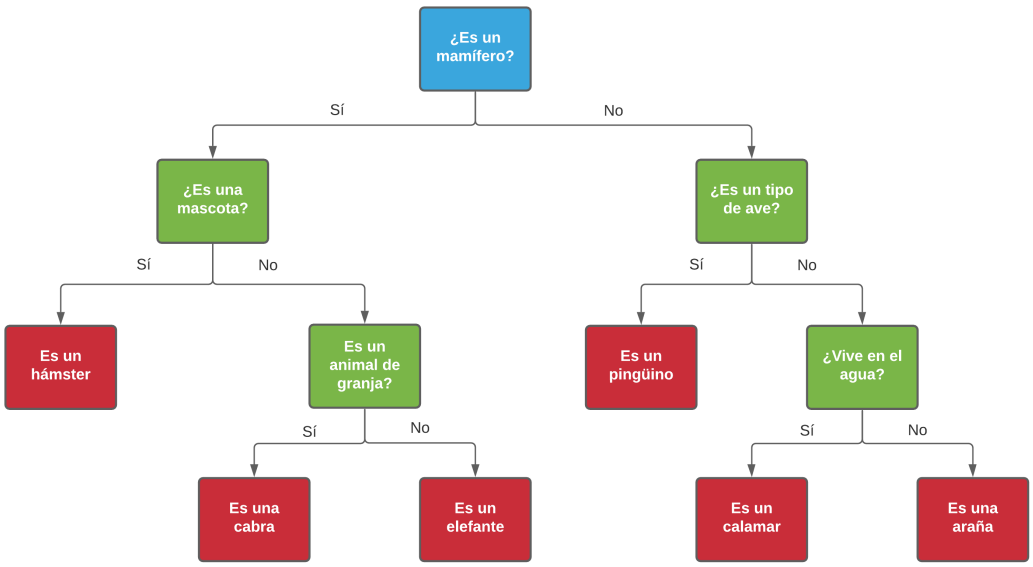


Figura 2.6: Ejemplo de árbol de decisión para clasificar animales

Existe también un algoritmo más complejo basado en los DT llamado Bosques Aleatorios (del inglés Random Forests (RF)), basado en construir una gran cantidad de DT en tiempo de entrenamiento y como salida devuelve la moda de las clases de los diferentes DT construidos. Este tipo de modelos son muy utilizados en la resolución de problemas porque aporta más consistencia que un modelo básico basado en DT ya que ignora errores individuales y se fija en el comportamiento de la mayoría.

2.3.3. Máquinas de Vectores de Soporte

La siguiente técnica que se va a explicar son las Máquinas de Vectores de Soporte (del inglés Support Vector Machines (SVM)). Las SVM son un tipo de clasificador binario lineal. Este tipo de modelos nos ayuda a lidiar con bases de datos más complejas donde los datos son multidimensionales y se usan en multitud de aplicaciones. Dada una serie de datos n-dimensionales, se trata de hallar un hiperplano o un conjunto de hiperplanos que separen las dos clases maximizando una cierta distancia (llamada *margen*) a los puntos de cada una [Nob06].

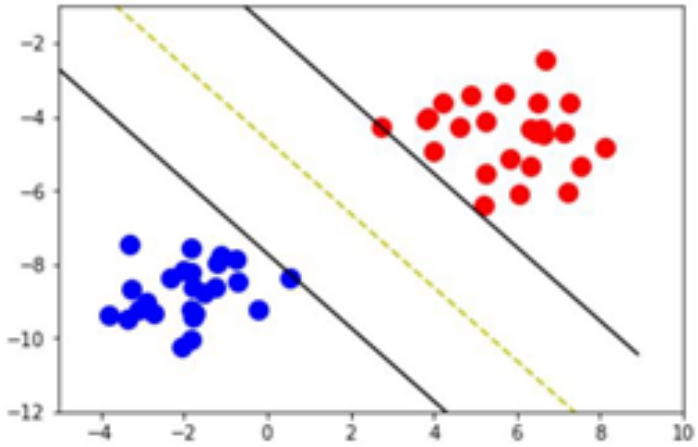


Figura 2.7: Ejemplo de aplicación de SVM en un espacio bidimensional [Gon20]

En la Figura 2.7 se ve un ejemplo de cómo se aplicaría este algoritmo para puntos bidimensionales. Los puntos que pasan por las rectas negras muestran los puntos más próximos al hiperplano (en este caso una recta al estar en un espacio bidimensional) y los cuales determinan el margen. Este caso es sencillo ya que se trata de un hiperplano lineal, pero existen ejemplos donde se necesitan hiperplanos no lineales para realizar la clasificación y para ello se necesitará la ayuda de una **función kernel**. Lo que permite esta función es conseguir aumentar la dimensionalidad del espacio en el que se está trabajando. Este tipo de técnica es muy utilizada para numerosas aplicaciones pero se encuentra fuera del alcance de este trabajo.

2.3.4. K-Vecinos Más Próximos

Otra técnica es la de los K Vecinos Más Próximos (del inglés k-Nearest Neighbors (k-NN)) donde se parte de unos datos de los que conocemos las clases y se escoge la moda de las clases de los k valores más próximos al dato que queremos clasificar. El valor de k es un valor fijado con anterioridad y dado un dato multidimensional $X = (x_1, \ldots, x_n)$ normalmente se define como más próximo al dato al valor $Y = (y_1, \ldots, y_n)$ que minimice la distancia euclídea entre los dos, es decir, el Y que minimice el valor

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Existen otras maneras de definir el concepto de proximidad y distancia como la distancia Manhattan o la distancia Minkowski [Pet09].

Esta técnica asume que los datos que son similares están próximos entre si, es decir, se asume que las clases de los k más próximos serán una buena aproximación para ver a qué clase pertenece el dato. Entrenar el modelo consiste simplemente en añadir los datos a la base de conocimiento para poder ver luego cuál de ellos está más cerca del nuevo dato dado.

Capítulo 3

Visión Artificial

La Visión Artificial (del inglés *Computer Vision* (CV)), es un campo de estudio que busca desarrollar técnicas que permitan a los ordenadores ver y entender imágenes y vídeos [FP12]. Uno de los modelos que ha demostrado tener mejor resultado para realizar esta tarea son las redes neuronales. Este capítulo detalla el funcionamiento de las redes neuronales artificiales en la Sección 3.1. En la Sección 3.2 se explican un tipo particular de las redes neuronales artificiales, las redes neuronales convolucionales. Se describe también de otro tipo de redes neuronales artificiales, las redes neuronales recurrentes en la Sección 3.3. Por último, en la Sección 3.4 se explica el proceso de entrenamiento de todas estas redes neuronales.

3.1. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (del inglés ANNs) son modelos simplificados del sistema nervioso humano. Son redes de *neuronas* computacionales altamente interconectados con la capacidad de responder a un estímulo de entrada y aprender a adaptarse al entorno [Pat98], [Mic18].

3.1.1. Estructura de una Única Neurona

Para entender el funcionamiento de una red neuronal es necesario entender cómo funciona cada una de ellas para luego entender cómo se conectan y cómo se forman las redes. Por ello, en esta sección se discutirá qué es una neurona y qué componentes tiene.

Básicamente cada neurona computacional coge una cierta cantidad de entradas (números reales) y calcula una salida (también un número real). Para denotar las entradas se utilizará $x_i \in \mathbb{R}$ con $i \in \{1, 2, \ldots, n\}$ donde n es el número de atributos de entrada, llamados normalmente como características (conocido en inglés como *features*). Esta

cantidad de características suele ser muy grande.

Existen numerosos tipos de neuronas, pero se explicarán las más sencillas, aquellas que para calcular la salida aplican una función a una combinación lineal de las entradas junto con un sesgo. Visto de manera matemática se puede ver como:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \tag{3.1}$$

Donde $(w_1, \ldots, w_n) \in \mathbb{R}^n$ representan los pesos (conocido en inglés como *weights*) y $b \in \mathbb{R}$ el valor de sesgo (conocido en inglés como *bias*). Se puede ver como se genera la salida a partir de la entrada en la Figura 3.1.

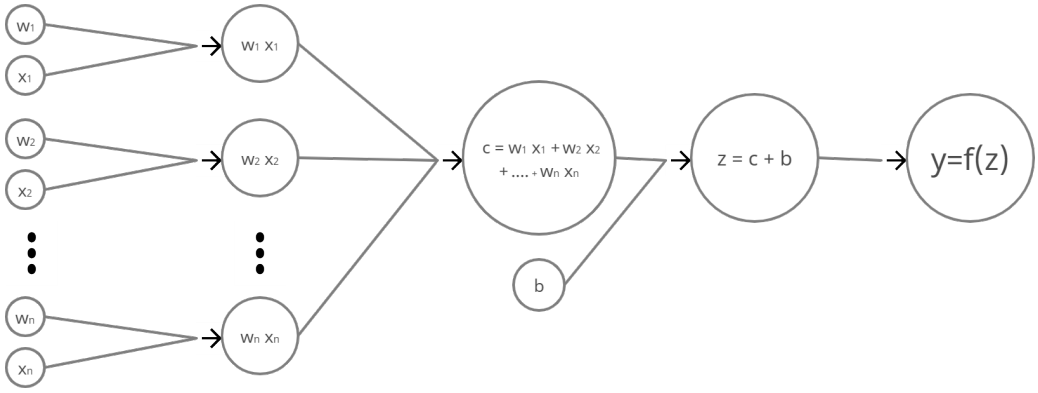


Figura 3.1: Gráfico computacional de una neurona

La f representa la función aplicada a la combinación lineal junto con el sesgo. Por lo tanto, la salida y será el resultado de aplicarle f a z. A la función f se le denomina función de activación (del inglés *activation function*).

Para simplificar dicha notación se utilizará notación de vectores con $x = (x_1, x_2, \ldots, x_n)$ y $w = (w_1, w_2, \ldots, w_n)$. Por lo que se tendrá $z = w\,x^T + b$ siendo x^T el vector traspuesto de x.

3.1.2. Funciones de Activación

Hay numerosas funciones de activación disponibles para cambiar la salida de la neurona. Se verán ahora las más utilizadas.

- **Función Identidad**: Esta es la función de activación más básica que se puede utilizar. Normalmente se denota como $I(z)$. Como su propio nombre indica, asigna a cada punto del dominio z, la misma salida. Matemáticamente se puede escribir como:

$$f(z) = I(z) = z \tag{3.2}$$

La Figura 3.2 muestra el gráfico de cómo se comporta la función identidad ante diferentes valores de z.

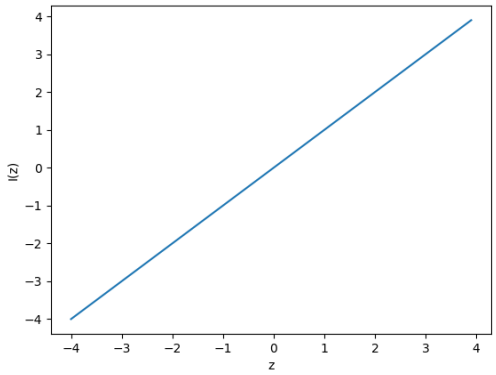


Figura 3.2: Gráfico de la función identidad

Esta función sirve para implementar la regresión lineal con una única neurona.

- **Función Sigmoide**: Esta función es una de las más utilizadas ya que devuelve valores entre el 0 y el 1. Normalmente se denota como $\sigma(z)$. La Figura 3.3 muestra como se comporta ante diferentes valores de entrada z. Su expresión matemática es:

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

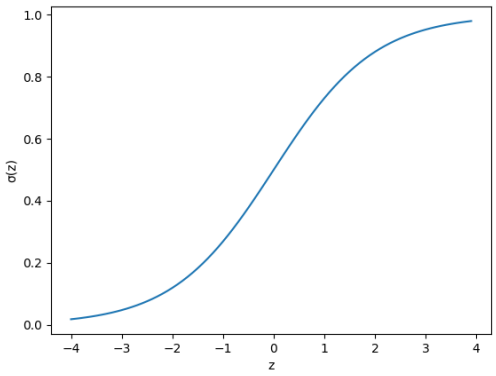


Figura 3.3: Gráfico de la función sigmoide

Se utiliza cuando se requiere que las salidas sean probabilidades y sobre todo en problemas de clasificación binaria. Una de las cosas a tener en cuenta es que si se tiene un valor de z muy grande (positivo o negativo), el ordenador podría redondear el valor de la salida a 0 o a 1, lo cual no podría suceder teóricamente.

- **Tangente Hiperbólica**: Utilizada cuando se quiere tener valores entre el -1 y 1. Se puede ver como una modificación de la función sigmoide para que los valores

negativos tengan su correspondiente valor de salida negativo. Normalmente se denota como $tanh(z)$. Su utilidad, al igual que la función sigmoide, radica en la resolución de problemas de clasificación binaria. Su expresión matemática se puede ver como:

$$f(z) = tanh(z) = 2\sigma(2z) - 1 = \frac{2}{1 + e^{-2z}} - 1 \tag{3.4}$$

Podemos ver su gráfico en forma de S en la Figura 3.4.

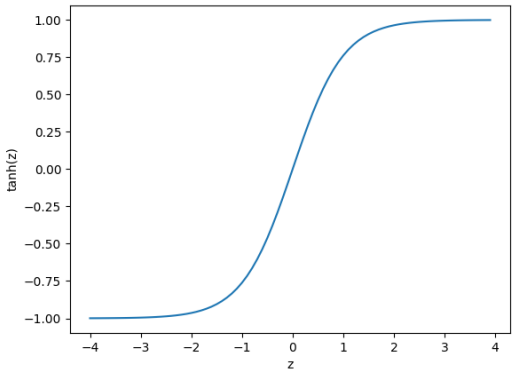


Figura 3.4: Gráfico de la tangente hiperbólica

- **ReLU**: La función $ReLU$ (del inglés Rectified Linear Unit (ReLU)) es también una de las funciones de activación más utilizadas debido a su versatilidad y que permite realizar el entrenamiento de manera más rápida ya que su cálculo es muy sencillo y rápido. Si el valor de z es positivo, es equivalente a la función identidad, mientras que si el valor de z es negativo, el valor de la salida será 0. Su expresión matemática se puede ver como:

$$f(z) = ReLU(z) = max(0, z) \tag{3.5}$$

El gráfico de la función se puede ver en la Figura 3.5.

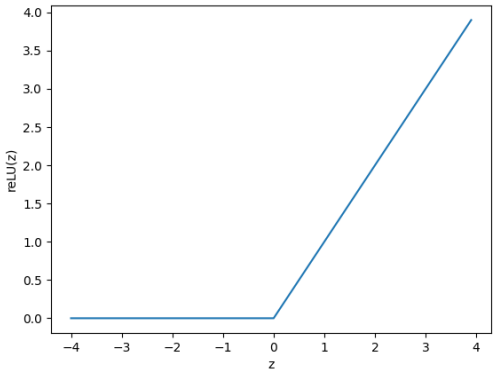


Figura 3.5: Gráfico de la función ReLU

- **Softmax**: Se verá ahora una de las funciones de activación más importantes para realizar problemas de clasificación con más de dos clases. Se utiliza en la última capa de la red neuronal para decidir cuál es la clase de salida elegida. La función *softmax*, denotada por $S(z)$, transforma un vector de k dimensiones en otro vector de k dimensiones de valores reales, cada uno entre 0 y 1, y cuya suma vale 1.

  Dados k valores reales $z = (z_1, \ldots, z_k) \in \mathbb{R}^k$ se define el vector resultado de aplicar la función *softmax* como $S(z) = (S(z)_1, \ldots, S(z)_k)$ donde

  $$S(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{3.6}$$

  Donde $i = 1, \ldots, k$. Como el denominador es siempre más grande que el numerador tenemos que $S(z)_i < 1$. Adicionalmente, se tiene que:

  $$\sum_{i=1}^{k} S(z)_i = \sum_{i=1}^{k} \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \frac{\sum_{i=1}^{k} e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = 1 \tag{3.7}$$

  Por lo tanto $S(z)$ se comporta como una probabilidad, ya que todos los elementos son menores que 1 y su suma es exactamente 1. Por lo tanto el valor $S(z)_i$ representa la probabilidad de que el dato de entrada pertenezca a la clase i. Esta cualidad hace que se utilice en prácticamente todas las redes neuronales que realizan tareas de clasificación cuando el número de clases es mayor que dos. Para elegir la clase a la que pertenece la entrada simplemente se escoge la componente del vector $S(z)$ con mayor probabilidad.

- **Función Swish**: Aunque no sea una función de activación muy utilizada, es una función muy prometedora para el mundo del DL. Su expresión matemática parte de la función sigmoide ya que se expresa como:

  $$f(z) = swish(z) = z\,\sigma(\beta z) = \frac{z}{1 + e^{-\beta z}} \tag{3.8}$$

  Donde β puede ser constante o un parámetro entrenable por el modelo. Esta función fue creada por el *Google Brain Team* y su estudio [RZL17] ha demostrado que simplemente reemplazando las funciones de activación ReLU por funciones Swish se mejora la precisión de la clasificación en *Imagenet* en un 0,9 %.

3.1.3. Aprendizaje de una Neurona

Una vez que se entiende bien cómo funciona una neurona computacional, se tiene que entender qué significa que una neurona aprenda. En la mayoría de redes neuronales, aprender significa buscar los pesos (se recuerda que cada neurona tiene su propio conjunto de pesos) y el sesgo de la red que minimice una cierta función escogida, normalmente llamada **función de coste** o **función de pérdida** y normalmente nombrada como J.

Existen numerosos métodos para buscar el mínimo de una función dada analíticamente. Sin embargo, en las redes neuronales utilizadas en la realidad, el número de pesos es tan grande que no es posible usar estos métodos. Por tanto, para calcular el mínimo, es necesario usar métodos numéricos. El más famoso de estos métodos es el llamado **descenso de gradiente**.

A continuación, se explica brevemente cómo funciona el descenso de gradiente. Dada una función $f(w)$ donde w es un vector de costes diferenciable, se puede calcular un mínimo local del espacio de pesos (i.e el valor w del dominio que minimiza la función $J(w)$) iterando de la siguiente manera:

1. Iteración 0: Se escoge un valor de los pesos inicial aleatorio w_0.
2. Iteración n: Los pesos de la iteración n, w_n, se calculan a partir de los valores de los pesos de la iteración anterior, w_{n-1}, usado la fórmula:

$$w_n = w_{n-1} - \gamma \nabla f(w_{n-1}) \tag{3.9}$$

Donde $\nabla J(w)$ indica el gradiente de la función de coste en el punto w, el cual es el vector cuyas componentes son las derivadas parciales de la función de coste con respecto a las componentes del vector de pesos w, es decir:

$$\nabla J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix} \tag{3.10}$$

El valor γ es el llamado **ratio de aprendizaje** (del inglés *learning rate*), cuya utilidad y significado se explicará más adelante en esta sección. Antes de eso, es necesario explicar cuándo se para de iterar y por tanto se termina el proceso.

Normalmente se define un valor $\epsilon \in \mathbb{R}$ y un valor $k \in \mathbb{N}$ tal que $|J(w_{q+1}) - J(w_q)| < \epsilon$ para todo $q > k$. En la práctica, esta comprobación es muy costosa por lo que se establece un cierto número de iteraciones y se deja el algoritmo funcionar durante ese número de iteraciones.

Existen variaciones muy utilizadas del Descenso Gradiente, la más común es el llamado Descenso Gradiente Estocástico (del inglés Stochastic Gradient Descent (SGD)).

El ratio de aprendizaje es uno de los parámetros de aprendizaje más importantes utilizados en el entrenamiento de una red neuronal. Es de vital importancia ya que dependiendo de su elección el modelo puede no converger o converger demasiado lento, por lo que nunca aprenderá. Este valor es diferente dependiendo del modelo y normalmente los valores utilizados varían desde 0.0001 hasta 0.1.

3.1.4. Funciones de Coste

Para terminar de explicar cómo funciona una neurona, se mostrarán ahora algunas de las funciones de coste más utilizadas. Es importante entender que dependiendo del tipo de problema que se quiera resolver se escogerá la función de coste más adecuada. Las funciones de coste se pueden dividir en dos grandes grupos según el problema que resuelven. Por un lado se tienen las funciones de coste utilizadas para problemas de clasificación y por otro lado las usadas para problemas de regresión.

Antes de pasar a ver ejemplos de coste para problemas de clasificación, se introduce algo de notación. En la Sección 3.1.1 se han denotado los datos de entrada como un vector $x = (x_1, x_2, \ldots, x_n)$ donde cada una de las componente es una característica. Este vector describe lo que se llama una observación, pero normalmente se realizan varias observaciones. Por lo tanto, se introduce un nuevo superíndice para indicar el número de observación de manera que el vector $x^{(i)}$ corresponde con el vector de características de la observación i-ésima y el valor $x_j^{(i)}$ con la característica j de la observación i-ésima. Se supone que el número de observaciones es m por lo tanto el conjunto de valores de entrada pueden ser escrito en forma de matriz con la siguiente notación:

$$X = \begin{pmatrix} x_1^{(1)} & \ldots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_n^{(1)} & \ldots & x_n^{(m)} \end{pmatrix} \tag{3.11}$$

donde cada fila de la matriz X corresponde con una característica y cada columna corresponde con una observación. De la misma manera, se puede poner los valores de salida y y los valores de z en función de cada observación de manera que la ecuación $z^{(i)} = w\,(x^{(i)})^T + b$ corresponde con el valor de z previamente definido para la observación i. Si se quieren representar todas las observaciones se utilizará la notación:

$$Z = (z^{(1)}\ z^{(2)} \ldots z^{(m)}) = w\,X^T + B \tag{3.12}$$

donde $B = (b, \dots, b) \in \mathbb{R}^m$. Por lo que la salida será

$$Y = (y^{(1)}\ y^{(2)} \dots y^{(m)}) = \left(f(z^{(1)})\ f(z^{(2)}) \dots f(z^{(m)})\right) = f(Z). \tag{3.13}$$

Solo para terminar de explicar esta notación se repasarán las dimensiones de los matrices involucradas.

- X tiene dimensión $n \times m$
- Z tiene dimensión $1 \times m$
- Y tiene dimensión $1 \times m$
- w tiene dimensión $n \times 1$
- B tiene dimensión $1 \times m$

Ahora que hemos terminado de formalizar la notación se mostrarán algunos de los ejemplos de funciones de coste.

3.1.4.1. Entropía Cruzada

La entropía cruzada (del inglés Cross Entropy (CE)) se utiliza en problemas de clasificación probabilística, es decir, cuando se quiere que la salida sea una predicción probabilista. En estos problemas se sabe la clase real a la que pertenecen los datos de entrada y se representa mediante un vector $t = (0, 0, \dots, 1, \dots, 0)$, donde el 1 está en la posición que representa a la clase a la que pertenecen los datos. La predicción, por tanto, será un vector de la forma $y = (y_1, y_2, \dots, y_n)$ donde cada $y_i \in [0, 1]$ representa la probabilidad de que el valor esté en la clase i y $\sum_{i=1}^{n} y_i = 1$. Por tanto, para calcular lo bien que se comporta la red neuronal, interesa ver la diferencia que hay entre los valores esperados t y los valores predichos y. Se recuerda que lo que se busca es una función que se minimice de manera que se modifiquen los pesos para mejorar el modelo a la hora de realizar la tarea de clasificación.

Para resolver esta cuestión surge la función de coste de entropía cruzada cuya fórmula para la observación i es:

$$L\left(t^{(i)}, y^{(i)}\right) = -\sum_{j=1}^{n} t_j^{(i)}\ \log y_j^{(i)} \tag{3.14}$$

Si se tiene más de una observación se tendrá la función de coste para todas las observaciones:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L\left(t^{(i)}, y^{(i)}\right) \tag{3.15}$$

Se utiliza el logaritmo para que la penalización sea más grande si la diferencia entre los valores está próximo a 1, mientras que la penalización es más baja si la diferencia es próxima a 0.

Un caso especial y muy utilizado es el de la entropía cruzada binaria (del inglés Binary Cross Entropy (BCE)). Es la función de coste más utilizada para problemas de clasificación binaria. Es una caso particular de la CE donde el número de clases es 2. Es decir su expresión sería:

$$J(w,b) = L\left(t^{(i)}, y^{(i)}\right) = -\sum_{j=1}^{2} t_j^{(i)} \log y_j^{(i)} = -[t^{(i)} \log y^{(i)} + (1 - t^{(i)}) \log(1 - y^{(i)})] \quad (3.16)$$

3.1.4.2. Hinge Loss

La segunda función de coste más comúnmente utilizada para problemas de clasificación es la llamada Hinge Loss (HL) y se suele utilizar para la evaluación de modelos de SVM. Se suele usar en problemas de clasificación donde la salida esperada t puede tomar los valores 1 y -1. Es una función bastante sencilla que tiene la siguiente expresión:

$$J(w,b) = l(y^{(i)}) = \text{máx}(0, 1 - t^{(i)} \cdot y^{(i)}) \quad (3.17)$$

3.1.4.3. Funciones Utilizadas para Regresión

Existen también funciones de coste utilizadas para problemas de regresión. En este caso no es tan importante su estudio ya que nos se centrará en un problema de clasificación, pero serán útiles definir algunas de ellas para luego utilizarlas para medir el error que tiene el modelo.

La primera función de coste es la llamada Error Cuadrático Medio (del inglés Mean Squared Error (MSE)). Como su nombre indica, esta función mide la media entre los cuadrados de los errores, siendo los errores la diferencia entre el valor esperado t y el valor actual. Hay que tener en cuenta que el error se mide para cada observación, por tanto su expresión es:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - w\,(x^{(i)})^T - b\right)^2 \quad (3.18)$$

Donde m es el número de observaciones.

Otra de las funciones más utilizadas es el llamado Error Absoluto Medio (del inglés MAE). Esta función es parecida a la anterior, solo que mide la media entre los valores absolutos de los errores. Es decir:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \left|y^{(i)} - w\,(x^{(i)})^T - b\right| \quad (3.19)$$

3.1.5. Estructura de una Red Neuronal

El poder real de las redes neuronales se presenta cuando unes miles e incluso millones de neuronas que interaccionan entre ellas para resolver un problema específico. La arquitectura de la red (cómo se conectan las neuronas entre ellas, cómo se comportan, etc) tiene un papel muy importante a la hora de ver cómo de eficiente es la red neuronal aprendiendo, cuan buenas son sus predicciones y qué tipo de problemas puede resolver. Las neuronas se organizan en **capas** y lo más importante es ver cómo las neuronas de diferentes capas se comunican entre si. La primera y más sencilla red neuronal ideada es la llamada **red neuronal prealimentada** (del inglés *feedforward neural network*), en la cual los datos entran a una capa de entrada y pasan capa por capa por la red hasta que llegan a la capa de salida. Por lo general, cada neurona de cada capa recibe como entrada la salida de todas las neuronas de una capa anterior y su salida se dirige a cada neurona de una capa posterior, véase la Figura 3.6. En este caso se dice que la red neuronal está totalmente conectada.

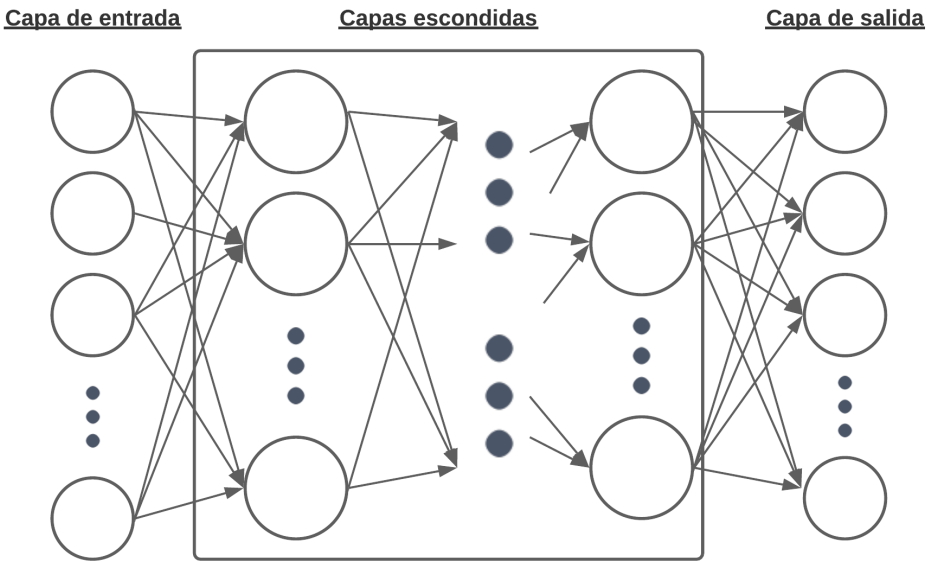


Figura 3.6: Diagrama de una red neuronal prealimentada multicapa

Las ANNs han demostrado ser efectivas para diversas tareas como el reconocimiento de patrones (e.g. reconocimiento visual y del habla), clasificación y compresión de datos. Funcionan de manera muy robusta resolviendo problemas de ruido, patrones de entrada incompletos y en aprendizaje adaptativo.

Existen varios tipos de ANNs utilizados con diferentes fines. Uno de los tipos principales son las Redes Neuronales Profundas (en inglés *Deep Neural Networks* (DNNs)), que son un caso particular donde hay múltiples capas entre las capas de entrada y salida. Un

tipo particular de DNNs son las CNNs que han demostrado ser muy efectivas en procesar datos visuales y otro tipo de datos bidimensionales y tridimensionales. También existen las llamadas Redes Neuronales Recurrentes (del inglés *Recurrent Neural Networks* (RNNs)), particularmente eficaces en aplicaciones como el reconocimiento de voz o en el caso de los vídeos, dado el componente continuo de los fotogramas.

En la siguientes secciones se centra el estudio en estos tipos de ANNs, sobre todo incidiendo en las CNNs utilizadas en el modelo construido.

3.2. Redes Neuronales Convolucionales y sus Componentes

Se pasa ahora a explicar un tipo particular de DNNs, las CNNs, utilizadas sobre todo en el campo de CV. Para ello se explicarán primero algunas operaciones particulares de este tipo de redes neuronales [Mic19].

3.2.1. Filtros

Uno de los componentes más importantes de las CNNs son los llamados filtros, los cuales son matrices cuadradas de dimensión $n_k \times n_k$ donde $n_k \in \mathbb{N}$ siendo normalmente un número pequeño como 3 o 5. A los filtros también se les denomina como núcleos (del inglés *kernels*). Estos filtros se utilizan para detectar patrones en las imágenes como líneas horizontales y verticales, cambios de luminosidad, etc.

3.2.2. Convolución

Una vez visto qué son los filtros se explicará una función muy importante para entender las CNNs, la convolución. La convolución se hace entre los llamados tensores. Estos tensores son objetos algebraicos que describen una relación entre conjuntos de objetos algebraicos de un espacio vectorial. Los tensores bidimensionales de entrada son normalmente matrices.

La convolución tiene como entrada dos tensores y produce como salida un tensor. Se denotará a la operación convolución con el operador $*$. Dados dos tensores con dimensión 3×3 la convolución funciona de la siguiente manera:

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_4 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix} * \begin{pmatrix} k_1 & k_2 & k_3 \\ k_4 & k_4 & k_6 \\ k_7 & k_8 & k_9 \end{pmatrix} = \sum_{i=1}^{9} a_i k_i \tag{3.20}$$

Es decir, multiplica cada elemento del primer tensor con su correspondiente de la segunda si existe y suma el resultado de todas las multiplicaciones.

Normalmente esta operación se realiza entre un tensor, denotado por A y un filtro. Mientras que la dimensión de A suele ser grande, como se mencionó anteriormente los filtros suelen tener dimensión 3×3 o 5×5. El tensor de entrada A es la imagen cuyas dimensiones pueden llegar a ser de $1024 \times 1024 \times 3$, donde 1024×1024 corresponde con la resolución y la última dimensión 3 con el número de canales de color (los valores RGB).

Dada la diferencia de dimensión es necesario entender cómo se aplica la convolución dada estas diferencias de dimensiones. Hay que aclarar que no se tiene en cuenta la dimensión del número de canales de color por el momento. La idea es empezar por la esquina superior izquierda de la matriz y coger una submatriz tan pequeña como lo sea la dimensión del filtro. Una vez aplicado el filtro a esa submatriz quedará un valor resultante que se guardará en una nueva matriz B. Para elegir la segunda submatriz con la que realizar la operación , se coge una submatriz trasladándose s valores hacia la derecha, es decir, se mueve s columnas hacia la derecha siempre que sea posible y si no es posible se coge la primera submatriz empezando por la izquierda y s filas por debajo de la que se estaba analizando. El valor s es el llamado paso (del inglés *stride*). Este proceso se itera hasta que se llega hasta la esquina inferior derecha de la matriz.

Explicado ahora formalmente el caso donde $k = 1$ se tendrá que el algoritmo genera un nuevo tensor B dado un tensor de entrada A y un filtro K siguiendo la fórmula:

$$B_{ij} = (A * K)_{ij} = \sum_{f=0}^{n_k-1} \sum_{h=0}^{n_k-1} A_{i+f,j+h} K_{i+f,j+h} \tag{3.21}$$

Donde $n_k \times n_k$ es la dimensión del filtro K.

Si se supone que la dimensión de A es $n_a \times n_a$, la dimensión de la matriz B $n_b \times n_b$ sería:

$$n_b = \left\lfloor \frac{n_a - n_k}{s} + 1 \right\rfloor \tag{3.22}$$

Donde $\lfloor x \rfloor$ es el *suelo* de x.

Por último, antes de pasar a otra operación, se incluye una explicación gráfica de como aplicar el filtro

$$K = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

En la Figura 3.7 se muestra cómo funcionaría esta operación en una matriz A de dimensión 4×4 ya que la fórmula es bastante enrevesada y difícil de entender.

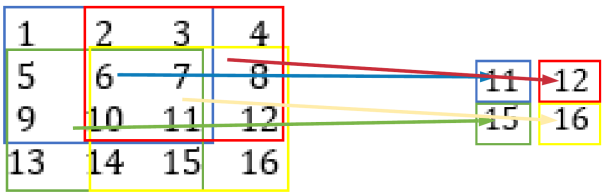


Figura 3.7: Ejemplo de la aplicación de la función de convolución

3.2.3. Pooling

La segunda operación fundamental en las CNNs es el *Pooling*. Esta operación es mucho más sencilla que la convolución. El tipo más común de *pooling* es el llamado *máximo pooling* (del inglés *max pooling*).

Esta operación se basa en los mismos principios que la convolución, se parte de la imagen y de una función a aplicar a las submatrices de la imagen. Las submatrices se cogen en el mismo orden que en la convolución, es decir, de arriba a abajo y de izquierda a derecha. Una vez se coge una submatriz, se queda con el máximo entre los valores de la submatriz y se guarda en una nueva matriz B. La Figura 3.7 también sirve como ejemplo de la aplicación de la operación de *pooling* si se cogen submatrices 3×3 con paso 1.

Otros tipos de *pooling* que existen es el *pooling promedio* (del inglés *average pooling*) el cual coge la media de los valores dentro de la submatriz o filtro y el *pooling sumario* (del inglés *sum pooling*) el cual coge la suma de todos los valores.

Existe también otro concepto importante a mencionar que es el concepto de *relleno* (del inglés *padding*), que como su nombre indica, se trata de añadir dimensiones a la matriz de entrada y rellenarla con ceros para que los filtros se puedan aplicar en el caso de que las dimensiones sean diferentes que las de la imagen de entrada. Por ejemplo si se quiere aplicar un filtro 3×3 a una imagen 4×4 con paso 2, no se tendrán suficientes valores para aplicar dos veces la función, por lo que será necesario aplicar el relleno.

3.2.4. Capas

Las operaciones de convolución y *pooling* explicadas en las secciones anteriores son utilizadas para construir las capas utilizadas en las CNNs. Existen tres capas principales en la construcción de las CNNs

- Capas de convolución
- Capas de pooling
- Capas totalmente conectadas

Las capas totalmente conectadas son precisamente las capas explicadas en la Sección 3.1.5, una capa donde las neuronas están conectadas a todas las neuronas de las capas anterior y posterior.

- **Capas de Convolución**: Una capa de convolución recibe como entrada un tensor (que puede ser tridimensional debido a los tres canales de color) y le aplica un determinado número de filtros, normalmente entre 10 y 32. Le aplica la convolución, añadiendo un sesgo y aplicando la función de activación $ReLU$ explicada en la Sección 3.1.2 para introducir no linealidad al resultado de la convolución y devuelve una matriz de salida B. En la Sección 3.2.2 se vio como aplicar la convolución con un único filtro. Para aplicar varios filtros a la vez es muy sencillo, simplemente el tensor de salida (se utiliza la palabra tensor porque ya no es una simple matriz) B no va a tener dos dimensiones si no que va a tener 3. Lo que se hace es añadir una dimensión para guardar los resultados de todos los filtros aplicados. Los pesos o parámetros de esta capa durante el entrenamiento son los elementos de los filtros. Es decir, entrenar estas redes consiste en ver los valores óptimos de los filtros para conseguir el resultado. Es decir si se tienen n_c filtros de dimensión $n_k \times n_k$ se tendrán $n_k^2 n_c$ parámetros en una capa convolucional.

- **Capas de Pooling**: La capa de *pooling* recibe como entrada un tensor y devuelve como salida otro tensor después de aplicarle la operación de *pooling* a la entrada. Es mucho más sencilla y no tiene parámetros de aprendizaje, pero depende de los valores del tamaño del filtro y el paso para realizar la operación. Normalmente en estas capas no se usa el relleno, ya que uno de los objetivos fundamentales de usar la operación de *pooling* es reducir la dimensionalidad de los tensores. Su objetivo es condensar la información después de una capa convolucional con la intención de crear una capa más compacta.

Normalmente en las CNNs se construyen primero una serie de capas de convolución seguidas de una capa de *pooling* que se utilizan para el aprendizaje de las características, para luego pasar a la capa totalmente conectada tal como se indica en la Figura 3.8. En esa figura se supone la aplicación de un filtro de dimensión 5×5 con paso 1 en las capas convolucionales y un filtro 2×2 con paso 2 en el caso de las capas de *pooling*.

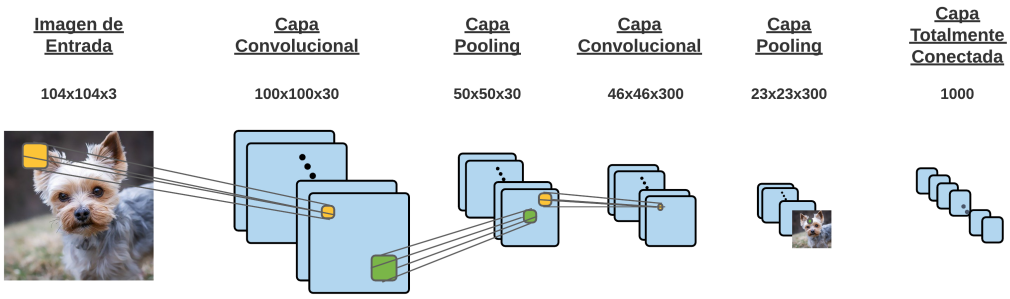


Figura 3.8: Representación de una red convolucional

Existen numerosas arquitecturas de CNNs con nombre dentro de la comunidad debido a su gran rendimiento y optimización. Algunas de las más importantes son: *Xception, VGG, ResNet, Inception, MobileNet, DenseNet, MobileNet*. La mayoría de ellas a su vez contienen varias versiones utilizables en diversos lenguajes de programación.

3.3. Redes Neuronales Recurrentes

A continuación se explicarán las llamadas redes neuronales recurrentes. Normalmente se utilizan cuando se trata con información secuencial, es decir, datos en donde importa el orden [MJ01]. El ejemplo más típico es el de series de palabras en una oración. La palabra recurrente viene de cómo funcionan este tipo de redes: aplican la misma operación a cada elemento de la oración, acumulando información de los términos previos.

Típicamente en la literatura, las RNNs se ilustran como en la Figura 3.9. Se ilustrará ahora qué representa cada uno de los elementos de la parte izquierda de la figura: x se refiere a la entrada, S a la memoria interna, W a un conjunto de pesos y U a otro conjunto de pesos. La parte derecha de la Figura 3.9 se debe leer de izquierda a derecha. La primera neurona evalúa en un tiempo t, produce una salida O_t y crea un estado de memoria interna S_t. La segunda neurona se evalúa en un tiempo $t+1$ después de la primera neurona, recibe como entrada tanto el siguiente elemento de la secuencia, x_{t+1}, y el anterior estado de memoria S_t. La segunda neurona luego genera una nueva salida, O_{t+1} y un nuevo estado de memoria interna, S_{t+1}. Este proceso se repite de esta manera para un número finito de neuronas.

Como se dijo anteriormente, existen dos conjuntos de pesos, W y U. El conjunto W se usa para los estados de memoria internos y U para el elemento de secuencia. Normalmente, cada neurona genera un nuevo estado interno de memoria siguiendo una fórmula que parece de la siguiente manera:

$$s_t = f\left(Ux_t + Ws_{t-1}\right) \tag{3.23}$$

Donde f denota una de las funciones de activación ya vistas, normalmente ReLU o la

tangente hiperbólica explicadas en la Sección 3.1.2.

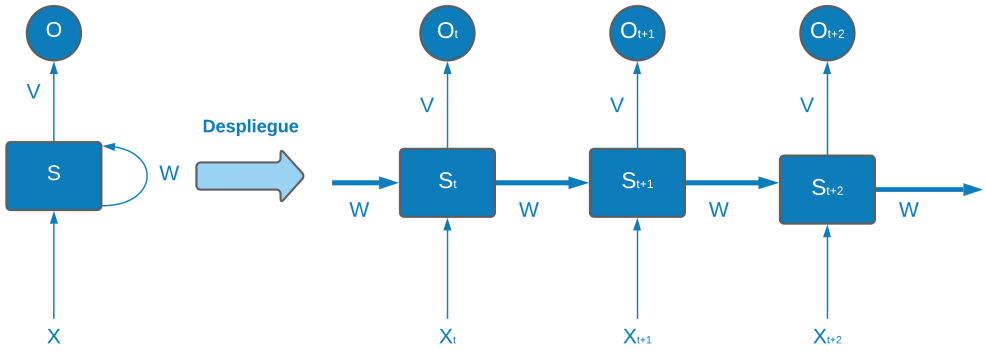


Figura 3.9: Representación esquemática de una Red Neuronal Recurrente

Uno de los tipos más importantes de RNNs es la Memoria a Largo y Corto Plazo (del inglés *Long Short Term Memory* (LSTM)), utilizada en tareas de DL y sobre todo cuando se está tratando con imágenes y videos.

3.4. Entrenamiento de las Redes Neuronales

Una vez creada una red neuronal, es necesario realizar un proceso que permita que la red neuronal aprenda a realizar la función para la que ha sido creada, a este proceso se le denomina *entrenar la red neuronal*. Entrenar la red neuronal significa ajustar los pesos de las diferentes capas para aumentar la precisión con la que realiza su función [Mic19]. El objetivo de esta sección es conseguir formas de hacer este proceso de entrenamiento eficiente, rápido y confiable. Existen numerosas formas de conseguir estos objetivos pero se enfocará en explicar los optimizadores.

Antes de nada es necesario entender qué se entiende por *iteraciones* y *ciclo* (del inglés *epoch*). Una iteración es cada uno de los pasos donde se actualizan los pesos. Un ciclo es un paso donde se entrena la red neuronal con todos los datos de entrenamiento. El número de datos que tiene cada iteración de un ciclo se denomina tamaño de lote (del inglés *batch size*).

Hasta ahora solo se ha visto cómo funciona el Descenso de Gradiente como mecanismo para minimizar la función de coste. Sin embargo esta no es la manera más eficiente ya que existen algunas modificaciones del algoritmo que permite hacerlo de una manera mucho más rápida y eficiente. Esta es un área muy activa de investigación y existen una gran cantidad de algoritmos, basados en diferentes ideas para realizar el aprendizaje de manera más rápida. Los más instructivos y conocidos son *Momentum, RMSProp* y *Adam*, pero el que generalmente es reconocido como el más rápido y mejor que los otros es el optimizador *Adam* [Rud16].

Uno de los problemas más importantes que aparece a la hora de entrenar una red neuronal es el llamado sobreajuste, más conocido como *overfitting*. Esto ocurre ya que la flexibilidad de las redes neuronales le permite aprender patrones del ruido, errores o incluso datos incorrectos. Este error suele ocurrir cuando se aprenden las características específicas de una base de datos concretas, haciendo que no sea capaz de predecir el resultado en otros datos que representan situaciones diferentes de las representadas en los datos de entrenamiento.

Existe también el efecto contrario denominado como *underfitting* que ocurre cuando el modelo no puede capturar adecuadamente la estructura fundamental de los datos.

Para solucionar el problema del *overfitting* existen varias técnicas. La más sencilla es la de incrementar los datos de entrenamiento con una mayor calidad. Existen maneras de mejorar la calidad de los datos de entrenamiento dependiendo del tipo de dato que sea. Por ejemplo, en el caso de las imágenes se puede cambiar el tamaño para que todas sean iguales, rotar la imagen, aumentarla, quitar el ruido, cambiar el color o convertirlas a blanco y negro, iluminar o oscurecer las imágenes, etc.

Otra de las técnicas para solucionar el *overfitting* es la regularización. Este término ha evolucionado mucho con el tiempo. La primera definición viene de los años 90 donde se reservaba solo como un término de penalización de la función de pérdida [B+95]. Más tarde ganó un significado mucho más amplio. Por ejemplo en [GBCB16] se define como c̈ualquier modificación que hacemos a un algoritmo de aprendizaje cuya intención es reducir su error de testeo pero no su error de entrenamiento.̈ También se entiende la regularización un método creado para solucionar el *overfitting* . Existen numerosos métodos de regularización, pero se verán los tres más comunes y conocidos: ℓ_1,ℓ_2 y dilución, más conocido como *dropout*. El *dropout* no todo el mundo lo considera como un método de regularización, pero como su objetivo también es solucionar el *overfitting* se considerará de tal manera en este trabajo.

Antes de estudiar que son la regularización ℓ_1 y ℓ_2, se introduce la notación para denotar la norma ℓ_p. Definimos la norma ℓ_p de un vector x con componentes x_i como:

$$\|x_p\| = \sqrt{\sum_i |x_i|^p} \quad p \in \mathbb{R} \tag{3.24}$$

Donde el sumatorio se realiza sobre todas las componentes del vector x.

Se empezará por el más instructivo de los métodos de regularización, ℓ_2. La regularización ℓ_2 consiste en añadir un término a la función de coste cuyo objetivo es reducir la complejidad del modelo cuando se tiene un número grande de características en la base de datos. Si se denota como w como el vector de pesos (incluyendo el sesgo) de nuestra red y m el número de observaciones y dada una función de coste $J(w)$ se definirá

la nueva función de coste como:

$$\widetilde{J}(w) = J(w) + \frac{\lambda}{2m}\|w\|^2 \tag{3.25}$$

El término adicional,

$$\frac{\lambda}{2m}\|w\|^2$$

se llama término de regularización y es exactamente la norma ℓ_2 de w al cuadrado multiplicado por el factor constante $\frac{\lambda}{2m}$. λ es el llamado parámetro de regularización.

La regularización ℓ_1 es análoga a la regularización ℓ_2 salvo que el término de regularización es

$$\frac{\lambda}{m}|w|$$

La idea de estos métodos es cambiar la función de coste para que los pesos se actualicen hacia cero, haciendo que la red sea menos compleja.

Por último, la idea del *dropout* es diferente: durante la fase de entrenamiento, se omiten o desactivan ciertas neuronas de una capa con una probabilidad definida. En cada iteración, se omiten diferentes neuronas, haciendo que en cada iteración se entrene a una red diferente.

Capítulo 4

Estado del Arte

En este capítulo se exponen los avances más recientes de la investigación de la estimación de edad en diferentes trabajos, mostrando las diferentes formas de abordar este tema. Cabe destacar que para realizar este estudio se ha centrado en la estimación a partir de la imagen de la cara de una persona. Antes de ver formas de estimar la edad de una persona, primero se mostrarán cuáles son los factores influyen en el reconocimiento de la edad en la Sección 4.1, para luego pasar a clasificar los diferentes trabajos en función de las técnicas utilizadas para crear el modelo en la Sección 4.2.

4.1. Factores que Influyen en el Reconocimiento de la Edad

Como se sabe, existen muchos factores que influyen a la hora de estimar la edad de una persona. Tanto la ropa que lleva, como la postura que tiene, puede cambiar la opinión que se tiene a primera vista de la edad que tiene una persona. En [ALLS18] se dice que el error humano a la hora de medir la edad una persona varía entre 2.07 y 8.62 dependiendo de diversos factores como la edad de la persona que lo mide, la edad del sujeto y la diferencia entre los dos. Normalmente se sobreestima la edad de los jóvenes [PS75]. Se suele estimar con una edad más próxima a la edad de la persona que estima y el propio género y las expresiones faciales también impactan en esta estimación [VELR12]. Cuánto más neutra es la expresión facial, más precisa es la estimación. También ocurre que se estima mejor la edad de una persona con el mismo género que el que la estima.

El objetivo es ver cuáles son los factores que influyen en mayor medida a un ordenador a la hora de estimar la edad de una persona. En [ALLS18] se ve que en el modelo conocido como DEX [RTVG15] y otros algoritmos como el de Microsoft Azure Cognitive Services [DS18] y Amazon AWS Rekognition [dCP20], el MAE es significativamente menor a la hora de estimar la edad de un hombre. En [VELR12] se demuestra que en las expresiones faciales de felicidad normalmente se subestima la edad, mientras que las sonrisas, el ceño

fruncido, la sorpresa y la risa pueden llegar a introducir líneas faciales que se confunden con arrugas, por lo que se subestima la edad de la persona. Otro de los factores principales es el ruido de las imágenes, en [FR13] se estudia su comportamiento aleatorio que afecta a las imágenes debido al brillo, color, etc. También el maquillaje influye en la estimación. En [DCR12] se vio que la presencia de maquillaje puede esconder ciertas imperfecciones faciales causadas por la edad como las arrugas y lugares oscuros que provocan que se subestime la edad. En [WLLK13] también se estudia como la raza de la persona influye a la hora de predecir su edad. Por último en [ABL+20] se estudio la correlación que tenían estos factores en menores de edad, viendo como variaba para las diferentes edades.

4.2. Técnicas Utilizadas para el Reconocimiento de la Edad

Hay muchas maneras de clasificar los trabajos realizados sobre este tema. Una posible clasificación se puede realizar viendo en qué método se basan para dividir los datos. Por un lado están los métodos basados en regresión, donde tratan las etiquetas de edad como valores numéricos y utilizan un regresor para calcular la edad. Por otro lado están los métodos de clasificación, donde se tratan diferentes edades o grupos de edades como etiquetas de clases independientes como en [ALKS20]. Por último aparecen los métodos basados en ranking, donde se utilizan series de clasificadores binarios simples para determinar el rango de la edad dada una cara. Sin embargo, parece más intuitivo dividir en función de cómo se ha realizado el modelo.

Al comienzo del estudio de la estimación de la edad de una persona se optó por un enfoque más humano del reconocimiento. Para ello, se utilizaban algunos de los rasgos faciales como ojo, nariz y boca y calculaban el tamaño y la distancia entre ellos como se puede ver en [KdVL99]. También existen técnicas como el Principal Componnent Analysis (PCA) y el Linear Discriminant Analysis (LDA) usadas en multitud de trabajo como [GGFH09], [SC09] y [GZSM07] utilizadas para elegir las mejores características y así conseguir disminuir la dimensión de los datos. El problema de este tipo de aproximación es que se necesitan imágenes muy claras y precisas.

Para mejorar este enfoque, surgió un nuevo enfoque basado en mejorar la calidad de las imágenes y de las características para luego utilizar un clasificador o regresor. Con este fin, se han usado diferentes técnicas como los llamados filtros de Gabor explicados en [GGFH09] y otros trabajos centrado en la extracción y mejora de la calidad de las características como en [FH08]. Además, también se utilizan las técnicas de PCA y LDA entre otras para reducir la dimensión de los datos como en [GFDH08] y [ALG+17]. Como clasificador se han usado diferentes técnicas como las SVM en [CLZ+12] y [EEH14a].

En último lugar ha surgido un nuevo enfoque basado en CNNs y otras técnicas de DL. Este último enfoque ha demostrado tener una gran precisión en los resultados sin

necesidad de tener un conocimiento específico de la materia muy amplio. La única pega de este enfoque es que se necesitan una gran cantidad de imágenes para entrenar el modelo ya que en una red neuronal profunda existen una gran cantidad de parámetros variables. Este último enfoque es el más utilizado actualmente ya que presenta mejores resultados.

De esta manera, se dividirán los trabajos según se enfoquen primero en extraer las características para luego usar un clasificador, llamado Aprendizaje Superficial (del inglés *Swallow Learning* (SWL)) o si se basan en una técnica de DL.

4.2.1. Aprendizaje Superficial

Proviene del término ingles SWL, consiste en extraer las características utilizando diferentes técnicas y luego clasificar las características extraídas utilizando un clasificador o regresor.

4.2.1.1. Extracción de Características

Después de empezar basándose en un enfoque más humano donde se extraía información sobre las arrugas y formas de la cara, en 2002 de la mano de [LTC02] se empezó a utilizar el algoritmo de Active Appearance Model (AAM), el cual se basa en extraer de manera conjunta la variación de la forma y textura de las caras humanas. Este algoritmo pasó a ser el modelo más utilizado para extraer las características durante unos años como se ve en [CCH10], [GYZ13], [LRBS09] y muchos más. Más tarde. en [GZSM07] se propuso el método Aging Pattern Subspace (AGES) donde se usa una secuencia de caras de un individuo en diferentes momentos de su vida para representar el proceso de envejecimiento.

Para mejorar la extracción de la información local de la cara se propuso un método basado en Spatially Flexible Patch (SFP) en [YLH08], donde se codifica la etiqueta de la edad, la apariencia local y la correspondiente coordenada espacial en un vector, para luego entrenar un Gaussian Mixture Modelling (GMM) para modelar las variaciones de todos los vectores [YLH08]. También se han utilizado otros métodos como el conocido en inglés como *Age Manifold* visto en [FXH07], [FH08] y [GFDH08]. En este método se aprende un patrón de edad de diferentes individuos en vez de un único individuo en diferentes edades. También se utilizan los Patrones Binarios Locales (del inglés Local Binary Patterns (LBP)) mostrado en [CLD13] y [GN08] ya que son más robustos frente a iluminación y efectivos para el análisis de texturas y clasificación.

Otro de los métodos ya comentados en la introducción de esta sección es la de los Filtros de Gabor, explicado en [GGFH09].

Más en la actualidad se han empezado a utilizar las CNNs para realizar la extracción de características llevando a modelos con un rendimiento robusto como en

[LLF+19],[ALG+17] y muchos más.

4.2.1.2. Determinación de la Edad

La estimación de la edad se trata de un problema de clasificación multiclase, por lo que es posible utilizar algoritmos tradicionales como las SVM ([CLZ+12], [EEH14a]) y k-NN ([XYXZ09]). Sin embargo, estos algoritmos no tienen en cuenta la relación que hay entre las etiquetas de las edades, pero en la estimación de la edad es un problema más serio predecir la edad de una persona de 30 años como una de 10 que como una de 25.

Para solucionar ese problema, se puede considerar como un problema de regresión ya que puede que genere mejores estimaciones de la edad, por ello se han utilizado técnicas como el Gaussian Process Regression (GPR) ([ZY10]) y Support Vector Regression (SVR) ([GGFH09]).

Otra opción es combinar varios clasificadores y regresores para capturar mejor el complicado proceso de envejecimiento y mejorar el rendimiento de la estimación [GFDH08].

También se han usado otras técnicas más particulares como en [LTC02] donde primeramente se clasifica la cara en un grupo de apariencias similares o en un grupo de edades y luego se utiliza una función de determinación de la edad creada específicamente para la predicción de la edad en ese grupo.

En [LLF+19] utilizan una serie de regresores locales diseñados utilizando la técnica de divide y vencerás, dando resultados diferentes, para luego pasar los resultados por una red de puertas que le dan un peso a los diferentes regresores locales.

4.2.2. Modelos Basados en Aprendizaje Profundo

La ventaja de los modelos de DL frente a los anteriormente explicados, es que no se necesita un proceso para seleccionar las características, sino que las características son seleccionadas automáticamente por la aplicación. La desventaja de este tipo de técnicas es que es necesaria una base de datos muy grande y un gran poder computacional. Sin embargo, se ha visto que los métodos basados en DL proporcionan una mayor precisión con respecto a los otros métodos, pero es muy difícil de interpretar qué características han sido usadas para llegar a un nivel más alto de precisión.

La mayoría de los modelos basados en CNNs utilizan arquitecturas conocidas como AlexNet, GoogleNet, ResNet y VGGNet preentrenadas normalmente en la base de datos de ImageNet [RDS+15]. En muy pocos trabajos se trata de desarrollar un nuevo modelo de CNNs desde 0.

El primer ejemplo que se explica es el modelo llamado AgeNet [LLK+15] donde se ha propuesto una CNNs profunda de 22 capas para estimar la edad que usa una combinación

de regresión y distribución de etiquetas basado en clasificación para estimar la edad final. También propone un método que ayuda a evitar el *overfitting* en una base de datos pequeña. Sin embargo, requiere tener un entrenamiento separado para modelos de regresión y clasificación.

Uno de los estudios más importantes es el [RTVG15], sistema llamado *Deep Expectation* (DEX). Utiliza como arquitectura base la red *VGG-16* preentrenada en ImageNet y luego optimizan el modelo en imágenes de caras con etiquetas de edad. La red *VGG-16* usa 16 capas entrenables con un tamaño de filtro pequeño de 3x3. Los resultados mostraron una mejora frente a hacer una regresión de la edad usando CNNs.

En [CZD+17] se propuso una arquitectura de CNNs basada en *ranking*. En esta se usa una combinación de arquitecturas CNNs entrenada en etiquetas de edad ordinales. Las salidas de las CNNs individuales se combinan para predecir la edad final. Este enfoque de estimación del error parece obtener mejores resultados en comparación con el enfoque de clasificación multiclase. El rendimiento de este método se evaluó con la base de datos MORPH.

En [DLL18] se propuso el modelo llamado CNN2ELM, como un modelo más complejo que incorpora CNNs y *Extreme Learning Machine* (ELM). Consiste en tres arquitecturas CNNs Age-Net, Gender-Net y Race-Net para extraer características relacionadas con la edad, género y raza de una imagen de una misma persona. La arquitectura está preentrenada en la base de datos de ImageNet. Luego utiliza el clasificador ELM para agrupar las edades y el regresor ELM para la estimación de la edad. Se optimizó en la base de datos de IMDB WIKI y tiene un rendimiento mucho mayor que la mayoría de arquitecturas y bases de datos conocidas. Esto se tiene porque utiliza fusión de decisiones para conseguir una decisión más robusta. Esta aproximación encuentra más rasgos distintivos de la imagen y luego combina la predicción en ellos para estimar la edad. Sin embargo, su rendimiento fue malo en el caso de una base de datos con fotos de menor calidad que en la que no se muestre la cara de frente.

Una limitación que afecta a todos los métodos anteriores es la poca cantidad de fotos de caras etiquetadas disponibles para la estimación de la edad. Como consecuencia en [DLL18] se propone una técnica de aumento de datos (del inglés *data augmentation*) para incrementar el tamaño de los datos de entrenamiento para la estimación de la edad. Para ello obtienen nuevas imágenes de entrenamiento aplicando una pequeña transformación como traslación, rotación o dar la vuelta a imágenes ya existentes de la base de datos.

En [ZGZC19] los autores propusieron un sistema de estimación de la edad combinando CNNs con LSTM. A este sistema le llamaron *Recurrent Age Estimation*. La CNNs se utilizo para encontrar características distintivas de las imágenes mientras que la LSTM se utilizaba para aprender los patrones de envejecimiento de una secuencia de características personalizadas. Para solucionar el problema del *overfitting* utilizan el llamado *Label*

Distribution Learning (LDL), método utilizado para representar las etiquetas y se basa en funciones de distribución de probabilidad condicionada. La mayor desventaja de este modelo es que necesita una base de datos bastante específica ya que necesita fotos de la cara de una persona en diferentes edades, preferiblemente en edades cercanas para que tenga una mayor utilidad la incorporación de la LSTM.

Por último, pero no por ello menos importante, se mostrará una tabla comparativa de los diferentes modelos explicados a lo largo de la sección.

4.2.3. Comparación de los Diferentes Modelos

En la Tabla 4.1 se muestra una comparativa de la precisión entre diferentes modelos explicados anteriormente. Cabe destacar que en el modelo AgeNet aparece la precisión medida en porcentaje ya que divide en 8 clases con diferentes grupos de edad y trata de adivinar a que grupo de edad pertenece el individuo. Se puede ver los modelos basados en DL tienen una mayor precisión que los basado en SWL como ya comentamos anteriormente.

Tabla 4.1: Comparación de los diferentes modelos

Modelo	Bases de datos usadas	Rendimiento
SFP [YLH08]	Concurso de YAMAHA [FH08]	MAE = 8.17
AGES [CZD+17]	FG-Net [PLTC16]	MAE = 6.22
	MORPH [RT06]	MAE = 8.07
BridgeNet [LLF+19]	MORPH II	MAE = 2.38
	FG-Net	MAE = 2.56
AgeNet [LLK+15]	Proporcionada por el concurso ICCV2015	Precisión en la edad: 61.3 %
Deep Expectation (DEX) [RTVG15]	IMDB-WIKI Dataset [RTG18]	MAE = 3.3345
Ranking CNN [CZD+17]	MORPH	MAE = 2.96
CNN2ELM [DLL18]	MORPH	MAE = 2.61
Recurrent Age Estimation [ZGZC19]	MORPH	MAE = 1.32
	FG-Net	MAE = 2.19

Capítulo 5

Algoritmo de Preprocesamiento de Imágenes

El método propuesto en este trabajo consiste en la creación de un proceso de pre-procesamiento de imágenes. Para probar este pre-procesamiento es necesario la creación de varios modelos para ver el efecto que tiene dichas mejoras en el entrenamiento de los modelos. El flujo seguido para realizar la contribución corresponde con la Figura 5.1. En la Sección 5.1 se crean dos modelos diferentes para probar las técnicas de pre-procesamiento de imágenes. Uno de los modelos es creado desde 0 mientras que el otro se basa en el Transfer Learning (TL). En la Sección 5.2 es donde se explican y se implementan las técnicas de pre-procesamiento de imágenes aplicadas. Este pre-procesamiento se compone de una primera parte donde se recortan, centran y alinean las caras y una segunda parte donde se aplican otras mejoras a la calidad de la imagen como puede ser la normalización y el aumento de datos. En las siguientes secciones se explicarán cada uno de los componentes de la Figura 5.1.

Por tanto, el objetivo del trabajo es analizar la eficacia de las técnicas de pre-procesamiento de imágenes propuestas en los modelos creados. El trabajo se puede encontrar en el siguiente enlace: `https://gitlab.fdi.ucm.es/miguel.franqueira/ageestimation`.

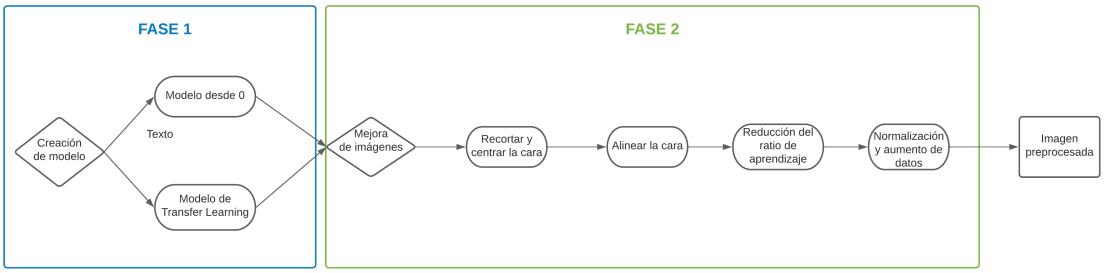


Figura 5.1: Esquema del proceso del método creado para realizar este trabajo.

5.1. Fase 1

Para probar las técnicas de pre-procesamiento de imágenes es necesario probarlo con un modelo en específico. Esto es debido a que cada modelo tiene un número diferente de parámetros que entrenar y, por tanto, un proceso de entrenamiento diferente. Por ello, en esta primera fase se explicarán los dos modelos creados para la experimentación. El primero de ellos sencillo, con muy pocos parámetros sin estar pre-entrenado, por lo que es más sensible a las imágenes de entrada. El segundo modelo está basado en TL, por lo que tiene muchos más parámetros entrenables y aporta más robustez ante los datos de entrada ya que se encuentra pre-entrenado en la base de datos *ImageNet*. Esta elección se debe a la búsqueda de ver las consecuencias del proceso de pre-procesamiento de imágenes en diferentes tipos de modelos. Por tanto, en esta fase se explica la estructura de ambos modelos con los que posteriormente se harán pruebas en la Sección 6.2.

5.1.1. Modelo 1: Red desde 0

Para realizar el primer modelo con el que experimentar se quiso crear una red convolucional muy sencilla, con muy pocos parámetros, ya que las pruebas se realizarán con una cantidad de imágenes no muy alta. De esta manera, la eficacia del proceso de pre-entrenamiento se verá más claramente.

Por ello, la red consiste únicamente en una primera capa convolucional con 32 filtros con tamaño de *kernel* 3×3 y función de activación ReLU, una segunda capa de *max pooling* con tamaño de filtro 2×2 y paso 2 con un *dropout* de 0,5, y por último una capa totalmente conectada con una única neurona con función de activación lineal.

Esta red es una de las redes convolucionales más sencillas. Se utiliza como función de activación de la última capa la función lineal (también llamada función de activación identidad explicada en la Sección 3.1.2) debido a que en la experimentación se llegó a la conclusión que el modelo aprendía mejor que utilizando la función de activación *softmax*.

5.1.2. Modelo 2: Red Basada en Aprendizaje por Transferencia

Para la creación del segundo modelo se quiso crear un modelo más robusto a los datos de entrada y más parecido a los creados en los diferentes artículos científicos para probar la eficiencia del proceso de pre-procesamiento de imágenes en un modelo más real y utilizado. Por ello, se optó por utilizar el aprendizaje por transferencia.

El aprendizaje por transferencia (del inglés TL) es un enfoque muy popular en donde una red entrenada para una tarea se reutiliza para una segunda tarea relacionada con la primera. Es muy útil y utilizada en reconocimiento de imágenes, ya que típicamente las primeras capas aprenden a detectar características genéricas y las últimas a detectar

características más específicas. Por tanto, se aprovecha la idea de que la red base (la entrenada para realizar otra tarea) ya ha aprendido a extraer las características más genéricas por lo que se evita la necesidad de aprenderlas otra vez. Sin embargo, para mejorar las predicciones hay que optimizar las predicciones de la nueva red para el caso específico. Esto se hace consiguiendo que la red aprenda características más específicas (típicamente pasa en las últimas capas de la red) que tienen que ver con la nueva tarea a resolver. Cuando la base de datos utilizada es mucho más pequeñas que la base de datos con la que se entrenó la red base, es una herramienta muy potente para evitar el *overfitting* de tu base de datos de entrenamiento. El proceso para llevar a cabo un modelo de TL se muestra en la Figura 5.2.

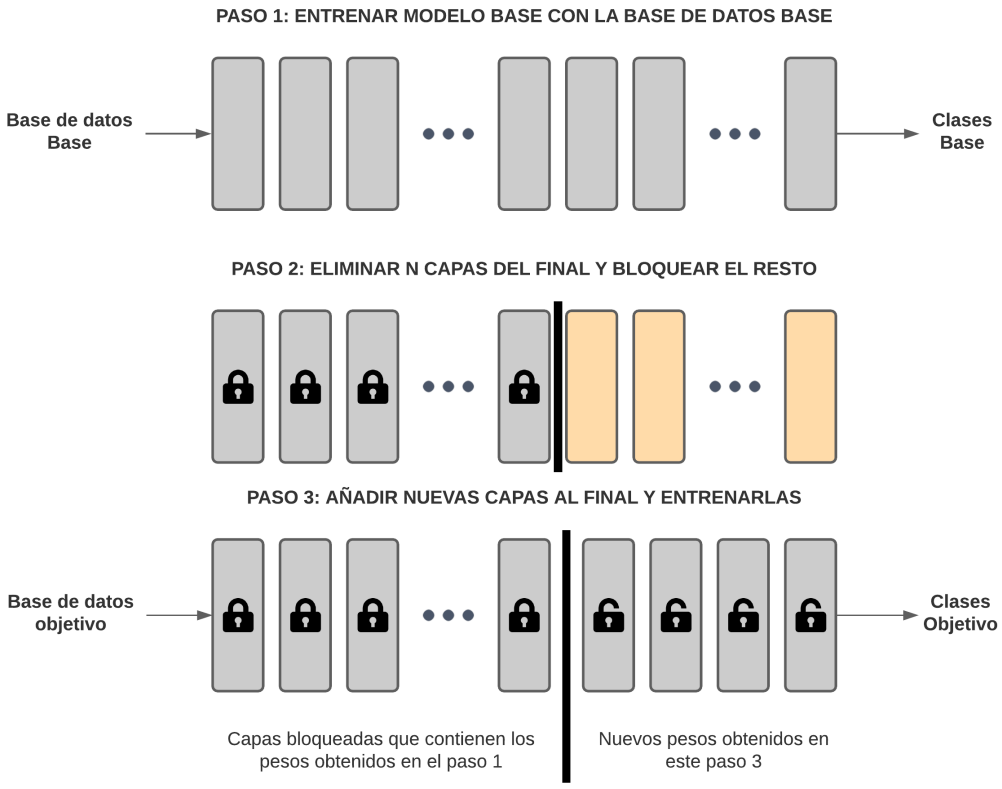


Figura 5.2: Representación esquemática del proceso de aprendizaje por transferencia

La principal ventaja de este enfoque es que el entrenamiento de un modelo de DL requiere una gran cantidad de recursos. La mayoría de las redes disponibles para su uso requirieron un entrenamiento en la base de datos de *ImageNet* de miles de horas de Graphics Proccessing Unit (GPU). Normalmente esto no es posible para los investigadores sin el hardware y los conocimientos necesarios. Con *Keras*, es realmente fácil su uso y permite resolver problemas de clasificación de imágenes con una precisión que no sería posible de otra manera. A continuación se explican las dos redes base principalmente utilizadas.

La primera red base utilizada es la llamada *Inception v3* [SVI+16] que, como su nombre indica, es la tercera edición de la red *Inception* [SLJ+15] realizada por Google y tiene una profundidad de 48 capas con 24 millones de parámetros entrenables. La red está preentrenada en más de un millón de imágenes de la base de datos *ImageNet*. Como resultado, la red ha aprendido una gran cantidad de representaciones de característica para un amplio espectro de imágenes. El principal foco de esta versión es el de utilizar menos potencia computacional modificando las arquitecturas *Inception* anteriores.

La segunda red utilizada fue la llamada *ResNet* [HZRS16] la cuál es una de las redes más utilizadas para realizar tareas de reconocimiento de imágenes. Cuando se entrenan redes profundas, llega un punto en el que el aumento de la profundidad hace que la precisión se sature y se degrade rápidamente. La red *ResNet* surge para combatir ese problema y resulta sencilla de optimizar y conseguir una precisión alta cuando la profundidad de la red incrementa, produciendo unos muy buenos resultados que hace que sea muy utilizada. Existen diferentes arquitecturas con distinto número de capas pero la utilizada para entrenar este modelo se trata de la *ResNet-50* que como su propio nombre indica se compone de 50 capas con más de 23 millones de parámetros entrenables. Al final su uso fue descartado en la experimentación

Para el segundo modelo, por tanto, se utilizó la red *Inception v3* a la que se le unió las mismas capas que la red creada desde 0 en la Sección 5.1.1, salvo que entre la capa de *max pooling* y la totalmente conectada, se añadió una nueva capa totalmente conectada con tantas neuronas como el número de edades diferentes disponibles en la base de datos y como función de activación la función *softmax*, como última capa se utilizó una única capa con función de activación lineal.

5.2. Fase 2

Después de un análisis de las diferentes bases de datos que contienen imágenes etiquetadas con la edad de una persona, se vio que muchas de las imágenes no contenían solo la cara y muchas de ellas contenían mucho ruido, por lo que era necesaria su modificación. Con el objetivo de mejorar la calidad de dichas imágenes y mejorar el entrenamiento de nuestro modelo se proponen diferentes mejoras y procesos que realizar sobre las imágenes. A continuación, se muestran las diferentes mejoras realizadas sobre las imágenes.

La primera de las mejoras realizadas fue el reescalado de las imágenes, ya que las diferentes bases de datos utilizaban imágenes de diferentes tamaños por lo que ha sido necesario llegar a un tamaño de imagen común con el que entrenar los modelos.

5.2.1. Recorte, Centrado y Alineamiento de la Cara

Una tarea muy importante a realizar es conseguir extraer la cara de las imágenes de entrada. Para ello, hay que reconocer la cara, recortarla de la imagen y centrarla en la nueva imagen. Para realizar este proceso se han utilizado diferentes técnicas basadas en DL incorporadas en el modelo de *DeepFace* [SO20], el cual permite realizar la detección facial con diferentes librerías y métodos como los de *OpenCV* [Bra00], *MTCNN* [ZZLQ16], *dlib* [Kin09] y *Single Shot Detector* (SSD) de una manera mucho más rápida y sencilla haciendo más fácil la programación y la comparación de los mismos.

Para realizar el recorte de las imágenes se extraen cuatro puntos sobre la imagen indicando las diferentes esquinas de la región en la que aparece la cara. Estos cuatro puntos forman un rectángulo que luego se utiliza para extraer la cara recortada. Para extraer la ubicación de los puntos se realiza un reconocimiento facial, donde se extrae la ubicación de diferentes características de la cara como pueden ser los ojos, la nariz, la boca y las orejas. Una vez reconocidas las diferentes características, se aplica un algoritmo basado en la ubicación de dichas características para devolver la posición de los cuatro puntos utilizados para recortar la cara. Existen diferentes algoritmos utilizados dependiendo de las características reconocidas, pero la mayoría se basan en ver la distancia entre las ubicaciones de las diferentes características para establecer la posición de los cuatro puntos.

Una vez recortada y centrada la cara, se observó que había muchos casos donde las caras aparecían giradas. Por ello, es necesario alinearlas para así conseguir que todas tengan más o menos la misma posición de los rasgos faciales. Para realizar este alineamiento es necesario reconocer la posición de los ojos. Una vez se sabe la posición de los ojos se traza una línea entre ambos que nos proporciona un ángulo de giro con el que rotar la imagen. Es decir, el segmento horizontal que pasa por un ojo y el segmento que une ambos ojos determina un ángulo que es utilizado como giro en la imagen, por notación se llamará a dicho ángulo de giro θ.

Un píxel de coordenadas (x_1, y_1) en la imagen tendrá como coordenadas en la imagen girada:

$$x_2 = \cos\theta * x_1 + \sin\theta * y_1 \tag{5.1}$$

$$y_2 = -\sin\theta * x_1 + \cos\theta * y_1 \tag{5.2}$$

Siendo (x_2, y_2) las nuevas coordenadas en la imagen girada. Los píxeles en la nueva imagen que no provienen de ningún píxel de la imagen inicial se rellenan en negro.

Una vez obtenida la imagen de la cara recortada, centrada y alineada se observó la posibilidad de aplicar más mejoras en la calidad de la imagen y el modelo para obtener mejores resultados.

5.2.2. Otras Mejoras de la Calidad de Imagen

Para mejorar la calidad de la imagen de las caras obtenidas en la sección anterior y disminuir la probabilidad de *overfitting* se añadieron al pre-procesamiento de las imágenes diferentes técnicas. A continuación se explicarán en detalle las técnicas utilizadas para mejorar la calidad de las imágenes.

- **Normalización**: Consiste en reescalar el valor de los píxeles de las imágenes para que tengan un nuevo rango de valores. Es un proceso clave a la hora de preprocesar imágenes y es realizado en casi todos los modelos de pre-procesamiento. En la mayoría de casos el reescalado de los píxeles se hace para que tengan valores entre 0 y 1. En el caso imágenes guardadas con el modelo de color RGB en los ordenadores, los valores de los píxeles contendrán numeros entre 0 y 255 por lo que el proceso de reescalado simplemente consistirá en dividir el valor de cada píxel en cada canal (existen tres canales cada uno correspondiente con un color primario) entre 255. De esta manera los valores de los píxeles estarán entre 0 y 1 como se quería.

- **Aumento de Datos**: proviene del término inglés *data augmentation* y es utilizado para evitar el *overfitting* consiguiendo nuevos datos de entrenamiento. El aumento de datos consiste en generar nuevas imágenes a partir de imágenes ya existentes aplicando algún tipo de transformación a ellas y utilizar las nuevas imágenes generadas como datos de entrenamiento. Las transformaciones más comunes son:

  - *Desplazar la imagen* un número determinado de píxeles horizontal o verticalmente. Si se tiene un píxel (x_1, y_1) para desplazarlo verticalmente se le tendrá que restar o sumar al valor x_1 un valor denotado por w dependiendo de si queremos subir o bajar la imagen respectivamente. Normalmente el valor w se calcula en función del largo de la imagen (un porcentaje de él). El caso de desplazar verticalmente es análogo cogiendo la segunda coordenada y_1 y el ancho de la imagen para establecer el valor w.
  - *Girar la imagen.* La manera de girar una imagen ya ha sido vista en la Sección 5.2.1 cuando se vio la manera de alinear la cara.
  - *Cambiar el brillo.* Existen numerosos algoritmos para realizar este proceso. Existen varios algoritmos para el caso de tener imágenes con el modelo de color RGB, el más sencillo de ellos añade un valor a cada uno de los canales si es posible para aclarar la imagen, mientras que se le resta un valor para oscurecerla. Esto es debido a que los colores correspondientes a valores más próximos al 255 de cada uno de los colores son más claros que los que tienen valores cercanos a 0.
  - *Aplicar zoom.* Existen diferentes algoritmos y maneras de aplicarlo, pero la

manera más sencilla consiste en recortar una parte de la imagen y crear una nueva imagen con ese recorte.

- *Dar la vuelta a la imagen.* Se puede realizar tanto verticalmente como horizontalmente. Si se quiere dar la vuelta a la imagen verticalmente, es decir, si se tiene la foto de la cara, que aparezca ahora la cara dada la vuelta, con la frente abajo y la boca arriba, se tendrá que realizar lo siguiente: para cada y entre 0 y la mitad del ancho se recorre cada x entre 0 y el largo de la imagen y se intercambian los valores de los píxeles en la posición (x, y) con los de la posición $(x, largo - y)$.

También se tratarán de realizar cambios en la red neuronal que no tienen que ver con la calidad de la imagen como es el ajuste de hiperparámetros. En un modelo de ML un hiperparámetro (del ingles *hyperparameter*) es un parámetro del modelo que se utiliza para controlar el proceso de aprendizaje. Para modificar sus valores en la librería de *Python* llamada *Keras* se utilizan los llamados *Callbacks*. En el caso de las CNNs, los principales hiperparámetros son:

- El ratio de aprendizaje, el cual indica cuánto actualizar los pesos en el algoritmo de optimización. Se pueden establecer diferentes métodos para reducir el ratio de aprendizaje y así optimizar el modelo. El más comúnmente utilizado es el de multiplicarlo por un valor de 0.1 cuando el la función de pérdida en validación no mejora entre dos *epoch* consecutivos.
- El número de *epoch*, es decir, el número de veces que todos los datos de entrenamiento pasan por la red neuronal. Se suele incrementar su número hasta que hay una diferencia muy pequeña entre el error de validación y el de entrenamiento.
- El tamaño de *batch*, es decir, el número de imágenes que pasan a la vez por el modelo.
- Inicialización de pesos. Se suele inicializar los pesos con valores aleatorios pequeños para prevenir neuronas "muertas".

El objetivo del Ajuste de Hiperparámetros (del inglés *Hyperparameter Tuning*) consiste en escoger los hiperparámetros óptimos para conseguir una mayor precisión del modelo.

Por tanto a las imágenes se le harán los siguientes cambios:

- Encontrar la cara en la imagen, recortarla, centrarla y por último alinearla.
- Reescalar el tamaño de las imágenes para que todas tengan tamaño común.
- Aplicarle técnicas de aumento de datos como girar la imagen, aplicar zoom, recortar la imagen, desplazar píxeles, etc.

- Normalizar los píxeles para que todos tengan valores entre 0 y 1.

Una vez modificadas las imágenes, también se harán mejoras a nuestros modelos como cambiar el ratio de aprendizaje en tiempo de ejecución y variar el número de *epoch*.

5.3. Tecnologías

Para realizar este trabajo, resultó de vital importancia la utilización de diversas librerías y herramientas. Estas tecnologías permiten facilitar el pre-procesamiento de las imágenes y la creación de los modelos en los que probar dicho pre-procesamiento. A continuación se explicarán las más importantes.

5.3.1. Librerías

Tanto para crear los modelos como para realizar el pre-procesamiento de las imágenes se ha utilizado como lenguaje de programación *Python*. Debido a la amplitud de tareas a realizar en el proyecto, se han utilizado diversas librerías para facilitar el proceso. Seguidamente, se explican las que más se han utilizado para el desarrollo.

- **Tensorflow** [Ten20]: Es una biblioteca de código abierto para ML desarrollada por *Google* para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar patrones. Puede ser utilizada para una gran cantidad de tareas pero se centra en el entrenamiento y detección de redes neuronales profundas. Las características que hacen que sea tan utilizadas son las siguientes: ejecuta eficientemente operaciones tensoriales de bajo nivel en la Central Processing Unit (CPU), GPU o Tensor Processing Unit (TPU), calcula el gradiente de expresiones diferenciables arbitrarias, escala el cálculo a muchos dispositivos, exporta programas (gráficos) a herramientas externas como servidores, navegadores y dispositivos móviles. Se ha utilizado en el desarrollo del trabajo ya que incorpora librerías muy útiles como *Keras* y *Cuda*.

- **Keras** [Ker20]: Es una biblioteca de código abierto escrita en *Python* y construida sobre *Tensorflow*. Es la Application Programming Interface (API) de alto nivel de *TensorFlow* 2: una interfaz accesible y altamente productiva para resolver problemas de aprendizaje automático, con un enfoque en el aprendizaje profundo moderno. Proporciona abstracciones esenciales y bloques de construcción para desarrollar y enviar soluciones de aprendizaje automático con una alta velocidad de iteración. Permite la creación y el entrenamiento de modelos de redes neuronales de manera muy fácil y sencilla.

Esta es la librería más utilizada en el desarrollo del trabajo, se ha utilizado para todas las tareas relacionadas con el procesamiento, creación y entrenamiento del modelo. Incorpora numerosas funciones de alto nivel que permiten crear capas y modelos de redes neuronales de manera sencilla, a su vez también incorpora funciones para poder optimizadores, funciones de pérdida y entrenarlos de manera muy sencilla. También incorpora diferentes funciones para el procesamiento de las imágenes de entrada, incluyendo funciones para permitir la normalización y el aumento de datos de las imágenes.

- **CUDA** [CUD20]: CUDA son las siglas de Compute Unified Device Architecture (CUDA) y es una plataforma de computación paralela y un modelo de programación desarrollado por *NVIDIA* para la computación en GPU. CUDA permite a los desarrolladores acelerar drásticamente las aplicaciones aprovechando la potencia de cómputo de las GPU. En las aplicaciones aceleradas por la GPU, la parte secuencial de la carga de trabajo se ejecuta en la CPU -que está optimizada para el rendimiento de un solo hilo- mientras que la parte de la aplicación que requiere más cálculo se ejecuta en miles de núcleos de la GPU en paralelo. Al utilizar CUDA, los desarrolladores programan en lenguajes populares como *C*, *C++*, *Fortran*, *Python* y *MATLAB* y expresan el paralelismo a través de extensiones en forma de unas pocas palabras clave básicas. CUDA permite su uso en *Python* mediante *Tensorflow* por lo que se ha utilizado para agilizar el entrenamiento de las redes neuronales creadas. Esto es debido a que los experimentos se realizaron en un equipo propio, por lo que la necesidad de la mejora de velocidad de los entrenamientos era vital.

- **Pandas** [pan21b]: Es una librería de *Python* utilizada oara el tratamiento de datos. Está escrita como una extensión de *Numpy* [num20] para la manipulación y análisis de datos. Permite leer y escribir fácilmente en ficheros en formato Comma-Separated Values (CSV), *Excel* y bases de datos *SQL*, permite acceder a los datos mediante índices o nombres para filas y columnas, ofrece métodos para reordenar, dividir y combinar conjuntos de datos, permite trabajar con series temporales y realiza todas estas operaciones de manera muy eficiente [pan21a]. En este trabajo se ha utilizado para la escritura y lectura de ficheros en formato CSV así como el tratamiento de los datos en esos ficheros, sobre todo a la hora de extraer la edad correspondiente a cada una de las imágenes.

- **OpenCV** [ope20]: Es una librería de software libre desarrollada por *Intel* dedicada al CV. Es una de las librerías más populares dedicadas al CV, siendo muy utilizada a nivel comercial, desde Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota,etc. La librería tiene más de 2500 algoritmos, que incluye algoritmos de ML y de CV para usar. Estos algoritmos permiten identificar objetos, caras, clasificar acciones humanas en vídeo, econtrar imágenes similares, reconocer escenarios y un

multitud de funcionalidades más. En este trabajo se ha utilizado para procesar imágenes, incluyendo el cambio de tamaño,su lectura y escritura y también los algoritmos de reconocimiento facial.

- **dlib** [Kin09]: Es una biblioteca de código abierta escrita en *C++* que contiene algoritmos de ML y herramientas para crear software complejo para resolver problemas reales. Es usada en la industria y el mundo académica en un amplio espectro de campos incluyendo robótica, dispositivos integrados, teléfonos móviles y en grandes entornos informáticos de alto rendimiento. También contiene algoritmos de reconocimiento facial de gran precisión. En el proyecto se ha utilizado como principal herramienta para el reconocimiento facial ya que ha demostrado tener los mejores resultados entre las diferentes técnicas probadas.

- **Glob** [Glo20]: Es un módulo de *Python* que encuentra todos los nombres de ruta que coinciden con un patrón especificado según las reglas utilizadas por el shell de *Unix*, aunque los resultados se devuelven en un orden arbitrario. Acepta el uso de diferentes caracteres como es el caso de que se utiliza para permitir cualquier secuencia de caracteres en su lugar. Ha resultado muy útil en el trabajo para encontrar las imágenes de una carpeta u subcarpetas.

- **Shutil** [shu20]: Es un módulo de *Python* que ofrece varias operaciones de alto nivel en archivos y colecciones de archivos. En particular, provee funciones que dan soporte a la copia y remoción de archivos. Muchas veces se utiliza junto con el módulo *os* [os20] que permite operaciones para archivos individuales. Muy utilizado en el trabajo para copiar imágenes entre diferentes carpetas e incluso para borrar subdirectorios sobrantes.

5.3.2. Herramientas

Además de utilizar librerías, se ha optado por utilizar diferentes herramientas y modelos ya creados, los cuáles ayudaban a la hora de realizar la detección de los rostros en una imagen. Vamos a centrar nuestro trabajo en explicar las dos más utilizadas.

- **DeepFace** [SO20]:Es un sistema de reconocimiento facial basado en DL creado por un grupo de desarrollo de Facebook. Engloba muchos de los *frameworks* más utilizados para el reconocimiento facial. Tiene varias utilidades como la verificación facial (ofrece la posibilidad de verificar si un par de caras pertenecen a la misma persona o a personas diferentes), buscar las caras de la base de datos con la que la cara de entrada corresponde, análisis de rasgos faciales y un multitud de características más. Para realizar cada una de estas tareas utiliza las técnicas más novedosas y con mejor rendimiento, convirtiéndose en uno de las herramientas más importantes sobre el tema.

Para el reconocimiento facial utiliza modelos que alcanzan una gran precisión como es el caso de FaceNet [SKP15] (99.65 %), ArcFace [DGXZ19] (99.40 %), VGG-Face [PVZ15] (98.78 %) y Dlib (99.38 %).

- **Face_recognition** [fac18]: Es una herramienta dispoible para reconocer y manipular caras con *Python*. Está construida utilizando la herramienta de reconocimiento facial de *dlib* basada en DL. Simplifica mucho el uso de esta librería en *Python* permitiendo encontrar diferentes caras en las imágenes, encontrar y manipular características faciales e identificar si dos rostros pertenecen a la misma persona. Se ha utilizado para realizar algunos test de reconocimiento de imágenes, pero al encontrar el sistema de *DeepFace* se sustituyó por este último. Otra de sus utilizaciones en el trabajo consistió en la localización de diferentes caras en un vídeo, así como filtrar las caras pertenecientes a diferentes personas, de manera que se guarda una única cara por cada persona que aparece en el vídeo.

Capítulo 6

Experimentos y Resultados

En este capítulo se describirán los experimentos realizados evaluando el rendimiento y la eficacia de cada uno de ellos. En la Sección 6.1 se presenta el contexto de la experimentación, seguida de la Sección 6.2 donde se explican las bases de datos estudiadas para realizar dicha experimentación. En la Sección 6.3 se realiza una evaluación de los modelos creados en la Sección 5.1, así como una explicación de cómo se llegó a la creación de los mismos. En la Sección 6.4 se realizan experimentos para probar el rendimiento de las técnicas de pre-procesamiento descritas en la Sección 5.2. En último lugar, en la Sección 6.5 se presenta una comparación entre los diferentes experimentos.

6.1. Contexto de la Experimentación

Antes de pasar a definir los experimentos, es necesario entender qué camino seguir para conseguir hallar la edad de una persona. Lo primero que hay que tener en cuenta es que para que una imagen sirva de entrada para una red neuronal, es necesario primero procesarla, ya que no todas las imágenes tienen la misma calidad o posición de la cara. Sobre todo el tamaño de imagen es muy importante ya que tiene que coincidir con el tamaño de la primera capa de la red neuronal.

Otro aspecto importante es que se va a trabajar con modelos basados en CNNs, realizando al principio modelos desde 0 y más adelante creando modelos basados en TL. Se verá que estos últimos son más eficientes ya que ya han sido preentrenadas con una gran cantidad de imágenes que permiten tener un comportamiento robusto ante tareas similares.

Por último, comentar que todas las pruebas se han hecho en un único equipo. El equipo utilizado es un equipo personal, por lo que fue necesario configurarlo con librerías como *CUDA* y *CUDNN* para que funcione con la GPU y así mejorar la velocidad de los experimentos. Este entorno de experimentación utilizado se podría resumir con las

características descritas en la Tabla 6.1.

Tabla 6.1: Entorno de experimentación

CPU	Memoria RAM	Tarjeta Gráfica
Intel Core i7-10710U 4.7GHz	16 GB	NVIDIA Geforce GTX 1650Ti Max Q

6.2. Bases de Datos Utilizadas en los Experimentos

Antes de ver los experimentos realizados, es necesario introducir algunas de las bases de datos más utilizadas así como los modelos utilizados para realizar los experimentos.

Uno de los principales problemas encontrados a la hora de realizar la experimentación es la escasez de bases de datos con imágenes de buena calidad etiquetadas con la edad de la persona. También se ha encontrado el problema de que las mejores son de pago como es el caso de [RT06], conocida como *MORPH* utilizada por la gran mayoría de los trabajos que investigan este tema junto con su posterior versión *MORPH Album II*. Otro problema es que la cantidad de imágenes que hay disponible es muy reducida, haciendo más difícil el entrenamiento de la red neuronal creada.

Después de experimentar con la librería *Keras*, la manera más sencilla que se encontró para reconocer la etiqueta de cada imagen es crear una carpeta con cada edad y meter en cada carpeta las imágenes cuya edad corresponda con la escrita en carpeta. Para realizar esta clasificación se tuvo que emplear diferentes procesos para las diferentes bases de datos.

- **FG-NET**: Una de las bases de datos utilizas es la llamada FG-NET [PLTC16] compuesta de 1002 imágenes de la cara de 82 personas de una edad entre 0 y 69. Es una de las bases de datos más utilizadas a la hora de realizar la estimación de la edad. No está disponible para descargar desde su página oficial pero se puede encontrar en otras fuentes. La principal desventaja de esta base de datos es la poca cantidad de imágenes y la principal ventaja es que tiene una buena calidad de imágenes.

- **MORPH**: La base de datos MORPH [RT06] está compuesta de 55,134 imágenes de la cara de 13,617 personas de una edad entre 16 y 77 años. Es la base de datos más utilizada entre los trabajos de estimación de la edad. La gran desventaja es que tiene un costo de 99 dólares por lo que no ha podido ser utilizada en este trabajo.

- **Adience**: Otra de las bases de datos utilizadas para muchos trabajos es la llamada Adience [EEH14b] compuesta de 26,580 personas de 2,284 personas diferentes, etiquetadas en 8 grupos de edades diferentes. Esta base se descartó para su uso porque en este trabajo los modelos se entrenan para averiguar la edad exacta y no un rango de edad.

- **IMDB-WIKI**: La base de datos IMDB-WIKI [RTG18] tiene la cantidad de imágenes más grande entre todas las bases de datos etiquetadas con la edad de la persona. Compuesta de 523,051 imágenes recolectadas de dos fuentes: IMDb (460,723 imágenes) y Wikipedia (62,328 imágenes). La edad de los artistas varía entre 1 y 99 años. La gran ventaja de esta base de datos que hace que sea muy utilizada es su gran cantidad de imágenes, resolviendo uno de los mayores problemas que se tiene a la hora de entrenar una red neuronal convolucional. Se trató de analizar la base de datos de IMDB-WIKI, tanto con las imágenes de la parte de IMDb como las de Wikipedia. Ambas tienen el problema de que vienen etiquetadas en un archivo *.mat* correspondiente al programa *Matlab* y como el objetivo era realizar el trabajo en *Python* se realizó un programa para pasar del archivo de *Matlab* a un formato CSV que resulta más manejable en *Python*. En este proceso también se trató de limpiar etiquetas con ruido, como valores nulos, y se calculó la edad de cada imagen a partir de los atributos de fecha de nacimiento y fecha de la foto proporcionadas en el archivo *.mat*. Después de un estudio de los resultados de esta clasificación se vio que la cantidad de datos mal etiquetados o de imágenes con una gran cantidad de ruido era demasiado grande como para poder ser utilizada en nuestro trabajo. Por lo tanto se ha preferido no utilizar esta base de datos ya que se ha valorado antes la calidad antes que la cantidad, siendo los datos mal etiquetados algo no deseable en este estudio. Por tanto se ha descartado su uso.

- **AgeDB**: La base de datos AgeDB [MPS+17] compuesta de 16,488 imágenes de 568 personas de una edad principalmente entre 5 y 95 años. Esta base de datos no es tan conocida como las anteriores, pero las imágenes tienen un buen etiquetado y la calidad de imagen no es mala. Por ello, dado que tiene una buena cantidad de imágenes, se ha usado para realizar este trabajo y así ver la diferencia una vez hayamos procesado las imágenes. Resultó sencillo extraer la edad ya que venía la edad en el propio nombre de la imagen, haciendo más sencillo el proceso de colocación en carpetas

- **UTKFace**: Una base de datos surgida en los últimos años es la llamada UTKFace [ZQ17] compuesta de más de 20,000 imágenes de una edad entre 0 y 116 años. Una gran ventaja de esta base de datos es que tiene una gran variedad de poses, expresiones faciales, iluminación y resolución. Todo esto unido con que la calidad de las imágenes es bastante grande, la ha convertido en una de las principales bases de datos utilizadas en el trabajo. En esta, también resultó sencillo extraer la edad ya que venía la edad en el propio nombre de la imagen. Cabe destacar que para la base de datos de UTKFace se incluyen dos bases de datos diferentes disponibles para descarga, la primera con las imágenes completas y la segunda con las imágenes con las caras recortadas. En los primeros experimentos se utilizaron las imágenes con las caras recortadas, pero más adelante se utilizaron las imágenes completas para

probar nuestro preprocesamiento y modelos.

- **APPA-REAL**: La última base de datos que se comentará es la llamada APPA-REAL [EA17] compuesta de 7591 imágenes de personas diferentes con un rango de edad entre 0 y 95 años. La gran ventaja que presenta es el amplio espectro de edades que comprende así como su buen etiquetado y calidad de imagen. Por estos dos motivos se ha utilizado para el entrenamiento de nuestros modelos. La base de datos viene dividida en tres carpetas, correspondiente al entrenamiento, validación y testeo. Para saber la edad correspondiente a cada imagen es necesario analizar un archivo en formato CSV con diferentes columnas. Una vez extraída la edad de cada una de las imágenes se coloca en una nueva carpeta que lleva como nombre su edad correspondiente.

Existen otras muchas bases de datos diferentes, pero la mayoría incluyen imágenes con ruido, mal etiquetadas o con un rango de edad reducido, haciendo más difícil su utilización para entrenar un modelo más general. La Tabla 6.2 muestra una comparación entre las diferentes bases de datos explicadas anteriormente.

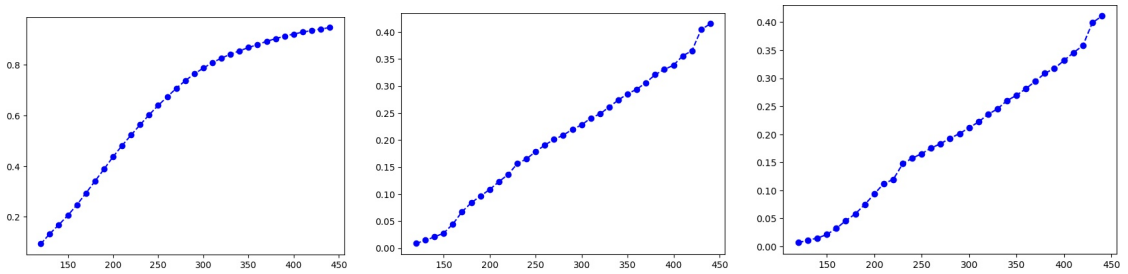
Tabla 6.2: Comparación de las diferentes bases de datos

Base de datos	Imágenes	Rango de edad
FG-NET	1,002	0-69
MORPH	55,134	16-77
Adience	26,580	8 grupos de edades
IMDB-WIKI	523,051	1-99
AgeDB	16,488	5-95
UTKFace	24,106	0-116
APPA-REAL	7,591	0-95

Otro de los procesos realizados fue el análisis del tamaño de imagen que tenían las fotos de las diferentes bases de datos. En un inicio se optó por confiar en la calidad de las imágenes que proporcionaban las diferentes bases de datos por lo que el procesamiento de las imágenes consistía en comprobar su tamaño y reescalar el tamaño para que todas tuviesen un tamaño común. Para realizar este proceso fue necesario realizar un análisis de la distribución de los tamaños de las diferentes imágenes para quedarse con un número lo suficientemente grande de ellas, el resultado del análisis en las bases de datos de AgeDB y APPA-REAL se muestra en la Figura 6.1.

Antes del análisis se intentó hacer funcionar los modelos con un tamaño de imagen muy bajo de 120x120 píxeles ya que permitía descartar muy pocas de las imágenes, pero después del análisis de los diferentes trabajos se llegó a la conclusión de que era más adecuado un tamaño de imagen de 230x230 píxeles ya que te permiten extraer mucha más información de la imagen. Las fotos con un tamaño menor al propuesto se descartan ya que no tiene

sentido reescalar una imagen a un valor de píxeles mayor, mientras que las que tienen una calidad más alta se reescalan para conseguir unificar el tamaño de las imágenes.



(a) AgeDB (b) Imágenes de entrenamiento de APPA-REAL (c) Imágenes de validación de APPA-REAL

Figura 6.1: Porcentaje de imágenes perdidas para diferentes tamaños de imagen

6.3. Evaluación de los Modelos de la Fase 1

Para realizar este trabajo se ha optado por la utilización de modelos construidos desde 0 en su mayoría. Esto es debido a que el objetivo no es conseguir la mayor precisión si no ver cómo evoluciona el entrenamiento en los diferentes modelos, así como ver como las mejoras propuestas para la calidad de imagen afectan a esta precisión.

En la primera fase, se optó por entrenar diferentes modelos sin incluir casi ninguna mejora en la calidad de las mismas. La única mejora que se hizo fue el reescalado de las imágenes para que todas ellas tuviesen el mismo tamaño.

6.3.1. Primeros Modelos

En los primeros modelos con los que se probó, se utilizó una red neuronal convolucional muy sencilla, para permitir observar el comportamiento de las bases de datos con un modelo sencillo. Un esquema básico del primer modelo se muestra en la Figura 6.2. Esta red se basa en la construida en [LH15] haciendo una ligera modificación eliminando la normalización de capas incluidas en el modelo propuesto por dicho trabajo. Después de muchas pruebas diferentes con las bases de datos de AgeDB, APPA-REAL y UTK-Face se vio que el modelo no tenía capacidad de aprendizaje tal como se construyó con la librería *Keras* de *Tensorflow* ya que el MAE permanecía constante tanto para los datos de entrenamiento, como los de validación en los diferentes *epoch*.

En estos experimentos también se probaron diferentes tamaños de imágenes y optimizadores para realizar el entrenamiento del modelo, dando lugar a diferentes valores pero siempre sin realmente aprender. Los valores del MAE variaba entre 26 (con un tamaño de imagen de 230x230) hasta 36 (con un tamaño de imagen de 120x120). También se

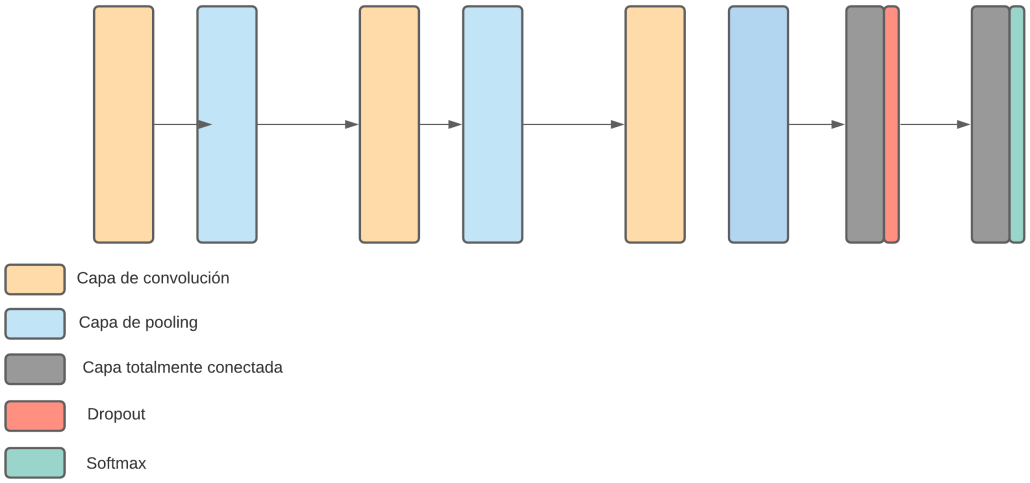


Figura 6.2: Esquema simplicado del Modelo 1

observó que el optimizador *Adam* disminuía ligeramente el valor del MAE frente a otros optimizadores.

También sirvió este proceso de creación para familiarizarse con la librería *Tensorflow* de *Python*, probando multitud de alternativas para realizar tanto el procesamiento de las imágenes y las etiquetas, como diferentes optimizadores, compiladores, funciones de error y demás características que te proporciona esta herramienta.

6.3.2. Modelos Más Complejos

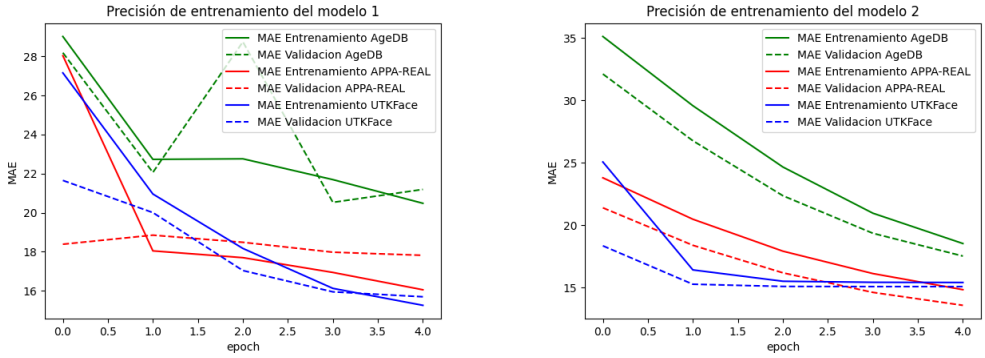
Dado que los anteriores modelos no funcionaban correctamente a pesar de haber modificado todos los datos de entrada, optimizadores, métricas y otras características, se llegó a la conclusión de que el modelo no era correcto por lo que se empezaron a diseñar nuevos modelos. Estos modelos parten del TL

En una primera versión del TL se probó a añadirle dos capas totalmente conectadas, una primera de 80 neuronas con una función de activación ReLU y una segunda con el número de clases y como función de activación la función *softmax*. Después de testear este nuevo modelo los resultados tampoco fueron buenos y después de un poco de estudio se llegó a la conclusión de que lo que no era correcto era utilizar la función de activación *softmax* en la última capa con un total de neuronas igual al número de edades diferentes de la base de datos. Esta es la razón por la que se diseñaron los modelos descritos en la Sección 5.1 con la última función de activación es una lineal.

Una vez llegados a este punto se pasaron a realizar las pruebas con los modelos descritos en la Sección 5.1.

Se realizó una primera prueba con un número de *epoch* reducido de 5 para probar la

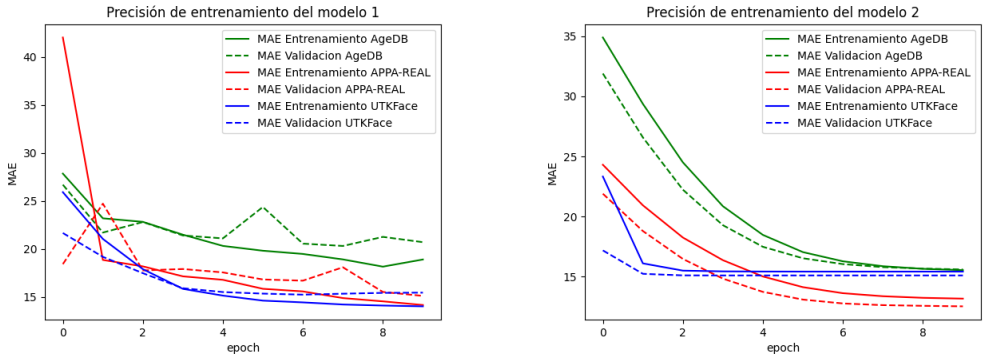
evolución de los dos modelos. Ambos modelos se entrenaron sobre las bases de datos de AgeDB, APPA-REAL y UTKFace dividiendo los datos en un 80 % para entrenamiento y un 20 % para validación, siendo la división de los datos de manera aleatoria. Para medir la precisión de los modelos se ha utilizado la función de error MAE. El resultado de entrenamiento de ambos modelos se muestra en la Figura 6.3.



(a) Modelo 1 (b) Modelo 2

Figura 6.3: Evolución del MAE durante el Experimento 1

Se observó que ambos modelos mejoraban al aplicarles más entrenamiento, por lo que se pasó a incrementar el número de *epoch* con los que se entrenaban ambos y así ver si alguno de los dos en algún momento convergía a un resultado. Durante este primer experimento, el resultado del entrenamiento de ambos modelos fue muy parecido debido al número reducido de *epoch* entrenado. En el segundo experimento se entrenaron ambos modelos durante 10 *epoch*. El resultado del experimento se puede ver en la Figura 6.4.



(a) Modelo 1 (b) Modelo 2

Figura 6.4: Evolución del MAE durante el Experimento 2

Durante este segundo experimento ya se muestra un comportamiento extraño a la hora de entrenar el primer modelo, habiendo picos de subida en algunos *epoch*, que denota la poca consistencia del modelo, al haberse construido con un número muy limitado de capas.

A su vez su precisión es menor para todas las bases de datos que la del modelo 2.

El modelo 2 se muestra más consistente, con un comportamiento que hace ver que tanto la validación como el entrenamiento van a tender a un valor de MAE. Un caso especial es el de la base de datos UTKFace que tras mejorar durante los dos primeros *epoch* permanece prácticamente constante.

Otra observación a tener en cuenta es que, tanto en la primera prueba como en la segunda, parece que ambos modelos tienden a tener una mayor precisión para la base de datos de APPA-REAL, lo que normalmente ocurre por la mejor calidad de los datos.

Por último, ya que el modelo de TL parece más prometedor, se probó a cambiar la red base *Inception V3* por una red *ResNet50* pero después de un entrenamiento se llegó a la conclusión de que era peor ya que el MAE de validación permanecía constante a pesar de solo haber cambiado la red base y tener el resto de capas de igual manera que el modelo creado con la red *Inception V3*. Por lo tanto, para el resto de los experimentos se usaron los modelos 1 y 2 previamente descritos y utilizados.

6.4. Fase 2

En esta fase probaremos la eficiencia y las técnicas descritas en la Sección 5.2. Para ello se realizarán primero unas pruebas para asegurar la calidad de las técnicas de recorte y centrado de las caras, para luego pasar a probar los modelos con las técnicas propuestas.

6.4.1. Recortar y Centrar la Cara

Uno de los problemas principales encontrados a la hora de estudiar las bases de datos es que la posición de las caras de las personas no estaba fija en las fotos y la mayoría de las fotos incluían mas de una persona. Por ello fue necesario ver herramientas que permitieran la detección de las caras de las imágenes. Esta investigación llevo a varios modelos de Github que realizaban esta tarea con una gran eficiencia. Algunos de los modelos más famosos son el *MTCNN*, los basados en la utilización de la librerías *OpenCV* y *dlib*, los modelos basados en SSD y otros modelos de diferentes repositorios.

Para probarlos se han descargado y entendido su funcionamiento para poder luego guardar la imagen de la cara recortada y así entrenar los modelos propuestos. Para realizar esto, primero se tuvieron que coger todas las imágenes de las diferentes bases de datos, aplicándole una función que permitiese extraer los datos de ubicación de las caras en las imágenes y luego a partir de los datos de la ubicación, guardar las imágenes de las caras recortadas en una nueva carpeta.

Una vez se realizó este proceso, se encontró el modelo *DeepFace* previamente explicado

en la Sección 5.2. Durante la fase de prueba con las diferentes bases de datos, el modelo *MTCNN* no llegaba a reconocer la cara de las personas, por lo que se descartó su uso. Un ejemplo de recorte con los diferentes métodos es la Figura 6.5.

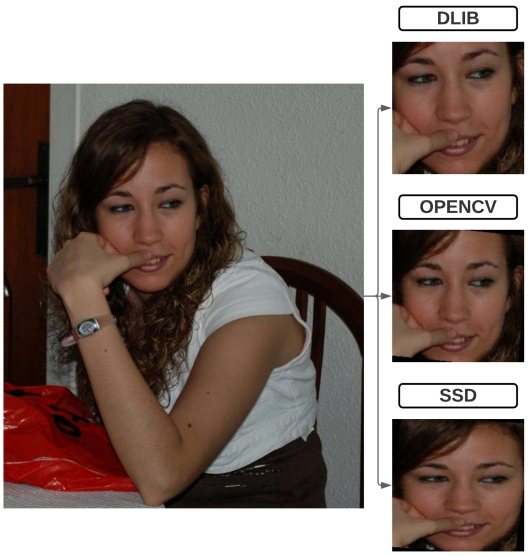


Figura 6.5: Recorte de la cara con diferentes métodos

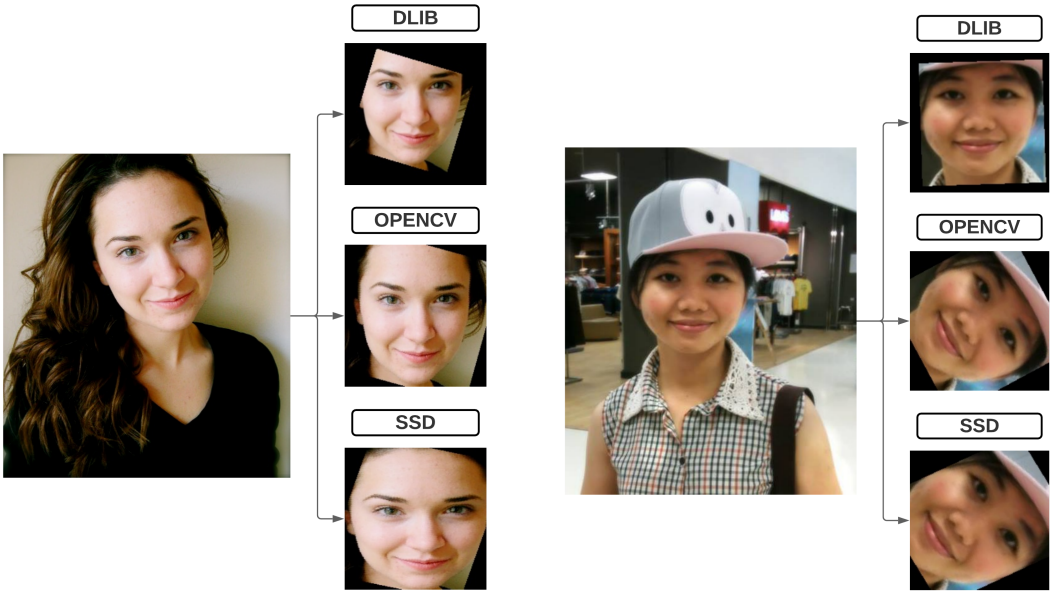
El resultado de la experimentación con las diferentes bases de datos usadas para entrenar los modelos se muestra en la Tabla 6.3. Cabe destacar que no se ha podido encontrar la base de datos AgeDB entera por lo que se ha utilizado una versión reducida de la misma y en los resultados se observa que el método basado en la utilización de *dlib* es el más consistente de ellos. El modelo basado en *OpenCV* también ha reconocido la mayoría de las caras, aunque en menor medida que en el caso de *dlib*. Por último, el modelo basado en SSD no ha sido de reconocer la mayoría de las caras de la base de datos de AgeDB, en el caso de la base de datos APPA-REAL ha llegado a reconocer menos de la mitad de las caras, pero en el caso de UTKFace si que ha reconocido una amplia mayoría. Esto muestra la necesidad de tener una gran calidad de imagen para que el modelo de SSD reconozca de manera efectiva las caras de las personas.

Tabla 6.3: Comparación del número de caras detectadas por cada modelo

Base de datos	**Número de imágenes iniciales**	**dlib**	**OpenCV**	SSD
AgeDB	7,143	6,761	5,661	395
APPA-REAL	7,591	6,190	5,153	3,319
UTKFace	24,108	23,372	22,172	23,469

6.4.2. Alineación de la Cara

Otro de los problemas de las imágenes es que muchas de las caras aparecían giradas, dificultando el aprendizaje de la red neuronal. Por ello se decidió buscar maneras de alinear todas las caras para así hacer más uniforme el conjunto de imágenes. Esto llevó al estudio de modelos como el de [BT17], pero este se descartó ya que no se centraba en realizar la tarea si no que más bien se centra en la extracción del contorno de la cara y las diferentes características para modelos en 2D y 3D. Más tarde se descubrió que el modelo de *DeepFace* incluye el alineamiento de las caras con los diferentes modelos, basado en la posición de los ojos para así establecer una línea entre los dos ojos y determinando el grado de rotación de la imagen. En la Figura 6.6(a) se ve un ejemplo de alineamiento de la cara con los diferentes métodos. Como se puede observar, *dlib* añade bastante ruido a la imagen ya que al alinear la cara los bordes de la imagen los deja en negro, lo cual puede ser perjudicial para el entrenamiento del modelo. Sin embargo, es mucho más preciso que los otros dos métodos a la hora de alinear la cara, ya que los otros dos pueden fallar en ocasiones como se puede ver en la Figura 6.6(b).



(a) Alineamiento con diferentes métodos (b) Mal alineamiento

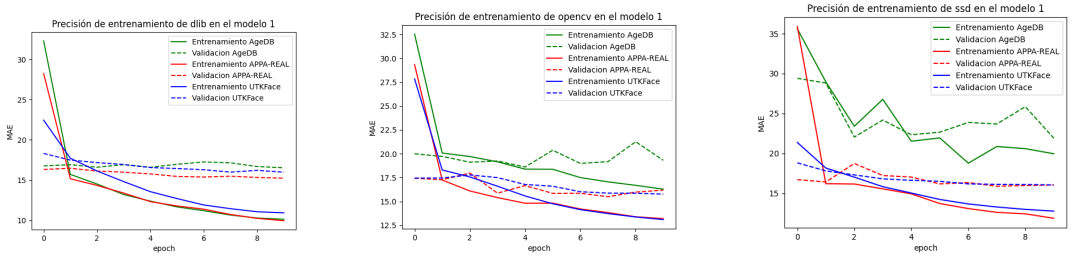
Figura 6.6: Ejemplos de alineamiento de la cara

El recorte y alineamiento de la cara permite extraer la información más relevante de la imagen, es decir, se consiguen las caras de todas las imágenes centradas y rectas.

6.4.2.1. Resultado del Entrenamiento con las Caras Recortadas, Centradas y Alineadas

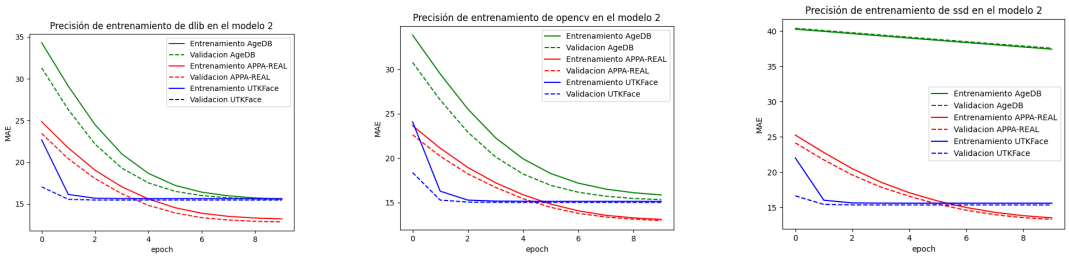
Para ver el cambio en el entrenamiento de los dos modelos con las diferentes técnicas para recortar, centrar y alinear la cara, se realizó un tercer experimento con las imágenes extraídas de los procesos anteriores. Para realizar el entrenamiento se optaron por utilizar las mismas características que en el entrenamiento de la Fase 1, cambiando únicamente los datos de entrada para así poder observar la diferencia entre ambos. El resultado de este entrenamiento para el Modelo 1 se puede ver en la Figura 6.7 y para el Modelo 2 en la Figura 6.8.

Como ya se observó en la anterior fase, se ve que el entrenamiento del modelo 1 es mucho menos consistente, habiendo grandes diferencias en algunos casos en el MAE como en el caso del entrenamiento en dlib donde hay una diferencia de más de 5 puntos entre los valores de entrenamiento y validación. Aun así, los resultados del entrenamientos son mejores que los del segundo experimento en el modelo 1, sobre todo en el caso de *dlib*, por lo que se puede considerar como un éxito. Por otra parte, entre los tres modelos el que tiene un comportamiento más robusto es el de *dlib* ya que el reconocedor basado en SSD presenta unos resultados más parecidos a los de los primeros experimentos. Por otra parte, el modelo 2 presenta unos resultados muy parecidos tanto entre *dlib* y *OpenCV* como con los resultados del segundo experimento de la Fase 1. Esto refuerza la idea que los modelos de TL son mucho más robustos, ya que han aprendido a extraer muchas más características de las imágenes por lo que son más generalizables. En este segundo modelo, la diferencia de rendimiento no es significativa. Por último destacar que en este último modelo el reconocedor basado en SSD no consiguió unos buenos resultados, dando a entender su mal funcionamiento y la introducción de ruido ya que el modelo funciona mejor sin utilizarlo a pesar de ser un modelo más robusto.



(a) dlib (b) OpenCV (c) SSD

Figura 6.7: Experimento 3: Evolución del MAE con las caras recortadas en el modelo 1



(a) dlib (b) OpenCV (c) SSD

Figura 6.8: Experimento 3: Evolución del MAE con las caras recortadas en el modelo 2

6.4.3. Otras Mejoras de la Calidad de la Imagen

Se muestra ahora cómo se aplicaron las diferentes técnicas explicadas en la Sección 5.2.2 para mejorar la calidad de la cara y disminuir la probabilidad de *overfitting*.

Se han probado a modificar los siguientes hiperparámetros de los modelo propuestos:

- El ratio de aprendizaje: Para modificarlo en los modelos, se utilizó un *Callback* que redujese el ratio de aprendizaje si no había una mejora de 0.001 en el valor de la función de pérdida en la validación, multiplicando el valor del ratio de aprendizaje por un factor de 0.1. Para comprobar por primera vez el valor de la función de pérdida se espera 5 *epoch* y cada vez que se cambia el valor del ratio de aprendizaje se espera 4 *epoch* antes de volver a cambiarlo para permitir adecuar a la red neuronal al entrenamiento con este nuevo ratio de aprendizaje. Esta modificación se realiza para generalizar más el modelo y permitir hacerlo más robusto. Para notar la diferencia de esta modificación es necesario entrenar el modelo con un alto número de *epoch*.
- El número de *epoch*: En el entrenamiento no se ha podido modificar e incrementar mucho su valor debido a las limitaciones del equipo y la gran cantidad de modelos a entrenar.
- El tamaño de *batch*: En este caso se utilizará un valor de *batch* de 2, debido a la limitación de memoria de la GPU utilizada, ya que la disminución del tamaño de *batch* disminuye el uso de memoria.
- Inicialización de pesos: En este caso cuando se utilice aprendizaje por transferencia no será necesario ya que los pesos del modelo base vienen inicializados del preentrenamiento en ImageNet. Para el modelo 1 se inicializan de manera aleatoria.

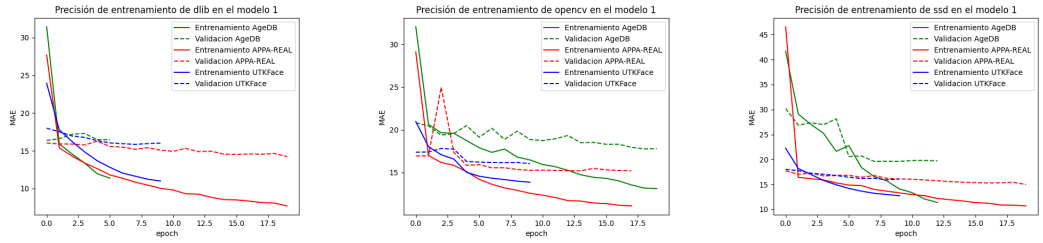
Existen otros hiperparámetros como la función de activación, número de capas ocultas, el número de neuronas de cada capa, el *dropout* y la elección del optimizador que están más ligados a la construcción de la red neuronal que no se optimizarán en este trabajo.

También se ha probado a normalizar y realizar aumento de datos en nuestro modelos como se explicará en los experimentos posteriores.

6.4.3.1. Resultado del Entrenamiento

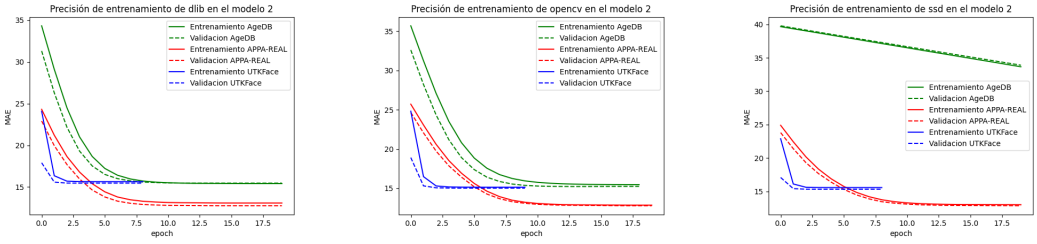
Para probar estas otras mejoras de calidad de la imagen se realizaron dos experimentos diferentes. Un primer experimento que probaba la disminución del ratio de aprendizaje y otro *Callback* que permite parar el entrenamiento en el caso de no detectar una mejora de la función de pérdida de un 1 % entre *epoch* dejando un mínimo de 6 *epoch* antes de parar el entrenamiento. El segundo experimento incorpora las técnicas de aumento de datos al modelo y normalización.

El resultado del primero de los experimentos se puede ver en las Figuras 6.9 y 6.10. En el caso del segundo modelo la introducción del *Callback* para parar el entrenamiento resulta muy beneficioso ya que se ve que el entrenamiento converge a un valor y simplemente se estaría haciendo un entrenamiento sin mucha utilidad. En el caso del modelo 1, la incorporación de este *Callback* no parece muy acertada debido a la fluctuación de los valores del MAE entre los diferentes *epoch*, parando el entrenamiento de algunos modelos que probablemente podían mejorar más su funcionamiento.



(a) dlib (b) OpenCV (c) SSD

Figura 6.9: Experimento 4: Evolución del MAE con las caras recortadas en el modelo 1

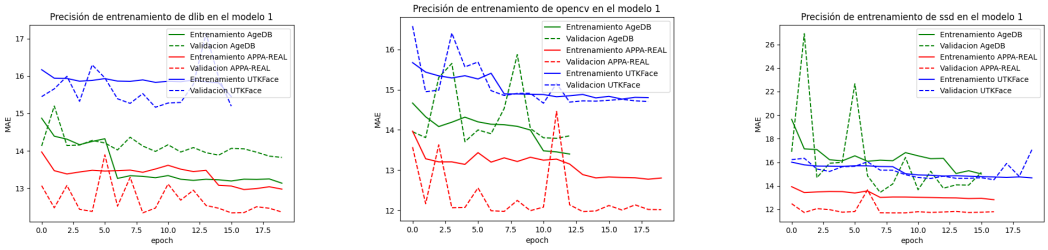


(a) dlib (b) OpenCV (c) SSD

Figura 6.10: Experimento 4: Evolución del MAE con las caras recortadas en el modelo 2

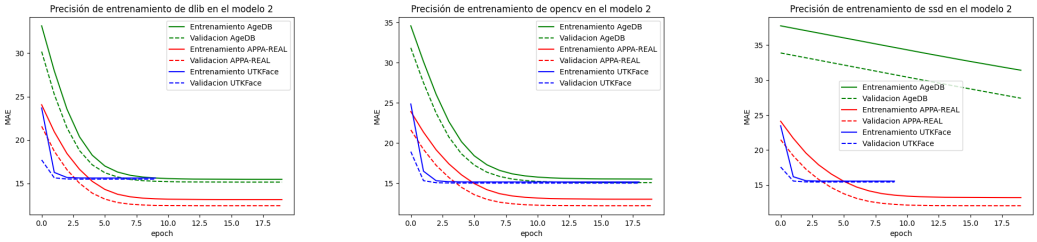
En el último experimento se reescalaron los píxeles para tener un valor entre 0 y 1 y se realizaron diferentes técnicas de aumento de datos. Se realizó una rotación de un máximo

de 40 %, un desplazamiento de un máximo de 20 % del ancho y largo de la imagen y un zoom máximo de 20 % a las imágenes. Todos estos cambios aplicados de manera aleatoria. Estos cambios se unieron a los ya realizados para entrenar un último modelo durante 20 *epoch* cuyo resultado se muestra en las Figura 6.11 para el modelo 1 y en la Figura 6.12 para el modelo 2. En este último experimento el comportamiento en el modelo 2 es análogo al del experimento anterior, remarcando que no importa tanto la calidad de las imágenes para un modelo basado en TL. El modelo 1 sin embargo, se ve que no mejora mucho los resultados frente al anterior experimento, habiendo una gran diferencia de valores entre las diferentes bases de datos y las diferentes técnicas utilizadas para recortar y centrar la cara. Una posible razón de esto es la escasez de entrenamiento del modelo, ya que aunque se han utilizado técnicas de aumento de datos, no se han entrenado los modelos con más imágenes, si no que se han entrenado con ellas modificadas.



(a) dlib (b) OpenCV (c) SSD

Figura 6.11: Experimento 5: Evolución del MAE en el modelo 1 con todas las mejoras



(a) dlib (b) OpenCV (c) SSD

Figura 6.12: Experimento 5: Evolución del MAE en el modelo 1 con todas las mejoras

6.5. Comparación de los Experimentos

Para comparar los diferentes modelos se cogerá la base de datos con mejores resultados con el fin de simplificar la tabla final. En este caso, la base de datos con mejor rendimiento es la de APPA-REAL. En el caso de los dos últimos experimentos, la técnica para centrar y alinear la cara con mejor rendimiento, es decir, dlib, ya que si no la tabla quedaría mucho más complicada y larga, siendo el principal objetivo de esta sección hacer un resumen

sencillo de los resultados de los experimentos mostrándolos de manera intuitiva. El valor mostrado del MAE en la tabla será el correspondiente al resultado después del último *epoch* tanto para entrenamiento como para validación. El resultado final de este estudio se muestra en la Tabla 6.4.

En la Tabla 6.4 se ve la mejora de los MAE durante el entrenamiento en los diferentes experimentos y modelos. El resultado resulta bastante positivo porque se ha llegado a bajar hasta 5 puntos en el valor de validación, sobre todo notando la diferencia de eficacia en el modelo 1 ya que está creado desde 0 y aun no tiene ningún peso aprendido, mientras que el modelo 2 al ser un modelo de TL ya viene con pesos preentrenados, por lo que es más consistente y difícil de mejorar debido a la calidad de los datos.

Tabla 6.4: Comparación de los diferentes experimentos

Experimento	**Entrenamiento/Validación**	**MAE**	
		Modelo 1	**Modelo 2**
Experimento 1	Entrenamiento	16.05	14.83
	Validación	17.81	13.58
Experimento 2	Entrenamiento	14.18	13.15
	Validación	15.13	12.51
Experimento 3	Entrenamiento	9.96	13.21
	Validación	15.24	12.85
Experimento 4	Entrenamiento	7.71	13.08
	Validación	14.25	12.75
Experimento 5	Entrenamiento	12.97	13.15
	Validación	12.36	12.45

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

A lo largo de este trabajo, se han visto diferentes maneras de crear modelos para estimar la edad de una persona a partir de una cara, incidiendo en los modelos basados en redes neuronales convolucionales. Se crearon varios modelos de prueba para dicha tarea, creando un pre-procesamiento para las imágenes con las que entrenar dichos modelos. El entrenamiento de estos modelos es muy costoso, requiriendo un gran poder computacional y muchas horas de trabajo. Para simplificar este proceso, se mostró que los modelos basados en TL actúan de manera mucho más robusta ante los datos de entrada, permitiendo ahorrar una gran cantidad de horas.

Sin embargo, la parte más importante ha sido la implementación de las técnicas de pre-procesamiento ya que han resultado muy eficaces a la hora de entrenar un modelo desde 0, permitiendo una mayor precisión en el entrenamiento de los modelos. Esta preparación previa de los datos y mejora de su calidad, puede servir para cualquier otro campo de estudio, lo que nos permite ver su gran utilidad. Este estudio muestra la importancia de tener una buena calidad de los datos de entrada a la hora de entrenar nuestro modelo, siendo muy necesario el gasto de tiempo en la mejora de estos datos.

7.2. Trabajo Futuro

Al ser la estimación de la edad una tarea muy novedosa, aún hay muchas vías de avance en la materia, como por ejemplo:

1. **Probar modelos de SWL**: A pesar de que durante toda la fase de experimentación se probaron varios modelos con diversas configuraciones, todos ellos estuvieron basados en técnicas de DL. Por tanto, utilizar modelos de SWL puede tener un gran

interés, ya que aunque los trabajos sobre este campo tienen una precisión menor, se podría observar mejor el efecto del pre-procesamiento en la etapa de la extracción de características.

2. **Aumento del número de modelos de prueba**: Otra posible vía de desarrollo es la experimentación de nuevos modelos, ya que existen diversos hiperparámetros como las funciones de activación, el número de capas ocultas, el número de neuronas de cada capa, el *dropout* de las capas y la elección de optimizador que se han modificado muy poco. Con respecto al TL existen también otras muchas redes disponibles para utilizar como red base en tu modelo como pueden ser *Xception*, *VGG19*, *ResNet152*, *InceptionResNet* y *MobileNet*.

3. **Aumento del tiempo de prueba de los modelos**: Al haber utilizado un equipo personal, la capacidad de entrenamiento del modelo ha sido limitado a pesar de haber utilizado una GPU. Una posible mejora del trabajo es aumentar el tiempo de prueba, aumentando el número de *epoch* y probando con diferentes tamaños de *batch* para los diferentes modelos.

4. **Incremento del número de bases de datos**: Una posible vía de desarrollo es aumentar el número de bases de datos utilizadas, así como la mezcla de varias de ellas para conseguir tener una mayor cantidad de datos para entrenar el modelo. Existen bases de datos como *MORPH*, *IMDB-WIKI* , *AFAD*, *MegaAge* y *AGFW* que pueden llegar a ser interesantes incorporarlas en el trabajo.

5. **Mejora del preprocesado de las imágenes**: Las técnicas vistas para el alineamiento y recorte de la cara a veces incorporan cierto ruido a la imagen, además de que en ciertas situaciones reescalan fotos de un tamaño menor que el objetivo, al tamaño objetivo. Este ruido resulta perjudicial para el entrenamiento del modelo, por lo que una posible vía de desarrollo puede ser la eliminación de ese ruido. También se pueden probar diferentes técnicas para cambiar algunas propiedades de la imagen como pueden ser el brillo, el contraste y el color. Para realizar esta tarea se podría crear un modelo basado en CNNs que sea entrenado para mejorar la calidad de las diversas imágenes y mejorando el resultado en el modelo.

6. **Análisis de vídeos**:En la experimentación se ha realizado un pequeño análisis de cómo se podrían extraer las distintas caras que aparecen en un vídeo. Para ello es necesario extraer todos los fotogramas de la imagen, y cada fotograma tratarlo como una nueva imagen. Para cada imagen se extraen las diferentes caras y se comparan con las caras extraídas en los fotogramas anteriores para buscar similitudes. De esta manera si dos caras en diferentes fotogramas pertenecen a la misma persona, solo se guarda una de ellas y se sigue con el análisis de los marcos. Para realizar esta tarea se ha utilizado el repositorio `github.com/ageitgey/face_recognition` que crea

un modelo basado en *dlib* para el reconocimiento y comparación de rostros en las imágenes. Otra posible vía de desarrollo sería la mejora del modelo de reconocimiento de caras en vídeos. Se ha experimentado muy poco sobre este tema en el trabajo, pero existen numerosos modelos que permiten realizar esta tarea, abriendo una nueva vía de desarrollo que puede basarse incluso en la combinación de varios de esos modelos. En este trabajo se ha trabajado con imágenes previamente etiquetadas y probadas en imágenes disponibles localmente en el ordenador, por lo que otra posible mejora es incorporar el trabajo el análisis de una imagen o vídeo en *streaming* ya que podría aumentar el número de aplicaciones reales del proyecto.

Resumen del Trabajo en Inglés

Capítulo 8

Introduction

8.1. Motivation

Since the beginning of computer science in the second half of the twentieth century, it has had an exponential growth that has led it to be one of the most booming scientific branches nowadays. Since the first computer systems, its objective has been clear: to receive input data to be processed and return an output containing the information requested from the data. Algorithms have played a key role in the development of humanity and the emergence of programming and programming languages has given a remarkable boost to our society today.

Numerous tasks today would be unfeasible if it were not for computer science, computer systems and algorithms. Computer science is involved in the lives of each and every person and improves our quality of life, allowing us to see relatives we do not live with in times of pandemic without any risk, to find out the news at the moment and a host of other things.

After decades of development of computer science, the branch of Artificial Intelligence arises, which completely changes the classical programming paradigm and allows us to perform tasks that would be unfeasible with classical computing. Specifically, in 1958 the so-called neural networks appeared with the intention of imitating human intelligence. Since then they began to grow at great speed driven by the advance of hardware, allowing the creation of more advanced machines that support more complex systems and had much more computing power. The great advantage of neural networks is that they can perform a very computationally expensive task very quickly once you train them to perform that task previously.

On the other hand, in the audiovisual field, until not so long ago, a specialized camera was required to be able to record a video and for its reproduction. In addition, the quality of the images and videos obtained was much lower than today. However, as technology

advanced, and with it the computer, the quality of cameras began to improve. With the creation of digital cameras, a first step was taken towards the representation of photos in a computer system. As early as 1998, the famous RGB code (red, green, blue) was created, which makes it possible to represent each pixel of an image as a color composition in terms of the intensity of the primary colors of light. The representation of photographs, and therefore of reality and our vision, allows us to tackle a myriad of previously impossible problems.

Tasks that require the identification of objects or some feature of an image usually have a high cost, so one of the most effective computational models to solve them are neural networks.

One of the most useful problems related to object identification is face recognition, giving rise to numerous applications. This system is used both to aid identification at airports and to facilitate police investigations, and is even used to detect symptoms of disease and provide a medical diagnosis.

Another problem that has arisen in recent years, given the success of facial recognition systems, is age identification. Most of these methods are based on a person's face, so it is first necessary to use a facial recognition system on the image to locate the face. In today's highly digitized world, there are numerous sources of photographic evidence that are routinely used by law enforcement to identify suspects and victims of crime both online and offline. Human characteristics such as age, height, weight, gender and hair color are often used by police officers and witnesses in their description of unidentified suspects. In certain circumstances, the age of the victim may determine the classification of the crime, for example, in child abuse investigations. Age estimation can also be used for access to adult entertainment venues, the purchase of age-restricted products such as alcohol and tobacco, age-targeted advertising and, recently, services such as *how-old.net* have been used for the recognition of Syrian refugee children.

Therefore, we propose to address this problem, which is quite current, with the aim of deepening the knowledge on the subject, making an in-depth study of the state of the art and the techniques used to solve the problem. Once that is understood, we propose to make tests that allow us to see how various models behave and how the input data influence when training the model. This search for knowledge on such a useful and interesting topic is the driving force behind this work.

8.2. Context

This Final Year dissertation was done within the Group of Analysis, Security and Systems (Group GASS, https://gass.ucm.es/, Group 910623 of the UCM catalogue of recognised groups), and as part of the THEIA research project (Techniques for Integrity

and Authentication of Multimedia Files of Mobile Devices), reference no. FEI-EU-19-04.

8.3. Object of the Investigation

Human vision allows us to recognize many characteristics present in images such as color, shape and texture of objects. Recognizing the multiple characteristics allows us, together, to identify the different objects in an image and therefore draw conclusions such as knowing the animals that appear in the image or even knowing the number of faces in the image. However, with the advance of technology and, above all, the advance of the AI systems, there is the possibility of training them to identify the characteristics of the images, even recognizing details that are not perceptible to the human eye. These small details are of vital importance when trying to perform the identification of some object or some task based on the analysis of an image.

The main objective of this work is to test different pre-processing techniques by measuring their effectiveness in the training process of different models based on convolutional neural networks (CNNNs) that perform age estimation in images.

8.4. Workplan

The realization of this project is divided into four phases:

1. **Research**: At the beginning, a learning process was carried out to acquire the knowledge to start with the subsequent development. In order to acquire specific knowledge about machine learning, about the *Python* library called *Tensorflow* and about image processing, 8 different courses were carried out for a total of 115 hours of training. These courses were carried out on the *Coursera* website thanks to the help of the Complutense University of Madrid, since they have an agreement that provides access to many courses for free. In addition, thanks to the work of the directors of this work, several platforms and tools were learned, such as *Google Scholar*, which would later be used to search for scientific articles where research is done on age estimation. Once the different ways of dealing with the study were understood and having concluded how they wanted to approach it, the development of the proposal began.

2. **Development**: Once it was understood how the project was to be approached, programming began. However, the research phase did not stop, but it ceased to be the main focus of the work. During this phase, the concepts about the programming language *Python* and libraries such as *Tensorflow* and *Pandas* learned in the courses taken in the previous phase were applied. During this phase, several databases were

also processed to train the model, found thanks to the scientific articles researched in the previous phase.

3. **Experimentation**: In this phase the developed prototypes were evaluated and the results obtained were analyzed. These results made it necessary to modify and develop new models, so this phase was carried out in parallel to the research and development stages. Once the first results were seen and new models were created, the pre-processing of the images was also experimented and its influence on the improvement of the training process was analyzed.

4. **Documentation**: Once the development started, the phase of writing and reviewing the final report of the project began. This stage began with the writing of the theoretical knowledge necessary to understand the project and with the compilation of the articles in the field of study for its explanation. Once the models were developed and experienced, an iterative writing process was carried out, looking for typos or possible sections to improve.

Table 8.1 shows the different activities carried out to develop the project.

Tabla 8.1: Work Plan Activities

Month	Activity
October	Learning concepts, tools and researching academic articles.
November	
December	
January	Creation of the most basic models and basic database processing.
February	
March	- Experimentation and optimization of the models, pre-processing. - Realization of the memory.
April	
May	
June	

8.5. Structure of the Work

The rest of the paper is organized in 8 chapters:

Chapter 2 explains the basic concepts of the AI, and then focuses on classification algorithms that can be used when dealing with age estimation from faces.

In Chapter 3, the operation of an artificial neural network is explained in detail, explaining each of its components. Some of its types and how to train them are also explained.

Chapter 4 shows the different models and techniques proposed for age estimation in images. Before explaining the techniques used in the various papers, some of the factors

that influence age recognition are shown. The papers are then divided according to the type of technique used to create the model, whether they are based on extracting features and then passing them through a classifier or whether they are based on deep learning. Finally, a comparison of several of the models previously explained in the chapter is made.

Chapter 5 describes the pre-processing techniques used to improve the quality of the images used to train our models. In turn, the models used to test the effectiveness of these techniques are explained. Finally, a brief explanation of the technologies used to perform both the models and the pre-processing of the images is given.

Chapter 6 details the experiments carried out to evaluate the effectiveness of the different models created, as well as the creation of these models and how the models explained in Chapter 5 arise from the experiments with the previous models. The experiments with the different image pre-processing techniques to improve the quality of the images and thus achieve a better training of the neural networks are also described. All these experiments are accompanied with graphs showing the results of the experiments, ending with a comparative section with the different models performed.

In Chapter 7 the main conclusions of this project are shown, as well as different possible improvements on the work done to investigate more about this topic.

Finally, the English translations of the Introduction and Conclusions are shown in Chapters 8 and 9.

Capítulo 9

Conclusions and Future Work

9.1. Conclusions

Along this proyect, we have seen different ways of creating models to estimate the age of a person from a face, focusing on models based on convolutional neural networks. Several test models were created for this task, creating a pre-processing for the images with used to train these models. The training of these models is very expensive, requiring large computational power and many hours of work. To simplify this process, it was shown that models based on TL perform much more robustly on the input data, saving a large number of hours.

However, the most important part performed has been the use of pre-processing techniques as they have proved to be very effective in training a model from 0, allowing a higher accuracy in training the models. This previous preparation of the data and improvement of its quality can be used for any other field of study, which allows us to see its great usefulness. This study shows us the importance of having a good quality of input data when training our model, being very necessary to spend time in the improvement of these data.

9.2. Future Work

Because age estimation is a very novel task, there are still many avenues of progress in the field, such as:

1. **Testing SWL models**: Although several models with different configurations were tested during the whole experimental phase, all of them were based on DL techniques. Therefore, using SWL models may be of great interest, since although the work in this field has a lower accuracy, the effect of pre-processing in the feature extraction

stage could be better observed.

2. **Increasing the number of test models**: Another possible avenue of development is the experimentation of new models, since there are several hyperparameters such as the activation functions, the number of hidden layers, the number of neurons in each layer, the *dropout* of the layers and the choice of optimizer that have been modified very little. With respect to the TL there are also many other networks available to use as a base network in your model such as *Xception*, *VGG19*, *ResNet152*, *InceptionResNet* and *MobileNet*.

3. **Increased model testing time**: Having used a personal computer, the model training capability has been limited despite having used a GPU. A possible improvement of the work is to increase the testing time by increasing the number of epoch and testing with different batch sizes for the different models.

4. **Increasing the number of databases**: A possible development path is to increase the number of databases used, as well as the mixture of several of them in order to have a diverse and larger amount of data to train the model. There are databases such as *MORPH* , *IMDB-WIKI*, *AFAD*, *MegaAge* and *AGFW* that can be interesting to incorporate in the work.

5. **Image preprocessing improvement**: The techniques seen for face alignment and cropping sometimes incorporate some noise into the image, and in some situations rescale photos from smaller than target size to target size. This noise is detrimental to model training, so one possible avenue for development may be to remove this noise. Different techniques can also be tried to change some image properties such as brightness, contrast and color. To accomplish this task, a model could be created and trained to improve the quality of the various images and improve the result in the model.

6. **Video analysis**:In the experimentation a small analysis of how the different faces that appear in a video could be extracted has been carried out. For this it is necessary to extract all the frames of the image, and treat each frame as a new image. For each image we extract the different faces and compare them with the faces extracted in the previous frames to look for similarities. In this way if two faces in different frames belong to the same person, only one of them is saved and we continue with the analysis of the frames. To perform this task it has been used in the `github.com/ageitgey/face_recognition` repository which creates a model based on *dlib* for face recognition and comparison in images. Another possible avenue for development would be to improve the model for face recognition in videos. There has been very little experimentation on this topic in the work, but there are numerous models that allow us to perform this task, opening a new avenue of development that may

even be based on the combination of several of these models. In this work we have worked with images previously labeled and tested on images available locally on the computer, so a possible improvement is to incorporate the work the analysis of an image or video in streaming as it could increase the number of real applications of the project.

Bibliografía

[ABL+20] F. Anda, B. A. Becker, D. Lillis, N. Le-Khac, and M. Scanlon. Assessing the influencing factors on the accuracy of underage facial age estimation. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8, 2020.

[ALG+17] A. Anand, R. D. Labati, A. Genovese, E.ñoz@, V. Piuri, and F. Scotti. Age estimation based on face images and pre-trained convolutional neural networks. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, 2017.

[ALKS20] Felix Anda, Nhien-An Le-Khac, and Mark Scanlon. Deepuage: Improving underage age estimation accuracy to aid csem investigation. *Forensic Science International: Digital Investigation*, 32:300921, 2020.

[ALLS18] F. Anda, D. Lillis, N. Le-Khac, and M. Scanlon. Evaluating automated facial age estimation techniques for digital forensics. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 129–139, 2018.

[Ayo10] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, 3:19–48, 2010.

[B+95] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[BT17] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks). In *International Conference on Computer Vision*, 2017.

[CCH10] Kuang-Yu Chang, Chu-Song Chen, and Yi-Ping Hung. A ranking approach for human ages estimation based on face images. In *2010 20th International Conference on Pattern Recognition*, pages 3396–3399. IEEE, 2010.

[Che16] Jim X Chen. The evolution of computing: Alphago. *Computing in Science & Engineering*, 18(4):4–7, 2016.

[CLD13] Wei-Lun Chao, Jun-Zuo Liu, and Jian-Jiun Ding. Facial age estimation based on label-sensitive learning and age-oriented regression. *Pattern Recognition*, 46(3):628–641, 2013.

[CLZ+12] Dong Cao, Zhen Lei, Zhiwei Zhang, Jun Feng, and Stan Z Li. Human age estimation using ranking svm. In *Chinese Conference on Biometric Recognition*, pages 324–331. Springer, 2012.

[CUD20] CUDA: Compute Unified Device Architecture. https://developer.nvidia.com/cuda-zone, March 2020.

[CZD+17] Shixing Chen, Caojin Zhang, Ming Dong, Jialiang Le, and Mike Rao. Using ranking-cnn for age estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5183–5192, 2017.

[dCP20] Bernardo Botelho Antunes da Costa and Pedro Silveira Pisa. Cloud strategies for image recognition. In *2020 4th Conference on Cloud and Internet of Things (CIoT)*, pages 57–58, 2020.

[DCR12] Antitza Dantcheva, Cunjian Chen, and Arun Ross. Can facial cosmetics affect the matching accuracy of face recognition systems? In *2012 IEEE Fifth international conference on biometrics: theory, applications and systems (BTAS)*, pages 391–398. IEEE, 2012.

[DGXZ19] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019.

[DLL18] M. Duan, K. Li, and K. Li. An ensemble cnn2elm for age estimation. *IEEE Transactions on Information Forensics and Security*, 13(3):758–772, 2018.

[DS18] Alessandro Del Sole. *Introducing Microsoft Cognitive Services*, pages 1–4. Apress, 2018.

[EA17] S Escalera X Baro I Guyon R Rothe. E Agustsson, R Timofte. Apparent and real age estimation in still images with deep residual regressors on appa-real database. In *12th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), 2017*. IEEE, 2017.

[EEH14a] E. Eidinger, R. Enbar, and T. Hassner. Age and gender estimation of unfiltered faces. *IEEE Transactions on Information Forensics and Security*, 9(12):2170–2179, 2014.

[EEH14b] Eran Eidinger, Roee Enbar, and Tal Hassner. Age and gender estimation of unfiltered faces. *IEEE Transactions on Information Forensics and Security*, 9(12):2170–2179, 2014.

[fac18] face_recognition: The world's simplest facial recognition api for Python. https://github.com/ageitgey/face_recognition, April 2018.

[FH99] H. Feng-Hsiung. IBM's Deep Blue Chess Grandmaster Chips. *IEEE Micro*, 19(2):70–81, March 1999.

[FH08] Y. Fu and T. S. Huang. Human age estimation with regression on discriminative aging manifold. *IEEE Transactions on Multimedia*, 10(4):578–584, 2008.

[FP12] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. Pearson, 2012.

[FR13] Mohd Awais Farooque and Jayant S Rohankar. Survey on various noises and techniques for denoising the color image. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, 2(11):217–221, 2013.

[FXH07] Yun Fu, Ye Xu, and Thomas S Huang. Estimating human age by manifold analysis of face pictures and regression on aging features. In *2007 IEEE International Conference on Multimedia and Expo*, pages 1383–1386. IEEE, 2007.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.

[GFDH08] G. Guo, Y. Fu, C. R. Dyer, and T. S. Huang. Image-based human age estimation by manifold learning and locally adjusted robust regression. *IEEE Transactions on Image Processing*, 17(7):1178–1188, 2008.

[GGFH09] G. Guo, Guowang Mu, Y. Fu, and T. S. Huang. Human age estimation using bio-inspired features. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 112–119, 2009.

[Glo20] Glob — Unix Style Pathname Pattern Expansion. https://docs.python.org/3/library/glob.html, March 2020.

[GN08] Asuman Gunay and Vasif V Nabiyev. Automatic age classification with lbp. In *2008 23rd International Symposium on Computer and Information Sciences*, pages 1–4. IEEE, 2008.

[Gon17] Andrés González. ¿qué es machine learning? https://cleverdata.io/que-es-machine-learning-big-data/, 2017.

[Gon20] Jorge De Andrés González. ¿cómo funciona un svm (support vector machines)? https://www.jparzival.com/blog/como-funciona-svm/, 2020.

[GYZ13] Xin Geng, Chao Yin, and Zhi-Hua Zhou. Facial age estimation by learning from label distributions. *IEEE transactions on pattern analysis and machine intelligence*, 35(10):2401–2412, 2013.

[GZSM07] Xin Geng, Zhi-Hua Zhou, and Kate Smith-Miles. Automatic age estimation based on facial aging patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2234–2240, 2007.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[KdVL99] Young H Kwon and Niels da Vitoria Lobo. Age classification from facial images. *Computer vision and image understanding*, 74(1):1–21, 1999.

[Ker20] Keras: the Python deep learning API. https://keras.io/, March 2020.

[Kin09] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[KS08] Carl Kingsford and Steven L Salzberg. What are decision trees? *Nature biotechnology*, 26(9):1011–1013, 2008.

[KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[LH15] Gil Levi and Tal Hassner. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2015.

[LLF+19] Wanhua Li, Jiwen Lu, Jianjiang Feng, Chunjing Xu, Jie Zhou, and Qi Tian. Bridgenet: A continuity-aware probabilistic network for age estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[LLK+15] Xin Liu, Shaoxin Li, Meina Kan, Jie Zhang, Shuzhe Wu, Wenxian Liu, Hu Han, Shiguang Shan, and Xilin Chen. Agenet: Deeply learned regressor and classifier for robust apparent age estimation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 16–24, 2015.

[LRBS09] Khoa Luu, Karl Ricanek, Tien D Bui, and Ching Y Suen. Age estimation using active appearance models and support vector machine regression. In *2009 IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–5. IEEE, 2009.

[LTC02] Andreas Lanitis, Christopher J. Taylor, and Timothy F Cootes. Toward automatic simulation of aging effects on face images. *IEEE Transactions on pattern Analysis and machine Intelligence*, 24(4):442–455, 2002.

[Mic18] Umberto Michelucci. *Applied Deep Learning*. Springer, 2018.

[Mic19] Umberto Michelucci. *Advanced applied deep learning: convolutional neural networks and object detection*. Springer, 2019.

[MJ01] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5, 2001.

[MP18] Rodrigo F Mello and Moacir Antonelli Ponti. *Machine learning: a practical approach on the statistical learning theory*. Springer, 2018.

[MPS+17] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: the first manually collected, in-the-wild age database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop*, pages 51–59, 2017.

[MRB+18] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. Machine learning for internet of things data analysis: a survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

[Nob06] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[num20] NumPy, The Fundamental Package for Scientific Computing with Python. https://numpy.org/, March 2020.

[ope20] Open Computer Vision Library (OpenCV). https://opencv.org/, March 2020.

[os20] os — Miscellaneous Operating System Interfaces. https://docs.python.org/3.9/library/os.html, March 2020.

[pan21a] La librería pandas. https://aprendeconalf.es/docencia/python/manual/pandas/, 2021.

[pan21b] Pandas: A Python Data Analysis Library. https://pandas.pydata.org/, April 2021.

[Pat98] Dan W Patterson. *Artificial neural networks: theory and applications*. Prentice Hall PTR, 1998.

[Pet09] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[PLTC16] Gabriel Panis, Andreas Lanitis, Nicholas Tsapatsoulis, and Timothy F Cootes. Overview of research on facial ageing using the fg-net ageing database. *Iet Biometrics*, 5(2):37–46, 2016.

[PS75] John B Pittenger and Robert E Shaw. Perception of relative and absolute age in facial photographs. *Perception & Psychophysics*, 18(2):137–143, 1975.

[PVZ15] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RN09] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Pearson, 3 edition, 2009.

[RT06] K. Ricanek and T. Tesafaye. Morph: a longitudinal image database of normal adult age-progression. In *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*, pages 341–345, 2006.

[RTG18] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision*, 126(2-4):144–157, 2018.

[RTVG15] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Dex: Deep expectation of apparent age from a single image. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, December 2015.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SC09] Lifeng Shang and Kwok-Ping Chan. Nonparametric discriminant hmm and application to facial expression recognition. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2090–2096. IEEE, 2009.

[shu20] Shutil — High-Level File Operations. https://docs.python.org/3/library/shutil.html, March 2020.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[SO20] Sefik Ilkin Serengil and Alper Ozpinar. Lightface: A hybrid deep face recognition framework. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 23–27. IEEE, 2020.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[Ten20] Tensorflow: An Open Source Machine Learning Framework for Everyone. https://www.tensorflow.org/, March 2020.

[Tur50] AM. Turing. Computing Machinery and Intelligence. *Mind*, 49:433–460, 1950.

[VELR12] Manuel C Voelkle, Natalie C Ebner, Ulman Lindenberger, and Michaela Riediger. Let me guess how old you are: Effects of age, gender, and facial expression on perceptions of age. *Psychology and aging*, 27(2):265, 2012.

[Ver20] Vaibhav Verdhan. *Supervised Learning with Python*. Springer, 2020.

[WLLK13] Xiaolong Wang, Vincent Ly, Guoyu Lu, and Chandra Kambhamettu. Can we minimize the influence due to gender and race in age estimation? In *2013 12th International Conference on Machine Learning and Applications*, volume 2, pages 309–314. IEEE, 2013.

[XYXZ09] Bo Xiao, Xiaokang Yang, Yi Xu, and Hongyuan Zha. Learning distance metric for regression by semidefinite programming with application to human age estimation. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 451–460, 2009.

[YLH08] Shuicheng Yan, Ming Liu, and Thomas S Huang. Extracting age information from local spatially flexible patches. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 737–740. IEEE, 2008.

[ZGZC19] Huiying Zhang, Xin Geng, Yu Zhang, and Fanyong Cheng. Recurrent age estimation. *Pattern Recognition Letters*, 125:271–277, 2019.

[ZQ17] Song Yang Zhang, Zhifei and Hairong Qi. Age progression/regression by conditional adversarial autoencoder. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

[ZY10] Yu Zhang and Dit-Yan Yeung. Multi-task warped gaussian process for personalized age estimation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2622–2629. IEEE, 2010.

[ZZLQ16] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.