



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Estimación de la Edad Legal en Antropología
Forense Utilizando Técnicas de Deep Learning

Autor

José Antonio López Palenzuela

Directores

Pablo Mesejo Santiago
Óscar Cordón García



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, 20 Junio de 2024



Estimación de la Edad Legal en Antropología Forense Utilizando Técnicas de Deep Learning

Autor

José Antonio López Palenzuela

Directores

Pablo Mesejo Santiago
Óscar Cordón García

Estimación de la Edad Legal en Antropología Forense Utilizando Técnicas de Deep Learning

José Antonio López Palenzuela

Palabras clave: Aprendizaje automático, aprendizaje profundo, aprendizaje supervisado, visión por computador, clasificación de imágenes, antropología forense, estimación del perfil biológico, estimación de la edad legal.

Resumen

La estimación de la edad legal es uno de los problemas que más interés suscita en el campo de la antropología forense. Un menor y un adulto no tienen los mismos derechos y obligaciones en la mayoría de marcos legales del mundo. Se recomienda evaluar el desarrollo óseo de dientes, muñeca y/o clavícula para realizar esta distinción de forma científica. No obstante, la praxis habitual en este contexto ha demostrado ser imprecisa y subjetiva. Este trabajo fin de grado presenta una solución a esas carencias: un método, basado en técnicas de aprendizaje automático, para determinar la edad legal de un sujeto a partir de una radiografía panorámica de su dentadura.

Utilizando 1741 ortopantomografías pertenecientes a individuos sudafricanos negros, una población nunca antes empleada en un estudio de esta índole, se implementan varios clasificadores basados en técnicas clásicas de aprendizaje automático y en aprendizaje profundo. Concretamente, se proponen dos redes neuronales convolucionales (*ResNet18* y *ResNet50*) y tres predictores clásicos (*Regresor Logístico*, *Support Vector Machines* y *Random Forest*) que utilizan el algoritmo *HOG* (*Histogram of Oriented Gradients*) como extractor de características. El resultado del estudio comparativo es un clasificador basado en la arquitectura *ResNet50* con una exactitud del 74.15 %, una sensibilidad del 76.64 % y una especificidad del 71.85 %, sobre imágenes nunca antes vistas. Utilizando una variante del algoritmo *Grad-CAM*, se dota a este mejor candidato de capacidad para argumentar sus predicciones. Gracias ello, un experto forense puede descartarlas, validarlas o incluso aprender de ellas.

El rendimiento obtenido no supera el Estado del Arte, pero la experimentación permite identificar que el grave desbalanceo de los datos es el origen de esta limitada capacidad, y que las técnicas habituales para solventarlo, sobremuestreo y aumento de datos, son parcialmente efectivas. Mejoran el rendimiento, pero no lo suficiente como para alcanzar el de expertos humanos que trabajan sobre la misma población o el de clasificadores similares que trabajan sobre poblaciones distintas.

Legal Age Estimation in Forensic Anthropology Using Deep Learning

José Antonio López Palenzuela

Keywords: Machine learning, deep learning, supervised learning, computer vision, image classification, forensic anthropology, biological profile estimation, legal age estimation.

Abstract

Legal age estimation is one of the key topics in forensic anthropology. A child and an adult do not have the same rights and obligations in most legal frameworks. Evaluating the bone development of teeth, wrist and/or clavicle is recommended to make this distinction scientifically. Nevertheless, standard practice in this context has shown to be imprecise and subjective. This bachelor's thesis presents a solution to these deficiencies: an automatic machine-leaning-based method to determine legal age from a panoramic radiography of the subject's teeth.

Using 1741 orthopantomographies belonging to black Southafrican individuals, a population that has never been subject of this kind of study, several classifiers are implemented based on classic machine learning techniques and deep learning. More precisely, two convolutional neuronal nets (*ResNet18* and *ResNet50*) and three classic predictors (*Logist Regressor*, *Support Vector Machine* and *Random Forest*) using *HOG* (*Histogram of Oriented Gradients*) as a feature extractor. As a result of this comparative study, a classifier based on *ResNet50* is obtained with a 74.15 % accuracy, 76.64 % sensibility and 71.85 % specificity on test data. Using a *GradCAM* variant, the ability to justify its predictions is given to this best candidate, a useful feature for human experts to validate or learn from them.

The proposed model performance does not surpass the state of the art, but the experimentation allowed, to identify the origin of this limited capacity, the dataset is highly unbalanced, and to test that usual techniques applied to solve this problem, oversampling and data augmentation, are partially effective. They enhance performance, but not enough to reach the level of human experts on the same population, or the results of similar classifiers on other kinds of populations.

Yo, **José Antonio López Palenzuela**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76656255S, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Antonio López Palenzuela

Granada a 19 de Junio de 2024.

D. **Pablo Mesejo Santiago**, Profesor del Área de Ciencias de la Computación e Inteligencia Artificial del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Óscar Cordón García**, Profesor del Área de Ciencias de la Computación e Inteligencia Artificial del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Estimación de la Edad Legal en Antropología Forense Utilizando Técnicas de Deep Learning*, ha sido realizado bajo su supervisión por **José Antonio López Palenzuela**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 20 de Junio de 2024.

Los directores:

Pablo Mesejo Santiago Óscar Cordón García)

Agradecimientos

A mi José Antonio y a mis Yolandas. A mi Lorena, a mis Colonos, a mis Serpientes, a mis Artistas y a mis Daltons. A todos los que creyendo en mí en mis momentos más bajos: Molina, Toté, Lucía, Gabri, Arturo, Teo, Yoli (bis), Lorena (bis), Jubago, Andrés, Alba, Lolo, Guille, Xema, Martín, Rocío, César, Alberto, Alba, Manu y Paula. A Óscar y Pablo por su esfuerzo y su paciencia. Y, por supuesto, a todos los que me olvido.

Índice general

1. Introducción	1
1.1. Definición del Problema	1
1.2. Motivación	3
1.3. Objetivos	5
1.4. Planificación del Proyecto	6
2. Fundamentos Teóricos	9
2.1. Aprendizaje Automático	9
2.1.1. Aprendizaje Profundo	11
2.2. Redes Neuronales	12
2.2.1. Arquitectura	12
2.2.2. Función Pérdida	14
2.2.3. Algoritmo de Aprendizaje	15
2.3. Redes Neuronales Convolucionales	19
2.3.1. Capas Convolucionales	23
2.3.2. Capas de Agrupación	25
2.3.3. Capas de Completamente Conectadas	25
2.3.4. Capas de Activación	26
2.3.5. Capas de Normalización	28
2.4. <i>Transfer Learning</i> y <i>Fine-Tuning</i>	28
3. Estado del Arte	31
3.1. Clasificación de Imágenes Utilizando Aprendizaje Automático	31
3.2. Estimación de la Edad Legal en Antropología Forense	31
3.3. Estimación de la Edad Legal en Antropología Forense Utilizando Aprendizaje Automático	37
4. Materiales y Métodos	41
4.1. Conjunto de Datos	41
4.1.1. Partición de Entrenamiento–Validación y Test	42
4.2. Métodos	44
4.2.1. Validación Cruzada	44
4.2.2. Sobremuestreo	46

4.2.3. Aumento de Datos	46
4.2.4. Métricas de Rendimiento de Clasificación	49
4.2.5. Aprendizaje Profundo	53
4.2.6. Aprendizaje Automático Clásico	59
5. Implementación	67
6. Experimentación	71
6.1. Experimentación - Aprendizaje Profundo	72
6.1.1. <i>ResNet18</i>	74
6.1.2. <i>ResNet50</i>	88
6.2. Experimentación - Aprendizaje Automático Clásico	91
6.2.1. HOG	92
6.2.2. Regresión Logística	92
6.2.3. SVM	93
6.2.4. Random Forest	94
6.3. Comparativa Global	95
6.3.1. Validación	95
6.3.2. Test	96
6.3.3. Explicabilidad	98
7. Conclusiones	103
8. Apéndice	107
Bibliografía	107

Índice de figuras

1.1.	Ortopantomografías de ejemplo	2
1.2.	Resumen esquemático de un modelo basado en ML o DL para estimar la edad legal a partir de una OPG	5
1.3.	Planificación temporal inicial	7
1.4.	Distribución temporal final	8
2.1.	Esquema del funcionamiento de una neurona. Transformación lineal (suma ponderada con desplazamiento) de un conjunto de señales de entrada (x_i) en base a unos parámetros (w_i) a la que se le aplica una función no lineal (σ) para generar una señal de salida (y).	13
2.2.	Ejemplo de arquitectura de una red neuronal totalmente conectada	14
2.3.	Entropía cruzada negativa en función de la probabilidad de la clase correcta.	16
2.4.	Ejemplo simplificado del efecto del valor de la tasa de aprendizaje en la optimización de una función	18
2.5.	Ejemplo simplificado del algoritmo de derivación “backpropagation”	20
2.6.	Ejemplo para ilustrar por qué la estructura espacial de una imagen es importante.	21
2.7.	Ejemplo de convolución para localizar parte de los bordes verticales de una imagen.	22
2.8.	Esquema habitual de una CNN utilizada para clasificación multiclase de imágenes.	23
2.9.	Ejemplo de aplicación de una capa de convolución	24
2.10.	Ejemplos de aplicación de una capa agrupación	26
2.11.	Funciones de activación y sus derivadas	27
3.1.	Protocolo español para la estimación de edad de potenciales menores	33
3.2.	Atlas de estados de Demirjian	34
3.3.	Cálculo del I_{3M}	35

3.4. Guía EASO para estimar de edad legal de un migrante	38
4.1. Varias OPG de ejemplo	41
4.2. Balanceo del conjunto de datos	42
4.3. Partición del conjunto de datos en entrenamiento-validación y test	44
4.4. Validación Cruzada	45
4.5. Balanceo de datos para una iteración de validación cruzada tras sobremuestreo	47
4.6. Distribución Beta	48
4.7. Bloque residual o “skip connection”	53
4.8. Arquitecturas <i>ResNet</i>	54
4.9. <i>Ejemplo LR-Find</i>	55
4.10. Ejemplo de GradCAM para clasificación	56
4.11. HOG	61
4.12. SVM. Ejemplo de hiperplano óptimo	63
4.13. SVM. Ejemplo de aplicación del Kernel	64
4.14. Ejemplo de árbol de decisión	65
6.1. <i>Experimentación LR-Find</i>	75
6.2. <i>ResNet18</i> . Experimentación Sobre muestreo. Conjunto de entrenamiento balanceado.	76
6.3. Aumento de Datos vía rotación	80
6.4. Experimentación Aumento de Datos. Sólo rotaciones. Reducción de la <i>SEN</i> con el aumento de p cuando $l = \{1\}$	80
6.5. Experimentación Aumento de Datos. Sólo rotaciones. Reducción del sobreajuste cuando $l = \{0, 1\}$	82
6.6. Aumento de Datos vía desplazamiento	83
6.7. Ejemplo de <i>MixUp</i> en función de λ_{MU}	86
6.8. Experimentación <i>Fine Tuning</i> . Sin aumento de datos. $d = 1$. Aumento de sobreajuste.	88
6.9. <i>Experimentación Fine Tuning. Sin aumento de datos. d = 1. Efecto de pFT y f.</i>	89
6.10. Imagen HOG sobre imagen del conjunto de entrenamiento. .	92
6.11. <i>GradCAM</i> . Verdadero Positivo. Caso Prometedor.	99
6.12. <i>GradCAM</i> . Verdadero Negativo. Caso prometedor.	100
6.13. <i>GradCAM</i> . Verdadero Positivo. Caso Anotación I.	101
6.14. <i>GradCAM</i> . Verdadero Positivo. Caso Anotación II.	102

Índice de cuadros

1.1.	Estimación de costes.	8
3.1.	Resultados de referencia para la estimación de la edad legal de sudafricanos negros a partir de OPG.	36
4.1.	Matriz de confusión para un clasificador binario.	49
6.1.	<i>ResNet18</i> . Experimentación Sobremuestreo. Comparación cuantitativa.	78
6.2.	<i>ResNet18</i> . Experimentación Aumento de Datos. Comparación cuantitativa.	79
6.3.	<i>ResNet18</i> . Experimentación Aumento de Datos. Sólo rotaciones. Comparación cuantitativa.	81
6.4.	<i>ResNet18</i> . Experimentación Aumento de Datos. Sólo traslaciones. Comparación cuantitativa.	84
6.5.	<i>ResNet18</i> . Experimentación Aumento de Datos. Traslaciones y rotaciones. Comparación cuantitativa.	85
6.6.	<i>ResNet18</i> . Experimentación Aumento de Datos. <i>MixUp</i> . Comparación cuantitativa.	86
6.7.	Experimentación <i>Fine Tuning</i> . Con aumento de datos. Comparación cuantitativa.	90
6.8.	<i>ResNet50</i> . Experimentación Aumento de Datos. Comparación cuantitativa.	91
6.9.	Resultados de los mejores candidatos sobre validación cruzada.	95
6.10.	Resultados de los mejores modelos sobre test.	96
8.1.	<i>ResNet18</i> . Experimentación Aumento de Datos. Sólo <i>MixUp</i> . Comparación cuantitativa.	107
8.2.	<i>ResNet18</i> . Experimentación Aumento de Datos. M^* + <i>MixUp</i> . Comparación cuantitativa.	108
8.3.	<i>ResNet18</i> . Experimentación Aumento de Datos. <i>MixUp</i> + M^* . Comparación cuantitativa.	109
8.4.	<i>ResNet50</i> . Experimentación Sobremuestreo. Resultados. . .	109

8.5. <i>ResNet50</i> . Experimentación Aumento de Datos. Sólo rotaciones. Comparación cuantitativa.	110
8.6. <i>ResNet50</i> . Experimentación Aumento de Datos. Sólo traslaciones. Comparación cuantitativa.	110
8.7. <i>ResNet50</i> . Experimentación Aumento de Datos. Traslaciones y rotaciones. Comparación cuantitativa.	111
8.8. <i>ResNet50</i> . Experimentación Aumento de Datos. Traslaciones y rotaciones. Comparación cuantitativa.	111
8.9. <i>ResNet50</i> . Experimentación Aumento de Datos. Sólo <i>MixUp</i> . Comparación cuantitativa.	112
8.10. <i>ResNet50</i> . Experimentación Aumento de Datos. <i>MixUp</i> + <i>M*</i> . Comparación cuantitativa.	113
8.11. <i>ResNet50</i> . Experimentación Aumento de Datos. <i>M*</i> + <i>MixUp</i> . Comparación cuantitativa.	114
8.12. Regresión Lineal. Comparación cuantitativa de resultados. . .	115
8.13. <i>Random Forest</i> . Comparación cuantitativa de resultados. . .	116
8.14. <i>SVM</i> . Comparación cuantitativa de resultados.	117

Capítulo 1

Introducción

1.1. Definición del Problema

Este Trabajo Fin de Grado (TFG) aborda uno de los problemas que mayor interés despierta en el contexto de la antropología forense a nivel internacional: cómo determinar la edad legal de un individuo. Más concretamente, propone un método automatizado, basado en técnicas de aprendizaje automático, para decidir si un individuo es mayor o menor de edad a partir de una radiografía panorámica de su boca.

La antropología forense (del inglés, *Forensic Anthropology* o FA) es la ciencia encargada del estudio del esqueleto para su aplicación en contextos médico-legales [1]. Los expertos de este campo extraen información del material óseo de un individuo para aproximar su sexo, edad, estatura, origen poblacional (ascendencia), intervalo post-morten (tiempo desde la muerte), traumatismos u otras patologías que afecten a los huesos. El individuo estudiado puede estar vivo o muerto. De entre todas las tareas enumeradas, la estimación de la edad atrae un interés especial. No sólo puede facilitar un proceso de identificación, como el resto de técnicas, sino que es determinante en conflictos legales como la recepción de migrantes en busca de asilo. Para ellos, ser menor o mayor de edad puede significar la diferencia entre ser aceptado o rechazado.

En la mayoría de países de la Unión Europea, incluida España, se recurre a la FA cuando un solicitante de asilo afirma ser menor de edad, pero no existe documentación o registros oficiales que lo demuestren [2, 3]. En este contexto, el informe de un antropólogo forense pasa a ser la evidencia sobre la que la autoridad competente decide si el individuo se considera mayor o menor de edad. Se recomienda que los peritos encargados de estos informes utilicen la muñeca, la clavícula y/o la dentadura para dar una estimación de la edad de la persona [4, 5]. El nivel de desarrollo de estas regiones anatómi-

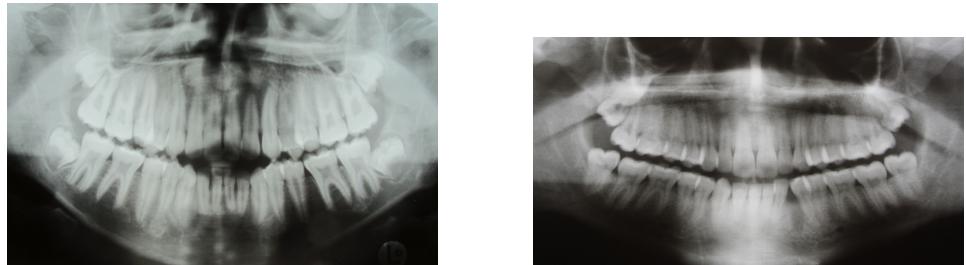


Figura 1.1: Ortopantomografías de ejemplo. Pertenecen al conjunto de datos utilizado en el proyecto. A la izquierda, un individuo menor de edad (14 años y 7 meses). A la derecha, un individuo mayor de edad (20 años y 3 meses).

cas (edad ósea) está relacionado con la edad del individuo al que pertenecen (edad cronológica). No obstante, esta relación está condicionada a factores como sexo, patologías, rango de edades, condición socioeconómica o incluso la raza. Se trata de una discusión aún abierta que el especialista debe tener en cuenta [6]. Utilizar un método de estimación basado en una población con rasgos distintos al individuo evaluado compromete la fiabilidad de los resultados [7].

Los dientes suelen ser escogidos sobre la muñeca o la clavícula por su mayor robustez a algunos de los factores mencionados, como patologías o condición socioeconómica [8]. De todos ellos, merece la pena destacar los terceros molares (comúnmente conocidos como “muelas del juicio”). Mientras que el resto de la dentadura completa su desarrollo en torno a los 12 o 14 años, el desarrollo de las muelas del juicio suele continuar hasta más de los 20 años, convirtiéndose en unos candidatos idóneos para evaluar la mayoría de edad de un individuo una vez comenzada la adolescencia [9]. El desarrollo de un diente no está definido por cómo de visible es desde fuera (erupción en la boca), sino por el estado de su corona y raíces. Esta información sólo es accesible extirpándolo o a través de la imagen resultante de una radiografía, una tomografía o una resonancia magnética. En este marco se ubican los datos con los que se trabaja en este TFG: radiografías panorámicas de la dentadura, también conocidas como ortopantomografías o OPG (Figura 1.1).

Aunque en Europa sean la vía más utilizada a falta de registros oficiales, los métodos habituales de FA para estimar la edad legal tienen un gran margen de mejora. De entre sus principales críticas, son relevantes para este proyecto que los resultados suelen presentar amplios márgenes de error [10] y/o dependen demasiado del criterio del experto que realiza la prueba [11]. Teniendo en cuenta que la FA apenas se ha beneficiado del gran avance en inteligencia artificial (del inglés, *Artificial Intelligence* o AI) de los últimos años, resulta relevante utilizar técnicas de aprendizaje automático (del

inglés, *Machine Learning* o ML) y de aprendizaje profundo (del inglés, *Deep Learning* o DL) para implementar un método automatizado que intente solventar esa falta de precisión y objetividad.

Por todo lo anteriormente expuesto, este TFG propone **un método automatizado, basado en técnicas de ML y DL, para estimar de edad legal de un individuo a partir de una OPG**.

1.2. Motivación

La FA necesita modernizarse. Este área de la ciencia se ha mantenido al margen de los grandes avances en el campo de la AI de la última década [11]. Las herramientas “inteligentes” ya son capaces de igualar o superar el rendimiento humano en tareas como el reconocimiento o la generación de imágenes, texto y audio [12, 13, 14]. Sorprende especialmente que no haya formado parte de los hitos alcanzados en la visión por computador (del inglés, *Computer Vision* o CV), teniendo en cuenta que buena parte de sus métodos, incluidos los métodos de referencia para estimar la edad legal, consisten en analizar de imágenes de forma visual. Un buen ejemplo a seguir para la FA sería la medicina clínica, campo en el que se utilizan el mismo tipo de imágenes (radiografías, tomografías o resonancias) y en el que ya se han evidenciado extensamente las bondades de aplicar este tipo de técnicas [15, 16]. Resumiendo, el uso de ML presenta resultados prometedores en otras disciplinas íntimamente relacionadas con la FA, por lo que el interés de fusionar ambas disciplinas está más que justificado.

El método resultante de esta fusión podría ser beneficioso en varios aspectos. Por un lado, su mero carácter automático reduciría notablemente el coste en tiempo y recursos de la técnica, ahorro indispensable en escenarios con un número alto de individuos a identificar o para acelerar la tramitación de peticiones de asilo. Por otro lado, el uso de ML podría solucionar la falta de precisión y objetividad que en ocasiones acusa la metodología forense. Respecto a la falta de objetividad, algunos de los métodos más utilizados en este contexto [17, 18, 19] se basan en comparar una región anatómica (visualmente o colocando marcadores a mano) con una referencia (tabla de valores o atlas de estados). Esta forma tan manual de operar es propensa errores y compromete la reproducibilidad de los resultados, así como la admisibilidad de los informes forenses en un proceso legal. Un ejemplo de esta falta de reproducibilidad es el siguiente caso: España, 2009, un migrante es datado con tres edades distintas en un intervalo de dos semanas por parte tres expertos distintos que utilizaron la misma técnica [17]. Dos peritos lo dataron como menor (16 y 17) y uno como adulto (18). No fue un caso aislado. La técnica sigue siendo una de las más utilizadas nacional e internacionalmente. En [7], pueden encontrarse éste y otros casos que reflejan

las limitaciones de la metodología forense para la estimación de la edad en España.

Los dos párrafos anteriores motivan la necesidad de aunar AI y FA para crear un método que determine la edad legal de un individuo. No obstante, merece la pena desarrollar por qué la FA y los métodos habituales para la estimación de la edad legal son relevantes, independientemente del enfoque basado en ML que propone este TFG. La FA es vital en tareas de la identificación humana. Son más baratas, más rápidas y pueden aplicarse en un abanico más amplio de escenarios que otros métodos de identificación como el test de ADN o el de huella dactilar [11]. Si bien estos últimos son más precisos, resulta imposible aplicarlos cuando no se tienen registros previos del individuo a identificar o cuando el cuerpo está en mal estado (los tejidos blandos que utilizan se deterioran más fácilmente que los huesos). La FA se convierte, por tanto, en la única herramienta disponible para labores tan relevantes como la identificación de desaparecidos, de víctimas de crímenes, de catástrofes naturales, de guerras, de atentados terroristas o en fosas comunes. Escenarios donde los restos de uno, cientos o miles de individuos pueden encontrarse desmembrados, quemados, desfigurados y/o mezclados con otros. Aparte de las ventajas y aplicaciones descritas en este apartado, los métodos forenses que permiten discernir si un individuo es mayor o menor de edad, tienen aún más relevancia. La distinción es indispensable en posibles casos de pedofilia y trata de personas, así como en el caso de migrantes no documentados. Ya se ha comentado que este último contexto de aplicación es el que más miradas atrae. No obstante, parece pertinente añadir algunas cifras que respalden de forma cuantitativa esta afirmación.

- En 2020, el número de migrantes internacionales en el mundo se estimó en 281 millones, un 3.5 % de la población mundial. De estos, casi un 15 % eran menores de edad [20].
- En 2022, el conjunto de países de la Unión Europea recibió 222.745 peticiones de asilo de supuestos menores (lo hacían por primera vez en el país en el que la solicitaban). Se acumularon a las peticiones todavía pendientes de resolución del año anterior. Se respondieron 212.195 peticiones de este acumulado (176.495 resoluciones en juzgado de primera instancia y 35.700 resoluciones definitivas). A final de año, 187.880 solicitudes quedaron pendientes de respuesta [21]. Se resolvieron menos peticiones de las que se recibieron y el acumulado pendiente a final de año no distaba demasiado de la cantidad recibida durante éste. Ésta es una tendencia que se cumple en casi todos los años de la última década.
- En 2022, España recibió 20.580 de las peticiones anteriores. También se acumularon a las todavía pendientes del año anterior. Se cerraron

14.580 peticiones (13.855 resoluciones en juzgado de primera instancia y 725 resoluciones definitivas). A final de año, 22.730 solicitudes quedaron pendientes de respuesta [21]. España fue aún más ineficiente que la totalidad de la Unión Europea.

1.3. Objetivos

El objetivo principal de este TFG es el desarrollo de **un método automatizado, basado en técnicas de ML y DL, para estimar la edad legal de un individuo a partir de una OPG**. Recuérdese que la edad legal hace referencia a la disyunción entre minoría o mayoría de edad.

La distinción entre ML y DL del enunciado anterior es intencional. El DL es una familia concreta de técnicas dentro del ML. En este TFG, se exploran sus diferencias proponiendo modelos basados en ambas aproximaciones. No obstante, las dos familias de modelos comparten una misma filosofía, utilizan un conjunto de OPG de individuos cuya edad legal se conoce (datos de entrenamiento) para aprender de forma autónoma (algoritmo de aprendizaje) a determinar la de un nuevo individuo de edad desconocida (véase la Figura 1.2).

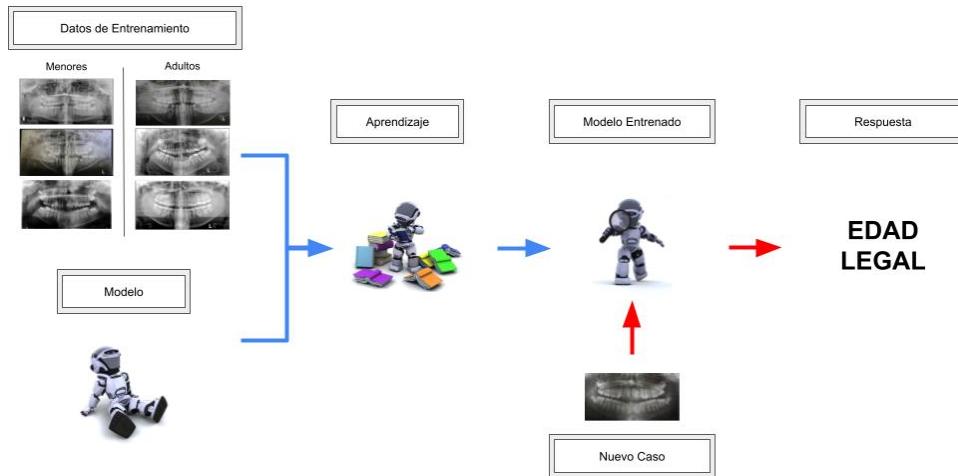


Figura 1.2: Resumen esquemático de un modelo basado en ML o DL para estimar la edad legal a partir de una OPG. Imágenes del robot extraídas de [22]

Alcanzar el objetivo principal mencionado requiere de la consecución de los siguientes objetivos parciales.

- Revisar de forma intensiva de la bibliografía relacionada con la estimación de la edad legal: migración, prácticas y recomendaciones nacionales e internacionales, metodología de FA y empleo de técnicas de ML, DL o CV en aplicaciones de FA.
- Diseñar, implementar y validar modelos basados enfoques tradicionales (ML) frente a modelos basados enfoques más novedosos (DL).
- Analizar comparativamente los resultados obtenidos. Se comparan las soluciones propuestas entre ellas y/o con el estado del arte.

1.4. Planificación del Proyecto

El TFG es una asignatura valorada en 12 créditos ECTS, teniendo en cuenta que un crédito equivale a 25 horas de trabajo, durante la etapa de planificación se estableció un total de 300 horas a repartir en el tiempo disponible para la realización del proyecto. Dado que la supervisión de los tutores estaba pactada desde el curso anterior, ese tiempo era de un curso completo, desde mediados de septiembre de 2023 a mediados de junio de 2024, un total de 9 meses. Se decidió repartir la carga de trabajo de forma homogénea. Esto sería beneficioso tanto para el alumno como para los tutores. No obstante, para el reparto de horas se consideró un intervalo de tiempo más garantista, 6 meses. De esta forma, la planificación contemplaba vacaciones, festivos e imprevistos de carácter menor. Considerando que cada mes tiene 4 semanas, el cómputo final fue de unas 12 o 13 horas semanales a dedicar al proyecto, 2 horas y media al día (dejando libres los fines de semana libres).

Este TFG está más orientado a la investigación que el desarrollo, por lo que una vez definidos los objetivos y necesidades del procedimiento experimental, los requisitos de la implementación no cambiaron demasiado. Requisitos que a su vez no son demasiado complejos, estrictamente, la implementación sólo necesita dar soporte a los experimentos a realizar, la fácil interacción y la usabilidad no eran focos principales del proyecto (aunque también se tuvieron en cuenta). Todos estos factores resultaron en la elección de una metodología de desarrollo software en cascada, caracterizada porque no permite retroceder a una fase del desarrollo del proyecto una vez finalizada. No obstante, este es un escenario bastante idealizado y poco realista, por lo que se optó por un desarrollo en cascada con retroalimentación, donde sí se permite volver a etapas anteriores para realizar ajustes. Cada fase o etapa del ciclo de vida del proyecto se definió de la siguiente forma.

- Análisis de requisitos: Reuniones iniciales con los tutores y expertos forenses para establecer el contexto (antropología forense), tema de la investigación (estimación de la edad legal a partir de OPG) y cómo

afrontarla (experimentos a realizar). También se incluye la revisión de la bibliografía relacionada. Se establecen así los requisitos y objetivos del TFG.

- Diseño: Descripción detallada de unas herramientas y un flujo experimental que permita al investigador ejecutar y analizar los experimentos de la forma más rápida y sencilla posible. Pruebas preliminares de concepto para determinar si es posible implementar todo lo requerido.
- Implementación: Implementar y depurar todas las funcionalidades descritas en la sección anterior. Merece la pena destacar la exploración y elección de herramientas externas a utilizar como *PyTorch* [23], *Scikit Learn* [24] y *Captum* [25].
- Pruebas: Realización de experimentos, tanto preliminares para volver a comprobar que la implementación es correcta, como para generar los resultados finales mostrados en la sección de experimentación.

Fase	Duración (semanas)	S	O	N	D	E	F	M	A	M	J	
		1	2	3	4	1	2	3	4	1	2	3
Análisis de Requisitos	8											
Diseño	8											
Implementación	8											
Pruebas	12											

Figura 1.3: Planificación temporal inicial. Se dejó algo más de tiempo para pruebas porque el entrenamiento de modelos de ML puede consumir bastante tiempo.

Teniendo todo lo anterior en cuenta, se planteó la planificación temporal mostrada en la Figura 1.3. Aunque el número de horas invertido fue el mismo que el planificado, el desarrollo temporal real del proyecto no estuvo exento de diferencias respecto al inicialmente planteado. La alteración más notable fue provocada por unas prácticas de empresa extracurriculares a jornada completa durante 4 meses que obligaron al estudiante a mudarse de Granada a Valencia. Estas se firmaron a mediados de noviembre, comenzaron en enero y acabaron en abril. Todo ello provocó una pausa en el desarrollo del proyecto durante el primer mes del año, una reducción del número de horas semanales dedicadas al proyecto durante los meses que duraron las prácticas, y el incremento correspondiente en la carga de trabajo en meses previos (cuando ya se sabía que se iban a realizar las prácticas) y posteriores. Otro factor que ralentizó el desarrollo del proyecto fue el uso de un marco software nunca antes utilizado para la mayor parte de la implementación: *PyTorch*. Además, esta no fue la elección inicial. Se comenzó con *FastAI* [26] por ser más sencilla de utilizar, pero su alto nivel de abstracción y limitadas funcionalidades impendían implementar lo requerido. Así pues, el desarrollo temporal final del proyecto puede observarse en la Figura 1.4.

Fase	Duración (semanas)	S	O	N	D	E	F	M	A	M	J	
		1	2	3	4	1	2	3	4	1	2	3
Análisis de Requisitos	8											
Diseño	8											
Implementación	8											
Pruebas	12											

Figura 1.4: Distribución temporal final. En azul, los períodos en los que se trabajó al ritmo previsto. En verde, se trabajó a un ritmo menor. En rojo, se trabajó a un ritmo mayor. También puede apreciarse la pausa de enero.

Finalmente, para estimar el coste del proyecto, se contabilizó el salario del ingeniero encargado y el equipo necesario para realizarlo. Para computar el salario se multiplicaron las horas de trabajo invertidas (aproximadamente 300 horas) por un salario de 35 €/hora, una cantidad habitual para un responsable de I+D o un investigador senior de una empresa tecnológica. A esta cantidad, tuvo que añadirsele el coste del equipo utilizado. Los resultados pueden encontrarse desglosados en el Cuadro 1.1.

Elemento	Coste [€]
Salario	4500.00
GPU (MSI RTX 3070 - 8GB)	559.90
CPU (AMD Ryzen 7 3700X - 3.6GHz)	319.90
Placa Base (MSI X470)	79.99
RAM (Corsair Vengeance LPX DDR4 3200 - 16GB)	66.01
Fuente de Alimentación (Corsair CV650 650W)	59.99
Caja	44.99
Periféricos	150.00
Otros	50.00
Total	5830.78

Cuadro 1.1: Estimación de costes. Para los componentes del equipo, se añade el precio de compra en 2022.

Capítulo 2

Fundamentos Teóricos

2.1. Aprendizaje Automático

El ML es una rama de la AI centrada en el desarrollo de algoritmos capaces de resolver un problema aprendiendo de un conjunto de datos de forma automática, en lugar de ejecutando una secuencia de instrucciones explícitamente programadas por un ingeniero [27]. Estos algoritmos, también conocidos como modelos, extraen información de los datos de entrada (patrones o relaciones entre instancias) y utilizan esta información para mejorar en la realización de una tarea, todo de forma autónoma. A este proceso de aprendizaje también se conoce como entrenamiento, ya que su carácter progresivo e iterativo recuerda al acondicionamiento físico de un deportista.

Este enfoque empírico, resolver un problema a partir de información extraída de los propios datos, convierten a los algoritmos de ML, también conocidos como modelos, en una opción idónea en el caso de problemas para los que una solución analítica es imposible o es demasiado costosa computacionalmente. Un ejemplo concreto de este tipo de problemas es el análisis de imágenes. Diferenciar si en una imagen hay un perro o un gato es sencillo para casi cualquier ser humano. Sin embargo, existen cientos de razas de perros, cientos de razas de gatos e infinitas formas de plasmarlos en una imagen, por lo que definir explícitamente un conjunto de instrucciones que permitan a un programa informático realizar esta tarea es sencillamente imposible. Un análisis análogo puede aplicarse al problema que aborda este TFG. Para un antropólogo especializado, estimar la edad legal de un individuo a partir de una OPG es sencillo, incluso puede describir el método que le ha llevado a esa conclusión para que otro ser humano realice la tarea. No obstante, convertir ese método en un algoritmo informático que reciba una OPG y genere la misma respuesta de forma automatizada, sin ML, es imposible.

En función del tipo de datos de los que se disponga, el tipo de tarea a resolver y la retroalimentación que recibe el modelo durante el aprendizaje, existen tres tipos de modelos ML: aprendizaje supervisado, no supervisado y aprendizaje por refuerzo [28]. En aprendizaje no supervisado, cada instancia del conjunto de datos está formada por un conjunto de características (un vector de cualquier dimensión de variables numéricas y/o categóricas), la tarea suele ser detectar agrupamientos de instancias (“clustering”), detectar instancias anómalas o reducir su dimensionalidad sin perder información y, durante el entrenamiento, el modelo evalúa su propia mejora sin ningún tipo de información externa. En aprendizaje supervisado, cada instancia del conjunto de datos está formada por un conjunto de características y una etiqueta (un número o una categoría), la tarea es predecir la etiqueta de una instancia dadas sus características y, durante el entrenamiento, el modelo evalúa su propio rendimiento comparando la etiqueta predicha con la etiqueta real del conjunto de casos que conoce. El aprendizaje por refuerzo es una aproximación mixta, el modelo no trabaja con etiquetas (como en aprendizaje no supervisado) pero, durante el entrenamiento, evalúa su rendimiento en base a una información externa (como en aprendizaje supervisado) que indica si lo hace bien o mal en términos generales, sin concretar cuándo o cómo operó incorrectamente. El problema que se trata en este TFG se enmarca en la segunda categoría, el aprendizaje supervisado. El objetivo es predecir la edad legal de un individuo (una categoría, mayor o menor de edad) a partir de una imagen (un tensor de números reales de dos o tres dimensiones). En términos más técnicos, se trata de un problema de clasificación binaria de imágenes.

Un modelo de aprendizaje supervisado recibe un dato de entrada y devuelve una etiqueta. El valor de la etiqueta predicha depende del valor de un conjunto de parámetros, también conocidos como pesos. Aprender significa ajustar el valor de estos parámetros con el fin de que la etiqueta devuelta sea la correcta para cualquiera de las instancias del conjunto de datos que se conoce (conjunto de entrenamiento). No obstante, el objetivo último no es que el modelo prediga correctamente las etiquetas de todo conjunto de entrenamiento. De hecho, esto es una tarea relativamente sencilla con un modelo lo suficientemente complejo (elevado número de parámetros) y/o un tiempo suficiente de entrenamiento. El objetivo es que el modelo prediga correctamente la etiqueta de casos fuera de este conjunto, que el modelo generalice [29]. Cuando un modelo no es capaz de generalizar, pero si predice bien las etiquetas del conjunto de entrenamiento, se dice que el modelo sobreajusta. El sobreajuste refleja que el modelo ha interiorizado información que sólo es relevante para los casos de entrenamiento (ruído estadístico). Encontrar el punto intermedio entre sobreajuste y generalización es el trabajo de un ingeniero informático en el contexto del ML.

2.1.1. Aprendizaje Profundo

El DL es un subconjunto de técnicas dentro del ML. Los modelos de esta subdisciplina, las redes neuronales, no necesitan que los datos de entrada sean sometidos a un proceso de extracción de características antes de que el modelo entrene con ellos [30]. Esta es la principal diferencia con el resto de técnicas de ML (conocidas como técnicas tradicionales): la extracción automatizada de características.

Para entender la importancia de este esta diferencia, primero debe explicarse qué es y por qué es necesaria la extracción de características en ML. Analícese el caso concreto de las imágenes. Aunque para un humano sea trivial interpretar una imagen, para un computador es mucho más difícil. La representación computacional de una imagen, un volumen de valores de una (blanco y negro) o tres dimensiones de profundidad (color), es muy sensible a fenómenos como cambios de iluminación, cambios de perspectiva, cambios de escala o posibles occlusiones (algunas partes de la imagen están tapadas). Por ejemplo, una misma foto con más o menos brillo, fácilmente reconocible como la misma para un humano, es totalmente distinta para un computador, no coincide el valor de casi ningún píxel de ambas imágenes. Puede aplicarse un razonamiento análogo a cualquiera de los fenómenos descritos previamente. Para solucionar estos problemas, las imágenes se someten a un proceso conocido como extracción de características. Las imágenes se transforman en una representación alternativa (conocida como descriptor de características) que debe encapsular la información más relevante de una imagen para poder predecir su etiqueta y ser robusta o incluso invariantes a los problemas mencionados. Básicamente, deben indicarle al modelo en qué características de los datos de entrada fijarse (y qué información desechar) para predecir su etiqueta. Esta práctica es especialmente necesaria en el caso de datos de alta dimensionalidad, categoría en la que también se enmarcan las imágenes. Cuanto mayor es la dimensionalidad de un dato, mayor es el riesgo de sobreajuste: más información irrelevante y más parámetros (más potencia) suele necesitar el modelo.

Para los modelos tradicionales de ML, esta etapa es manual e independiente del entrenamiento del predictor. En el caso de las imágenes, se escoge un método del amplísimo catálogo existente para extraer el descriptor de todo el conjunto de entrenamiento (HOG, SIFT, KAZE, SURF, FAST ...) [31] y después se entrena el predictor pertinente (SVM, “random forest” o regresor logística) con dichos descriptores. El problema es que la generalidad y, por extensión, la conveniencia de un descriptor para una tarea específica (como determinar la edad legal de un individuo a partir una ortopantomografía) es bastante limitada. Por contra, para los modelos basados en DL, esta primera etapa es automática y no es independiente del entrenamiento del predictor. Las redes neuronales aprenden, simultánea y automáticamente,

te, a extraer características relevantes de los datos de entrada y a predecir sus etiquetas en base a ellas.

Por un lado, esta automatización libera al ingeniero de tener que encontrar el extractor de características adecuado, una tarea tediosa y difícil, que suele requerir de conocimiento experto del problema concreto que quiera resolverse. Esto le permite, por ejemplo, implementar un modelo que prediga la edad legal de un individuo a través de una OPG sin haber estudiado antropología forense. Por otro lado, cuando existe una cantidad de datos suficiente, proporciona mucho mejores resultados que la extracción manual. Así lo demuestran los resultados sobre ImageNet, el “benchmark” de referencia internacional en CV [32]. Desde su irrupción en 2012, las redes neuronales convolucionales siempre han formado parte del mejor modelo propuesto en tareas de clasificación, segmentación, localización o detección de imágenes. El rendimiento humano en clasificación lo superaron allá por 2015 [33].

2.2. Redes Neuronales

2.2.1. Arquitectura

Las redes neuronales artificiales o simplemente redes neuronales son modelos formados por un conjunto de unidades simples de cómputo llamadas neuronas [34]. Vagamente inspiradas por el funcionamiento de las células homónimas, estas unidades toman unas señales de entrada, las transforman linealmente en base a unos parámetros (suma ponderada con desplazamiento) y aplican una transformación no-lineal al resultado (función de activación) para generar una señal de salida. Consultar la Figura 2.1. La aplicación de una transformación no-lineal es vital para este tipo de modelos. Sin ellas, la potencia expresiva de la red sería similar al de una única neurona, el perceptrón, un modelo que sólo puede predecir la etiqueta de un dato de entrada si esta es una combinación lineal de sus características [35]. Con ellas, la potencia expresiva de la red se dispara, son capaces de aproximar una amplia variedad de funciones complejas. De hecho, una red con solamente una capa oculta es ya un aproximador universal; es decir, puede aproximar cualquier función continua [36].

Las neuronas se estructuran en módulos llamados capas. En las redes neuronales completamente conectadas, el caso más básico de red neuronal, todas las neuronas de una capa toman todas las salidas de las neuronas de la capa anterior como señales de entrada, a excepción de la primera capa. Las neuronas de la primera capa (capa de entrada) toman como señales entrada las instancias del conjunto de datos. En el caso de la clasificación de imágenes, la entrada son los píxeles de la imagen. La estructura de la última capa (capa de salida) depende del problema a solucionar. Si se tra-

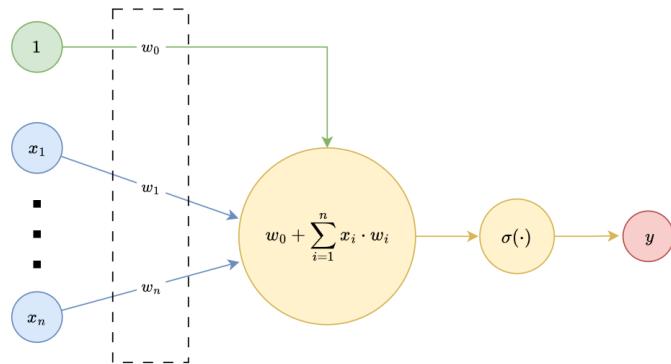


Figura 2.1: Esquema del funcionamiento de una neurona. Transformación lineal (suma ponderada con desplazamiento) de un conjunto de señales de entrada (x_i) en base a unos parámetros (w_i) a la que se le aplica una función no lineal (σ) para generar una señal de salida (y).

ta de un problema de regresión (la etiqueta es un número real), la última capa suele estar formada por una neurona sin transformación no lineal, la salida de la red se interpreta directamente como la etiqueta. Si se trata de un problema de clasificación multiclas (más de dos clases), la última capa suele estar formada por tantas neuronas como clases, la salida de cada una se interpreta como la probabilidad sin normalizar de que el dato procesado pertenezca a esa clase (también conocida como “logit”). En el caso de la clasificación binaria, la última capa suele estar formada por una única neurona cuya función de activación sea una sigmoide. De esta forma, la salida es un valor numérico entre 0 y 1 que se interpreta como la probabilidad de que la imagen pertenezca a la clase notada como “1” (también conocida como clase positiva). La probabilidad de la clase notada como “0” (también conocida como clase negativa) es $1 -$ la salida de la red. A las capas que no son ni la primera ni la última, se les conoce como capas ocultas. En la Figura 2.2 puede encontrarse un esquema de una red neuronal en el que se pueden identificar todos los elementos descritos hasta el momento.

La arquitectura descrita en el párrafo anterior es la responsable de que las redes neuronales puedan extraer características de forma automática, el rasgo diferencial de DL. La operación que realiza cada capa puede interpretarse como una transformación de una representación de los datos de entrada en otra; es decir, cada capa transforma un conjunto de características en otro. Como el funcionamiento de las neuronas y, por extensión, el de las capas dependen de parámetros que la red aprende de forma automática, la extracción de estas características también es automática. Otro aspecto importante de las características extraídas por cada capa es que cada vez son más complejas. Las primeras capas encuentran relaciones simples entre

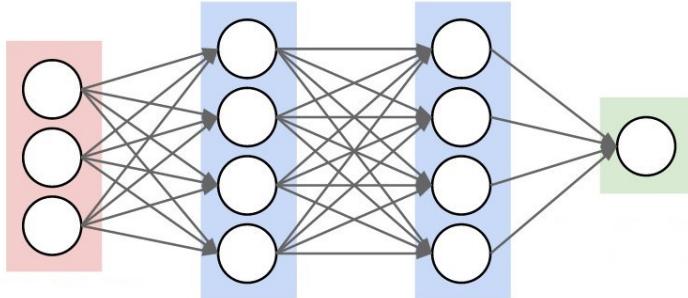


Figura 2.2: Ejemplo de una red neuronal totalmente conectada poco profunda. Capa de entrada con tres neuronas (rojo). Dos capas ocultas de cuatro neuronas (azul). Capa de salida con una neurona (verde). Tanto el número de neuronas por capa como el número de capas ocultas, puede modificarse (añadir o eliminar) para adaptarse al problema concreto. Imagen extraída de [37].

los datos, mientras que las siguientes capas utilizan esta información para inferir características progresivamente más complejas. Este tipo de características, jerárquicas y progresivamente más complejas, proporciona muy buenos resultados para datos de alta dimensión, ya que las relaciones entre componentes de una misma instancia de datos suelen tener la misma naturaleza, jerárquicas y complejas; tómese por ejemplo el caso de las imágenes.

Esta estructura también puede interpretarse como una función (f) cuya salida ($f(W, x)$) para una entrada concreta (x) depende del valor de un conjunto de parámetros (W). Aprender o entrenar consiste en modificar el valor de estos parámetros para maximizar el rendimiento del modelo en la predicción de etiquetas sobre el conjunto de entrenamiento. Para completar la definición de estos modelos, solo quedarían definir dos conceptos: cómo modificar de los parámetros de los que depende la salida (algoritmo de aprendizaje) y cómo medir el rendimiento del modelo (función pérdida).

2.2.2. Función Pérdida

La función pérdida mide el rendimiento de un modelo. Más concretamente, mide cómo de mal (o bien) opera dicho modelo sobre un conjunto de datos determinado. Para el caso concreto de una red que clasifica imágenes, la función pérdida toma la salida del modelo para un conjunto de imágenes (probabilidad de cada una de las clases posibles para cada imagen), calcula la pérdida para cada una de ellas y computa la suma o el valor medio. Cuanto más alto sea este valor, menor es el número de imágenes correctamente clasificadas y viceversa. Una imagen se clasifica correctamente cuando la clase con mayor probabilidad predicha corresponde a la clase real (etiqueta) de

la imagen. No obstante, clasificar correctamente todas las imágenes no es el único factor que suele tenerse en cuenta, también suele importar lo “segura” que está la red: cuánto más alta sea la probabilidad de la clase positiva, aunque ya sea la clase de mayor probabilidad, mejor. Puede hacerse un análisis análogo para las probabilidades de las clases negativas, pero en este caso el objetivo es que la probabilidad asociada sea lo más baja posible.

Para este proyecto se escogió una de las funciones pérdidas más habituales en el contexto de la clasificación, la entropía cruzada negativa binaria media. Sea una red definida por el conjunto de parámetros (W), un conjunto de imágenes (X) de etiqueta binaria conocida (Y), x_i una imagen concreta de este conjunto, y_i la etiqueta real de dicha imagen y $f(W, x_i)$ la predicción de la red para dicha imagen (probabilidad de la clase positiva), la entropía cruzada negativa binaria media (\bar{L}_{BCE}) puede definirse de la siguiente forma.

$$\bar{L}_{BCE}(X, Y, W) = -\frac{\sum[(1-y_i)\cdot \log(1-f(W, x_i)) + (y_i)\cdot \log(f(W, x_i))]}{N}$$

Esta función es interesante por varios motivos. En primer lugar, es diferenciable respecto a las salidas del modelo, que a su vez son diferenciables respecto a los parámetros de las neuronas de la red. Esto permite que la función perdida, y por extensión el rendimiento del modelo, pueda optimizarse de forma automática con métodos basados en el descenso de gradiente (algoritmos detallados más adelante 2.2.3). En segundo lugar, como ya se anticipó, no sólo tiene en cuenta si el modelo acierta o falla, sino que tiene en cuenta la certeza o incertidumbre con la que lo hace. No es lo mismo fallar prediciendo la clase correcta con un 0.01 de probabilidad que con un 0.49 de probabilidad. Cuanto mayor es probabilidad de la clase correcta (y_i), menor es el valor de esta función perdida. La Figura 2.3 representa el valor de la entropía cruzada negativa ($-\log(p)$), para una sola imagen, en función del valor de la probabilidad predicha para la clase correcta (p). En la práctica, el uso de esta función hace que el modelo no sólo entrena hasta acertar, sino hasta acertar con la mayor seguridad posible.

2.2.3. Algoritmo de Aprendizaje

Para el caso concreto de la clasificación de imágenes, aprender consiste en modificar el valor de los parámetros de la red para maximizar el número de imágenes correctamente clasificadas. Un posible criterio para optimizar estos parámetros podría ser la búsqueda local aleatoria. Se genera un conjunto inicial aleatorio de parámetros y se modifica aleatoriamente. Si el modelo resultante es mejor (clasifica correctamente un mayor número de imágenes), se repite el proceso con el nuevo valor de los parámetros como conjunto ini-

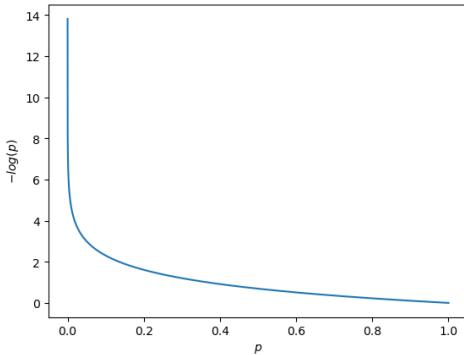


Figura 2.3: Entropía cruzada negativa ($-\log(p)$) en función de la probabilidad predicha para la clase correcta (p) para una sola imagen.

cial. Si el modelo resultante es peor, se descarta el nuevo conjunto de valores y se repite el proceso con el conjunto inicial de parámetros. Sin embargo, este proceso es poco eficiente para modelos con muchos parámetros, como es el caso de las redes neuronales.

La alternativa que se escoge en el DL es el descenso de gradiente (del inglés, *Gradient Descend* o GD) o variantes de éste [38]. Aunque esta explicación se centra en la versión clásica de GD, los principios aquí expuestos son la base del resto de variantes. Se trata de un método de optimización. Permite encontrar óptimos locales de una función diferenciable cualquiera de forma iterativa. Únicamente necesita el valor de dicha función y el de su gradiente en un punto del espacio en el que están definidos. Su funcionamiento se basa en que el gradiente de una función en un punto informa de cómo deben modificarse (en magnitud y sentido) cada una de las componentes de ese punto para maximizar el valor de la función a nivel local. Cuanto mayor sea el gradiente sobre una componente (derivada parcial de la función sobre ese componente) más aumentará el valor de la función si se aumenta el valor de esa componente de forma infinitesimal. Por ello, para maximizar una función, es razonable modificar la componente de forma proporcional (factor λ o tasa de aprendizaje) a la magnitud de su gradiente y en el mismo sentido. Si se hace en sentido contrario, se estaría minimizando la función.

Bajo este pretexto, cobra sentido la función pérdida. Una función diferenciable respecto a los parámetros de la red que refleja cómo de mal rinde el modelo y cuyo valor se conoce para un punto. Calculando el valor del gradiente respecto a los parámetros del modelo en ese mismo punto, puede aplicarse GD (actualizar los parámetros de forma proporcional y sentido contrario al gradiente de forma iterativa) para minimizar la función perdida, maximizando así el rendimiento del modelo. Recuérdese que la función perdida es opuesta al rendimiento. Merece la pena incidir en que la optimi-

zación ocurre a nivel local, luego el modelo alcanzado no tiene por qué ser el mejor en términos absolutos. Puede estancarse en óptimos locales, aunque en la práctica, no suele ser un problema para las redes neuronales [39, 40].

Así pues, el entrenamiento de un modelo que utiliza GD como algoritmo de aprendizaje podría resumirse de la siguiente forma, también conocida como bucle de entrenamiento.

1. Para las imágenes del conjunto de entrenamiento (X), se genera la salida del modelo ($f(W, X)$) y, junto a las etiquetas reales dichas imágenes (Y), se calcula el valor de la función pérdida ($L(W, X, Y)$). Etapa conocida como “paso hacia delante” o ”forward pass”.
2. Utilizando el algoritmo de “backward propagation”, se computa el gradiente de la función pérdida y se actualizan los parámetros del modelo en base a este gradiente para minimizarla. Etapa conocida como paso hacia atrás o “backward pass”.

$$W_{i+1} = W_i - \lambda \cdot \frac{\partial L(W, X, Y)}{\partial W_i}$$

3. Se repite el proceso hasta que sea pertinente.

Tasa de Aprendizaje

El gradiente de la función pérdida es útil para actualizar los parámetros del modelo; no obstante, GD y sus variantes admiten un hiperparámetro adicional que modula la agresividad de esta actualización: la tasa de aprendizaje (λ). En el caso concreto de GD, λ se multiplica por el gradiente antes de restárselo a los parámetros. Controla la agresividad de la actualización de forma proporcional.

El valor de λ depende del criterio del ingeniero y se determina de forma experimental. Su papel es fundamental en el proceso de optimización. Si λ tiene un valor demasiado alto, el proceso de optimización no converge, incluso puede divergir. El entrenamiento de una red no diferiría demasiado de una búsqueda aleatoria. Si, por el contrario, λ tiene un valor demasiado bajo, la convergencia es demasiado lenta. El entrenamiento de una red requeriría muchas más iteraciones para alcanzar un rendimiento aceptable y corre un mayor riesgo de estancarse en un mínimo local. En la Figura 2.4 puede encontrarse un ejemplo simplificado de los dos fenómenos a la hora de optimizar una función que depende de un único parámetro.

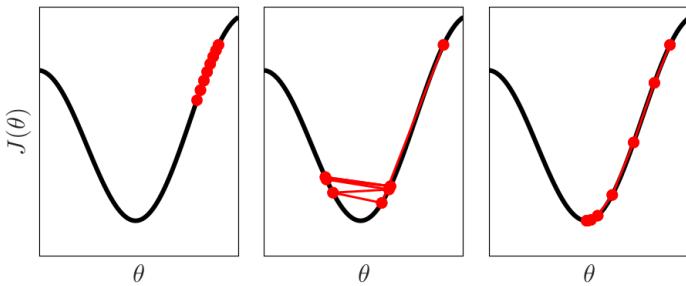


Figura 2.4: Ejemplo simplificado del efecto del valor de la tasa de aprendizaje (λ) en la optimización por descenso de gradiente de una función que depende de un sólo parámetro ($J(\Theta)$). Si λ es demasiado baja, la convergencia es muy lenta (izquierda). Si λ es muy alta, el proceso no converge (centro). Si λ tiene un valor óptimo, el proceso converge de forma rápida (derecha). Imagen extraída de [41].

Tamaño del “Batch”

El algoritmo de aprendizaje descrito exige que la función pérdida y su gradiente se evalúen sobre todo el conjunto de entrenamiento en cada iteración del bucle de entrenamiento para actualizar los parámetros del modelo. No obstante, en la práctica, esto no suele ser así. Suele utilizarse una variante conocida como descenso de gradiente por “mini-batches”. En esta variante, se optimizan los parámetros del modelo sobre un subconjunto aleatorio de casos. Obviamente, este método no garantiza que, en cada iteración, se minimice la función pérdida para todo el conjunto de entrenamiento. Sin embargo, permite actualizar el modelo de forma menos costosa (sin tener que evaluar todos los casos) en base a una guía (función pérdida y gradiente sobre un subconjunto de casos) que no suele ser una mala aproximación de la pérdida para el conjunto de datos completo. Algo es especialmente útil cuando el tamaño del conjunto de entrenamiento es muy alto. Otra ventaja de emplear este método es que introduce una aleatoriedad en el proceso de optimización que puede permitir escapar de óptimos locales. Normalmente, el conjunto de entrenamiento se divide en “mini-batches” disjuntos de forma aleatoria con los que se alimenta el modelo de forma secuencial. Cada vez que se agotan los “mini-batches”, se dice que el modelo ha completado una época y se repite el proceso de división “mini-batches” aleatorios.

El caso más extremo de descenso de gradiente por “mini-batches” se conoce como descenso de gradiente estocástico, aunque en muchas ocasiones se utiliza este nombre para referenciar al descenso de gradiente por “mini-batches”. En este caso, los “mini-batches” están formados por un sólo caso. No suele utilizarse porque no permite procesar casos en paralelo (ralentizando dramáticamente el tiempo de entrenamiento) y la pérdida sobre un sólo

caso puede no ser suficientemente representativa de la pérdida para todo el conjunto de entrenamiento.

Algoritmo de Derivación

Merece la pena destacar el algoritmo que se utiliza para calcular el gradiente de la función pérdida en un punto: “backward propagation” [42, 43]. Consiste en aplicar la “regla de la cadena” para calcular la derivada parcial de la función pérdida respecto a cada uno de los parámetros de la red. Un método de derivación mucho más eficiente las otras alternativas existentes: inferir la expresión analítica o calcular de forma computacional. Estas alternativas se vuelven inviables con el aumento de la complejidad de la arquitectura y/o el número de parámetros, algo que limitaría la potencia de los modelos. Por ilustrar de forma cuantitativa la necesidad de este algoritmo de derivación: la red neuronal más pequeña que se utiliza en este proyecto tiene más de 11 millones de parámetros. Imagínese calcular de forma analítica o computacional la derivada de la función que representa el modelo para cada uno de ellos. En la Figura 2.5 puede encontrarse un ejemplo simplificado de aplicación de “backpropagation”.

2.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (del inglés, *Convolutional Neural Networks* o CNN) son un tipo especial de red neuronal que tiene en cuenta que el dato de entrada es una imagen [45, 46]. Más concretamente, la arquitectura de las CNN, así como las operaciones que realizan, conservan la característica fundamental de una imagen para interpretarla, su estructura espacial. La estructura espacial de una imagen hace referencia a que el valor de un píxel sólo tiene sentido teniendo en cuenta el valor de los píxeles de su entorno. Consultar la Figura 2.6.

El problema de las redes neuronales convencionales (totalmente conectadas) es que esta estructura espacial se pierde al introducir las imágenes en el modelo, ya que una imagen debe transformarse en un vector de una sola dimensión. Uno pudiera imaginar que este problema pueden solucionarlo por sí solas, descubriendo/aprendiendo esta estructura durante el entrenamiento. No obstante, su elevado número de parámetros hace que el modelo sobreajuste antes de conseguirlo. Si, por ejemplo, se quisiera alimentar una red totalmente conectada con imágenes a color de 256 píxeles de alto y ancho, solamente las neuronas de la primera capa oculta tendrían $256 \cdot 256 \cdot 3 = 196.608$ parámetros cada una. Imagínese al añadir varias capas con varias neuronas. Las CNN solucionan este problema disponiendo las neuronas de cada capa de forma tridimensional, limitando el número de salidas de la capa anterior

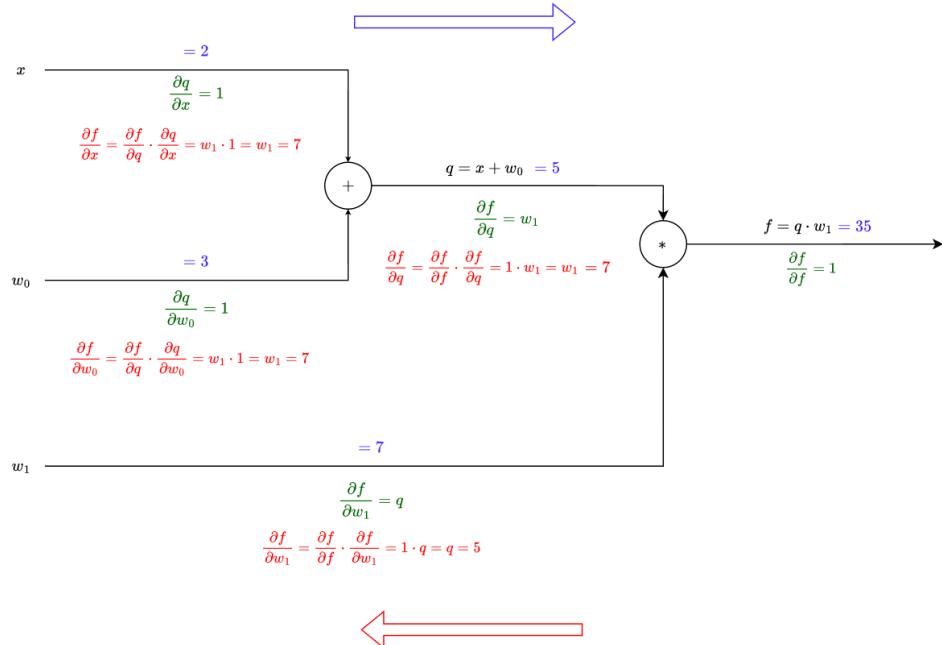


Figura 2.5: Ejemplo simplificado del algoritmo “backpropagation”. Sea la función $f(w_0, w_1, x) = (x + w_0) \cdot w_1$ y un punto en el que se evalúa dicha función $(w_0, w_1, x) = (3, 7, 2)$. El objetivo es calcular el valor del gradiente de la función (derivadas parciales) sobre x , w_0 y w_1 en ese punto. Antes de aplicar “backpropagation” la función debe descomponerse en funciones intermedias (f y q) cuyas derivadas parciales respecto a sus entradas sean triviales (verde). En este caso se utiliza un grafo computacional para ilustrar la descomposición. Después de computar el valor de la función perdida en $(3, 7, 2)$, y con ello el valor de las funciones intermedias ($q = 7$ y $f = 35$, en azul), “backpropagation” consiste en calcular el valor de la derivada parcial de $f(w_0, w_1, x)$ respecto a cada estado intermedio respecto aplicando la “regla de la cadena” de forma regresiva (rojo). En última instancia se obtiene el valor del gradiente sobre x , w_0 y w_1 . La notación utilizada no es arbitraria, $f(w_0, w_1, x)$ podría interpretarse como la función perdida de una red sencilla (w_0 y w_1) para una entrada x . En azul el “forward pass” y en rojo el “backward pass”. Ambos deben su nombre al sentido en el que se realizan las operaciones o fluye la información, de entrada a salida (hacia delante) o de salida a entrada (hacia atrás). Esquema inspirado en [44].



Figura 2.6: Ejemplo para ilustrar por qué la estructura espacial es vital para interpretar la información contenida en una imagen. A la izquierda, una foto del padre del redactor de este TFG (un conjunto de valores ordenados en el espacio de forma tridimensional). A la derecha la misma foto (mismo conjunto valores) pero eliminando su estructura espacial (posición en el espacio de los valores alterada de forma aleatoria).

que una neurona recibe (campo receptivo) y haciendo que las neuronas de una misma capa con una misma profundidad comparten parámetros (“parameter sharing”). Analizar por separado cada una de estas decisiones es algo confuso, así que a continuación se expone el funcionamiento emergente.

En una primera etapa (extracción de características), cada capa de la CNN transforma un volumen de entrada en un volumen de salida. Un volumen es un objeto de tres dimensiones, anchura, altura y profundidad o canal. El primer volumen que procesa una CNN es una imagen en escala de grises o RGB, un volumen con uno o tres canales de profundidad (respectivamente). Para generar el volumen de salida, las capas aplican alguna de las siguientes operaciones al volumen de entrada: una convolución o una agrupación. Convolucionar un volumen consiste en aplicarle un conjunto filtros convolucionales. La aplicación de cada filtro convolucional genera un mapa bidimensional (una imagen en escala de grises) en el que aparecen señaladas (píxeles con mayor intensidad) las ubicaciones de alguna característica del volumen al que se le ha aplicado, de ahí que se conozca como mapas de característica. Consultar la Figura 2.7. El volumen de salida se crea concatenando los mapas de características generados. La red aprende/ajusta el valor de los parámetros que definen los filtros durante el entrenamiento para extraer características relevantes en la predicción de la etiqueta. Como cada capa se alimenta de la salida de la anterior, los mapas de características generados por las capas iniciales señalan las ubicaciones de patrones sencillos (como bordes o esquinas), mientras que las siguientes capas generan mapas

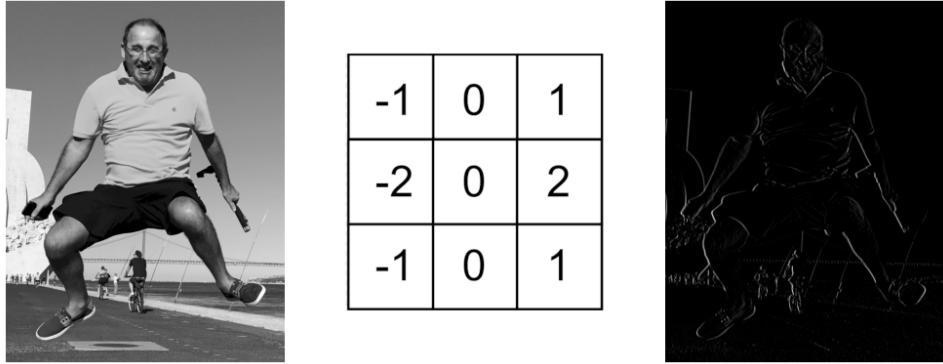


Figura 2.7: Ejemplo de convolución para localizar los bordes verticales de una imagen. A la izquierda, la imagen original en escala de grises. En el centro, el filtro de convolución empleado (filtro de sobel). A la derecha el mapa de características generado. Los píxeles del mapa señalan (con una mayor intensidad) las posiciones de la imagen original donde hay un cambio de intensidad (de menor a mayor, algunos bordes verticales). También se aplica una ReLU sobre el mapa generado (función de activación explicada en la Subsección 2.3.4).

que señalan las ubicaciones patrones complejos (hasta caras o animales) resultado de combinar la información de mapas de características más simples. Un proceso jerárquico similar al ocurrido en redes neuronales convencionales. La aplicación sucesiva de filtros también provoca que las últimas capas tengan en cuenta información proveniente de una mayor región de la imagen original para detectar características (tienen un mayor campo receptivo). Un efecto deseable, ya que los patrones complejos de una imagen suelen tener un mayor tamaño que los patrones simples que los forman. Entre capas de convolución, las CNN suelen intercalar capas de agrupación. Estas capas reducen la resolución en ancho y alto del volumen procesado aplicando, eliminan información prescindible de los mapas de características. Esto es deseable, ya que el objetivo de todo este proceso es convertir un dato con una dimensionalidad muy alta (imagen) en un conjunto de características que, aun encapsulando la información más relevante para predecir su etiqueta, tenga la menor dimensión posible (y a menor dimensión, menor sobreajuste).

En una segunda etapa (cabecera), los mapas de características generados por la última capa se convierten en un vector de características. Generalmente, esto se consigue añadiendo una capa extra que reduce la resolución del volumen de mapas a $1 \times 1 \times n_{mapas}$ e interpretando este volumen como un vector. Cada componente de este vector es una característica extraída de la imagen. Estas características se utilizan como la entrada de una red neu-

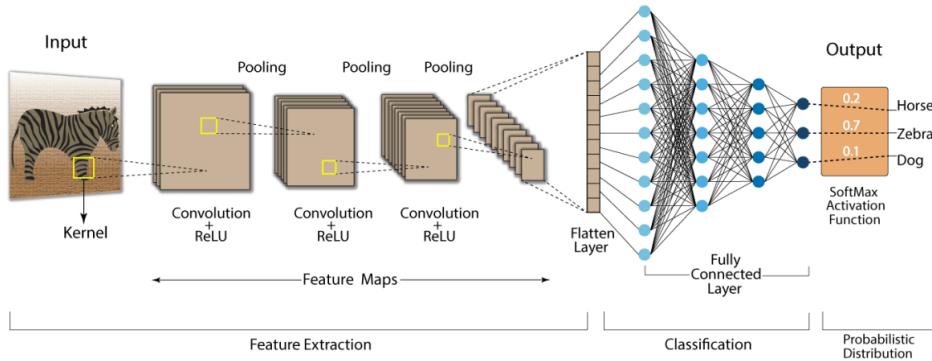


Figura 2.8: Esquema habitual de una CNN utilizada para clasificación multiclase de imágenes. Los detalles concretos de cada elemento mencionado se desarrollan a lo largo de esta subsección. Figura extraída de [47].

ronal completamente conectada cuya salida se adapta al problema deseado. Los parámetros de esta cabecera también se aprenden de forma automática. Dado que el predictor utilizado sigue siendo una red neuronal completamente conectada, es deseable que las características extraídas no tengan una dimensión demasiado alta, para evitar el sobreajuste.

En conclusión, a través de una estructura secuencial de capas de convolución “entrenables” (intercaladas eventualmente por capas de agrupación) y una cabecera totalmente conectada también “entrenable”, las redes CNN son capaces de aprender, automática y simultáneamente, a extraer características útiles para predecir la etiqueta de una imagen y a predecirla. Nótese que ambas operaciones (convolución o agrupación) trabajan de forma conjunta para extraer características que tengan en cuenta la estructura espacial de la imagen, el problema fundamental de las redes neuronales convencionales. Consultar la Figura 2.8 para el esquema de una CNN utilizada para clasificación multiclase de imágenes.

2.3.1. Capas Convolucionales

Una capa convolucional aplica un conjunto de filtros convolucionales a un volumen de entrada para generar, con cada uno de ellos, un mapa de características. La utilidad de este proceso ya ha sido discutida. Sin embargo, queda por detallar lo que es un filtro convolucional y cómo se aplica. Un filtro convolucional es un volumen de parámetros y un parámetro extra para el desplazamiento. Aplicar un filtro sobre un volumen de entrada consiste en generar una matriz (mapa de características) cuyo píxel ij -ésimo es el resultado de combinar linealmente el píxel ij -ésimo del volumen original con los píxeles de su entorno en base a los valores del filtro. Por clarificar, un píxel

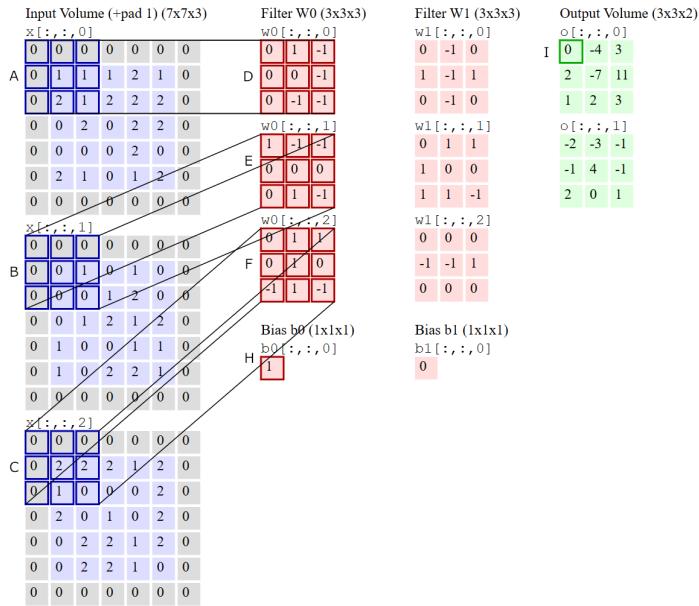


Figura 2.9: Ejemplo de aplicación de una capa convolución. Cada volumen se expresa como un conjunto de matrices. En azul, el volumen de entrada. En gris, el relleno espacial añadido (“padding”). En rojo, los filtros de la capa de convolución aplicada (w_0 y w_1). Ambos utilizan un “stride” igual a 2. En verde, el volumen de salida. Su profundidad coincide con el número de filtros aplicados. Se señalan los valores utilizados para calcular el valor del píxel (0,0) del mapa de características generado por el filtro w_0 (I). Se realiza la siguiente operación: $I = A \odot D + B \odot E + C \odot F + H = (-4) + (0) + (3) + (1) = 0$. Imagen extraída de [37].

es un vector con tantos elementos como profundidad tenga el volumen al que pertenece. La red aprende/optimiza el valor de los parámetros de estos filtros para extraer características de las imágenes que minimicen la función pérdida. A modo de resumen y ampliación, a continuación se enumeran los hiperparámetros que definen una capa convolucional. La función de dichos hiperparámetros y el proceso completo quedan perfectamente ejemplificados en la Figura 2.9.

- Número de filtros. Define la profundidad del volumen de salida. Cada uno genera un mapa de características bidimensional. Esto puede interpretarse como que cada uno aprende a localizar una característica distinta del volumen de entrada. El conjunto de mapas resultante se concatena para generar un nuevo volumen tridimensional. Por ejemplo: una imagen RGB está formada por tres mapas de características concatenados, uno detecta el color rojo, otro el verde y otro el azul.

- Tamaño del filtro. Define el entorno de un píxel. Los píxeles de la imagen original que se tienen en cuenta para generar cada píxel del mapa generado. El ancho y alto de un filtro suele ser igual e impar (como 3x3, 5x5 o 1x1) y su profundidad debe ser igual a la del volumen al que se aplica. El tamaño del filtro condiciona el tamaño del volumen de salida.
- “Padding”. En ocasiones es deseable expandir el volumen de entrada, a lo ancho y alto, antes de aplicar un filtro; es decir, añadir bordes de relleno. Imagínese que se quiere aplicar un filtro 3x3x1 sobre un volumen 5x5x3. Sin borde de relleno, ¿cómo aplicaría el filtro sobre el píxel (0,0) del volumen?. El “padding” de un filtro define cuántas unidades se expande las dimensiones espaciales con valores de relleno (generalmente, 0). Condiciona las dimesiones del mapa de características generado.
- “Stride”. Define cada cuántos píxeles del volumen original se aplica filtro. Con un “stride” de 1, el filtro se aplicaría sobre todo el volumen de entrada. Con un “stride” de 2, el filtro se aplicaría de forma alternativa: un píxel sí un píxel no. Evidentemente, cuánto mayor sea, menor es la dimensión del mapa de características generado. Como puede observarse, las capas convolucionales con un “stride” > 1 también reducen la dimensión del volumen procesado (función propia de las capas de agrupación).

2.3.2. Capas de Agrupación

Una capa de agrupación o capa de “pooling” reduce la resolución a lo ancho y alto de un volumen de entrada. Su objetivo es eliminar información redundante y/o reducir la complejidad de la representación del volumen procesado y, con ello, minimizar el potencial sobreajuste. Se genera un nuevo volumen de salida en el que el píxel ij-ésimo suele ser el resultado de tomar el valor medio (“average pooling”) o el máximo (“max pooling”) del entorno del píxel ij-ésimo del volumen original. Dos hiperparámetros controlan esta operación: “pool size”, que define el entorno de un píxel y “stride”, ya explicado para las capas convolucionales. En la Figura 2.10 puede encontrarse un ejemplo de ambas técnicas sobre un volumen formado por un único mapa de características.

2.3.3. Capas de Completamente Conectadas

También conocidas como cabecera completamente conectada. Se trata de una red completamente conectada como la descrita en la Subsección 2.2.1.

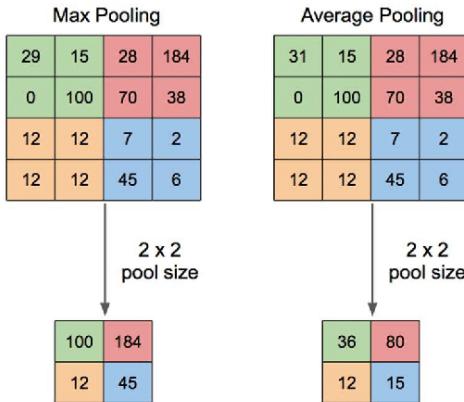


Figura 2.10: Ejemplos de aplicación de una capa de agrupación [48] sobre un volumen formado por un único mapa de características de 8×8 . Ambos utilizan un “stride” de 2 (se aplica cada dos píxeles). Ambos tienen un tamaño de 2×2 . A la izquierda, se aplica “max pooling”, cada entorno de píxeles (mismo color) se reduce a uno sólo cuyo valor corresponde al máximo de ese entorno. A la derecha “average pooling”, cada entorno se reduce a la media de valores de ese entorno.

Toman como entrada los últimos mapas de características, previamente reducidos a un vector en el que cada componente es un mapa de 1×1 . Su salida se adapta al problema concreto. El principal hiperparámetro que modula esta cabecera es el número de neuronas por capa y el número de capas ocultas.

En este TFG, las cabeceras utilizadas tienen siempre la misma arquitectura. La capa de entrada está formada por tantas neuronas como características genere la primera etapa (extractor de características) de la CNN. La capa de salida está formada por una única neurona. No hay capas ocultas. El valor que devuelve la neurona de la capa de salida puede interpretarse como la probabilidad sin normalizar de que la imagen procesada pertenezca a la clase positiva. A esta probabilidad se le aplica una sigmoide para normalizarla entre 0 y 1 (consultar Figura 2.11).

2.3.4. Capas de Activación

Al igual que en las redes neuronales convencionales, cada combinación lineal ejecutada por una capa de convolución o por una capa completamente conectada va acompañada de una operación no-lineal (función de activación). Recuérdese que estas transformaciones evitan que la potencia de la red sea tan limitada como una sola neurona y permiten a la red aproximar funciones no-lineales. En las CNN, estas operaciones son aplicadas por

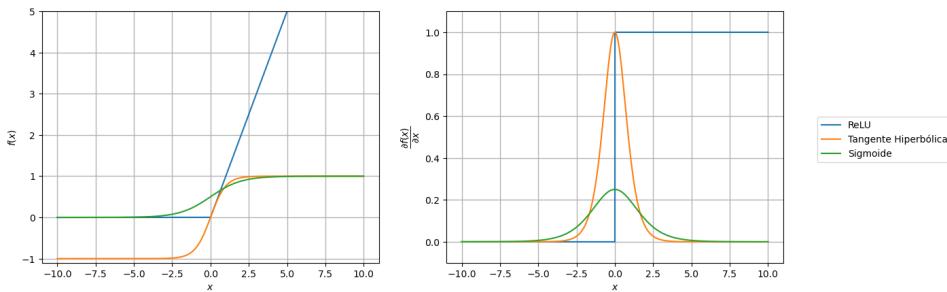


Figura 2.11: Funciones de activación mencionadas y sus derivadas. $\text{ReLU} = \max(0, x)$. $\text{Sigmoidal} = \frac{1}{1+e^{-x}}$. Tangente hiperbólica (sigmoide transformada) $= \tanh(x)$. Nótese como el gradiente de sigmoidal y la tangente saturan cuando $x > 1$ y $x < -1$. La ReLU, por el contrario, se mantiene constante y no nula cuando $x > 1$.

una capa independiente situada a continuación de las convolucionales y las completamente conectadas. Habitualmente se utilizaban operaciones como la sigmoidal o la tangente hiperbólica; sin embargo, desde hace años se ha estandarizado su uso de ReLU (“Rectified Linear Units”).

Propuestas en este contexto por *AlexNet* [49], las ReLU permiten que el entrenamiento de un modelo converja más rápido. La sigmoidal y la tangente hiperbólica saturan cuando su valor crece por encima de 1 y por debajo de -1; es decir, el valor de su gradiente para estos valores se vuelve prácticamente nulo, se desvanece (“vanishing gradients”). Un gradiente muy bajo ralentiza la optimización de los parámetros de la red y con ello el aprendizaje, llegando incluso a imposibilitarlo. Es por esto que para entrenar redes profundas se utilizan las ReLU. Siguen saturando al decrecer por debajo de 0, pero generalmente queda un porcentaje mínimo de neuronas que sigue entrenando. Consultar la Figura 2.11. Otras ventajas de las ReLU, respecto al resto de funciones de activación, es su simplicidad computacional o su capacidad para devolver el valor cero. La capacidad de devolver valores nulos deriva en representaciones con un importante número de componentes nulos, representaciones menos complejas (“sparse representations”). Recuérdese que una representación más sencilla es más deseable, ya que conservan únicamente las características más importantes, lo que deriva en modelos que tienden a un menor sobreajuste. En este TFG, las capas convolucionales utilizan ReLU como función de activación.

En el caso de la cabecera completamente conectada, como ya se ha comentado, se aplica una sigmoidal para normalizar la salida entre 0 y 1 (probabilidad de que la imagen procesada pertenezca a la clase positiva).

2.3.5. Capas de Normalización

Los datos que procesa una capa (ya sea la primera capa o una capa oculta) suelen normalizarse. Suelen trasformarse linealmente para que los valores de todas las componentes/características se encuentren en un mismo rango. Esto, además de hacer que una característica no prime sobre otra por su diferente escala, acelera la convergencia del aprendizaje, lo hace más robusto al valor de hiperparámetros como la tasa de aprendizaje o los valores iniciales de los pesos modelo y, generalmente, permite obtener mejores resultados. Lo habitual es utilizar un tipo de normalización concreta conocida como “batch normalization” [50]. Antes de introducir los datos en el modelo y después de que una capa convolucional procese un “mini-batch” de volúmenes, el objeto cuatridimensional resultante (ancho, alto, profundidad y “mini-batch”) se normaliza por canales. Es decir, se toma el i -ésimo mapa de características de cada volumen del “mini-batch”, y se normaliza cada píxel de dichos los mapas en función del valor de ese píxel en todos los mapas (media y desviación típica). Para las capas ocultas, esta normalización también incluye dos parámetros “entrenables” que permiten al modelo transformar linealmente el resultado de la normalización para cada canal.

El motivo por el que “batch normalization” es tan efectiva todavía es una discusión abierta. La justificación habitual, propuesta en el artículo original de 2015, afirma que esto se debe a que la normalización reduce el desplazamiento de la covarianza interna de los datos procesados (ICS), una forma de referirse a que durante del entrenamiento una capa se enfrenta a datos distribuidos de una forma distinta a cada iteración. No obstante, más tarde se demostró que esta premisa no es cierta, alterando de forma aleatoria e intencionada el ICS y observando que no empeoraban los resultados [51], donde también proponen que el verdadero impacto de esta técnica aparece sobre el perfil de la función pérdida. La normalización lo “suaviza” significativamente, lo que resulta en un comportamiento más estable de su gradiente y, en última instancia, las bondades inicialmente descritas (convergencia más rápida y robusta del proceso de optimización, así como mejores resultados). También sigue en duda cuando debe aplicarse esta normalización, si antes o después de la función de activación. En este TFG se escoge la primera opción (antes de la capa de activación), por ser la más habitual.

2.4. *Transfer Learning* y *Fine-Tuning*

Como su propio nombre indica, el *Transfer Learning* (TL) es un conjunto de técnicas que consisten en utilizar el conocimiento que un modelo adquiere en una tarea para resolver otra. Estos métodos son especialmente útiles cuando no se cuenta con datos de entrenamiento suficientes como pa-

ra entrenar la red “desde cero” (a partir de parámetros aleatorios) sin que esta presente un sobreajuste exagerado. En el contexto de la clasificación de imágenes, la forma más habitual de TL consiste en tomar una red entrenada sobre el conjunto de imágenes genérico y masivo como *ImageNet* (con un millón doscientas mil imágenes y mil clases) y reemplazar su cabeza completamente conectada (preparada para mil clases) por una nueva adaptada al problema concreto. A priori, sólo pueden entrenarse los pesos de esta nueva cabecera; es decir, se utiliza la red pre-entrenada como extractor de características fijo (parámetros no actualizables, congelados) mientras que los pesos de la cabecera se aprenden/optimizan para solucionar la nueva tarea de clasificación en base a dichas características.

El TL se fundamenta en que las características extraídas para un problema de clasificación pueden utilizarse para otro. Esta afirmación parece razonable para problemas muy parecidos, pero ... ¿Qué tienen que ver las características que permiten diferenciar distintas razas de perro en *ImageNet* con las características que permiten diferenciar a un adulto de un niño? Efectivamente, las características más complejas de las imágenes que permiten diferenciar unas clases de otras, poco se parecen de un problema a otro. Sin embargo, las características que extraen las primeras capas (las más profundas), que a su vez son las más difíciles de entrenar, sí que probablemente sean comunes. Recuérdese que cuánto más profunda es una capa, más simples son las características que detecta, como bordes o esquinas, patrones simples que componen cualquier característica compleja, ya sea para diferenciar razas de perro u ortopantomografías. De esta forma, el TL libera al ingeniero de la parte más costosa del entrenamiento: entrenar las capas más profundas de la red. Recuérdese que cuánto más profunda es una capa, menor es el gradiente de sus parámetros y más lenta y costosa es su optimización.

El *Fine-Tuning* (FT) consiste en intentar sacar provecho del poco sentido que tiene el TL para las capas menos profundas. El FT consiste en re-entrenar (descongelar) las capas más superficiales del extractor de características pre-entrenado para ajustarlas al nuevo problema. El extractor de características pre-entrenado ya no es fijo, se optimiza. Naturalmente, cuánto mayor es la profundidad que alcanza el descongelamiento, mayor es la probabilidad de que el modelo sobreajuste, ya que el modelo tiene mayor potencia expresiva (número de parámetros “entrenables”). Merece la pena mencionar que no tiene sentido descongelar una capa sin hacer lo propio con todas las capas que existan por encima de ella. Las características extraídas por una capa dependen de lo extraído por la anterior, luego el ajuste de una capa no tendría sentido si no se ajustan también las siguientes a ella.

Capítulo 3

Estado del Arte

3.1. Clasificación de Imágenes Utilizando Aprendizaje Automático

El ML, en concreto el DL, representan el estado del arte en la clasificación automatizada de imágenes [52]. Desde su irrupción en 2012, las CNN siempre han formado parte de todas las soluciones ganadoras en el “benchmark” de referencia, *ImageNet* [32], superando el rendimiento humano en 2015 [33]. Existe un amplio abanico de arquitecturas concretas de CNN que pueden utilizarse como punto de partida para resolver un problema. No es común proponer una nueva arquitectura construida desde cero. De hecho, recuérdese que la práctica habitual no sólo consiste en reutilizar la arquitectura (o parte de ella) sino también el valor de los parámetros tras pre-entrenarlas en otra tarea (“Transfer Learning” y “Fine Tuning”). De todas las CNN disponibles, merece la pena destacar las redes *ResNet* [53]. Desde su aparición en 2015, se ha convertido en la arquitectura por defecto a la hora de solucionar un problema de clasificación sin tener que inventar una nueva arquitectura. Los detalles a cerca de estructura y su popularización se exploran en la Sección 4.2.5, pero su rasgo principal es el uso de bloques residuales (de ahí su nombre).

3.2. Estimación de la Edad Legal en Antropología Forense

El problema que intentan resolver los modelos implementados en este TFG es estimar la edad legal de individuo (¿edad > 18?). A día de hoy, la FA es la forma más fiable de dar respuesta a esta pregunta en ciertos contextos, siendo el más relevante el caso de migrantes en busca de asilo que

dicen ser menores, pero para los que no existe documentación o registros oficiales que respalden esta afirmación [10]. Por su mayor importancia, esta subsección se centra en este caso concreto de aplicación.

A nivel nacional e internacional, cuando un migrante sin documentación solicita asilo como menor, se recomienda que un antropólogo forense evalúe varias de las siguientes regiones anatómicas: carpo (muñeca), dientes y clavícula [4, 5]. En la Figura 3.1 puede observarse que el protocolo de referencia en España está construido en torno a la evaluación de estas tres regiones. Existen otros métodos para estimar la edad de un individuo vivo que, aplicando un simple umbral, podrían utilizarse en este contexto. Métodos que analizan otras partes del esqueleto, como el extremo de la cuarta costilla [54, 55], las suturas craneales (articulaciones fibrosas que conectan los diferentes huesos del cráneo) [56, 57], la síntesis del pubis (pelvis) [58, 59] o la superficie del ileón (pelvis) [60, 61]. No obstante, no son la práctica habitual en este contexto y/o son menos fiables para el potencial rango de edad de los sujetos evaluados, desde niños hasta jóvenes-adultos.

En este caso sólo se dispone de la OPG de un individuo, información relativa al desarrollo de los dientes, por lo que sólo se desarrolla el estado del arte para esta región anatómica. Los métodos relacionados con la muñeca y la clavícula quedan en segundo plano. Bajo este pretexto, las opciones más utilizadas son el método de Demirjian [18] o variantes de este, como el estándar de Mincer [62]. No obstante, una alternativa más moderna, el test de Cameriere [63], ha demostrado igual o mejores resultados para numerosos grupos poblacionales [64, 65, 66, 67].

El método de Demirjian [18] permite estimar la edad cronológica de un individuo (no estrictamente la edad legal) a partir de una OPG. Primero, se puntuía el grado de desarrollo de cada uno de los siete dientes permanentes de la mitad inferior izquierda de la dentadura. La puntuación de cada diente se obtiene comparando visualmente el estado de su corona y de sus raíces en base a un atlas de estados. Consultar la Figura 3.2. A continuación, se suman de estos valores parciales para obtener la puntuación madurativa, un valor que refleja el grado de desarrollo dental del individuo. Finalmente, esta puntuación global se transforma en edad cronológica a través de gráficos o tablas de correspondencia específicos. Estas tablas asignan a cada puntuación madurativa, una edad cronológica (incluyendo una estimación margen de error). Para construir una tabla, se toma una muestra de individuos de edad conocida de una población específica (por ejemplo, hombres-sanos-caucásicos-españoles de entre 0 y 18 años), se determina la puntuación madurativa de cada individuo con el atlas y se calcula, para cada puntuación madurativa, la distribución de la edad cronológica de los individuos con esa puntuación (media, desviación típica y percentiles). La media y la desviación típica de esta distribución permiten obtener, para cada puntuación, un

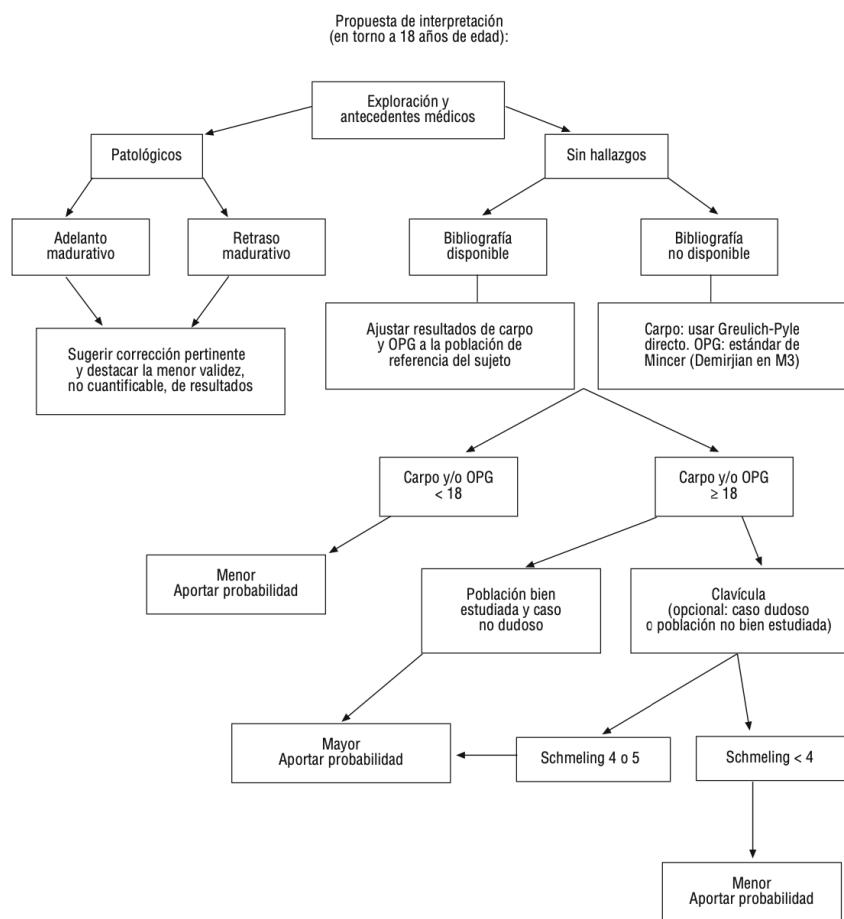


Figura 3.1: Protocolo español para la estimación de edad de potenciales menores. En Greulich-Pyle se evalúa la muñeca (carpo) [17]. En Schmeling se evalúa la clavícula [19]. En Demijram/Mincer se evalúa el tercer molar (dientes) a partir de una OPG [62]. Figura extraída de [4].

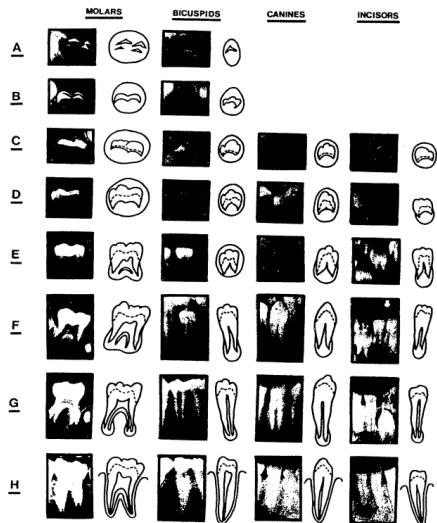


Figura 3.2: Atlas de estados de Demirjian para puntuar el desarrollo de un diente. Siete estados madurativos, notados alfabéticamente para no asumir que duran lo mismo (de arriba a abajo). Cuatro tipos de dientes (de izquierda a derecha): molares, premolares, caninos e incisivos. No se incluyen las descripciones de apoyo que acompañan a cada estado y facilitan la reproducibilidad de las puntuaciones. Figura extraída de [18].

intervalo de edad real aproximado con una certeza de un determinado porcentaje. El papel de estas tablas es fundamental, no sólo para Demirjian sino para cualquier otro método forense. Ciertos factores como el sexo, la raza, patologías o el origen socioeconómico pueden afectar a la correspondencia entre edad cronológica o real y edad biológica. Cómo y cuánto afectan sigue siendo un debate lejos de estar cerrado [68]. Por ello, para ser garantista, es necesario utilizar tablas específicas a la población a la que el individuo estudiado pertenece. En caso de que no exista un estudio lo suficientemente concreto sobre la población a la que el individuo pertenece, deben realizarse otras pruebas forenses que permitan a la autoridad competente tomar una decisión más informada. El problema de Demirjian aparece cuando la edad supera los 14 años. Todos los dientes suelen finalizar su desarrollo en torno a los 12-14 años, a excepción de los terceros molares (muelas del juicio), que suelen continuar hasta más allá de la mayoría de edad [9]. Sólo quedan por desarrollarse los terceros molares, luego estos son el único potencial indicador de la edad legal restante.

El estándar de Mincer [62] es una de las variantes de Demirjian que sólo evalúan el desarrollo de los terceros molares para estimar la edad legal de un individuo. En estas variantes, si el desarrollo de los dientes evaluados se encuentra igual o por encima de un determinado estado, el individuo se

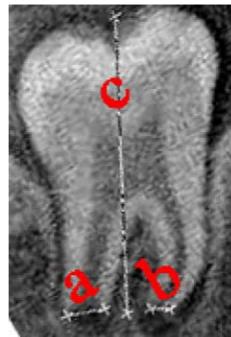


Figura 3.3: Cálculo del I_{3M} , ratio entre el ancho de la apertura de las raíces (a y b) y el largo (c) del tercer molar inferior izquierdo. $I_{3M} = \frac{c}{a+b}$. Si las raíces están cerradas ($a + b = 0$), entonces $I_{3M} = 0$. Figura extraída de [69].

considera mayor de edad con una determinada probabilidad. Dicha probabilidad y el estado utilizado como umbral dependen del grupo poblacional concreto al que pertenece el individuo, no obstante el estado que mejores resultados suele proporcionar es el H. El método/estándar de Mincer se refiere al primer artículo que aplicó este enfoque. En él, se analizaron las OPG de 823 hombres y mujeres estadounidenses de entre 14.1 y 24.9 años, mayoritariamente caucásicos. La conclusión extraída fue que los terceros molares no pueden utilizarse para estimar la edad concreta del individuo con un margen de error aceptable (amplitud media de la estimación, 8 años), pero sí para discernir su edad legal. Más concretamente, un 90 % de los hombres y un 92 % de las mujeres cuyas muelas del juicio inferiores habían sido calificadas como estado H, eran mayores de edad. No obstante, un considerable número de individuos adultos quedaron incorrectamente clasificados como menores (falsos menores), en torno a la mitad de los casos clasificados como E, F o G. Respecto al resto de estados (A-B-C-D), la probabilidad de falso-menor fue mínima. Desde su aparición, varios estudios han comprobado la aplicabilidad y/o adaptado este estándar a distintos grupos poblacionales como españoles [64], turcos [65], austriacos [66], japoneses, alemanes o sudafricanos [67]. Todas las publicaciones anteriores concluyen que el estado H es el mejor umbral, pero insisten en el elevado número de falsos-menores. Algunos discuten la posibilidad de utilizar otro estado como umbral para reducir ese número de falsos-menores, pero esto dispara el número de falsos-mayores, menores calificados como mayores de edad. Un error mucho más grave desde un punto de vista moral y legal. La ley es más garantista con los derechos de un potencial menor que con los de un adulto, es preferible un exceso de falsos-menores a un exceso de falsos-mayores.

El test de Cameriere [63] nace para solucionar el exceso de falsos-menores del estándar de Mincer, sin aumentar el número falsos-adultos. Consiste en

calcular el ratio entre ancho de la apertura de las raíces y el largo del tercer molar inferior izquierdo (I_{3M}) y utilizarlo como umbral para determinar la mayoría de edad de un individuo. Si $I_{3M} < 0.08$, entonces el individuo es mayor de edad (test positivo) y viceversa. Consultar la Figura 3.3. En el estudio comparativo [69], para la muestra poblacional utilizada, hombres y mujeres caucásicos entre 14 y 23 años, este criterio devuelve una sensibilidad del 70 % (mayores de edad predichos como tal), una especificidad del 95 % (menores de edad predichos como tal) y una exactitud (tasa de acierto) del 83 %, independientemente del sexo. Unos resultados significativamente mejores que el estándar de Mincer (mayor de edad si estado madurativo igual a H), sobre todo en términos de sensibilidad (debido a la menor cantidad de falsos-menores o falsos-negativos): 58 % de sensibilidad (12 puntos peor), 98 % de especificidad (3 puntos mejor) y 83 % de exactitud (similar). La validez (y en algunos casos, la mejora) del método no sólo ha sido demostrada para otras muestras poblacionales [70, 71, 72, 73, 74]; sino que, de acuerdo con [75], es extensible a individuos de cualquier etnia, incluida la sudafricana. No obstante, de acuerdo con [76], el rendimiento de la clasificación sí que varía de forma significativa en función del origen poblacional, un incentivo más a favor de utilizar un método sólo cuando haya sido correctamente validado para la población concreta estudiada. Se trata de una discusión aún abierta.

En [77] se aplica y valida el test de Cameriere para estimar la edad legal de un conjunto de individuos sudafricanos negros. Comparado con este TFG, se utilizan los mismos datos (OPG), el mismo grupo poblacional (sudafricanos negros, hombres y mujeres), se intenta dar respuesta a la misma pregunta ($\text{edad} \geq 18?$). Todo ello, añadido a que éste test se considera una alternativa fiable para un amplio abanico de poblaciones, incluida la sudafricana, convierten a los resultados de este estudio en un marco de referencia apropiado para los modelos implementados en este trabajo. Consultar la Figura 3.1 para los resultados concretos del estudio. De antemano, los modelos implementados cuentan con una ventaja: no presenta ningún tipo de error intra o inter-observador que comprometa la reproducibilidad de los resultados (aunque en la publicación referida, también sean mínimos).

		Edad Real	
		< 18	≥ 18
Test	< 18	489	62
	≥ 18	21	261

Cuadro 3.1: Matriz de confusión de referencia para la estimación de la edad legal de sudafricanos negros a partir de OPG. No se incluyen el error intra-inter observador. Exactitud (ACC) del 0.9004. Sensibilidad (SEN) del 0.8080. Especificidad (SPE) del 0.9588. Resultados extraídos de [77].

Antes de cerrar esta subsección, merece la pena contextualizar la práctica forense de forma más general. De acuerdo con el organismo internacional de referencia (EASO o EUAA), cualquiera de los métodos descritos anteriormente no debería ser suficiente para determinar la edad legal de una persona, sus resultados deberían ser acompañados por la evaluación de otras regiones anatómicas o de las mismas regiones con otras técnicas. A su vez, el informe forense resultante también debería formar parte de un proceso multidisciplinar en el que se tengan en cuenta otros factores, como el desarrollo social y psicológico del individuo (enfoque holístico de la estimación). En la Figura 3.4 puede consultarse un esquema del protocolo recomendado por la EASO o EUAA para estimar la edad de un potencial menor migrante.

3.3. Estimación de la Edad Legal en Antropología Forense Utilizando Aprendizaje Automático

De acuerdo con una de las pocas recopilaciones del estado del arte en materia de inteligencia artificial y antropología forense [11], la mayoría de métodos para estimar la edad y, por extensión, la edad legal de un individuo, se centran en el análisis de radiografías de la mano de un individuo. Además, desde 2017, las CNN son la herramienta “inteligente” que más se utiliza por sus mejores resultados [78, 79, 80, 81]. No obstante, existen otras soluciones, también basadas en CNN, que evalúan distintas regiones como la rodilla [82], la mandíbula y el fémur (conjuntamente) [83], la pelvis [84], el pecho [85] o los dientes [86, 87, 88, 89, 90].

La mayoría de publicaciones que proponen modelos basados en DL y trabajan con imágenes de los dientes tienen como objetivo estimar la edad de un individuo, no estimar su edad legal explícitamente; es decir, proponen regresores, no clasificadores. Un ejemplo de ello, es [86], donde se utilizan 4035 OPG de individuos croatas de entre 19 y 90 años de edad (edad media 38.17 años) para entrenar regresores basados en distintas arquitecturas de CNN. El objetivo, predecir la edad de un individuo a partir de la OPG completa o de un único diente. El mejor modelo alcanza un error medio absoluto de estimación de 3.96 años en el primer caso (OPG completa) y de 6.30 años en el segundo (imagen del mejor diente por separado). Otro ejemplo sería [88], donde se utilizan 2.289 OPG de individuos españoles entre 4 y 89 años. El objetivo entrenar al mismo tiempo tres redes: una para estimar la edad, otra para estimar el sexo y otra que utiliza las características extraídas por ambas para estimar la edad. El resultado es que la tercera red (la que tiene en cuenta el sexo) presenta resultados significativamente mejores que la primera (la que no lo tiene en cuenta), logrando un error medio absoluto de 2 años y 10 meses. Una mejora que refleja que el sexo puede afectar a la estimación de la edad.

3.3. Estimación de la Edad Legal en Antropología Forense Utilizando Aprendizaje Automático

38

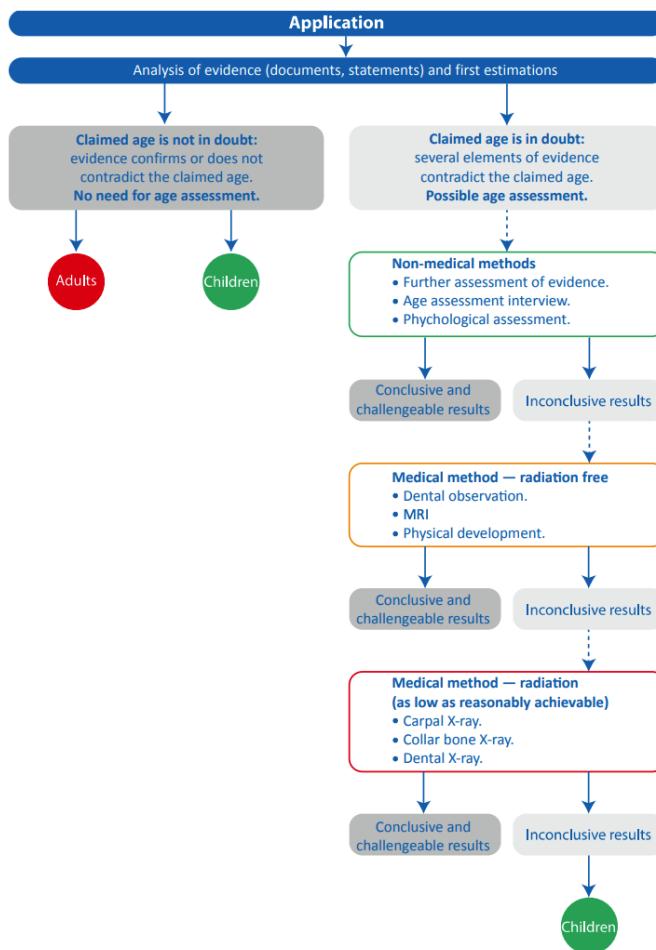


Figura 3.4: Resumen esquemático de la guía de la EASO (ahora EUAA) para la estimación de edad de migrantes no documentados. Se omiten los mecanismos específicos que salvaguardan los derechos del potencial menor. A pesar de ser el último eslabón de la cadena, la mayoría de países a los que va dirigida esta guía fundamenta sus decisiones únicamente en los informes forenses [3].

Dos de los pocos artículos encontrados en los que se utilizan imágenes dentales para entrenar clasificadores en vez de regresores son [89] y [90]. En el segundo, se utilizan 4003 OPG, 2149 mujeres y 1809 hombres, todos habitantes de la zona centro-oeste de Brasil, sin patologías o grandes deformaciones en la dentadura. Edades entre los 8 y los 22.9 años, distribuidas entre ambos límites de forma homogénea. Utilizando una CNN *DenseNet121* [91] y aumentando los datos con ligeras traslaciones de hasta un 10 % del tamaño original de la OPG, se alcanza una $ACC \approx 0.88$, con una $SEN \approx 0.87$

y una $SPE \approx 0.88$. En el primero, se utilizan 10257 OPG, 4.579 hombres y 5.678 mujeres, todos de origen chino, sanos, sin patologías o dientes ausentes. Edades entre los 5 y los 25 años. 64.89 % menores de edad. Se entrenan varios clasificadores binarios, basados en distintas arquitecturas de CNN (*EfficientNet* [92] y *ResNet101* [53]), que permiten diferenciar si un individuo es mayor o menor de 14, 16 y 18 años, umbrales de edad relevantes para el código penal chino. Los modelos no utilizan las OPG completas; o hacen uso de la región relativa a los 8 dientes inferiores del lateral izquierdo, o únicamente emplean la región relativa al tercer molar izquierdo. El rendimiento de los clasificadores se compara con el de dos antropólogos, que aplican el método de Demirjian [18] (como ya se comentó, práctica habitual en este contexto). La conclusión es que, para todos los umbrales, las CNN son una alternativa igual o mejor que los expertos humanos. Para 14 y 16 años, se obtienen mejores resultados cuando se utilizan los 8 dientes mandibulares izquierdos. Para los 18 años, el análisis de únicamente el tercer molar ofrece mejores resultados. El mejor modelo para los 18 años, *EfficientNet-B3*, alcanza $ACC \approx 0.92$, $SEN \approx 0.93$ y $SPE \approx 0.92$. Para arrojar algo de luz sobre por qué los clasificadores ofrecen mejores resultados, se utiliza *Grad-CAM*, una técnica de explicabilidad que permite visualizar las regiones de la imagen original que más influyen en la predicción del clasificador [93]. Esta técnica se utiliza también en este proyecto y puede encontrarse desarrollada con mayor detalle en la Subsección 4.2.5. En este caso, *GradCAM* confirma que las redes “se fijan” en regiones que también utilizan los expertos para estimar la edad dental, como el tamaño de la cavidad interna del tercer molar (utilizado por el método de Cameriere [63]), pero combina esta información con regiones que típicamente no se utilizan, como la distancia entre dientes adyacentes o la distancia entre dientes de leche y permanentes, razón por la que podrían presentar mejor rendimiento que los expertos. El artículo insiste en la problemática habitual: estos resultados no tienen por qué ser extensibles a otros grupos poblacionales. Salvando las distancias, parte de este TFG puede interpretarse como una reproducción de este estudio sobre una muestra de individuos de origen sudafricano, grupo poblacional para el que no se encontraron publicaciones de esta índole (ML/DL + estimación de la edad legal + OPG).

Capítulo 4

Materiales y Métodos

4.1. Conjunto de Datos

En este TFG se trabaja con 1.741 imágenes. Cada imagen corresponde a la OPG de un individuo sudafricano de raza negra. Ningún individuo presenta más de una OPG. La resolución de las imágenes no es siempre la misma, ni todas las radiografías están escaneadas de la misma forma. De cada individuo se conoce la edad en la que se realizó la radiografía, que oscila entre los 5.15 y los 27.74 años, con una media de 14.09 años. En la Figura 4.1 pueden encontrarse algunas OPG de ejemplo.

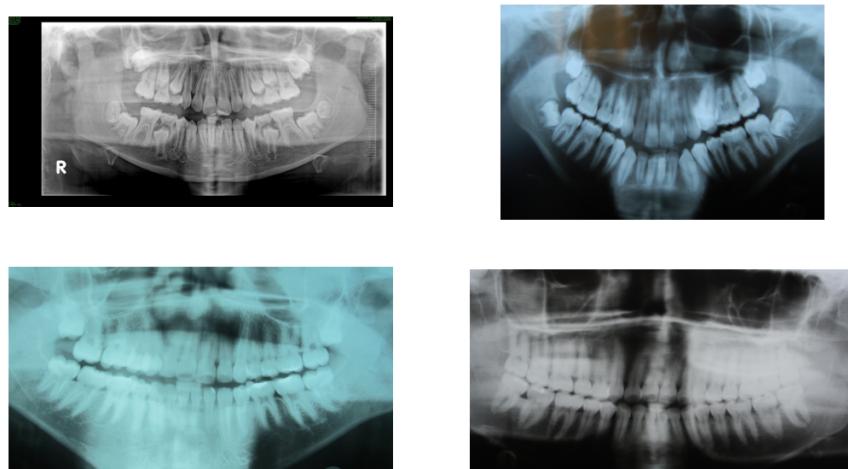


Figura 4.1: Varias OPG de ejemplo. Todas corresponden al conjunto de datos de este proyecto. Arriba a la izquierda, el individuo más joven con 5.77 años. Arriba a la derecha, un individuo de 14.32 años. Abajo a la izquierda un individuo de 18.08 años. Abajo a la derecha, el individuo de mayor edad, con 27.74 años.

El problema a resolver no consiste en estimar la edad (regresión), sino la edad legal (clasificación binaria), por lo que se crea un nuevo conjunto de etiquetas binario para el conjunto de datos. Si una imagen pertenece a un individuo mayor de edad ($\text{edad} \geq 18$), entonces la etiqueta a predecir es 1 (casos positivos). Si una imagen pertenece a un individuo menor de edad ($\text{edad} < 18$), entonces la etiqueta es 0 (casos negativos). En la Figura 4.2 puede consultarse la distribución de este nuevo conjunto de etiquetas categóricas. El número de casos positivos es mucho menor que el número de casos negativos, el conjunto está desbalanceado (82.72 – 17.28 %). Matizar que enfocar el problema como una clasificación no es estrictamente necesario, podría haberse implementado un regresor para computar la edad de un individuo y transformar este resultado a edad legal en función de si supera o no el umbral de los 18 años. Esta es una decisión arbitraria, en trabajos futuros, podría explorarse esta posibilidad.

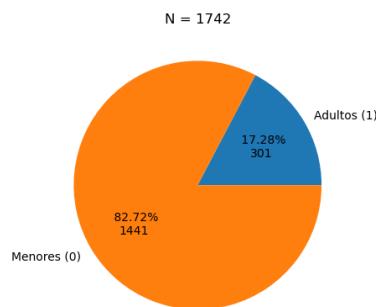


Figura 4.2: Balanceo del conjunto de datos.

4.1.1. Partición de Entrenamiento–Validación y Test

El objetivo último de los modelos implementados es generalizar, es decir, clasificar correctamente imágenes con las que no ha entrenado. Además, el funcionamiento y, por extensión, la capacidad de generalización de un modelo depende de un conjunto de parámetros (pesos de los filtros convolucionales y de las capas completamente conectadas) e hiperparámetros (como el tamaño de “mini-batch”, el tipo de normalización o la tasa de aprendizaje). Teniendo todo lo anterior en cuenta, el conjunto de imágenes se dividió aleatoriamente en dos grupos disjuntos: entrenamiento-validación y test. A continuación, se desarrolla por qué.

El conjunto entrenamiento-validación permite obtener el valor de los parámetros e hiperparámetros de un modelo. Por un lado, el modelo utiliza el conjunto de entrenamiento para aprender automáticamente los pesos del modelo. Por otro lado, el ingeniero utiliza el conjunto de validación para

escoger los hiperparámetros del modelo. El ingeniero, durante y después del entrenamiento, evalúa dicho modelo sobre el conjunto de validación y compara los resultados con los de otros modelos que utilizan el mismo conjunto de entrenamiento pero distinta configuración de hiperparámetros. El objetivo del especialista es encontrar la combinación de hiperparámetros que mejor modelo genera. Recuérdese que las imágenes del conjunto de validación no forman parte del conjunto de entrenamiento, por lo que el resultado sobre validación es una estimación de la capacidad de generalización de los modelos, del mejor modelo. Para no introducir ningún sesgo, la parte de los datos que se destina a validación suele escogerse aleatoriamente. Esta forma de validar, conocida como “hold-out”, no es la que se utiliza en este proyecto, pero simplifica la explicación de este apartado. El protocolo concreto utilizado puede consultarse en la subsección 4.2.1, pero podría definirse como aplicar “hold-out” varias veces.

El conjunto de test permite estimar de forma realista el rendimiento del modelo. Efectivamente, los resultados sobre el conjunto de validación aproximan el rendimiento real del modelo, su capacidad de generalización. No obstante, esta aproximación es demasiado optimista, ya que los valores de los hiperparámetros del modelo se escogen específicamente para maximizar los resultados de validación. El modelo está “sobre-ajustado” al conjunto o protocolo de validación. Para resolver este exceso de optimismo, se reserva una parte del conjunto original de imágenes, el conjunto de test. Un conjunto de imágenes que no debe utilizarse ni para entrenar ni para validar. **El modelo que mejores resultados presente en el protocolo de validación, se entrena sobre el conjunto completo de entrenamiento-validación y se evalúa sobre el conjunto test.** Esta estimación es lo suficientemente realista como para comparar el modelo implementado con un experto humano. Añadir que el conjunto de test suele estar balanceado para evaluar los resultados sobre ambas categorías sin que ninguna métrica de rendimiento presente un valor “engañoso” (consultar Subsección 4.2.4 para más información al respecto).

Así pues, en este TFG, se reservaron para labores de testeo un conjunto aleatorio balanceado de imágenes que constituye el 20 % del conjunto de datos original. Esto resulta en un 80 % de imágenes aún más desbalanceadas (de 82.72 – 17.28 % a 90.89 – 9.11 %) para entrenamiento y validación. En la Figura 4.3 puede observarse el balanceo de esta partición.

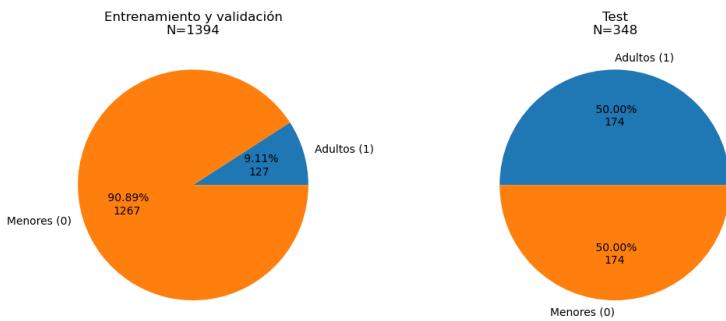


Figura 4.3: Partición del conjunto de datos. A la izquierda, el balanceo del conjunto reservado para entrenamiento y validación. A la derecha, el balanceo del conjunto reservado para test.

4.2. Métodos

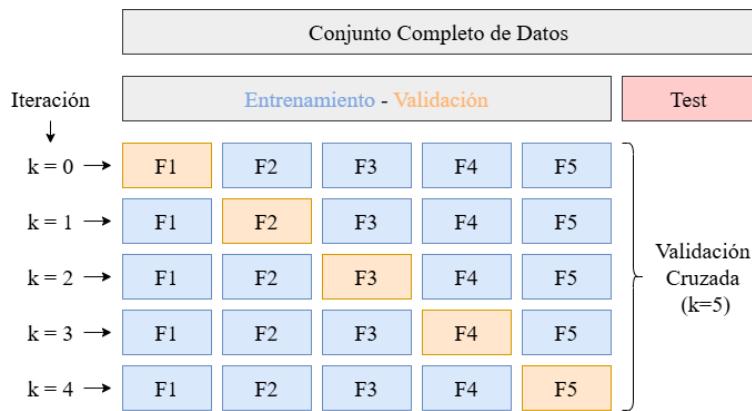
4.2.1. Validación Cruzada

Como ya se ha comentado, “hold-out” no es la técnica de validación que se utiliza en este proyecto. Reservar aleatoriamente una fracción del conjunto entrenamiento-validación puede provocar que el conjunto de validación escogido sea especialmente favorable o perjudicial para una combinación concreta de parámetros e/o hiperparámetros. Para solucionar esta falta de fiabilidad, se recurre a la validación cruzada. Este protocolo de validación consiste en dividir el conjunto entrenamiento-validación en k particiones y repetir el entrenamiento y validación de un modelo k veces. En cada una de las iteraciones, se valida el modelo con una partición (distinta cada vez) y se entrena con las $k - 1$ particiones restantes. En la Figura 4.4a puede consultarse un esquema donde se resume este proceso de partición de datos y cómo casa con la partición previa en entrenamiento-validación y test.

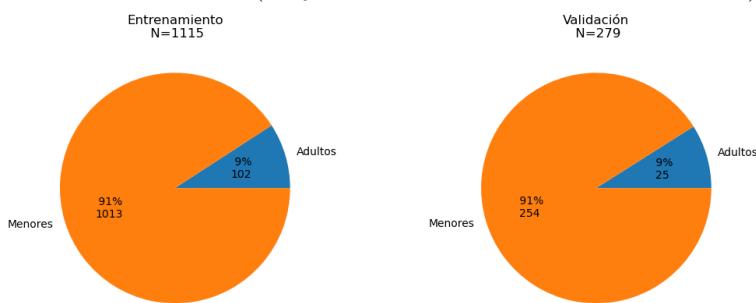
Además de las curvas de entrenamiento, cada ejecución de la validación cruzada genera un valor concreto para cada una de las métricas de rendimiento evaluadas, como pueden ser la exactitud o la precisión. El valor de cada una de estas métricas se obtiene promediando el valor que presenta dicha métrica para el mejor modelo encontrado en cada iteración de la validación cruzada. En otras palabras, se obtienen k -mejores modelos (uno de cada iteración) y se hace una media de su rendimiento. La desviación típica también suele calcularse para, en caso de medias similares, escoger el candidato con menor variabilidad. El criterio para determinar el mejor modelo es potestad del ingeniero.

En este TFG, el conjunto de datos entrenamiento-validación se dividió aleatoriamente en 5 particiones estratificadas; es decir, particiones que

mantienen el balanceo del conjunto original de los datos. En cada una de las iteraciones de la validación cruzada, se valida el modelo con una partición distinta, que supone un 20 % del conjunto entrenamiento-validación, y se entrena con el resto, 4 particiones que suponen un 80 % del conjunto entrenamiento-validación. En la Figura 4.5 puede consultarse el balanceo de los conjuntos de entrenamiento y validación para cada iteración de la validación cruzada. El criterio para determinar mejor modelo en base a los resultados para cada una de las iteraciones se concreta en la Sección 4.2.4.



(a) Resumen esquemático de la partición de datos para validación cruzada con $k = 5$. El conjunto de test no se utiliza. Los subconjuntos F son disjuntos. En cada iteración se valida con un distinto (conjunto de validación de la iteración) y se entrena con los 4 restantes (conjunto de entrenamiento de la iteración).



(b) Balanceo de datos de la validación cruzada. A la izquierda, el balanceo del conjunto de entrenamiento de la primera iteración de la validación cruzada ($k = 0$). A la derecha, el balanceo del conjunto de validación de esa misma iteración. El resto de iteraciones presenta la misma distribución.

Figura 4.4: Validación Cruzada.

4.2.2. Sobremuestreo

La mayoría de modelos de ML, especialmente las redes neuronales, suelen interiorizar cualquier sesgo que tenga el conjunto de entrenamiento, incluida la distribución o el balanceo de las etiquetas. Si una red se entrena con un conjunto tan desbalanceado como el disponible (90.89 % de menores, 9.11 % de adultos), se obtiene un clasificador que atribuye la clase mayoritaria a casi cualquier imagen. Un modelo que ni siquiera es capaz de presentar un rendimiento aceptable para la clase minoritaria en el conjunto de entrenamiento. Si no se pueden recopilar más datos, la forma más sencilla de solucionar este problema es el sobremuestreo: duplicar imágenes de la clase minoritaria hasta balancear el conjunto de entrenamiento. Existen otras técnicas para paliar el desbalance como: el sub-muestreo, que consiste eliminar imágenes de la clase mayoritaria hasta balancear el conjunto, descartada porque sacrifica más de un 80 % del ya escaso conjunto de imágenes; una función perdida que da un mayor peso a las imágenes del conjunto minoritario, descartada por equivalente a sobremuestrear a nivel de “mini-batch”; o utilizar modelos y estrategias no tan sensibles a datos desbalanceados como aquellos que se utilizan para la detección de anomalías (“one-class SVM”, “Isolation Forest” o “bagging” con subconjuntos de datos balanceados).

En este TFG, para resolver el desbalance de los datos, se aplica sobre-muestreo sobre el conjunto de entrenamiento de cada iteración de la validación cruzada. El conjunto de validación de cada iteración no se balancea. En la Figura 4.5, puede observarse el balanceo de los datos de entrenamiento y validación de una iteración de la validación, después de aplicar el sobre-muestreo. El algoritmo de sobremuestreo fue un sobre-muestreo aleatorio sin reemplazo que se repite hasta tener el mismo número de casos de ambas clases.

4.2.3. Aumento de Datos

Para paliar el sobreajuste, cuando este exista, puede optarse por el aumento de datos [94]. Esta técnica de regularización consiste en generar nuevas imágenes resultado de aplicar ciertas transformaciones a las imágenes ya existentes. Estas nuevas imágenes se añaden posteriormente al conjunto de entrenamiento, emulando la obtención de nuevos datos. Deformar las imágenes puede parecer contra-intuitivo a la hora de mejorar el rendimiento del modelo, pero lo que se intenta es que el modelo no aprenda y decida en base a información irrelevante de la imagen, no interiorice ruido estadístico, no sobreajuste. Esta técnica suele utilizarse junto al sobre-muestreo, para que las nuevas imágenes generadas de la clase minoritaria no sean duplicados idénticos. Algunos ejemplos de estas transformaciones pueden ser: rotaciones, traslaciones, recortes, re-escalados, difuminar la imagen, ensalzar los

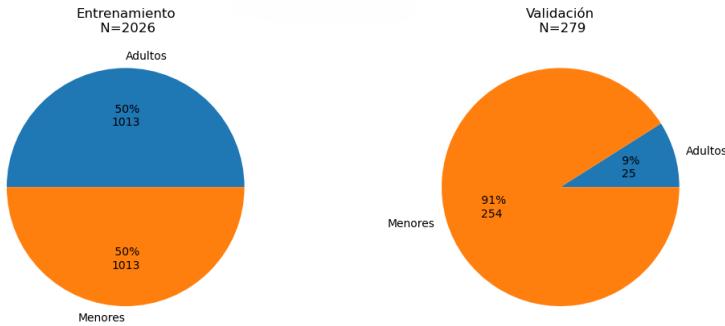


Figura 4.5: Balanceo de datos para una iteración de validación cruzada tras sobremuestreo. A la izquierda, el balanceo del conjunto de entrenamiento de la primera iteración de la validación cruzada. A la derecha, el balanceo del conjunto de validación de esa misma iteración. El resto de iteraciones presenta la misma distribución.

bordes de la imagen, ocultar parcialmente regiones de la imagen, cambios de contraste, inyecciones de ruído, mezcla de imágenes y etiquetas (*MixUp* y *CutMix*) ... Otro grupo de técnicas de aumento de datos son aquellas que utilizan redes generativas.

En este TFG, el aumento de datos se aplica antes de procesar un “minibatch” de imágenes del conjunto de entrenamiento (no del conjunto de validación). Aunque no se esté generando explícitamente un nuevo conjunto de entrenamiento aumentado, a cada época las imágenes con las que el modelo trabaja son distintas, equivalente a entrenar un nuevo conjunto de entrenamiento aumentado. A esta forma de generar un nuevo conjunto de datos se conoce como “online”. La alternativa sería crear un nuevo conjunto transformado “offline”; es decir, aplicar las transformaciones y almacenar el nuevo conjunto de datos en disco. No obstante, este enfoque no es habitual, ya que despilfarra mucha memoria. Otro motivo por el que se renunció a este enfoque, es que para acelerar el tiempo de ejecución, se cargaba el conjunto de datos preprocesado (normalizado y reescalado) y sobremuestreado (si es que se aplicaba sobremuestreo) en memoria. Realizar el aumento “offline” habría limitado severamente el tamaño del conjunto de datos aumentado al tamaño de la memoria.

MixUp

La técnica de aumento de datos *MixUp* consiste en generar una instancia de datos (imagen y etiqueta) combinando 2 instancias del conjunto de datos [95]. El píxel ij -ésimo de la nueva imagen (I_{ij}^2) es resultado de combinar linealmente los píxeles ij -ésimos de ambas imágenes (I_{ij}^0 y I_{ij}^1) en base a un

factor λ_{MU} . El valor de λ_{MU} se encuentra siempre entre 0 y 1. Este factor se extrae aleatoriamente de una “distribución beta” con $\alpha = \beta$ [96]. En *MixUp*, α y β siempre tienen el mismo valor, así que a partir de este punto, el hiperparámetro α_{MU} hace referencia al valor de ambos. En la Figura 4.6, puede comprobarse cómo afecta α_{MU} a la distribución.

$$I_{ij}^2 = \lambda_{MU} \cdot I_{ij}^0 + (1 - \lambda_{MU}) \cdot I_{ij}^1$$

Las etiquetas de las imágenes se combinan de forma análoga, pero requieren ser codificadas al estilo “one-hot” previamente. Esto significa que se sustituyen por un vector con tantas componentes como etiquetas (2 en este caso) con todas las componentes igual a 0 salvo una, que corresponde a la etiqueta extraída. En el caso de las imágenes con etiqueta negativa (=0), la primera componente es 1. En el caso de las imágenes de etiqueta positiva (=1), la segunda componente es 1. Esta transformación también requiere “adaptar” la salida de los modelos. En lugar de ser un único valor p entre 0 y 1, correspondiente a la probabilidad normalizada de que la clase positiva, la salida es una dupla en la que el primer elemento es $1 - p$ y el segundo p . La función pérdida también debe adaptarse.

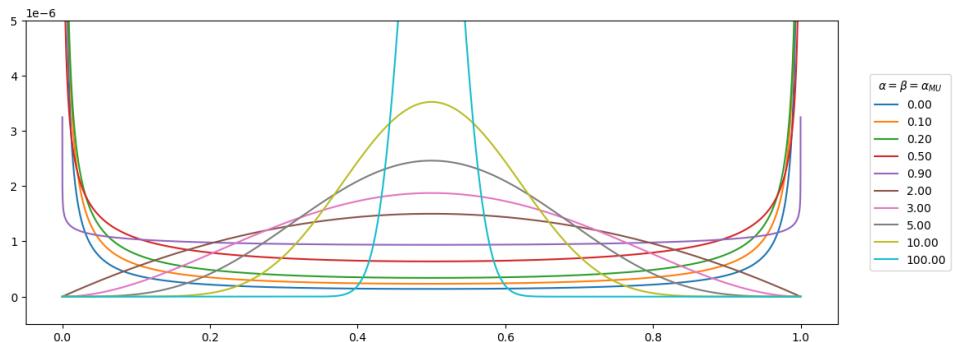


Figura 4.6: “Distribución beta” en función de α y β con $\alpha = \beta = \alpha_{MU}$. Como puede observarse, cuánto mayor sea el valor de α_{MU} , más probable y agresiva es la combinación de imágenes en *MixUp*.

De acuerdo con los autores de *MixUp*, el poder regularizador de esta técnica reside en obligar al modelo a comportarse de forma lineal entre instancias. Se aumenta el sesgo del modelo, se reduce su varianza, en pro de una mejor generalización, un balance más que habitual en el contexto del ML. Otra opción popular que también se basa en la mezcla de imágenes y etiquetas es *CutMix*. En lugar de obtener una nueva imagen resultado de mezclar todos los píxeles de las dos imágenes originales, subregiones completas de la imagen se sustituyen por subregiones del mismo tamaño de otras imágenes del conjunto de datos [97].

4.2.4. Métricas de Rendimiento de Clasificación

Para evaluar el rendimiento de un clasificador, no sólo es útil la función pérdida, sino que existe un conjunto de funciones que permiten interpretar cómo rinde el modelo para cada una de las clases (mayores de edad = 1 = positiva o menores de edad = 0 = negativa). Toda métrica en este contexto procede de una estructura de datos conocida como matriz de confusión. Los elementos de esta matriz desglosan el rendimiento de un clasificador en función de la clase real y la predicha para cada instancia del conjunto de datos evaluado. Consultar el Cuadro 4.1.

		Clase Real	
		0	1
Clase Predicha	0	TN	FN
	1	FP	TP

Cuadro 4.1: Matriz de confusión para un clasificador binario. Permite desglosar el rendimiento de un clasificador en función de la clase real y predicha de cada instancia del conjunto de datos evaluado.

- TP. Verdaderos positivos. Imágenes cuya clase real es positiva (1) y se predice positiva (1).
- FP. Falsos positivos. Imágenes cuya clase real es negativa (0) y se predice positiva (1).
- TN. Verdaderos negativos. Imágenes cuya clase real es negativa (0) y se predice positiva (0).
- FN. Falsos negativos. Imágenes cuya clase real es positiva (1) y se predice negativa (0).

En este TFG, aparte de la función pérdida y la matriz de confusión, los modelos se evalúan los modelos sobre las siguientes métricas de rendimiento: la exactitud (*ACC*), la sensibilidad (*SEN*) y la especificidad (*SPE*). Además de ser las métricas que se evalúan en Estado del Arte (facilitando así, la comparativa), describen el funcionamiento del modelo de forma más práctica que la matriz. La exactitud mide de forma relativa cuántas veces acierta la predicción del modelo para ambas clases. Se incluye porque es la más general y fácil de interpretar, pero puede ser bastante engañosa. El buen rendimiento para una de clase, podría enmascarar un pésimo funcionamiento para la otra. Además, este enmascaramiento es usual cuando se entrena a partir de conjuntos de datos desbalanceados, aunque se hayan balanceado sintéticamente (como es el caso). Por ello, se incluyen también la sensibilidad y la especificidad, métricas que desglosan el funcionamiento para cada clase.

La sensibilidad determina cuántas imágenes de las realmente positivas se predijeron como tal. La especificidad determina cuántas imágenes realmente negativas se predijeron como tal. Sensibilidad y especificidad suelen ser una balanza. Cuanto más se intenta que un modelo rinda bien para una clase, peor rinde para otra.

$$\begin{aligned} ACC &= \frac{TP + TN}{TP + FP + TN + FN} \\ SEN &= \frac{TP}{TP + FN} \\ SPE &= \frac{TN}{TN + FP} \end{aligned}$$

Lo garantista, a la hora de elegir un mejor modelo si estas métricas discrepan, sería elegir aquel con mayor especificidad. Como ya se ha comentado, ética y legalmente es más tolerable clasificar a un mayor de edad como menor (mayor número de falsos negativos, menor sensibilidad) a clasificar un menor de edad como mayor (mayor número de falsos positivos, menor especificidad). No obstante, el alto grado de desbalance de los datos empuja a los modelos a tener, de antemano, una especificidad muy alta y una sensibilidad muy baja. En pro de no beneficiar a los modelos que más interiorizan este sesgo, el criterio para elegir un mejor modelo fue una media ponderada de sensibilidad y especificidad que se repite para distintos factores de ponderación. Más información en la siguiente subsección. Merece la pena mencionar que para evaluar un modelo, no es sólo útil el valor final de estas métricas (análisis cuantitativo) si no cómo este ha evolucionado a lo largo del entrenamiento (análisis cualitativo). Las curvas de entrenamiento muestran son la forma usual de representar esta evolución de las métricas rendimiento.

Criterio de Evaluación - Aprendizaje Profundo

Durante toda la experimentación, el criterio para determinar el mejor modelo de una iteración de validación cruzada, fue escoger aquel candidato con mejor (más alta) $M(\alpha_M)$. El criterio para determinar la validación cruzada con mejores resultados fue M_{CV} (más baja).

$$\begin{aligned} M(\alpha_M) &= \alpha_M \cdot SEN + (1 - \alpha_M) \cdot SPE \\ M_{CV}(\alpha_M) &= |\overline{CV}_{SEN}(\alpha_M) - \overline{CV}_{SPE}(\alpha_M)| \end{aligned}$$

\overline{CV}_X representa el valor medio de la métrica X en validación cruzada. Se calculó como la media del valor de dicha métrica para el mejor modelo encontrado en cada iteración. Como el mejor modelo se escoge vía $M(\alpha_M)$,

\overline{CV}_X también depende de α_M . En otras palabras, en cada iteración se almacena el modelo con más alta $M(\alpha_M)$ y a continuación se hace una media de los 5 valores. Como ya se ha mencionado, ambos criterios dependen de un factor de ponderación α_M . Este factor se incluyó para evitar seleccionar candidatos que en promedio rinden bien para ambas clases, pero que en realidad favorecen significativamente más a una que a la otra (un 10% más). Cada vez que deben compararse dos modelos vía validación cruzada, se repite la evaluación para $\alpha_M = \{0.4, 0.5, 0.6\}$ y se escoge el candidato con mejor (más baja) M_{CV} .

Esta forma de evaluar los resultados se incluyó porque favorece que ambas métricas presenten un valor medio alto y que el candidato elegido no muestre preferencia por ninguna de las clases, aunque sea a costa de sacrificar algo de rendimiento para alguna de ellas. Inicialmente, se consideró utilizar simplemente la media aritmética de SEN y SPE para determinar los mejores modelos de cada iteración de la validación cruzada ($\alpha_M = 0.5$). No obstante, experimentación preliminar demostró que este criterio provocaba la elección de modelos que presentaban un rendimiento muy bajo para una de las clases sobre candidatos que presentaban un rendimiento equilibrado y medianamente alto para ambas. En concreto, se veían especialmente beneficiados conjuntos de hiperparámetros que interiorizaban el desbalance de los datos, ya que este sesgo empuja a los modelos a tener de base una SPE alta y una SEN baja. Matizar que este criterio se ignoró en escenarios en los que un análisis cualitativo de las curvas de aprendizaje es suficiente o más informativo.

Criterio de Evaluación - Aprendizaje Automático Clásico

La biblioteca de *Python* utilizada para este tipo de modelos fue *Scikit Learn*, una biblioteca mucho menos flexible que *PyTorch*. Así pues, el criterio de evaluación tuvo que adaptarse. La limitación más problemática de la biblioteca es que no se encontró la forma de almacenar el mejor modelo encontrado durante el entrenamiento de un modelo atendiendo a un criterio propio (recuérdese $M(\alpha_M) = \alpha \cdot SEN + (1 - \alpha) \cdot SPE$). Los resultados para cada iteración de la validación cruzada no corresponden a evaluar el mejor modelo encontrado, sino a evaluar el modelo que queda tras la última iteración/época del entrenamiento. Por un lado, esto provoca que el término α_M desaparezca a la hora de evaluar los candidatos, ya que sólo se utilizaba para esta selección del mejor modelo durante el entrenamiento. Por otro lado, compromete la optimalidad de $M_{CV} = |SEN - SPE|$ como criterio para escoger el mejor modelo. Al no existir una etapa previa en la que se intenta maximizar explícitamente el valor de SEN y SPE , el riesgo de que M_{CV} presente un valor engañoso es más alto; por ejemplo, cuando ambas métricas presentan un valor bajo. Para paliar este problema, tras escoger el

mejor candidato en base M_{CV} , se comprobó que este estuviese entre los 5 mejores candidatos en base a $\frac{1}{SEN + SPE}$. De no ser así, se prueba con el siguiente candidato con menor M_{CV} hasta encontrar uno que satisfaga las condiciones. Una solución inspirada en la propuesta en [98], donde para cada métrica se crea un ranking y si escoge como mejor modelo aquel con mejor posición media. Esta solución fue especialmente útil cuando el número de candidatos era elevado y este valor engañoso para M_{CV} no puede advertirse a simple vista.

Cuantificación del Sobreajuste

Un modelo sobreajusta cuando lo que aprende no es generalizable; es decir, el rendimiento es alto para el conjunto de entrenamiento, pero no se traduce fuera de este. El modelo decide en base a características de los datos del conjunto de entrenamiento que en realidad no son relevantes (ruído estadístico). Este fenómeno puede observarse en las curvas de aprendizaje. Si el modelo sobreajusta, a medida que pasan las épocas, la distancia y área entre las curvas de entrenamiento y validación aumenta (divergen). La curva de entrenamiento continúa mejorando su valor, mientras que la curva de validación se estanca o empeora. Cuanto más pequeño y menos diverso sea el conjunto de entrenamiento, cuanto más tiempo (épocas) entrene el modelo y cuanto más complejo sea (número de parámetros entrenables); mayor es el riesgo de sobreajuste, mayor es la probabilidad de que el modelo se adapte demasiado a minimizar el error sobre el conjunto de entrenamiento. El fenómeno del sobreajuste puede apreciarse a simple vista en las curvas de aprendizaje. No obstante, en ocasiones puede ser necesario cuantificarlo de forma precisa y reproducible, por ejemplo, a la hora de determinar si un modelo sobreajusta más que otro.

En este proyecto, para cuantificar el sobreajuste de un modelo, se computó la diferencia entre el valor de una métrica de rendimiento para el conjunto de entrenamiento y el valor de esa misma métrica para el conjunto de validación. Como se utiliza el protocolo de validación cruzada, en realidad se utiliza el promedio de esta diferencia para el mejor modelo encontrado en cada una de las iteraciones, el índice $O_{CV}(X)$.

$$O_{CV}(X) = \overline{CV}_X^T - \overline{CV}_X^V$$

X es la métrica evaluada. \overline{CV}_X es el valor medio en validación cruzada de la métrica (media aritmética del valor que presenta X para el mejor modelo encontrado en cada iteración). Los superíndices T y V indican sobre qué conjunto de datos se valora la métrica, entrenamiento y validación respectivamente.

4.2.5. Aprendizaje Profundo

ResNet

Como ya se ha comentado, *ResNet* [53] es la arquitectura de referencia a la hora de solucionar un problema de clasificación de imágenes cuando no se quiere inventar una nueva arquitectura. El factor diferencial de este tipo de CNN es el añadido de módulos residuales o “skip connections”.

Hasta la aparición de las arquitecturas *ResNet*, la profundidad de las CNN estaba limitada por un fenómeno conocido como “vanishing gradient”. Este fenómeno hace referencia a un parámetro de la red que no puede optimizarse porque el gradiente de la función pérdida sobre él es mínimo o nulo. Sin módulos residuales, esto ocurre para los parámetros de las capas más profundas, el gradiente se disipa conforme se retropropaga aplicando la regla de la cadena de capas más superficiales a capas más profundas (algoritmo de “backpropagation”). Los módulos residuales o “skip connections” conectan, a través de la suma, la salida de una capa más profunda con la salida de otra más superficial (en ocasiones esta debe proyectarse linealmente la salida de una de ella para que ambas capas tengan la misma dimensión y poder sumarse). Como la derivada parcial de la suma es la función identidad, esto provoca que a la hora de repropagar el gradiente de la función pérdida aplicando la regla de la cadena, la capa más profunda reciba el mismo gradiente que la capa más superficial. Enlazando “skip connectinos” se crea una especie de autopista que permiten que el gradiente alcance las capas más profundas de la red. En la Figura 4.7 puede consultarse el aspecto esquemático de una “skip connection”.

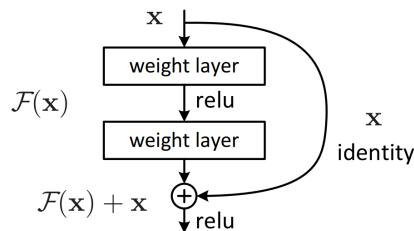


Figura 4.7: Bloque residual o “skip connection”. Se conecta la salida y la salida de dos capas no consecutivas. Figura extraída de [53].

Otros aspectos reseñables de estas arquitecturas son el uso de capas de normalización por “batches” entre capas convolucionales (“batch normalization”, explicado en la subsección 2.3.5) o el uso de módulos *Inception*, convoluciones 1x1 que se aplican antes de convoluciones más costosas (con filtros más grandes, como 3x3 o 5x5) para reducir la dimensionalidad (en profundidad) de las representaciones generadas (volúmenes) y, por exten-

sión, el número de parámetros necesarios [99]. Ambas técnicas facilitan el entrenamiento de redes más profundas. No hay capas dedicadas exclusivamente a la reducción (capas de agrupación).

En este TFG, se utilizan dos redes de este tipo: *ResNet18* y *ResNet50*. La primera presenta 18 capas, más de 11 millones de parámetros entrenables y una exactitud. La segunda presenta 50 capas y con 23 millones de parámetros entrenables. Ambas están pre-entrenadas sobre *ImageNet*. Sobre el conjunto de test de este “benchmark” (100.000 imágenes y 1000 clases posibles), presentan respectivamente una exactitud del 69.76 % y 76.13 % respecto la clase con mayor probabilidad (“top-1-accuracy”) y del 89.08 % y 92.86 % respecto a las cinco clases con mayor probabilidad (“top-5-accuracy”). Rendimiento extraído de [100]. En la Figura 4.8 puede encontrarse una comparativa de estas y otras arquitecturas de *ResNet*.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\left[\begin{array}{c} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 2$	$\left[\begin{array}{c} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 3$	$\left[\begin{array}{c} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{c} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{c} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$
conv3_x	28×28	$\left[\begin{array}{c} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 2$	$\left[\begin{array}{c} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 4$	$\left[\begin{array}{c} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{c} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{c} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 8$
conv4_x	14×14	$\left[\begin{array}{c} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 2$	$\left[\begin{array}{c} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 6$	$\left[\begin{array}{c} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 6$	$\left[\begin{array}{c} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 23$	$\left[\begin{array}{c} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 36$
conv5_x	7×7	$\left[\begin{array}{c} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 2$	$\left[\begin{array}{c} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 3$	$\left[\begin{array}{c} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	$\left[\begin{array}{c} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	$\left[\begin{array}{c} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 4.8: Arquitecturas *ResNet*. Cada fila representa un bloque secuencial de capas convolucional que se aplica tantas veces como esté indicado (también de forma secuencial) al volumen de entrada. Por ejemplo, el bloque señalado indica que el volumen de entrada es procesado de forma consecutiva por 4 capas de convolución sucesivas (2 veces un bloque de 2 capas), todas aplican filtros 64 filtros de 3x3. Figura extraída de [53].

Adam

El algoritmo de optimización utilizado para todos los modelos basados en CNN es una variante del descenso de gradiente conocida como Adam [101]. Se trata de uno de los optimizadores más utilizados porque su convergencia, respecto a otras alternativas como, el descenso de gradiente por “mini-batches” suele ser más rápida y robusta al valor de la tasa de aprendizaje o al valor inicial de los parámetros del modelo. Este algoritmo adapta la tasa de aprendizaje (λ) a cada parámetro, en cada iteración [38]. Asigna

una mayor tasa de aprendizaje a aquellos parámetros que más afectan a la función pérdida y viceversa. Cuánto afecta un parámetro a la función pérdida se calcula utilizando el gradiente de esta sobre parámetro en esa iteración y ponderándolo con el valor del gradiente en iteraciones previas.

LR-Find

LR-Find es un método experimental que permite estimar un valor adecuado para la tasa de aprendizaje de un modelo [102]. Una alternativa mucho más eficiente y óptima que el ensayo y error. El método consiste en entrenar el modelo con varios “mini-batches” de imágenes (generalmente, una época), aumentando a cada “mini-batch” procesado la tasa de aprendizaje (desde un valor excesivamente bajo a un valor excesivamente alto) y almacenar el valor de la función pérdida para iteración. El resultado es una gráfica como la mostrada de la Figura 4.9. A continuación se selecciona el valor de λ en base a este gráfico, el siguiente párrafo describe el criterio concreto.

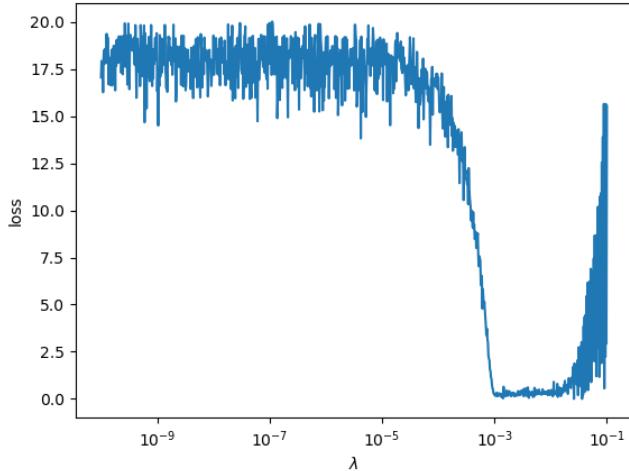


Figura 4.9: Ejemplo LR-Find para un modelo arbitrario. Número de mini-batches procesados 10.000. Cota mínima $\lambda = 1 \cdot 10^{-10}$. Cota máxima $\lambda = 1$. Un valor adecuado para la tasa de aprendizaje estaría entre $\lambda = 1 \cdot 10^{-4}$ y $\lambda = 1 \cdot 10^{-2}$.

Los primeros valores de λ se descartan por ser demasiado pequeños. La función pérdida presenta un valor muy alto, lo que significa que el modelo apenas mejora ya que λ es tan pequeña que apenas permite modificar los parámetros modelo. Los últimos valores también se descartan, esta vez por ser demasiado grandes. La función pérdida se dispara, lo que significa que la actualización de los parámetros es demasiado agresiva, ya que λ presenta un

valor excesivamente alto. Entre ambas etapas, existe un conjunto de valores para los que la pérdida decrece o se mantiene mínima. Esto significa que λ es lo suficientemente grande como para que la actualización tenga efecto notable, sin que este diverja. Cualquier valor perteneciente a este intervalo es aceptable; sin embargo, suele escogerse un λ un orden de magnitud menor que el λ para el que el modelo empieza a aumentar pérdida. De este modo, se es garantista (es poco probable que el modelo diverja) a la vez que se arriesga con una λ lo más grande posible. Recuérdese que a mayor λ sin divergir, más rápido converge el modelo. En este TFG, se utiliza *LR-Find* antes de entrenar toda CNN para estimar la tasa de aprendizaje apropiada.

GradCAM

Uno de los grandes problemas del aprendizaje automático es que los modelos comportan como una “caja negra”. Reciben una entrada y generan una salida, que puede ser correcta o incorrecta, sin aportar ningún tipo de explicación al respecto. *GradCAM* (*Gradient Class Activation Map*) aparece para paliar esta falta de explicabilidad en el caso concreto de modelos basados en CNN [93]. Tras predecir la etiqueta de una imagen, esta técnica consiste generar un mapa de calor/activación que señala los píxeles de la imagen que más han afectado a la salida del modelo. Cuanto mayor es la intensidad del píxel ij -ésimo del mapa de activación, mayor es la influencia del píxel ij -ésimo de la imagen original. Superponiendo este mapa a la imagen original, pueden localizarse a simple vista las regiones de la imagen que más han condicionado la predicción del modelo.

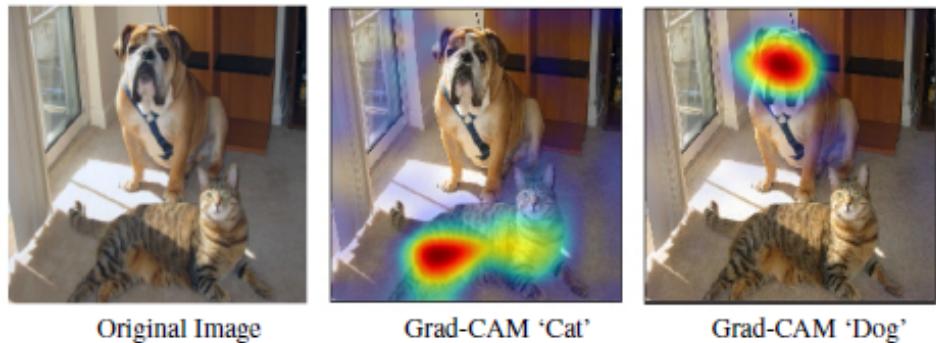


Figura 4.10: Ejemplo de GradCAM para un modelo de clasificación. A la izquierda la imagen original. En el centro, superpuestos, la imagen original y el mapa de activación que señala las regiones que maximizan la clase “gato”. A la derecha ídem para la clase “perro”. Imagen extraída de [93].

El mapa de activación se obtiene combinando los últimos mapas de características generados por la CNN. Se suman de forma ponderada y cada

uno utiliza como coeficiente la media del gradiente de sus píxeles respecto a la salida del modelo. Por ello, en el caso concreto de la clasificación, donde se generan tantas salidas (puntuaciones o probabilidades) como clases, pueden extraerse varios mapas. Cada uno señala los píxeles de la imagen procesada que más condicionan la probabilidad predicha para una clase concreta. Consultar Figura 4.10. A continuación se detalla cómo obtener uno de estos mapas de activación. Sea y_c la probabilidad predicha por una CNN para la clase c , A^k uno de los mapas de características generados por la última capa y A_{ij}^k un píxel del mapa A^k .

1. Se genera un único coeficiente (α_k^c) para cada A^k como la media de la derivada de y_c respecto a A_{ij}^k . Esto es equivalente a aplicar un “global average pooling” a cada mapa de características y calcular la derivada respecto al píxel resultante. Recuérdese que la derivada estima la influencia/importancia de una componente de un punto sobre el valor función en ese punto; en este caso, la influencia/importancia de un píxel de un mapa de características sobre la probabilidad predicha para una clase concreta.

$$\alpha_c^k = \sum_i \sum_j \frac{\partial y_c}{\partial A_{ij}^k}$$

2. Se genera un mapa de activación ($L_{Grad-CAM}^c$) para la clase y_c combinando linealmente A_{ij}^k utilizando los coeficientes α_c^k .

$${}^*L_{Grad-CAM}^c = \sum_k \alpha_c^k \cdot A^k$$

3. Se aplica una ReLU. Esto condiciona la información que contiene el mapa, sólo se señalan las regiones que afectan positivamente a la probabilidad de una clase; es decir, aquellas que inducen al modelo a pensar que la clase correspondiente es c . El resto de regiones, como tienen un gradiente negativo, condicionan al modelo a predecir lo contrario.

$$L_{Grad-CAM}^c = \text{ReLU}({}^*L_{Grad-CAM}^c)$$

4. Se re-escala al tamaño de la imagen original.

Fundamentalmente, este método funciona porque aprovecha los dos rasgos más característicos de las CNN: los mapas de características generados por la última capa convolucional localizan las características verdaderamente importantes a la hora de realizar la predicción y todavía conservan la estructura espacial de la imagen original (en cierto grado). Una CNN predice la clase a la que pertenece una imagen en base a la información que contienen

los mapas de características de la última capa y estos mapas señalan las regiones de la imagen donde se encuentran las características más complejas que ha logrado aprender la imagen, lo verdaderamente relevante para la predicción. Por lo tanto, cuanto mayor sea la influencia de un mapa sobre la predicción (estimada por el gradiente medio de sus píxeles), mayor será la influencia de las regiones de la imagen que señala ese mapa.

Aunque el algoritmo descrito se limite al caso concreto de la clasificación, nótese que puede aplicarse siempre que la salida del modelo sea derivable respecto a los últimos mapas de características generados. Puede utilizarse en otros problemas como la descripción automática de imágenes o la regresión, donde también suele interesar sustituir la ReLU por una operación de valor absoluto. También existen otras variantes como *HiResCAM*, donde en lugar de utilizar el mismo coeficiente para todos los píxeles de un mismo mapa (α_k^c), cada píxel se pondera en base al gradiente de y_c sobre él [103]. Esta variante soluciona que, en ocasiones, el mapa de *GradCAM* señala regiones que en realidad no son relevantes para la predicción o no señala regiones que sí lo son. No obstante, en este proyecto ambos métodos son equivalentes. Las arquitecturas utilizadas reducen los últimos mapas de características a un único número realizando una media global (“global average pooling”), lo que provoca que la derivada de y_c sobre todos los píxeles de un mismo mapa sea la misma, y la media de varios valores iguales es igual/equivalente a cualquiera de los valores.

En este TFG, se utiliza *GradCAM* tras entrenar la CNN con mejores resultados para comprobar en qué parte de una OPG se fijan para determinar la edad legal. Esta información permite comprobar si el modelo se está centra en información irrelevante, como el fondo de la radiografía o etiquetas que acompañan a las radiografías (síntomas de sobreajuste), y si las regiones sobre las que fundamenta su decisión difieren de las que suelen emplear los especialistas en otros métodos del estado del arte, como el tercer molar. Matizar que el algoritmo utilizado difiere ligeramente del *GradCAM* original para clasificación. Recuérdese que la salida de las CNN propuestas es un número real entre 0 y 1 correspondiente a la posibilidad de que la OPG introducida pertenezca a un mayor de edad, no un vector con tantos valores como clases. Los mapas de características con menores gradientes, indican la posición de características que minimizan la salida del modelo; es decir, regiones de la imagen que empujan al modelo a predecir que el individuo es menor de edad. Los mapas de características con mayores gradientes, indican la posición de características que maximizan la probabilidad de ser menor. Eliminando la ReLU final del algoritmo, el mapa de activación resultante de esta versión de *GradCAM* señala al mismo tiempo las regiones que afectan positiva (mayor intensidad) y negativamente (menor intensidad) a la predicción, sin tener que ejecutar *GradCAM* para cada clase. Matizar que también se incluyó un umbral mínimo de intensidad en valor absoluto para

descartar las regiones poco relevantes.

Early Stopping

Una de las formas más sencillas y efectivas para evitar sobre-entrenar un modelo demasiado es el “Early-Stopping”. Esta técnica consiste en detener el entrenamiento cuando el valor de la función pérdida sobre el conjunto de validación diverge o empeora por encima de un umbral (tolerancia) durante un número de épocas seguidas (pacienza). Esta tolerancia y paciencia son hiperparámetros a fijar experimentalmente. Suele habitual incluir un límite máximo de épocas, además del “Early Stopping”, por si el valor de estos hiperparámetros (tolerancia y paciencia) no es adecuado. Además de la función pérdida, cualquier otra métrica aplicable al conjunto de validación podría utilizarse con el mismo propósito (otro hiperparámetro).

En este TFG, además de para impedir el sobreajuste, esta técnica también se empleó para evitar tiempo y esfuerzo computacional innecesarios derivados de tener que fijar el número óptimo de épocas de forma experimental. Matizar que ni siquiera el “Early Stopping” es capaz de impedir que el modelo diverja, aunque sea en levemente, por lo que también se almacenó el mejor modelo encontrado durante el entrenamiento.

4.2.6. Aprendizaje Automático Clásico

Descriptor HOG

Como ya se ha discutido en profundidad en esta memoria 2.1.1, antes de entrenar un modelo, lo habitual es someter los datos a una extracción de características. Transformarlos en nueva representación conocida como descriptor de características. Para los modelos clásicos de ML, la elección de este modelo es independiente del entrenamiento, lo que la hace manual y sujeta al criterio del ingeniero. En este caso se utilizó la técnica HOG [104], del inglés “Histogram of Oriented Gradients”, por ser una de las más habituales.

Como su propio nombre indica, HOG consiste en generar una nueva representación, un vector de números reales, a partir de la distribución de la orientación del gradiente de una imagen. La intuición detrás de este algoritmo es que el gradiente, útil para identificar los bordes de una imagen, contiene la información necesaria para la interpretación de una imagen, ya que son los bordes, y no las regiones planas, los que contienen la información útil o efectiva para interpretar una imagen. A continuación se describe el proceso en detalle. Para ejemplificar cada etapa, se recomienda encarecidamente las figuras de [105], son sumamente ilustrativas pero demasiadas

como para incluirlas todas de forma estructurada en esta memoria.

1. Normalizar la imagen para reducir el efecto de los cambios locales de iluminación. Este paso no es estrictamente necesario.
2. Re-escalar la imagen. Este paso tampoco es estrictamente necesario se re-escalaron las imágenes al mismo tamaño que la CNN evaluadas requirieron. De esta forma, las condiciones iniciales de ambos modelos son similares y la comparación, más justa.
3. Computar las derivadas de la imagen a lo largo (y) y ancho (x). Esto puede conseguirse convolucionando la imagen con un filtro de derivada, también conocido como sobel, en cada dirección. El resultado son dos imágenes: en una ($\frac{\partial I}{\partial x}$) los bordes verticales aparecen resaltados; en la otra ($\frac{\partial I}{\partial y}$), los bordes horizontales. Nótese que para imágenes con más de un canal, el resultado serían dos derivadas por canal. No obstante, la implementación del algoritmo utilizada toma únicamente el valor del canal dominante para cada píxel; es decir, cada píxel se transforma en la componente del canal cuya derivada tiene un mayor valor. Esto puede interpretarse como una cierta robustez del algoritmo a los cambios de color.
4. Calcular la magnitud (g) y la orientación (Θ) del gradiente en cada píxel aplicando las siguientes expresiones. Sean g_x y g_y el valor de las derivadas en un píxel. El resultado son dos matrices con tantas componentes como píxeles tiene las derivadas. Consultar la Figura 4.11.

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\Theta = \arctan\left(\frac{g_x}{g_y}\right)$$

5. Los valores de las matrices anteriores se dividen en celdas. El número de valores o píxeles por celda es un hiperparámetro a elección del ingeniero.
6. Cada celda se transforma en un histograma. El número de contenedores de estos histogramas es otro hiperparámetro a elección del ingeniero. La orientación del gradiente se encuentra en el rango $[0, 360]^\circ$. Este rango se divide entre el número de contenedores. Cada píxel de la celda “vota” una orientación en función de la magnitud de su gradiente. Merece la pena destacar que el rango de la orientación suele reducirse a $[0, 180]^\circ$ para algunas tareas como la detección de via-andantes,

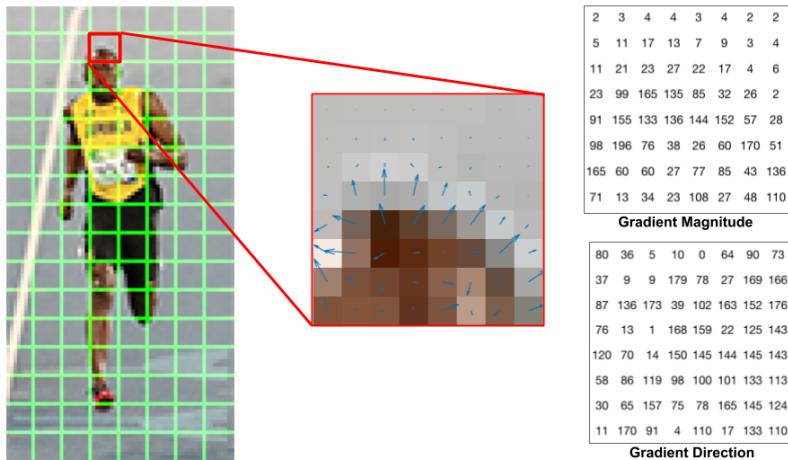


Figura 4.11: HOG. Magnitud (g) y orientación (Θ) del gradiente de una celda 8x8. Proceso de extracción de características HOG. Imagen extraída de [105].

aplicación original de este algoritmo. Los resultados son empíricamente mejores. De esta forma, se considera que, por ejemplo, 90° y 270° , tienen la misma orientación.

7. Normalización de los histogramas. En lugar de normalizar cada histograma por separado, se tienen en cuenta las celdas de su alrededor. El tamaño de esta ventana deslizante de normalización es también otro hiperparámetro. El “stride” de esta ventana suele ser igual al tamaño de la celda, de forma que cada celda afecta a varias componentes de la representación final. Este proceso intenta hacer la representación más robusta a cambios de iluminación, sombras y contraste. El resultado es un histograma normalizado por cada vez que se aplica la normalización.
8. Concatenar los valores de cada histograma normalizado y de todos los histogramas generados en un único vector de una sola dimensión.

No se hizo un estudio experimental de los hiperparámetros descritos previamente, se utilizaron los valores utilizados en el artículo original. Por un lado, ajustar experimentalmente estos hiperparámetros dispara la cantidad de ensayos a realizar, que son además independientes a los modelos utilizados, también con un buen número de hiperparámetros a determinar. Por otro lado, el objetivo de este proyecto es la aplicación de DL, el uso de técnicas clásicas de ML se incluyó a modo de comparativo. Dicho lo cual, el valor por defecto de los hiperparámetros utilizados es el siguiente: re-escalado de la imagen a 64×128 , 8x8 para el tamaño de celda, $[0, 180]^\circ$ como rango

de valores para la orientación, 9 contenedores por histograma, tamaño de 16×16 y “stride” de 8 para la ventana de normalización.

Regresión Logística

La regresión logística (en inglés *Logist Regression* o LR) es un modelo lineal. Esto quiere decir que la etiqueta predicha para una instancia responde a una combinación lineal de las componentes de dicha instancia. Dada su incapacidad para modelar funciones no lineales, suele presentar peores resultados que modelos más complejos como las redes neuronales; no obstante, su evaluación se incluyó en este proyecto a modo comparativo.

El algoritmo consiste tomar una instancia del conjunto de datos ($x = (1, x_1, \dots, x_n)$), que debe ser un vector de números reales, y combinar linealmente sus parámetros en base a unos pesos ($W = (w_0, w_1, \dots, w_n)$). En el caso de la clasificación binaria, el resultado del producto con desplazamiento anterior, es un número real al que posteriormente se le aplica una sigmoide (σ). La salida de la sigmoide y del modelo ($f(W, x)$) está acotada entre 0 y 1, por lo que puede interpretarse como la probabilidad de que la instancia procesada pertenezca a la clase positiva. Nótese que este modelo también puede interpretarse como una red neuronal formada por únicamente una neurona con una función de activación sigmoide.

$$f(W, x) = \sigma(W^t \cdot x) = \frac{1}{1 + e^{-(W^t \cdot x)}}$$

Para encontrar el valor de W suele optimizarse la entropía cruzada (extensamente discutida en 2.2.2). Esta función pérdida admite dos términos de regularización: L_1 y L_2 . El peso de esta penalización puede ajustarse a través de hiperparámetro λ_L .

$$\begin{aligned} L_1(W) &= \lambda_L \cdot \sum_{i=0}^n \|w_i\| \\ L_2(W) &= \lambda_L \cdot \sum_{i=0}^n w_i^2 \end{aligned}$$

Tanto el optimizador utilizado, como el tipo y agresividad de la regularización, son hiperparámetros de este modelo. En este TFG, se experimentó con ellos para encontrar el modelo con mejores resultados.

Support Vector Machines

Los SVM (del inglés *Support Vector Machines*) [106] son modelos que pueden utilizarse tanto para clasificación como para regresión. En el contexto

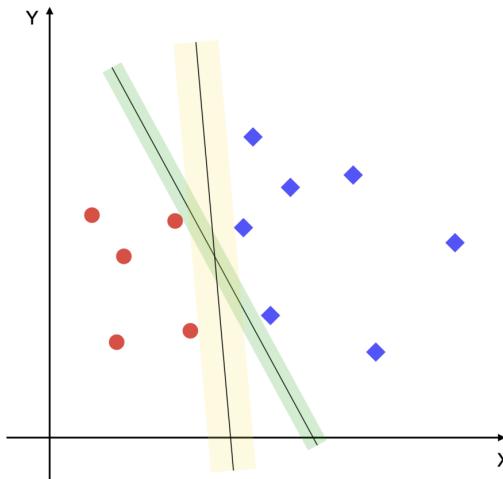


Figura 4.12: Ejemplo de hiperplano óptimo. Resaltado en verde, un hiperplano que clasifica correctamente todos los casos, pero no es óptimo. Resaltado en amarillo, un hiperplano óptimo. Imagen extraída de [107].

to de la calificación binaria, estos modelos tratan de encontrar una frontera (hiperplano) en el espacio en el que están definidos los datos de entrada, que separe de forma óptima las instancias de cada clase. Un hiperplano es un subespacio con una dimensión menos que el espacio en el que se encuentra (una recta en un espacio bidimensional, un plano 2D en un espacio tridimensional) que divide el espacio en dos subespacios. Todas las instancias de un mismo subespacio tienen la misma etiqueta. Un hiperplano es óptimo cuando maximiza el margen entre clases; es decir, cuando maximiza la distancia entre sí mismo y las instancias de datos más cercanos de cada clase. A estas instancias fronterizas se les conoce como vectores de soporte. Consultar la Figura 4.12 para una representación visual de la diferencia entre una frontera óptima y una que no.

Uno pudiera pensar que estos modelos sólo pueden encontrar fronteras lineales. No obstante, recurren a un recurso conocido como kernel, que les permite clasificar adecuadamente instancias de datos aunque no sean linealmente separables. Los kernels son funciones transforman los datos a un nuevo espacio de mayor dimensionalidad donde puede que si sean linealmente separables. Al proyectar el hiperplano de vuelta en el espacio original, este será una frontera no lineal. Existen distintos tipos de kernels (gaussianos, polinómicos, sigmoides, radial ...). El kernel a utilizar es un hiperparámetro que debe elegir el ingeniero. Consultar la Figura 4.13.

Lo último que quedaría por definir es cómo se calcula ese hiperplano óptimo. Un hiperplano está caracterizado por su vector normal (W) y su desplazamiento (b). Ergo, para calcular el plano, sólo hay que obtener el

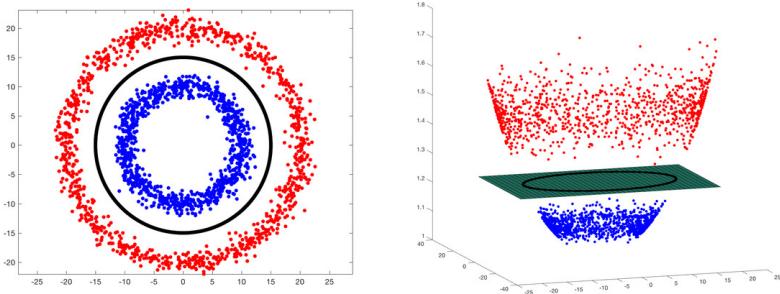


Figura 4.13: Ejemplo de aplicación de kernel. Aplicando un kernel gaussiano ($X' = e^{-(x_0^2+x_1^2)}$) a un conjunto de datos que no es linealmente separable en 2D, se obtiene un nuevo conjunto de datos donde. Imagen extraída de [108].

valor de W y b .

- En el caso de datos linealmente separables (“Hard Margin SVM”), lo único que importa es que el hiperplano resultante maximice el margen entre clases. Puede inferirse que para lograr este objetivo debe minimizarse el valor de $\|W\|$, teniendo en cuenta que para toda instancia del conjunto de entrenamiento (x_i) y su respectiva etiqueta (y_i , que puede ser 1 o -1) debe cumplirse que $y_i \cdot (W \cdot x_i - b) \geq 1$.
- En el caso más realista, datos que no son linealmente separables (“Soft Margin SVM”), las condiciones anteriores nunca se cumplirán. Puede inferirse que para calcular el valor de W y b debe minimizarse la siguiente función pérdida. Sea x_i una instancia del conjunto de datos y y_i su etiqueta, con un valor igual a 1 y -1.

$$\bar{L}_H(W, b, x_i, y_i) = \frac{1}{n} \sum_{i=0}^n \max(0, 1 - y_i \cdot (W \cdot x_i - b))$$

Esta función se conoce como “Hinge Loss” y es mayor cuánto mayor sea la confianza con la que erra en la clasificación de una instancia, aunque también penaliza si acierta con poca seguridad. Para mantener la importancia de maximizar el margen entre clases, se añade el término $\|W\|^2$ controlado por el factor λ_{SVM} , otro hiperparámetro a disposición del ingeniero. Finalmente, la función a minimizar queda de la siguiente forma.

$$\bar{L}_H(W, b, x_i, y_i) = \frac{1}{n} \sum_{i=0}^n \max(0, 1 - y_i \cdot (W \cdot x_i - b)) + \lambda_{SVM} \cdot \|W\|^2$$

Random Forest

Random Forest (RF) [109, 110] se caracteriza por ser un modelo “ensemble”; es decir, está formado por un conjunto de modelos cuyas predicciones se combinan. En este caso, se trata de un conjunto de árboles de decisión. Un árbol de decisión es una estructura que divide jerárquica y sucesivamente el espacio dimensional en el que están definidos los datos con los que trabaja en modelo. Consultar la Figura 4.14.

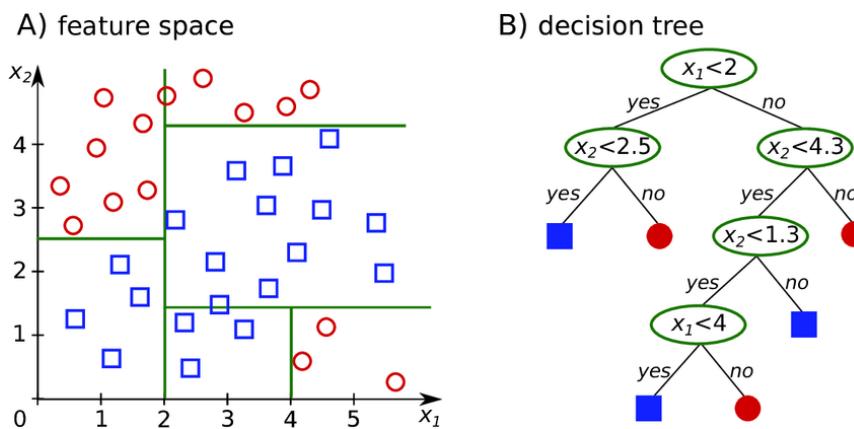


Figura 4.14: Ejemplo simplificado de árbol de decisión para un problema de clasificación binaria de instancias bidimensionales. Imagen extraída de [111].

- Como su propio nombre indica, un árbol de decisión puede representarse como una estructura formada por un nodo raíz, un conjunto de nodos internos, un conjunto de nodos hoja y las ramas que los unen.
- Cuando un árbol procesa una instancia, esta se introduce por el nodo raíz hasta alcanzar alguna de las hojas, que está asignada a una etiqueta. Esta es la etiqueta predicha por el modelo para la instancia procesada. Para atravesar el árbol, en cada nodo se evalúa una característica de la instancia y, en función de su valor, se redirige por una rama u otra.
- Durante el proceso de entrenamiento, se construye el árbol. Partiendo de un único nodo (futuro nodo raíz) se evalúa de forma intensiva qué característica y qué umbral sobre el valor de esta divide mejor al conjunto de entrenamiento en base a un criterio de pureza. Este criterio pureza debe favorecer la homogeneidad de los subconjuntos resultantes de la división y es un hiperparámetro a fijar por el ingeniero. El entrenamiento se detiene cuando se alcanza una profundidad máxima, cuando se alcanza un nivel de pureza aceptable o cuando no quedan instancias suficientes para ser divididas.

Como puede intuirse, uno de los problemas más graves de estos modelos es el sobreajuste. De no ajustar adecuadamente los hiperparámetros de parada y pureza, el árbol puede continuar creciendo hasta alcanzar un error nulo sobre todos los casos de entrenamiento, creando subdivisiones en el espacio demasiado concretas y poco generalizables (proveniente de un “outlier” o de un caso mal clasificado). Para evitar este exceso de sobreajuste aparece el “ensemble” más básico de árboles, también conocido como “bagging”. Propuesto en [112], consiste en entrenar varios árboles de decisión, cada uno con una partición distinta de los datos de entrenamiento. Las predicciones se toman por voto mayoritario simple. Esto hace que los errores por sobreajuste queden amortiguados, ya que, siendo distintos los datos con los que entranan, difícilmente la mayoría de árboles cometan los mismos errores debidos al sobreajuste.

No obstante, RF va más allá. Se añade una capa más de aleatoriedad sobre “bagging” para aumentar la diversidad de los árboles generados y así reducir el sobreajuste. Durante el entrenamiento, cada vez que se escoge qué característica y umbral para ella evaluar en un nodo, en lugar de disponer todas las características posibles, se limita a un número aleatorio de estas (otro hiperparámetro a fijar por el ingeniero). Esto evita que si existe una característica y/o un umbral que permite dividir el conjunto de casos mucho mejor que el resto, no todos los árboles la elijan como primer nodo.

Recapitulando, los hiperparámetros de este modelo son: el número de árboles, la profundidad máxima de estos, el número de características a evaluar en cada nodo, el criterio para evaluar la calidad de una partición (criterio de pureza) y el número mínimo de ejemplos que debe tener un nodo para no ser considerado hoja. En este TFG, se evaluaron distintas versiones de este modelo combinando distintos valores de estos hiperparámetros.

Capítulo 5

Implementación

La realización de los experimentos descritos en esta memoria implicó la implementación de un marco software en el que poder llevarlos a cabo de forma eficiente. Puede encontrarse subido y documentado en el repositorio <https://github.com/Palenchuekla/TFG>. Como es habitual en el campo del ML y la ciencia de datos, el lenguaje de programación utilizado fue *Python* [113]. El amplio abanico de bibliotecas que ofrece, como *Pandas* [114], *Numpy* [115], *Scikit Learn* [24], *Captum* [25] y, especialmente, *PyTorch* [23], dinamizan el desarrollo en este contexto. Respecto a otras alternativas como *fastai* [26], utilizada durante grado, *PyTorch* se escogió por su menor el nivel de abstracción. Las labores del ingeniero van más allá de hacer una llamada a la API para cargar los datos y el modelo, otra para entrenar y otra para evaluar. *PyTorch* obliga al programador a implementar el bucle de entrenamiento y evaluación, así como funcionalidades que se dan hecho en otras librerías, como guardar el mejor modelo encontrado o el *Early Stopping*. Aunque esto pueda parecer un inconveniente, aporta mucha más flexibilidad y transparencia durante el desarrollo. Además, permite entender realmente lo que está ocurriendo.

La mayor parte del desarrollo y ejecuciones se realizó de forma local, en una máquina como la descrita en Subsección 1.4. Como editor se utilizó *Visual Code Studio* [116] y, aunque se creó una pequeña API de funciones que puede ser ejecutada desde línea de comandos, el código está diseñado para *Jupyter Notebooks* [117], un entorno que permite ejecutar *Python* de forma interactiva, bastante habitual en el contexto del ML. Para gestionar la instalación y las dependencias de cada una de las bibliotecas utilizadas se utilizó la herramienta *Conda* [118], lo aporta una gran portabilidad al marco desarrollado. Para replicar este entorno de ejecución, puede utilizarse la propia herramienta *conda* y el fichero *YAML* [119] del repositorio.

En el caso de las técnicas clásicas de ML, el diseño y la implementación fueron relativamente sencillos. La mayor parte de las funcionalidades

necesarias estaban incluidas de antemano en la biblioteca *Scikit Learn*. En el caso de los modelos DL, la tarea fue algo más compleja. De todas las funcionalidades implementadas, merece la pena destacar las siguientes: las funciones **train**, **LR-Find** y **justify** y las clases **RAMDataset**, **SingleLogitResNet** y **CustomMixUp**. Puede encontrarse una documentación más detallada de estas y otras funcionalidades en el repositorio.

- **RAMDataset**. Esta clase permite, a partir de un archivo “.csv” en el que a cada fila se especifica la ruta de una imagen y su respectiva etiqueta, cargar en memoria dicho conjunto de imágenes pre-procesado. Esta precarga reduce dramáticamente el tiempo de ejecución, de lo contrario, deberían pre-procesarse y cargarse en memoria todas las imágenes de un “mini-batch” cada vez que éste tiene que ser procesado por el modelo. El constructor de esta clase admite como parámetros: la ruta del “.csv”, el nombre de las columnas que contienen las rutas y las etiquetas de las imágenes, las transformaciones a aplicar como pre-procesado (normalizado y re-escalado en este proyecto) y la posibilidad de informar, con una barra de progreso, del estado de la carga.
- **SingleLogitResNet**. Esta clase permite instanciar un clasificador binario cuya arquitectura sea la descrita en esta memoria 2.2.1. El extractor de características puede ser cualquier arquitectura *ResNet* ofrecida la biblioteca *PyTorch*. De esta forma, el flujo de experimentación es independiente a la CNN utilizada como base. A futuro, añadir otras cabeceras completamente conectadas u otros extractores de características sería bastante sencillo.
- **CustomMixUp**. Implementación propia de la técnica de aumento de datos *MixUp*, descrita en la Subsección 4.2.3. Admite α_{MU} como parámetro. Aunque se planea hacerlo, esta funcionalidad aún no está incluida en la versión estable de *PyTorch*.
- **train**. Una función que permite entrenar y validar un clasificador **SingleLogitResNet**. La función admite como parámetros: algoritmo de optimización, política para alterar la tasa de aprendizaje durante el entrenamiento (planificador), función pérdida, número máximo de épocas, conjunto de datos utilizados para entrenar y validar, qué métricas se evalúan durante entrenamiento y validación, *Early Stopping* (métrica a evaluar, paciencia y tolerancia), técnicas de aumento de datos a aplicar al conjunto de entrenamiento (a qué clase y con qué probabilidad aplicarlas), posibilidad de *MixUp* y su α_{MU} (a aplicar antes o después que el resto de técnicas de aumento de datos), posibilidad de almacenar y cargar el mejor modelo encontrado durante el entrenamiento (así como la métrica evaluada), la ruta de un directorio en el

que almacenar todos los resultados y la posibilidad de informar con gráficas y barras de progreso del estado del entrenamiento. Esta función hace la mayor parte del trabajo pesado de todos los experimentos. Como resultado, devuelve el modelo entrado (última iteración o mejor modelo encontrado, según se haya especificado) y crea un directorio en el que se almacena: un fichero con los el valor de los parámetros para el mejor modelo encontrado, un “.csv” con el valor de las métricas de rendimiento a cada época (junto al tiempo de ejecución de esta) y un “.png” con las curvas de aprendizaje (entrenamiento y validación) para cada métrica especificada.

- **LR-Find.** Una función que permite ejecutar la técnica homónima (expuesta en la Subsección 4.2.5) para encontrar una tasa de aprendizaje adecuada. Recibe como parámetros: un modelo **SingleLogitResNet**, la función pérdida, el conjunto de datos, las técnicas de aumento de datos, si se utiliza *MixUp* y su α_{MU} (a aplicar antes o después que el resto de técnicas de aumento de datos), exponente mínimo para la tasa de aprendizaje ($\lambda_{min} = 1 \cdot 10^{e_{min}}$), exponente máximo ($\lambda_{max} = 1 \cdot 10^{e_{max}}$) y el número de iteraciones (= número de mini “batches” a procesar = número de tasas de aprendizaje a evaluar) y un directorio donde almacenar los resultados. Como resultado, en la ruta especificada crea directorio con un “.csv” donde se desglosa el valor de la función pérdida a cada iteración o valor de la tasa de aprendizaje.
- **justify.** Dado un modelo **SingleLogitResNet**, la última capa convolucional de éste, las transformaciones que hace antes procesar una imagen (re-escalado y normalizado en este caso concreto) y una imagen cuya etiqueta debe predecirse, genera la predicción justificada del modelo. Una Figura similar a las de la Subsección 6.3.3.

Para cada experimento DL, se ejecutó un cuaderno de *Jupyter Notebooks* en el que: se cargan los datos en memoria (**RAMDataset**); se especifica el valor de los hiperparámetros de entrenamiento (**SingleLogitResnet**, **LR-Find** y/o **CustomMixUp**); se define un directorio en el que almacenar un informe de los hiperparámetros utilizados y los resultados; y, finalmente, se lanza la validación cruzada para el modelo definido (sucesivas llamadas a **train**). Puede encontrarse un ejemplo de este flujo experimental en el repositorio. Al final de la ejecución, colgando del directorio especificado, el usuario cuenta con la siguiente estructura de archivos:

- **model.txt.** Contiene un esquema de arquitectura utilizada donde se detalla el número de capas, el tipo de capa, los hiperparámetros, número de parámetros congelados y descongelados o la dimensionalidad de los datos de entrada y salida.

- **report.txt.** Contiene un resumen del resto de hiperparámetros utilizados: semilla para inicializar procesos aleatorios, optimizador, tasa de aprendizaje, tamaño del “batch”, función pérdida, *Early Stopping* (pacienza, tolerancia y métrica estudiada), pre-procesado de las imágenes (como el escalado y/o la normalización) o la técnica de aumento de datos utilizada.
- **initial_values.pt.** Uso interno. Contiene el valor inicial de los parámetros para el modelo. De esta forma, en todas las iteraciones de la validación cruzada se parte del mismo punto.
- **partition_i.** Un directorio para cada iteración de la validación cruzada similar al generado por la función **train**.

Capítulo 6

Experimentación

El objetivo final de esta sección es obtener el mejor clasificador posible. Para ello, primero se buscó la mejor combinación de hiperparámetros para cada “familia” de modelos vía validación cruzada: *ResNet18*, *ResNet50*, *RL*, *SVM* y *RF*. A continuación, el mejor candidato de cada familia se entrenó sobre el conjunto completo de entrenamiento-validación y se evaluó sobre el conjunto de test. El rendimiento sobre este conjunto (nunca antes visto) es una estimación realista de la capacidad de generalización de dichos clasificadores, por lo que permite escoger el mejor de ellos y compararlos con resultados de otros estudios recogidos en el Estado del Arte. O bien a lo largo de la sección, o bien en el Apéndice, pueden encontrarse Cuadros con los resultados en bruto de todos los experimentos realizados. Por motivos legibilidad, no se incluyeron todos en esta sección.

Para evaluar el rendimiento de los modelos, recuérdese que las métricas analizadas son, además de la función pérdida (*Loss*), la exactitud (*ACC*), la especificidad (*SPE*), la sensibilidad (*SEN*) y el sobreajuste para la sensibilidad (*OSEN*). La *ACC* refleja cómo de bien funciona el clasificador para ambas clases, pero se incluye *SEN* y *SPE* para evitar conclusiones engañosas derivadas de trabajar con un conjunto de datos desbalanceado, como es el caso. La *SEN* refleja lo bien que funciona el clasificador para la clase positiva (mayores de edad). La *SPE* refleja lo bien que funciona el clasificador para la clase negativa (menores de edad). *OSEN* mide la diferencia entre *SEN* para el conjunto de entrenamiento y *SEN* para el conjunto de validación, el sobreajuste del modelo para la clase positiva. Más información en la Subsección 4.2.4.

En el caso de las redes neuronales, recuérdese también que el criterio para elegir el mejor modelo en una iteración de la validación cruzada fue $M(\alpha_M)$, mayor media ponderada de *SEN* y *SPE*; que el criterio para determinar los mejores resultados de validación cruzada fue $M_{CV}(\alpha_M)$, menor diferencia entre *SEN* y *SPE*; y que cada vez que deben evaluarse los resultados de la

validación cruzada, se utilizan varios factores de ponderación para la elección del mejor modelo en cada iteración, $\alpha_M = 0.4, 0.5, 0.6$. En el caso los modelos de ML clásico, este criterio se adaptó dadas las limitaciones de la biblioteca utilizada. Como M_{CV} pasó a ser un indicador menos seguro, también se tuvo en cuenta la media de SEN y SPE a la hora de escoger el mejor candidato. Más información en la Subsección 4.2.4.

6.1. Experimentación - Aprendizaje Profundo

Antes de realizar cualquiera de los experimentos, primero se pre-procesaron los datos.

- Re-escalado del conjunto de datos (entrenamiento, validación y test). Para todos los modelos evaluados, las imágenes se re-escalaron a $224 \times 224 \times 3$. Estas dimensiones son habituales para este tipo de problemas, proporcionan una velocidad de cómputo aceptable sin perder demasiada información. De hecho, puede consultarse los modelos utilizados están preentrenados con las imágenes de *ImageNet* reescaladas a estas dimensiones.
- Normalización el conjunto de datos (entrenamiento, validación y test). Las imágenes también se normalizaron para todos los modelos evaluados. La normalización (por canal) se hace en base a las métricas (media y desviación típica) de *ImageNet*. En esta memoria, ya se han justificado las bondades de normalizar los datos; no obstante, no se ha justificado por qué debe hacerse en base a las métricas de *ImageNet*. Esto se debe a que todos los modelos utilizados están preentrenados en este conjunto de datos y durante dicho pre-entrenamiento, las imágenes se normalizan en base a estas métricas. Podría interpretarse como que los modelos están acostumbrados a que la realidad se muestre de una determinada forma, de forma normalizada.

La mayoría de experimentos realizados con modelos de DL siguieron la siguiente estructura.

1. Escoger una CNN preentrenada en *ImageNet* y añadirle una cabecera para adaptarla al problema de la clasificación binaria. En este proyecto, esta cabecera siempre está compuesta por una red neuronal completamente conectada, donde la capa de entrada tiene tantas neuronas como mapas de activación genera la última capa convolucional y la cada de salida está compuesta por una única neurona de salida cuya función de activación es una sigmoide. No se utilizaron capas ocultas.

El número de mapas o entradas depende de la arquitectura concreta, para *ResNet18* es de 512 y para *ResNet50* es de 2048.

2. Escoger manualmente el valor ciertos hiperparámetros: optimizador, tamaño del “batch”, número máximo de épocas, métrica para evaluar el mejor modelo encontrado y métrica, tolerancia y paciencia del *Early Stopping*.

Si no se especifica nada al contrario, el modelo con el que se experimenta utilizó los siguientes valores: optimizador ADAM (con los hiperparámetros predeterminados); tamaño del “batch” de 32 imágenes, $M(\alpha_M)$ como la métrica evaluada para encontrar el modelo y para *Early Stopping*, que además se usa con una tolerancia de 0.01 y una paciencia de 10 épocas; número máximo de épocas de 50.

No se experimentó para hallar el valor óptimo de estos hiperparámetros porque no suelen jugar un rol diferencial en los resultados si tienen un valor aceptable, que a su vez es fácil de estimar sin una experimentación rigurosa.

3. Estimar experimentalmente un valor adecuado para la tasa de aprendizaje (λ) vía *LR-Find*. Número de “batches” procesados 1000. Tamaño del “batch” 32. Cota mínima $\lambda = 1 \cdot 10^{-10}$. Cota máxima $\lambda = 1 \cdot 10^{-1}$. Crecimiento exponencial de λ a cada iteración. Puede consultarse el significado de todos estos hiperparámetros en la Subsección 4.2.5.
4. Entrenar y validar el modelo vía validación cruzada. Recuérdese que en caso de aplicar sobremuestreo de la clase minoritaria o aumento de datos, estos se aplican sobre el conjunto entrenamiento que la iteración de validación cruzada está utilizando. Ídem para el aumento de datos.
5. Evaluar resultados. Cada ejecución de la validación cruzada genera unas curvas de entrenamiento y un valor medio para cada una de las métricas analizadas (resultado de promediar el rendimiento de los mejores modelos encontrados a cada iteración). Estos resultados se utilizaron para analizar lo ocurrido y comparar distintas configuraciones de hiperparámetros.

Merece la pena mencionar que, durante los experimentos, se utilizaron semillas para garantizar la reproducibilidad de los resultados y la igualdad de condiciones. Estas se fijan antes de realizar particiones aleatorias, antes de sobremuestrear y/o aumentar los datos, así como antes de comenzar el entrenamiento. También se almacenan los valores iniciales de cada modelo con el mismo propósito. Las fuentes de aleatoriedad que afectan al modelo son las siguientes: los valores iniciales de los pesos, la partición de entrenamiento-validación y test y la de la validación cruzada, el mezclado

aleatorio del contenido de los “batches” a cada época y las transformaciones aplicadas durante el aumento de datos.

6.1.1. *ResNet18*

Esta arquitectura se utilizó para, en primera instancia, motivar el uso de algunas de las técnicas descritas en la sección anterior. Más concretamente, el uso de *LR-Find* para estimar una tasa de aprendizaje óptima y el del sobremuestreo para balancear los datos. A continuación, se experimentó con la agresividad y la naturaleza del aumento de datos y la agresividad del *Fine-Tuning* para encontrar el mejor modelo posible basado en esta arquitectura.

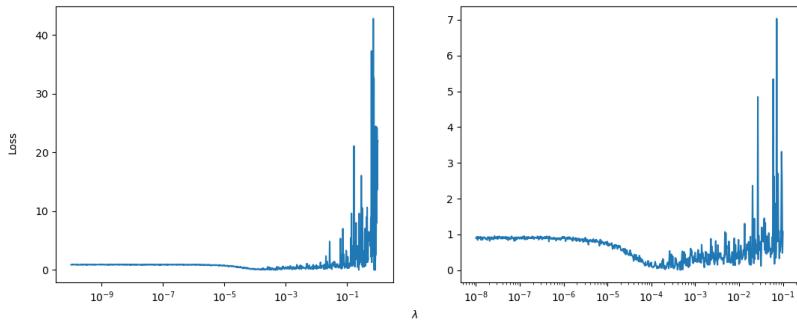
LR-Find

El objetivo de esta sección es ilustrar experimentalmente que *LR-Find* es un método apropiado para determinar la tasa de aprendizaje de un modelo λ . Para ello, se ejecutó *LR-Find* para una *ResNet18*. Más concretamente, se entrenó el modelo con 1000 “mini-batches” de 32 de imágenes cada uno. A cada “mini-batch” procesado, se aumentó exponencialmente el valor de la tasa de aprendizaje, desde $\lambda = 1 \cdot 1^{-10}$ hasta $\lambda = 1$. El resultado obtenido puede consultarse en la Figura 6.1a.

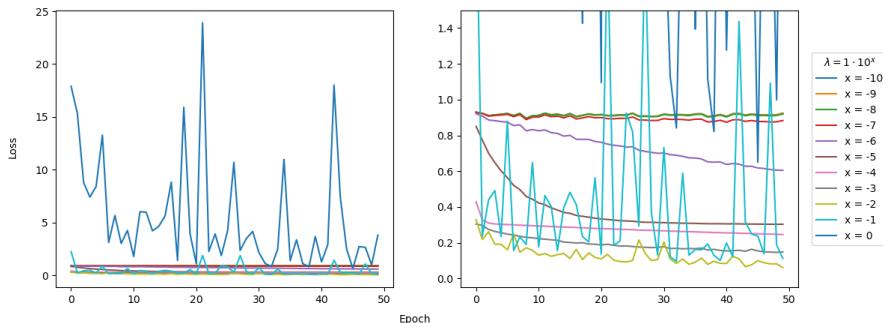
Como ya se justificó en la Subsección 4.2.5, analizando la pendiente de la gráfica, un valor entre $1 \cdot 1^{-5}$ e $1 \cdot 1^{-3}$ para λ sería apropiado. Sin embargo, se quiso respaldar experimentalmente esta afirmación. Se comprobó cómo evoluciona la curva de entrenamiento del modelo durante 50 épocas para algunos valores de λ que se evalúan durante *LR-Find*. En la Figura 6.1b puede observarse que si λ es muy baja, el modelo apenas mejora, y si la λ es muy alta, el modelo se comporta de forma errática, como una búsqueda aleatoria, pudiendo llegar incluso a divergir. Como se sugirió inicialmente, el entrenamiento converge de forma más controlada y a un mínimo más óptimo cuando $\lambda = [1 \cdot 1^{-5}, 1 \cdot 1^{-3}]$.

Sobremuestreo

El objetivo de esta sección es comprobar si es útil o necesario sobremuestrear la clase minoritaria para mejorar el rendimiento del modelo. Recuérdese que este sobremuestreo consiste en duplicar el número de imágenes de la clase minoritaria hasta que el número de casos de cada clase sea el mismo (hasta que se balanceen los datos). Se entrenó y validó una misma *ResNet18* sin sobremuestrear y sobremuestreando el conjunto de entrenamiento de cada iteración de la validación cruzada. La red tenía todos los pesos congelados, salvo los de la cabecera completamente conectada. En ambos casos se utilizó

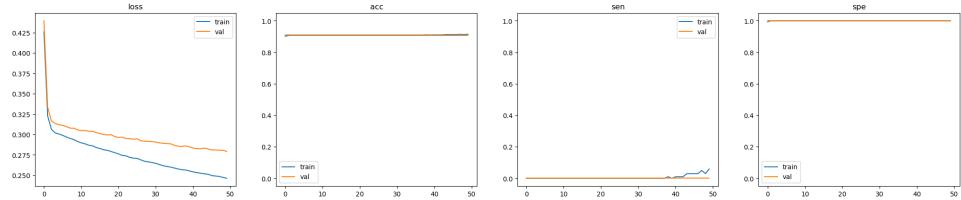


(a) Resultados LR-Find para la *ResNet18*. Misma gráfica a ambos lados, en la derecha se suprime los valores exageradamente altos que dificultan el análisis de la región de interés de la curva. Número de “mini-batches” procesados 1000. Cota mínima $\lambda = 1 \cdot 10^{-10}$. Cota máxima $\lambda = 1$. Crecimiento exponencial de λ a cada iteración.

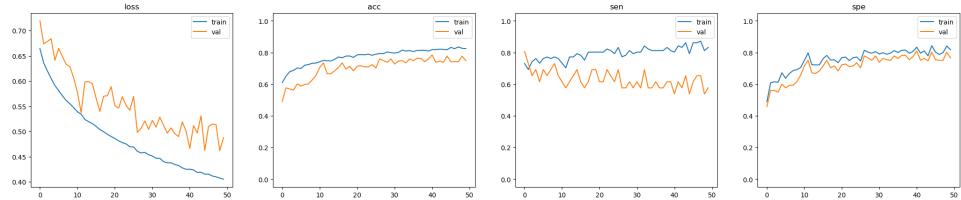


(b) Función pérdida de una misma *ResNet18* en función del valor de la tasa de aprendizaje (λ). Para todo λ se utilizó el mismo conjunto de hiperparámetros, el mismo conjunto de datos, los mismos valores iniciales para los parámetros del modelo y, aunque el orden y el contenido de los “mini-batches” procesados a cada época sea aleatorio, también es el mismo para todos.

Figura 6.1: Resultados de la experimentación para *LR-Find*.



(a) Conjunto de entrenamiento desbalanceado. El modelo interioriza el balanceo.



(b) Resultados para un conjunto de entrenamiento balanceado via sobre-muestreo. El modelo no interioriza el balanceo.

Figura 6.2: Curvas de entrenamiento y validación de una misma *ResNet18* que entrena con un conjunto de entrenamiento desbalanceado (a) y balanceado (b). De izquierda a derecha, función pérdida (*loss*), exactitud (*ACC*), sensibilidad (*SEN*) y especificidad (*SPE*). Resultados correspondientes a una misma iteración de la validación cruzada ($k = 2$).

una tasa de aprendizaje de $\lambda = 1 \cdot 10^{-4}$, un valor adecuado de acuerdo con *LR-Find*. Se entrenó en ambos casos durante 50 épocas, sin *Early Stopping*, para tener una visión más completa de lo ocurrido y porque el experimento no requería un exagerado tiempo de ejecución.

En la Figura 6.2a, puede observarse cómo, entrenando con un conjunto de entrenamiento desbalanceado, el modelo interiorizó el desbalance de los datos. El modelo casi siempre respondía que la imagen procesada pertenece a la clase mayoritaria. Las curvas relativas a la función pérdida pueden hacer creer que el modelo aprendió a clasificar imágenes y que este conocimiento extrapolaba fuera del conjunto de entrenamiento. De hecho, la *ACC* confirma que el modelo es un buen clasificador. No obstante, el bajo valor de *SEN* y el alto valor de *SPE*, denotan que lo que ocurrió realmente es que el modelo aprendió a responder siempre que la imagen procesada pertenecía a la clase mayoritaria, que el individuo analizado es siempre menor de edad. A continuación se entrenó el mismo modelo sobre-muestreando el conjunto de entrenamiento a cada iteración de la validación cruzada.

En la Figura 6.2b, puede observarse cómo, entrenando con un conjunto de entrenamiento balanceado, el rendimiento del modelo mejoró notablemente para la clase minoritaria (*SPE*). El modelo no interiorizó el desbalance, no aprendió a clasificar siempre como la clase mayoritaria. También es reseñable que la curva de la función pérdida para el conjunto de validación,

aún manteniendo una tendencia descendente, presenta un comportamiento algo más errático (picos). Esto se debe a que el conjunto de validación de cada iteración de la validación cruzada no se sobremuestrea. En cada iteración de la validación cruzada, el conjunto está formado por (individuo arriba o abajo): 253 menores (91 %) vs 25 mayores (9 %). Ese exagerado desbalance provoca que, aunque a cada época el modelo mejore globalmente para ambas clases, si la mejora beneficia levemente a la clasificación de la clase minoritaria en detraimiento de la clasificación para la clase mayoritaria (acción-reacción que suele ser habitual), el error global para el conjunto de validación aumente, ya que hay muchas más imágenes de la clase mayoritaria. El desbalance es tan exagerado que empeorar levemente para la clase mayoritaria dispara el error medio de la función pérdida para todo el conjunto. Estos picos puede dificultar el análisis del rendimiento del modelo; no obstante, por ello se utilizan otras métricas independientes a ese desbalance (*SPE* y *SEN*).

El anterior análisis comparativo demuestra que el sobremuestreo permite obtener un mejor modelo, puede consultarse la mejora de cuantitativa en el Cuadro 6.1. No obstante, los resultados obtenidos en esta Subsección motivaron potenciales formas de mejorar el rendimiento de este mismo modelo: aumento de datos y *Fine Tuning*. Por un lado, se observó que el modelo sobreajustaba para los casos de la clase minoritaria, fenómeno que sugería el uso de aumento de datos, al menos sobre esa clase. En la Figura 6.2b, puede observarse que el área entre la curva de *SEN* para el conjunto de entrenamiento y el de validación no es despreciable. Concretamente, el sobreajuste medio en validación cruzada del modelo utilizando sobremuestreo fue de $O_{CV}(SEN) = 0.1197$ (revisite la Subsección 4.2.4). Esta pérdida de rendimiento para el conjunto de validación es razonable, aunque se hubiera balanceado el conjunto de entrenamiento vía sobremuestreo de la clase minoritaria, las nuevas imágenes generadas eran duplicados idénticos de las imágenes originales. Esta falta de diversidad en las imágenes de la clase minoritaria provoca que el modelo sobreajuste para ellas. Por otro lado, nótese que el sobreajuste era mínimo para la clase mayoritaria. Esto sugería que aumentar el número de épocas y/o la potencia del modelo vía *Fine Tuning* podría mejorar el rendimiento de la red para esta clase. No obstante, ambas soluciones (más épocas y más potencia) podrían derivar en un aún mayor sobreajuste para la clase antes minoritaria. Es por esto que la experimentación con estos dos factores se pospuso hasta intentar reducir el sobreajuste para la clase antes minoritaria vía aumento de datos.

Aumento de Datos

El objetivo de esta es comprobar si el aumento de datos es una técnica útil para obtener un mejor rendimiento después de balancear los datos vía

Modelo	α_M	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	<i>M_{CV}</i>
Desbalanceado	0.4	0.3400	0.9103	0.0157	1.0000	0.9843
	0.5	0.3400	0.9103	0.0157	1.0000	0.9843
	0.6	0.3400	0.9103	0.0157	1.0000	0.9843
Balanceado	0.4	0.5013	0.7762	0.6852	0.7853	0.1001
	0.5	0.5068	0.7704	0.6929	0.7782	0.0853
	0.6	0.5330	0.7489	0.7169	0.7522	0.0353

Cuadro 6.1: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento desbalanceado y balanceado. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$).

sobremuestro de la clase minoritaria. El problema fundamental encontrado a la hora enfrentar esta subsección de la experimentación es que existen infinidad de formas para aumentar los datos (desplazamientos, rotaciones, cambios de contraste, cambios de color, cambios de nitidez, introducción de ruido, *MixUp*, *GAN* ...), que pueden utilizarse por separado o combinadas, que pueden aplicarse sólo a las imágenes de una clase o ambas y que además la mayoría dependen de uno o más hiperparámetros propios (proporción de imágenes p transformadas, α_{MU} para *MixUp*, amplitud del giro para las rotaciones, cotas máximas y mínimas para desplazamientos y cambios de contraste, agresividad del cambio de nitidez ...). Además, el tiempo y esfuerzo computacional por época se dispara con el número de técnicas combinadas. Una clara futura mejora de este estudio sería la exploración de estas y otras técnicas de aumentos de datos de forma intensiva.

Considerando todo lo anterior, el enfoque seguido fue el siguiente. Primero, se exploraron por separado dos técnicas relativamente sencillas: ligeras traslaciones y rotaciones. Estas se escogieron por ser intuitivas y seguras, entendiendo como segura que la etiqueta de una imagen no cambia después de ser transformada. Un perito forense puede interpretar una OTP aunque esta esté girada o ligeramente desplazada, por lo que tiene sentido obligar al modelo a decidir en base a características invariantes a estas transformaciones. Después, se experimentó con la técnica *MixUp*. Esta se seleccionó por todo lo contrario a las dos anteriores, es poco intuitiva y poco segura. Difícilmente un perito forense podrá interpretar dos OTP mezcladas linealmente. Finalmente, se comprobó si tanto por separado como combinadas, el uso traslaciones, rotaciones y *MixUp* beneficiaba o empeoraba el rendimiento del modelo. Recordar que en todos los casos se utilizó una *ResNet18* con todos los pesos congelados, salvo los de la cabecera completamente conectada, y un conjunto de datos balanceado vía sobremuestreo de la clase minoritaria.

En otras palabras, en esta Subsección se presentan 5 nuevos candidatos a mejor combinación de hiperparámetros para *ResNet18*: uno que utiliza sólo rotaciones para aumentar los datos, otro que utiliza sólo traslaciones, otro que combina rotaciones y traslaciones, otro que utiliza sólo *MixUp* y otro que combina *MixUp* con la técnica de aumentos de datos sencilla que mejores resultados ofreció (rotaciones, traslaciones o rotaciones y traslaciones). Puede consultarse un resumen comparativo en el Cuadro 6.2. De este cuadro pueden extraerse las siguientes conclusiones. Por un lado, que en todos los casos se reduce el sobreajuste de la clase positiva, una de las razones que motivaron el uso de este tipo de técnicas. Por otro lado, la mejor forma obtenida para aumentar los datos fue combinar rotaciones y traslaciones, aplicando de forma aleatoria alguna de las dos con una probabilidad del $p = 0.5$.

DA	α_M	Loss	ACC	SEN	SPE	M_{CV}	O_{SEN}
Ninguna (6.1)	0.6	0.5330	0.7489	0.7169	0.7522	0.0353	0.1197
R (6.3)	0.5	0.5423	0.7446	0.7326	0.7458	0.0132	0.0061
T (6.4)	0.5	0.5344	0.7382	0.7483	0.7372	0.0111	0.0565
R y T (6.5)	0.5	0.5396	0.7475	0.7486	0.7475	0.0011	0.0111
<i>MixUp</i> (6.6)	0.6	0.5499	0.7310	0.7329	0.7309	0.0020	0.1025
M* y MU (6.6)	0.5	0.5447	0.7417	0.7403	0.7419	0.0016	0.0281

Cuadro 6.2: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado de distinta forma. DA = Técnica de aumento de datos. R = Rotaciones. T = Translaciones. R y T = Combinación de R y T. M* = Transformación que mejores resultados ofrece de entre R, T y R y T. MU = *MixUp*. Cada forma de aumentar los datos incluye los resultados del mejor candidato y una referencia a los resultados parciales. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

Rotaciones

Primero se exploró el uso de rotaciones. La técnica consistió en aplicar aleatoriamente rotaciones de 90, 180 y 270 grados a las imágenes del conjunto de entrenamiento. En la Figura 6.3 puede encontrarse un ejemplo de estas transformaciones sobre una misma imagen. Definido el posible ángulo de giro, existen 2 hiperparámetros más a fijar: la proporción de imágenes transformadas p y qué clase de imágenes deben transformarse l , aquellas pertenecientes a la clase positiva (1), a la negativa (0) o a ambas (0 y 1); y la proporción de datos aumentados se fijó en p . Así pues, se lanzaron varios procesos de validación cruzada para probar las siguientes combinaciones:

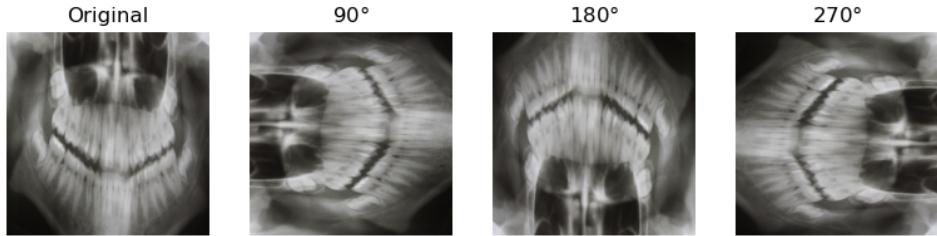


Figura 6.3: Aumento de datos vía rotación. Una misma imagen puede ser considerada como 4 imágenes distintas. Las imágenes aparecen re-escaladas pero sin normalizar para facilitar la interpretación.

$l = \{1\}$ y $p = 0.5$, $l = \{1\}$ y $p = 0.9$, $l = \{0, 1\}$ y $p = 0.9$ y $l = \{0, 1\}$ y $p = 0.5$. No se probó a aumentar sólo las imágenes de la clase negativa ($l = \{0\}$), ya que es innecesario. En la subsección anterior, se observó que esta clase apenas presentaba sobreajuste. El número de épocas utilizado fue 50 nuevamente. La tasa de aprendizaje, cuya optimalidad se comprobó vía *LR-Find*, también fue de $\lambda = 1 \cdot 10^{-4}$.

En el caso de aumentar sólo las imágenes de la clase positiva ($l = \{1\}$), se observó que el aumento de datos era muy perjudicial para el rendimiento del modelo sobre esa misma clase, aunque a priori fuesen las únicas que pareciesen necesitarlo por su mayor sobreajuste. Aumentó el sobreajuste sobre la clase positiva, el valor de *SEN* se desplomó para el conjunto de validación. Además, cuánto más agresivo fue el aumento de datos (mayor p), más grave fue el desplome. En la Figura 6.4 puede observarse este desplome para una de las iteraciones de la validación cruzada.

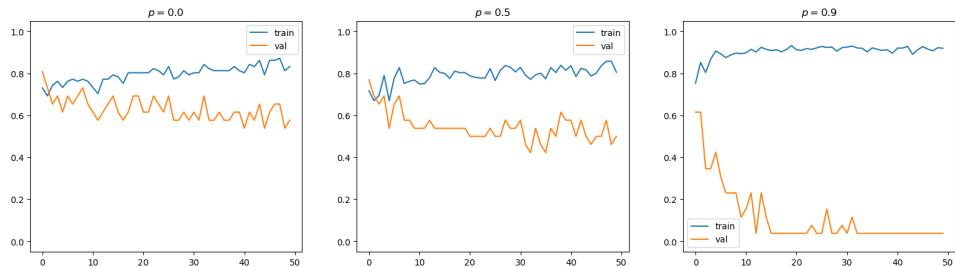


Figura 6.4: Curvas de entrenamiento y validación para *SEN* en función de p cuando sólo se aumentan imágenes de la clase positiva ($l = \{1\}$) vía rotaciones. Resultados correspondientes a una misma partición ($k = 2$).

En el caso de aplicar rotaciones sobre ambas clases ($l = \{0, 1\}$), se observó que, tanto $p = 0.5$ como $p = 0.9$, la sensibilidad no se desplomaba y el sobreajuste de la clase positiva se reducía en la mayoría de iteraciones de la validación cruzada. En la Figura 6.5, puede observarse que para casi todas

las iteraciones de la validación cruzada, el sobreajuste se reduce. Para comprobarlo de forma cuantitativa, se utilizaron los resultados de $O_{CV}(SEN)$, consultar Cuadro 6.3.

Por último, destacar que la forma de aumentar los datos que mejores resultados produce es aumentar ambas clases con una frecuencia del 50% ($p = 0.5$ y $l = \{0, 1\}$). En el Cuadro 6.3 pueden encontrarse los resultados para cada combinación de l y p evaluadas. Además, los resultados obtenidos de este mejor candidato son mejores que sin aumentar los datos.

α_M	l	p	$Loss$	ACC	SEN	SPE	M_{CV}	O_{SEN}
0.4	{1}	0.5	0.4735	0.7983	0.6148	0.8168	0.2021	0.1917
		0.9	0.4293	0.8644	0.2840	0.9226	0.6386	0.6126
	{0, 1}	0.5	0.5110	0.7747	0.6778	0.7845	0.1066	0.0639
		0.9	0.5400	0.7575	0.6455	0.7688	0.1232	0.0796
0.5	{1}	0.5	0.4735	0.7983	0.6148	0.8168	0.2021	0.1917
		0.9	0.5177	0.7675	0.4489	0.7995	0.3505	0.4086
	{0, 1}	0.5	0.5423	0.7446	0.7326	0.7458	0.0132	0.0061
		0.9	0.5400	0.7575	0.6455	0.7688	0.1232	0.0796
0.6	{1}	0.5	0.5048	0.7761	0.6385	0.7900	0.1516	0.1768
		0.9	0.6345	0.6614	0.5514	0.6724	0.1210	0.2347
	{0, 1}	0.5	0.5679	0.7102	0.7646	0.7049	0.0598	-0.0308
		0.9	0.6063	0.6513	0.7495	0.6418	0.1077	-0.0112

Cuadro 6.3: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía rotaciones de 90°, 180°, o 270°. La red se entrenó transformando sólo las imágenes de la clase positiva ($l = \{0\}$) o las imágenes de ambas clases ($l = \{0, 1\}$) con una frecuencia de $p = \{0.5, 0.9\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

Traslaciones

Experimentación análoga a la hecha para las rotaciones, pero en lugar de aplicar aleatoriamente rotaciones de 90, 180 y 270 grados a las imágenes del conjunto de entrenamiento, se aplican traslaciones en ambos ejes, de hasta un 20% de las dimensiones de la imagen original (ancho o largo). En la Figura 6.6 pueden observarse algunos ejemplos del resultado de esta transformación sobre una misma imagen. Se escogió límite del 20% porque la pérdida de información para un porcentaje mayor se consideró excesiva. Al desplazar una imagen, se pierde información porque la política para llenar los píxeles que quedaron sin valor fue llenar con píxeles negros. Aunque



Figura 6.5: Curvas de entrenamiento y validación para *SEN* en función de la proporción de imágenes transformadas (p) cuando se aumentan imágenes de ambas clases ($l = \{0, 1\}$) vía rotaciones de 90° , 180° o 270° . Para todas las iteraciones (k), el sobreajuste (área entre curva de entrenamiento y validación) se reduce significativamente (salvo para $k = 2$).

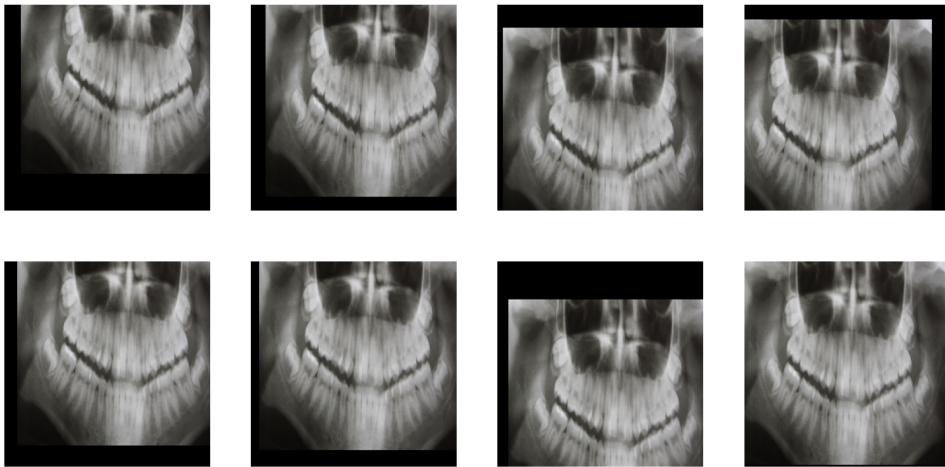


Figura 6.6: Aumento de datos vía desplazamiento. Magnitud del desplazamiento aleatoria hasta un máximo del 20% de las dimensiones de la imagen original. Política de relleno, píxeles negros. Las imágenes aparecen reescaladas pero sin normalizar para facilitar la interpretación.

no se exploraron, pueden aplicarse otras políticas de relleno.

Los resultados y conclusiones extraídos también fueron similares a los de las rotaciones. Si sólo se aumenta la clase positiva, el rendimiento del modelo se desploma para esta misma clase (*SEN*). Un desplome que aumenta con la proporción de imágenes aumentadas. Si se aumentan ambas clases, independientemente de p , el sobreajuste sobre la clase positiva se reduce ligeramente. En este caso no se han incluido las figuras porque sería redundante. Respecto a la mejor combinación de hiperparámetros encontrada, empataron dos candidatos: $\{p = 0.5, l = \{0, 1\}\}$ y $\{p = 0.9, l = \{0, 1\}\}$. Se escogió aquel con $p = 0.5$ porque la siguiente forma de aumentar datos explorada fue combinar ambas técnicas, y $p = 0.5$ ofrece mejores resultados también en el caso de la rotación. Al igual que en el caso de las rotaciones, esta forma de aumentar los datos genera mejores resultados que no utilizarla e incluso mejores resultados que las rotaciones.

Rotaciones y Traslaciones

Una vez observado que aumentar los datos a través de tanto rotaciones como traslaciones ofrecía mejores resultados que entrenar el modelo sin aumentar datos, se probó a combinar ambas técnicas. Existen dos formas de combinar dos tipos de transformaciones de datos: o bien aleatoriamente se aplican ambas técnicas sobre una misma imagen; o bien aleatoriamente se escoge una de las transformaciones para cada imagen. Se probaron ambos casos. En el caso de aplicar ambas transformaciones sobre una misma imagen,

α_M	l	p	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{SEN}
0.4	{1}	0.5	0.4425	0.8199	0.5658	0.8453	0.2795	0.2631
		0.9	0.4027	0.8737	0.4394	0.9171	0.4777	0.4539
	{0, 1}	0.5	0.4775	0.7934	0.6615	0.8066	0.1451	0.0857
		0.9	0.5118	0.7647	0.7166	0.7696	0.0529	0.0304
0.5	{1}	0.5	0.4875	0.7927	0.6129	0.8106	0.1977	0.2130
		0.9	0.4626	0.8228	0.5194	0.8532	0.3338	0.3569
	{0, 1}	0.5	0.5344	0.7382	0.7483	0.7372	0.0111	0.0565
		0.9	0.5202	0.7547	0.7323	0.7569	0.0246	0.0242
0.6	{1}	0.6	0.5660	0.7180	0.6849	0.7215	0.0366	0.1102
		0.9	0.4904	0.7991	0.5425	0.8247	0.2823	0.3279
	{0, 1}	0.5	0.5554	0.7224	0.7637	0.7182	0.0455	0.0492
		0.9	0.5433	0.7381	0.7483	0.7372	0.0111	0.0171

Cuadro 6.4: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía traslaciones de hasta el 20 %. La red se entrenó transformando sólo las imágenes de la clase positiva ($l = \{0\}$) o las imágenes de ambas clases ($l = \{0\}$) con una frecuencia de $p = \{0.5, 0.9\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

primero se rotó y después se desplazó. Respecto al valor del hiperparámetro l (qué clases aumentar), dado que el efecto de l es el mismo para translaciones y rotaciones por separado, se asumió que el efecto sería el mismo combinando ambas técnicas y por ello se descartó $l = \{0, 1\}$. Experimentación preliminar no documentada confirmó esta sospecha. No tenía sentido probar a aumentar sólo la clase positiva, ya que provoca que se desplome la sensibilidad del modelo. Respecto al valor del hiperparámetro p , como ya se ha comentado, se empleó un $p = 0.50$, ya que es el valor que ofrece el mejor resultado en ambos casos. Recuérdese que el límite de épocas fue 50 y la tasa de aprendizaje, cuya optimalidad se comprobó vía *LR-Find*, es de $\lambda = 1 \cdot 10^{-4}$.

Nuevamente, se observó el mismo fenómeno que en el caso de las translaciones y las rotaciones: se reduce ligeramente el sobreajuste para la clase positiva. Respecto a qué forma de combinar ambas transformaciones proporciona mejores resultados, no se observaron diferencias significativas, salvo que aplicar de forma combinada ambas transformaciones a una misma imagen duplicaba el tiempo de ejecución medio por época. Finalmente, en el Cuadro 6.5 puede observarse una comparativa de los resultados medios obtenidos. La mejor forma de combinar ambas técnicas es aplicando o una rotación o una traslación a cada imagen aleatoriamente.

Combinación	α_M	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{SEN}
R or T	0.4	0.5168	0.7726	0.7166	0.7782	0.0616	0.0618
	0.5	0.5396	0.7475	0.7486	0.7475	0.0011	0.0111
	0.6	0.5476	0.7381	0.7566	0.7364	0.0202	0.0206
R and T	0.40	0.5047	0.7711	0.6855	0.7798	0.0942	0.0720
	0.50	0.5452	0.7274	0.7486	0.7253	0.0233	0.0085
	0.60	0.5452	0.7274	0.7486	0.7253	0.0233	0.0085

Cuadro 6.5: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía rotaciones de 90° , 180° o 270° y traslaciones de hasta el 20 %. Para cada imagen transformada, o bien se escoge una de las transformaciones (R or T) o bien se combinan ambas (R and T). Proporción de imágenes transformadas $p = \{0.5\}$. Se aumentaron imágenes de ambas clases $l = \{0, 1\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

MixUp

Como ya se ha comentado, esta técnica se escogió por su peculiar y contra-intuitiva naturaleza. Recuérdese que *MixUp* consiste en mezclar linealmente dos instancias de datos (imágenes y etiquetas) para generar nuevos casos con los que entrena el modelo. La agresividad de esta mezcla depende del valor λ_{MU} , que a su vez se extrae de una distribución “beta” definida por el hiperparámetro α_{MU} . En la Figura 6.7 pueden encontrarse varios ejemplos de *MixUp* para dos mismas imágenes en función de λ_{MU} . En esta Subsección, primero se compararon los resultados de aplicar esta técnica para distintos valores de $\alpha_{MU} = \{0.01, 0.1, 1.0, 10, 100\}$. Se escogieron estos valores porque la distribución resultante varía notablemente de un caso a otro, compruébese en la Figura 4.6. A continuación, se probó a utilizar de forma conjunta *MixUp* y la técnica de aumento de datos que mejores resultados había proporcionado hasta el momento: rotaciones (de entre 90° , 180° o 270°) y traslaciones (de hasta un 20 %) aplicadas aleatoriamente con una probabilidad de $p = 0.5$ a imágenes de ambas clases $l = \{0, 1\}$ de forma no conjunta (R or T). Por simplicidad, se notó esta forma de aumentar los datos como M*. Se exploraron dos formas de combinarlas: primero rotando o trasladando las imágenes a mezclar y después aplicando *MixUp*; y primero mezclando las imágenes vía *MixUp* y después rotando o trasladando la imagen resultante.

En el caso de aplicar solamente *MixUp*, se observó que, en general, cuánto mayor es la agresividad del mezclado (mayor α_{MU}) peores resultados se obtienen en términos de *ACC*, *SEN* y *SPE*. No obstante, el mejor modelo encontrado supera el rendimiento en validación cruzada al candidato que

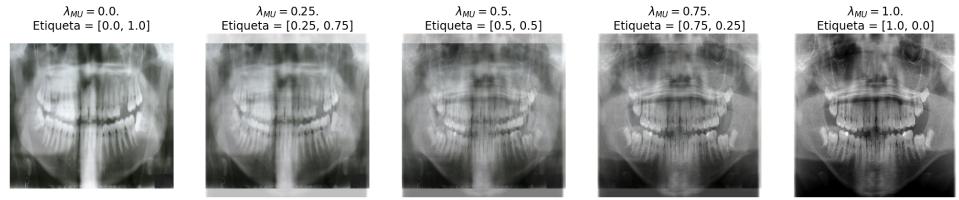


Figura 6.7: Ejemplo de *MixUP* aplicado sobre dos mismas imágenes de distinta etiqueta (o tiene por qué ser así) en función de λ_{MU} . Las imágenes y las etiquetas se obtienen combinando linealmente ambas imágenes o etiquetas en base al factor λ_{MU} . Las imágenes aparecen re-escaladas pero sin normalizar para facilitar la interpretación.

no utiliza ninguna técnica de aumento de datos. También fue reseñable la reducción del sobreajuste con el aumento de λ_{MU} , aunque este fue en detrimento del rendimiento del modelo, especialmente para la clase negativa (*SPE*). En el caso de aplicar *MixUp* junto a M^* , se observaron los mismo fenómenos, tanto como para aplicar primero M^* y después *MixUp* como en el caso contrario. Los Cuadros comparativos de los 3 experimentos pueden consultarse en el Apéndice, en esta Subsección sólo se incluyó el mejor de cada caso por legibilidad (45 casos).

DA	α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}
<i>MU</i> (8.1)	0.6	0.10	0.5499	0.7310	0.7329	0.7309	0.0020
$M^* + MU$ (8.2)	0.5	0.01	0.5346	0.7468	0.7486	0.7467	0.0019
<i>MU + M*</i> (8.3)	0.5	0.01	0.5447	0.7417	0.7403	0.7419	0.0016

Cuadro 6.6: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* y M^* con distintos valores para α_{MU} . DA = Técnica de aumento de datos utilizada. $M^* = M$ or T con $p = 0.5$ y $l = \{0, 1\}$. $MU = MixUp$. $MU + M^* =$ Primero *MixUp* y después M^* . $M^* + MU =$ Primero M^* y después *MixUp*. Cada forma de aumentar los datos incluye los resultados del mejor candidato y una referencia a los resultados parciales. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$).

Fine-Tuning

Una vez intentada la reducción del sobreajuste sobre la clase positiva vía aumento de datos, se exploró la posibilidad de mejorar el rendimiento vía *Fine Tuning*. Como ya se comentó en la Subsección 2.4, esta técnica consiste en permitir que durante el proceso de optimización se actualice el valor de parámetros pertenecientes a capas más profundas que los de la cabe-

cera completamente conectada, libración conocida como descongelamiento de parámetros o capas. La optimización de parte del extractor de características puede interpretarse como adaptar las características que aprende el modelo al problema concreto. Así pues, se procedió a entrenar y validar varios modelos basados en la arquitectura *ResNet18*, alterando únicamente los parámetros o capas descongeladas a cada vez.

En general, cuánto más profundo es un parámetro, menor debe ser la tasa de aprendizaje con la que se actualiza. Esto se debe a que, cuánto más profunda es una capa, más simples son las características que extrae o aprende, por lo que menos “responsables” son sus parámetros de lo bien o mal que rinde el modelo y menos deberían cambiar durante el proceso de optimización. No existe un criterio estandarizado sobre qué capas descongelar y cómo reducir la tasa de aprendizaje para cada una de estas. La política escogida fue la siguiente: descongelar hasta las 4 últimas capas de *ResNet18* y reducir la tasa de aprendizaje un orden de magnitud para las capas descongeladas (factor multiplicativo de reducción, $f_{FT} = 0.1$). Además, una capa i -ésima no se descongeló hasta pasadas p_{FT} épocas entrenando con la capa inmediatamente menos profunda descongelada; es decir, la penúltima capa no se descongela hasta entrenar durante p_{FT} épocas con la última capa descongelada, ídem para antepenúltima y penúltima. El número de capas descongeladas o profundidad de la congelación se controló a través de un nuevo hiperparámetro, d . El máximo de capas descongeladas fue 4 ($d = 4$) y el mínimo ninguna ($d = 0$). Volviendo a la Figura 4.8, se consideraron como las últimas 4 capas de *ResNet18*, las capas de $conv5_x$. En esta capa se aplican secuencialmente 4 filtros de $3 \times 3 \times 512$, con su correspondiente capa de normalización (estas también cuentan con parámetros entrenables). Si $d = i$, se descongeló hasta la i -ésima capa menos profunda de estas 4, junto a las capas de normalización correspondientes. Si $d = 0$, se descongeló sólo última capa. Si $d = 4$, se descongelaron todas las de $conv5_x$. El valor del resto de hiperparámetros se mantuvo similar al de los experimentos anteriores, a destacar: 50 épocas y tasa de aprendizaje de la última capa $\lambda = 1 \cdot 10^{-4}$.

Primero se aplicó *Fine Tuning* a una *ResNet18* que sólo utilizaba sobremuestreo (sin aumento de datos) con $p_{FT} = 10$ y $f_{FT} = 0.1$. Sólo se utilizó $d = \{1\}$. El objetivo era comprobar experimentalmente algo que ya se anticipó en esta memoria: la aplicación de *Fine Tuning* dispara el ya notable sobreajuste para la clase negativa (*SEN*), derivando en un peor rendimiento. En efecto, fue así. Puede observarse visualmente el aumento de sobreajuste en la Figura 6.8. Añadir que sólo se utilizó $d = \{1\}$ porque no tenía sentido seguir aumentando d , lo único que se consigue es aumentar y/o adelantar el sobreajuste del modelo. Estos resultados motivaron el uso de una técnica de regularización como el aumento de datos para intentar reducir este sobreajuste. Antes de continuar, uno pudiera pensar que con unos hiperparámetros distintos, la política escogida para fijar λ no provo-

caría estos resultados. Con $d = 1$, se comprobaron experimentalmente los efectos de p_{FT} y f_{FT} para una de las particiones de la validación cruzada ($k = 2$), consultar la Figura 6.9. Reduciendo o aumentando p_{FT} , lo único que se conseguía era adelantar o retrasar el punto de inflexión en el que se dispara el sobreajuste. Respecto a f , aumentándolo, se adelanta/incrementa el sobreajuste, y reduciéndolo, también se reduce el sobreajuste, pero en el mejor de los casos se obtienen resultados similares a los de no usar *Fine Tuning*.

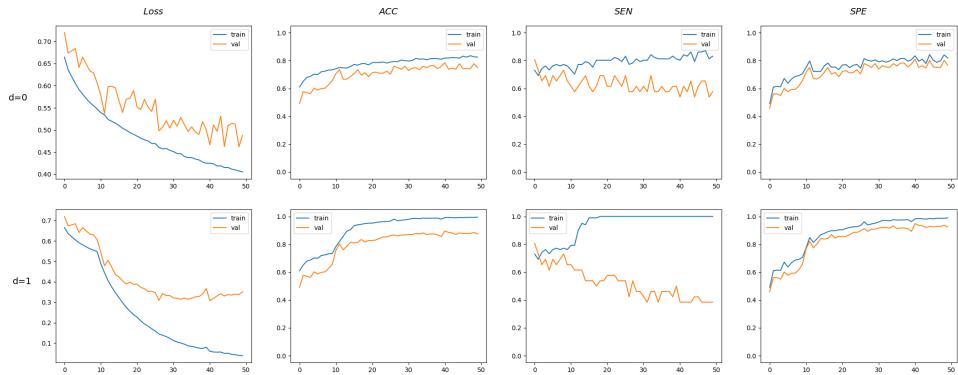
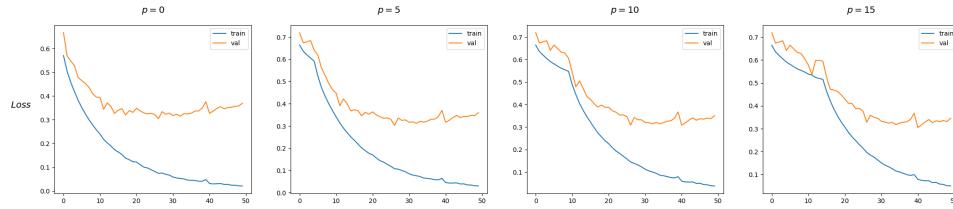


Figura 6.8: Curvas de entrenamiento y validación para una misma *ResNet18* que entrena con un conjunto de datos balanceado vía sobremuestreo y aplicando *Fine Tuning* con distinto d . Arriba, sólo están descongelados los pesos de la cabecera completamente conectada ($d = 0$). Abajo, también se descongela última del extractor de características ($d = 1$).

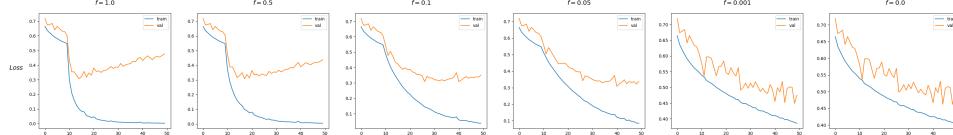
A fin de combatir ese sobreajuste, se aplicó *Fine Tuning* con $d = \{1, 2, 3, 4\}$, $p_{FT} = 5$ y $f_{FT} = 0.1$ sobre el candidato con mejores resultados que utilizaba aumento de datos hasta el momento: una *ResNet18* que además de sobremuestrear las imágenes de la clase positiva, aumentaba las imágenes de ambas clases rotándolas o trasladándolas de forma aleatoria, con una probabilidad del $p = 0.5$. Los resultados no mejoraron rendimiento de este mejor candidato, pueden consultarse en el Cuadro 6.7. En las curvas de aprendizaje puede observarse el mismo fenómeno que al aplicar *Fine Tuning* sin aumento de datos: el sobreajuste para la clase positiva se dispara cuántos más parámetros del modelo se descongelan (mayor d). En conclusión, *Fine Tuning* (con $p_{FT} = 5$, $f_{FT} = 0.1$ y $d = \{1, 2, 3, 4\}$) y aumentar los datos (R or T con $p = 0.5$ y $l = \{0, 1\}$) no reporta mejoras de rendimiento.

6.1.2. *ResNet50*

El objetivo de esta Subsección fue comprobar si utilizando un modelo más potente, *ResNet50*, se obtenían mejores resultados. Comprobar si la profundidad de la red jugaba un papel diferencial. Para ello, se repitió toda la



(a) El valor p_{FT} no evita el sobreajuste, solo marca el punto de inflexión en el que éste se dispara.



(b) El valor de f_{FT} modula el sobreajuste, pero en el mejor de los casos se obtiene un modelo igual a uno que no utilice *Fine Tuning* ($f_{FT} = 0.0$).

Figura 6.9: Curvas de entrenamiento y validación de la función pérdida para una misma *ResNet18* que entrena con un conjunto de datos balanceado vía sobremuestreo, sin aumento de datos y aplicando *Fine Tuning* con $d = 1$, $p_{FT} = \{0, 5, 10, 15\}$ y $f_{FT} = \{1.0, 0.5, 0.1, 0.01, 0.001, 0\}$. Resultados correspondientes a una misma iteración de la validación cruzada ($k = 2$).

experimentación relativa al sobremuestreo, aumento de datos y *Fine Tuning* para esta nueva arquitectura.

Primero, se entrenó y validó una *ResNet50* balanceando el conjunto de entrenamiento a cada iteración de la validación cruzada vía sobremuestreo de la clase negativa. Los resultados pueden consultarse en el Cuadro 8.4. Al igual que en el caso de *ResNet18*, el modelo presentaba un notable sobreajuste para la clase sobremuestreada (positiva), por lo que se procedió a probar las diferentes técnicas de aumento de datos exploradas en la subsección anterior: sólo rotaciones de entre 90° , 180° o 270° (R); sólo traslaciones de hasta un 20 % del tamaño original de la imagen (T); traslaciones y rotaciones, aplicadas sobre una misma imagen (T and R) o de forma desjunta, unas veces una y otras otra (T or R); *MixUp* y *MixUp* junto al mejor candidato de las técnicas anteriores, R or T con $l = \{0, 1\}$ y $p = 0.5$. Los resultados pueden consultarse en el Cuadro 6.8. A diferencia de la Subsección anterior, en el caso de aplicar traslaciones, no se probó a aumentar sólo las imágenes de la clase positiva, ya que los resultados para el aumento vía sólo rotaciones confirmaban que con *ResNet50* también se desploma el rendimiento para la clase positiva si sólo se aumentan las imágenes de esta clase. Todas las técnicas de aumento de datos reportaron una mejora significativa en el rendimiento, especialmente para la clase positiva, pero el mejor candidato es aquel que utiliza *MixUp* con $\alpha_{MU} = 0.1$ (primero) combiando con R or T con $l = \{0, 1\}$ y $p = 0.5$ (después). Finalmente, se experimentó aplicando

α_M	d	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{SEN}
0.5	0	0.5396	0.7475	0.7486	0.7475	0.0011	0.0111
0.4	1	0.3911	0.8393	0.6302	0.8603	0.2301	0.2744
	2	0.3292	0.8787	0.5828	0.9084	0.3256	0.3509
	3	0.3373	0.8679	0.6062	0.8942	0.2880	0.3351
	4	0.3348	0.8694	0.6062	0.8957	0.2896	0.3351
0.5	1	0.4306	0.8156	0.6618	0.8311	0.1693	0.2288
	2	0.3474	0.8629	0.6065	0.8887	0.2822	0.3219
	3	0.3373	0.8679	0.6062	0.8942	0.2880	0.3351
	4	0.3348	0.8694	0.6062	0.8957	0.2896	0.3351
0.6	1	0.5367	0.7338	0.7406	0.7332	0.0074	0.0838
	2	0.5015	0.7568	0.7252	0.7601	0.0349	0.1145
	3	0.5347	0.7324	0.7406	0.7316	0.0090	0.0885
	4	0.5347	0.7324	0.7406	0.7316	0.0090	0.0885

Cuadro 6.7: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía R or T con $p = 0.5$ y $p = \{0, 1\}$ y aplicando *Fine Tuning* con distinta profundidad (d), $p_{FT} = 5$, $f_{FT} = 0.1$ y $d = \{1, 2, 3, 4\}$. Se incluyen las métricas del candidato a mejorar ($d = 0$). Los datos se transformaron escogiendo una de las transformaciones (rotación o traslación) para cada imagen. Proporción de imágenes transformadas $p = \{0.5\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

Fine Tuning a este candidato más prometedor resultante de las pruebas con aumento de datos. La intención inicial era probar a descongelar hasta las últimas 4 capas ($d = \{1, 2, 3, 4\}$) con una política similar a la de la Subsección anterior ($p_{FT} = 5$ y $f_{FT} = 0.1$). Para *ResNet50*, esas últimas capas corresponden a 4 capas convolucionales que aplican filtros de $3 \times 3 \times 512$ (y sus respectivas capas de normalización). No obstante, los malos resultados para $d = 1$, denotaban que seguir entrenando, aumentando el número de capas descongeladas, carecía de sentido. Al igual que para *ResNet18*, descongelar la última capa del extracto de características, disparó el sobreajuste para la clase positiva, la *SEN* para el conjunto de validación se desplomó, mientras que la del conjunto de entrenamiento se disparaba.

DA	α_M	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	<i>M_{CV}</i>	<i>O_{SEN}</i>
Ninguna (8.4)	0.5	0.4883	0.7927	0.6852	0.8035	0.1182	0.2142
R (8.5)	0.6	0.5407	0.7590	0.7640	0.7585	0.0055	0.0857
T (8.6)	0.6	0.5098	0.7805	0.7403	0.7846	0.0442	0.0677
R or T (8.7)	0.5	0.5412	0.7611	0.7637	0.7608	0.0029	0.0944
R and T (8.8)	0.6	0.5430	0.7647	0.7560	0.7656	0.0096	0.0453
<i>MixUp</i> (8.9)	0.6	0.5371	0.7554	0.7557	0.7553	0.0004	0.1550
<i>MixUp + M*</i> (8.10)	0.5	0.5400	0.7640	0.7637	0.7640	0.0003	0.0899
<i>M* + MixUp</i> (8.11)	0.5	0.5733	0.7482	0.7486	0.7482	0.0004	0.0183

Cuadro 6.8: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado de distinta forma. DA = Técnica de aumento de datos. Ninguna = Sólo sobremuestreo. R = Rotaciones. T = Translaciones. R or T = Rotaciones o translaciones (de las imágenes aumentadas, algunas imágenes se rotan y otras se trasladan). R and T = Rotaciones y translaciones (toda imagen aumentada se rota y se traslada). M* = Transformación que mejores resultados ofrece de entre R, T, R or T y R and T = R or T (con $l = \{0, 1\}$ y $p = 0.5$). *MixUp + M** = Primero *MixUp* y después M*. M* + *MixUp* = Primero M* y después *MixUp*. Cada forma de aumentar los datos incluye los resultados del mejor candidato y una referencia a la tabla con los resultados parciales. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

6.2. Experimentación - Aprendizaje Automático Clásico

El objetivo de esta sección es ofrecer un marco comparativo para los resultados de DL. Dado que el tiempo de ejecución de los experimentos es mucho menor que para las redes neuronales, en lugar de ejecutar de realizar un análisis cuantitativo y cualitativo posterior a cada combinación de hiperparámetros e intentar diagnosticar lo ocurrido, se probaron todas las combinaciones de hiperparámetros propuestos y se escogió el mejor candidato (filosofía *Grid Search* [120]). No se aplicaron técnicas de aumento de datos. Sí que se experimentó con la posibilidad de balancear el conjunto de datos vía sobremuestreo. Añadir que se mantuvo la partición utilizada para entrenamiento-validación, así como las particiones utilizadas en cada iteración de la validación cruzada.

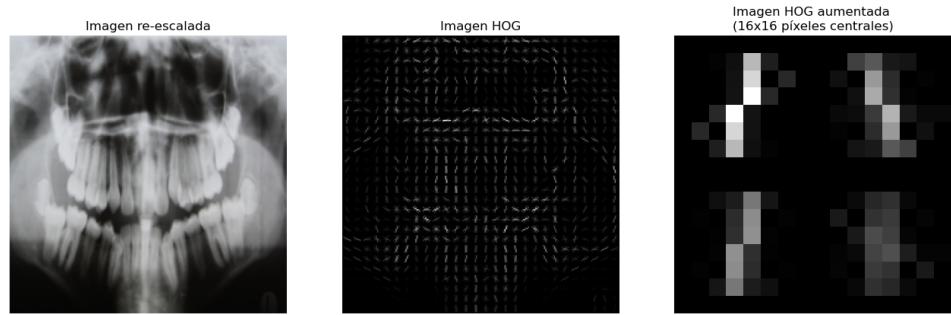


Figura 6.10: Imagen HOG sobre imagen del conjunto de entrenamiento. Reescalado a 224×224 para facilitar la comprensión del lector.

6.2.1. HOG

Recuérdese que la experimentación con modelos de ML clásico está dividida en dos etapas: extracción de características y entrenamiento del modelo. Todos los modelos estudiados en esta Subsección compartieron el mismo proceso de extracción de características, el algoritmo HOG. Como ya se justificó en la Subsección 4.2.6, se utilizó el valor por defecto de los hiperparámetros que definen este algoritmo: normalización por canales de las imágenes, reescalado a 128×64 , 8×8 para el tamaño de celda, $[0, 180]^\circ$ como rango de valores para la orientación, 9 contenedores por histograma, tamaño de 16×16 y “stride” de 8 para la ventana de normalización. Estos valores, resultaron en que cada imagen se transformó en un vector de 3.780 componentes antes de ser procesada por un modelo, ya sea durante entrenamiento o validación.

A la hora de visualizar el HOG de una imagen, en lugar de utilizar el vector de características, suele utilizarse la imagen HOG. Recuérdese que, durante la aplicación de HOG, se computa el histograma de gradientes de un grupo de píxeles conocido como celdas. En esta imagen, se representa visualmente el histograma de cada una esas celdas. Tiene el mismo tamaño que la imagen original, y sus píxeles se encuentran también agrupados por celdas. En cada celda se encuentran tantas líneas concéntricas como contenedores haya por histograma. Cada una de estas líneas representa un intervalo para la orientación del gradiente. Cuánto mayor sea el número de píxeles de la celda cuyo gradiente se encuentra en este intervalo y mayor sea su magnitud, mayor es la intensidad de la línea asociada. En la Figura 6.10 puede encontrarse un ejemplo de una imagen HOG para una de las imágenes utilizadas.

6.2.2. Regresión Logística

Para este tipo de modelos se utilizó como algoritmo de optimización una variante del descenso de gradiente conocida como *SAGA* [121], una tasa

de aprendizaje adaptativa y automática recomendada en la documentación de la biblioteca utilizada y un número máximo de iteraciones igual a 50 (aunque el entrenamiento puede detenerse antes si el proceso de optimización converge). Los hiperparámetros anteriores se mantuvieron fijos durante toda la experimentación. No obstante, sí se experimentó con los siguientes:

- Balanceo de los datos vía sobremuestreo.
- Tipo de regularización: $r = \{ \text{Ninguna}, L_1, L_2 \}$.
- Factor de regularización: $C = \frac{1}{\lambda_{LR}} = \{0.0001, 0.001, 0.01, 0.1, 1.0, 10.0\}$.
A mayor C , menor regularización y viceversa.

Primero se entrenó un regresor sin ningún tipo regularización con el conjunto de datos original. Al igual que en el caso de las redes convolucionales, el modelo interioriza el grave desbalance de los datos y aprende a clasificar casi todas las imágenes como positivas, así lo refleja su muy baja *SEN* y muy alta *SPE*. A continuación, se repitió el experimento para el conjunto de datos balanceado vía sobremuestreo de la clase minoritaria. En este caso, los resultados mejoraron, pero el rendimiento seguía siendo bastante malo. Finalmente, se probaron todas las combinaciones de C y r posibles (C no tiene sentido si $r = \text{Ninguna}$) utilizando sobremuestreo (dado que reportó mejores resultados en primera instancia). En Apéndice puede encontrarse un Cuadro 8.12 que recoge los resultados de toda la experimentación descrita. De los 4 mejores resultados, 3 de ellos utilizan regularización L_2 . No obstante, en base el criterio establecido, el mejor modelo encontrado utilizaba una regularización $r = L_1$ con un factor de $C = 0.1$.

6.2.3. SVM

Para este tipo de modelos, se experimentó con todas las combinaciones posibles de los siguientes hiperparámetros:

- Factor para la regularización L_2 : $C = \frac{1}{\lambda_{SVM}} = \{0.001, 0.01, 0.1, 1.0, 10.0\}$.
A mayor C , menor regularización y viceversa.
- Kernel lineal, polinomial, radial/gaussiano o sigmoide. $k = \{l, p, g, s\}$.
- Grado del kernel polinomial: $g = \{2, 5, 10\}$. $g = 0$ no se utiliza porque es equivalente a utilizar un kernel lineal.
- Factor γ , también conocido como σ en la bibliografía. Afecta a kernels polinómicos y radiales. Cuánto mayor sea mayor, mayor es la probabilidad de sobreajuste del modelo, genera fronteras de decisión más abruptas y complejas. Se utilizaron los valores recomendados por la

biblioteca utilizada: $\gamma = \left\{ \frac{1}{n_f}, \frac{1}{n_f * Var(X_{train})} \right\}$. Valores sugeridos por la biblioteca utilizada. Donde n_f es el número de características de una instancia de datos (3790). $Var(X_{train})$ es la varianza de las componentes de un vector resultado de concatenar las componentes de todas las instancias del conjunto de entrenamiento.

Al igual que en el caso anterior, primero se entrenó un modelo con los hiperparámetros por defecto ($C = 1.0$, kernel radial y $\gamma = \frac{1}{n_f * Var(X_{train})}$), utilizando y sin utilizar sobremuestreo de la clase minoritaria para balancear el conjunto de entrenamiento en cada iteración de la validación cruzada. En efecto, el sobremuestreo mejoró los resultados, por lo que se procedió a utilizarlo para el resto de la experimentación. A continuación, se procedió a ejecutar una batería experimental en la que se probaron todas las combinaciones posibles de valores de los hiperparámetros especificados. 55 candidatos. En el Apéndice puede encontrarse un Cuadro 8.14 con los resultados para los 20 modelos con mejor M_{CV} . El mejor modelo encontrado utilizaba un kernel sigmoide, una $\gamma = \frac{1}{n_f * Var(X_{train})}$ y un factor de regularización $C = 10$.

6.2.4. Random Forest

Para este tipo de modelos, los candidatos analizados utilizaban “bootstrapping”, es decir, cada árbol utilizaba tantas instancias como tuviese el conjunto de datos, pero las obtenía muestreando de forma aleatoria y con reemplazo el conjunto de entrenamiento. Esto resulta en que, de todas las instancias con las que entrena un árbol, aproximadamente dos tercios de ellas no estaban duplicadas. Respecto al resto de hiperparámetros, se probaron todas las combinaciones de los siguientes valores (el significado de cada uno de ellos puede revisitarse en la Subsección 4.2.6):

- Número de estimadores (n): 50 (por defecto) o 100.
- Criterio de pureza o separación (c): Gini (por defecto) o entropía cruzada [122].
- Número máximo de características a evaluar en cada nodo (max_f). Tomando como referencia el número de instancias del conjunto de datos ($n_f = 3.780$), se utilizó: $max_f = \sqrt{n_f}$ (valor por defecto) y $max_f = \log_2(n_f)$. Valores propuestos en la documentación de la biblioteca utilizada.
- Número de instancias mínimas para no considerar a un nodo hoja (n_{leaf}): 2 (valor por defecto, no hay hojas impuras), un 10 %, un 20 % o un 30 % del número de instancias del conjunto de entrenamiento (n_{train}). Sustituye a la profundidad máxima como criterio de parada.

Al igual que en los casos anteriores, primero se entrenó un modelo con los parámetros por defecto, utilizando y sin utilizar sobre|muestreo de la clase minoritaria. En efecto, el sobre|muestreo mejoró los resultados, por lo que se procedió a utilizarlo en el resto de la experimentación. A continuación se lanzó la batería experimental que probaba todas las combinaciones posibles de los hiperparámetros especificados. En el Apéndice puede consultarse un Cuadro 8.13 con todos los resultados. Se observó que el hiperparámetro n_{leaf} dominaba sobre el resto. Cuanto mayor era su valor, mejores resultados, sin excepción. El mejor modelo encontrado utilizaba $n = 100$, Gini como criterio de pureza, $\max_f = \log_2(n_f)$ y $n_{leaf} = 0.3 \cdot n_{train}$.

6.3. Comparativa Global

6.3.1. Validación

Modelo	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>
<i>ResNet18</i>	0.7475	0.7486	0.7475
<i>ResNet50</i>	0.7640	0.7637	0.7640
<i>LR</i>	0.7912	0.4486	0.8256
<i>SVM</i>	0.6636	0.6369	0.6661
<i>RF</i>	0.7267	0.5434	0.7451
<i>Dummy</i>	0.5064	0.4966	0.5075

Cuadro 6.9: Resultados de los mejores candidatos sobre validación cruzada. Se incluye el mejor candidato para cada familia de modelos: *ResNet18*, *ResNet50*, *LR*, *SVM* y *RF*. También se incluyó un *Dummy*, un modelo que predice etiquetas de forma totalmente aleatoria. Señalado el considerado como mejor modelo.

En el Cuadro 6.9 se recopiló el rendimiento en validación cruzada para los mejores candidatos de cada familia de modelos, así como los resultados para un clasificador que responde de forma aleatoria, un *Dummy*. Sólo se incluyó *ACC*, *SEN* y *SPE* porque el número de candidatos es suficiente bajo como para poder hacer un análisis sin recurrir a un criterio automático de evaluación. Por un lado, todos los modelos aparentan aprender algo, así lo demuestra que el rendimiento siempre estuviese por encima de los del *Dummy* aleatorio. Por otro lado, los modelos basados en DL obtuvieron mejores resultados que los basados ML clásico. El perseguidor más cercano, el candidato de *SVM*, se encontraba a una distancia de aproximadamente 10 puntos porcentuales en cada métrica. Por último, el candidato de *ResNet50* fue el modelo con mejor rendimiento en validación cruzada, presenta el mejor valor para todas las métricas, salvo para la *ACC*, donde *LR* logra el valor

máximo. No obstante, la baja *SEN* de *LR* denota que el alto valor de *ACC* es engañoso, por lo que no debe tenerse en cuenta.

6.3.2. Test

Modelo	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>
<i>ResNet18</i>	0.7241	0.7126	0.7356
<i>ResNet50</i>	0.7414	0.7644	0.7184
<i>LR</i>	0.6695	0.5287	0.8103
<i>SVM</i>	0.6580	0.4828	0.8333
<i>RF</i>	0.6379	0.5345	0.7414
<i>Dummy</i>	0.4626	0.4828	0.4425
<i>Humano</i>	0.9004	0.8080	0.9588
<i>C</i>	0.9230	0.9260	0.9190
<i>B</i>	0.8750	0.8700	0.8790

Cuadro 6.10: Resultado de mejores modelos sobre test. Se incluyen también los resultados del Estado del Arte para un experto humano que trabaja con la misma población (sudafricanos negros) [77], y para dos clasificadores automáticos que utilizan materiales y métodos similares, pero que se aplican sobre poblaciones distintas, chinos (*C*) [89] y brasileños (*B*) [90]. Señalado el candidato de este proyecto con mejores resultados.

A continuación, para seleccionar el mejor modelo y comparar los modelos implementados con los del Estado del Arte, se procedió a evaluar el rendimiento del mejor candidato de cada familia con el conjunto de test (previo entrenamiento con todo el conjunto entrenamiento-validación). Pueden encontrarse en el Cuadro 6.10. A priori, esta práctica (elegir el mejor modelo en base al conjunto de test), podría considerarse errónea. Por norma general, ninguna información del conjunto de test debe condicionar la elección del mejor modelo, incluido el rendimiento del predictor sobre él. Esto provoca que el mejor modelo esté sobre-ajustado al conjunto de test. No obstante, una vez fijados los hiperparámetros del mejor candidato para cada familia de modelos a través de los resultados en validación, suele considerarse excesivo no utilizar los resultados sobre el conjunto de test para elegir el mejor de ellos. El argumento que se esgrime es que si el conjunto de test es lo suficientemente representativo de la realidad, no hay problema alguno con que el mejor modelo esté sobreajustado a él. De hecho, es deseable. En este proyecto se asumió esta premisa. De esta forma, los resultados sobre test se utilizaron para escoger el mejor modelo y compararlo con los resultados de referencia recogidos en el Estado del Arte: expertos humanos que realizan la misma tarea sobre la misma población estudiada, sudafricanos negros [77]; y dos clasificadores automáticos que utilizan similares materiales (OPG) y

métodos (CNN) a los de este proyecto, pero que se entrenan y evalúan sobre una población distinta, brasileños [90] y chinos [89].

Por un lado, *ResNet50* siguió considerándose como el mejor modelo implementado. Tiene peor *SPE* que *ResNet18*, pero bastante mejor *SEN*, tanto que la media aritmética de ambas es más alta. También la *ACC* de *ResNet50* es mejor que la de *ResNet18*. Por otro lado, el rendimiento de los modelos ML clásicos siguió muy por debajo de los modelos basados en DL. Finalmente, respecto a la comparativa con los resultados del Estado del Arte, los resultados del mejor modelo implementado (*ResNet50*) son peores que los de cualquiera de los estudios de referencia. El rendimiento humano se encuentra muy alejado. Sin embargo, a favor del modelo implementado, también debe tenerse en cuenta que sus predicciones son instantáneas y perfectamente reproducibles, no existe error inter o intra-observador. El rendimiento de los clasificadores automáticos (*C* y *B*) también se encuentra alejado. No obstante, merece la pena mencionar que en ambos estudios se trabaja con un conjunto de datos mucho más grande y balanceado. En el caso de *C*, se utilizan 10257 imágenes con un 64.89 % de menores y un 55.11 % de mayores de edad. En el caso de *B*, se utilizan 4003 imágenes, no se explica la proporción de mayores y menores, pero se indica que las edades estaban repartidas de forma homogénea entre los 8 y los 22.9. Además, en ningún punto del artículo se hace referencia al balanceo de los datos, lo que probablemente implique que no existía. Esto es una ventaja bastante significativa. Recuérdese que los modelos DL implementados se encontraban limitados por la falta de casos y variedad para la clase positiva (clase minoritaria). Durante los experimentos, el sobreajuste, y con ello el fin de la mejora durante el entrenamiento, ocurría siempre para la clase minoritaria *SEN*. Como suele decirse: “un modelo es tan bueno como los datos con los que se alimenta”.

Existen otras diferencias entre los materiales y métodos de este proyecto y *B* y *C*, como que exploran otras arquitecturas o que en el caso de *C*, los modelos no trabajan con la OPG completa, sólo con la sección inferior izquierda (si se divide en 4 partes iguales), pero probablemente no sean tan significativas. En cualquier aso, es algo que podría comprobarse en trabajos futuros.

Merece la pena detenerse a discutir las discrepancias entre los resultados de validación y test. En el caso de los candidatos de DL, son razonables. Son poco significativas y conducen a la misma conclusión, que *ResNet50* es el mejor modelo implementado. Esto denota que las decisiones tomadas durante el proceso experimental fueron adecuadas y que el protocolo de validación es una estimación bastante cercana al rendimiento real (test). Además, se cumple el motivo por el que reservar un conjunto de test era necesario en primera instancia. Los resultados del protocolo de validación son una esti-

mación más optimista que el rendimiento real (sobre test). Tanto *ResNet50* como *ResNet18* presentan peores resultados sobre test. Recuérdese que esto se debe a que durante la experimentación, se ajusta el valor de los hiperparámetros para maximizar específicamente los resultados del protocolo de validación. En el caso de los candidatos de ML, las diferencias son más significativas, pero también razonables e irrelevantes. Se deben a que no modelan con precisión suficiente la realidad que el conjunto de validación y test representan. Sus decisiones son demasiado arbitrarias. Su bajo rendimiento en ambos casos (validación y test) y cercanía al caso *Dummy*, respalda esta afirmación.

6.3.3. Explicabilidad

A parte de su rendimiento, una de las grandes críticas que puede hacerse al mejor modelo encontrado (*ResNet50*) es que se comporta como una “caja negra”. Recibe una OPG y predice si pertenece a un menor o a un adulto sin ningún tipo de explicación que justifique el acierto o el error. Por contra, un experto humano puede justificar el porqué de sus predicciones. Para aportar explicabilidad a las predicciones del mejor candidato, se utilizó el método *GradCAM*. Este método permite visualizar las regiones de una imagen que más afectan a la predicción del modelo. Pueden consultarse más detalles en la Subsección 4.2.5.

Para comprobar la utilidad de esta funcionalidad, se tomó el clasificador encontrado, se entrenó para todo el conjunto de entrenamiento-validación y se aplicó *GradCAM* para varias imágenes seleccionadas aleatoriamente del conjunto de test. En esta memoria se incluyen algunos de estos casos. La adaptación utilizada de *GradCAM* genera, para cada imagen, dos filtros: $G_{p=1}$ y $G_{p=0}$. $G_{p=1}$ señala con píxeles de mayor intensidad las regiones de la imagen que empujan al modelo a predecir que el individuo es mayor de edad (amarillo, en este caso) y con tonos más oscuros, las regiones que lo empujan a lo contrario (violeta). $G_{p=0}$ señala lo mismo, pero invirtiendo las intensidades, con píxeles de mayor intensidad, las regiones que empujan al modelo a predecir menoría de edad, y con tonos más oscuros, las regiones que lo empujan a predecir mayoría de edad. Ambos filtros son el mismo, pero invertido.

Para algunos de los casos visualizados, el modelo parecía fijarse en características relevantes, véanse las Figuras 6.11 y 6.12. Sin embargo, en otras ocasiones, las regiones más influyentes coincidían con regiones irrelevantes, como anotaciones escritas sobre la radiografía, véanse las Figuras 6.13 y 6.14. Esto fue de utilidad para concluir que debería aplicarse algún tipo de preprocesado para eliminar estas anotaciones. En cualquier caso, el verdadero potencial de esta herramienta no es que un ingeniero informático compruebe

si está valorando las regiones anatómicas correspondientes, si no que lo haga un experto forense.

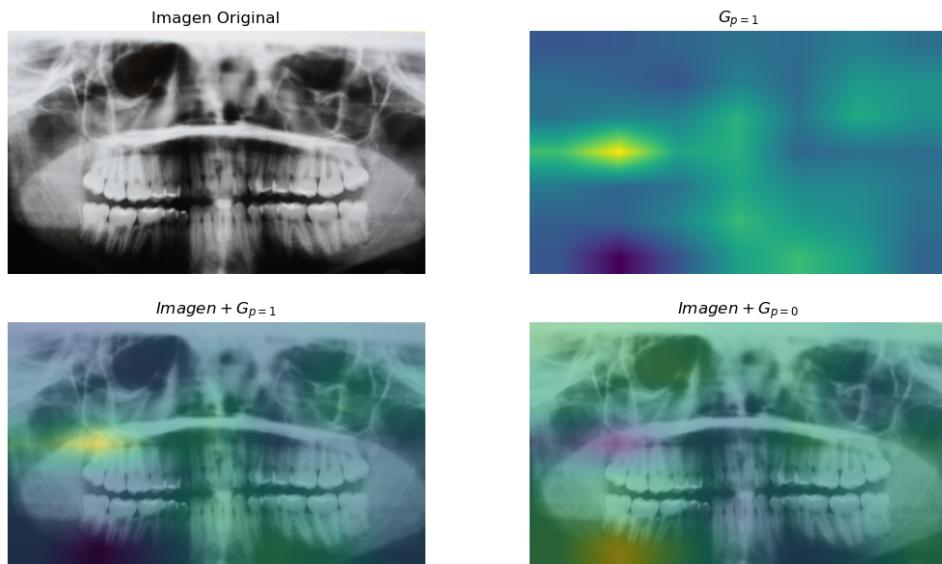


Figura 6.11: Resultados de *GradCAM* para un verdadero positivo. Edad del individuo = 19.49 años. Etiqueta predicha = 1 (con una probabilidad de $p = 0.73$). Filtro de mayor interés = $G_{p=1}$. Este caso se ha incluido porque sugiere que el modelo se centra en el tercer molar superior izquierdo para predecir la mayoría de edad.

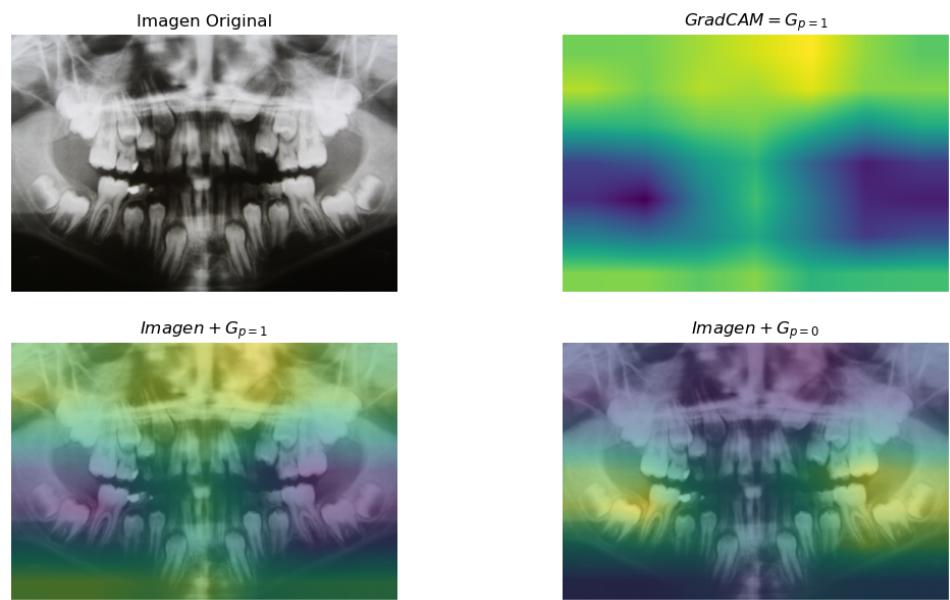


Figura 6.12: Resultados de *GradCAM* para un verdadero negativo. Edad del individuo = 9.95 años. Etiqueta predicha = 0 (con una probabilidad de $p = 0.90$). Filtro de mayor interés = $G_{p=0}$. Este caso se ha incluido porque sugiere que el modelo se centra en el prematuro estado de desarrollo de los molares de ambos lados de la boca. Un atributo realista.

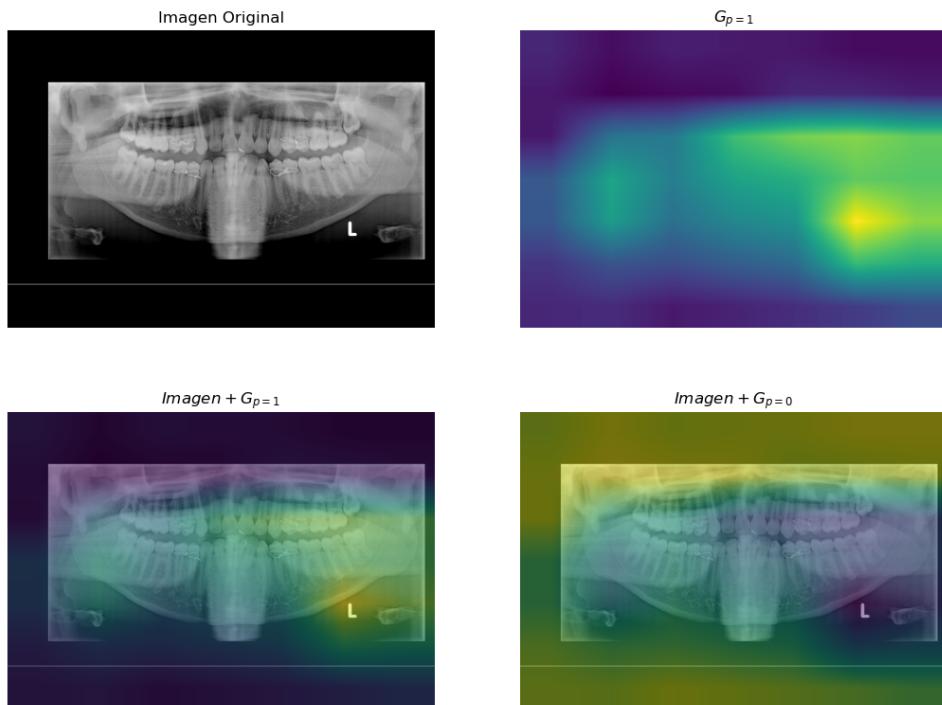


Figura 6.13: Resultados de *GradCAM* para un verdadero positivo. Edad del individuo = 20.65 años. Etiqueta predicha = 1 (con una probabilidad de $p = 0.94$). Filtro de mayor interés = $G_{p=1}$. Este caso se ha incluido porque la región de la imagen que más influye a la predicción del individuo corresponde a una anotación en la radiografía.

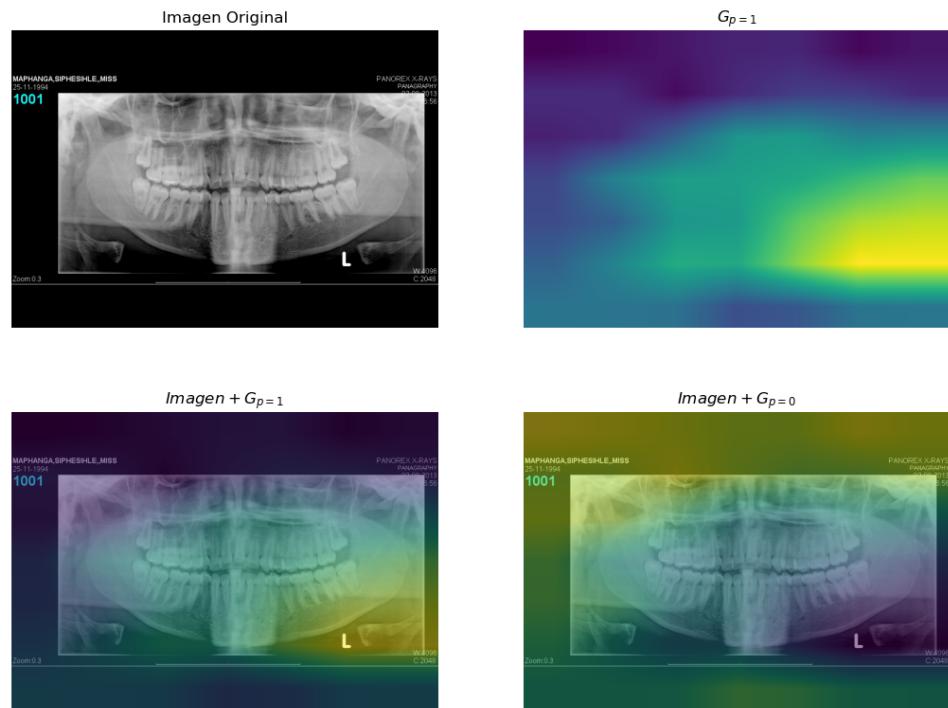


Figura 6.14: Resultados de *GradCAM* para un verdadero positivo. Edad del individuo = 20.24 años. Etiqueta predicha = 1 (con una probabilidad de $p = 0.97$). Filtro de mayor interés = $G_{p=1}$. Este caso se ha incluido porque la región de la imagen que más influye a la predicción del individuo vuelve a ser la misma anotación en la radiografía que para 6.13. No ocurría en todos los casos verdaderamente positivos, pero sin en los suficientes como para considerarlo un problema. Sugiere que tal vez exista un sesgo en los datos, la anotación “L” aparece mayoritariamente en el caso de los mayores de edad.

Capítulo 7

Conclusiones

La estimación de la edad legal a partir de los dientes de un individuo es uno de los problemas que mayor interés suscita en el ámbito de la antropología forense. En la mayoría de marcos legales del mundo, un menor y un mayor de edad no tienen los mismos derechos y deberes. Esta diferenciación es especialmente importante en el caso de migrantes indocumentados en busca de asilo. El análisis del desarrollo óseo de su dentadura puede determinar si su petición es aceptada o rechazada. A pesar de su especial relevancia, los métodos habitualmente utilizados en este contexto han demostrado ser imprecisos y subjetivos. Este TFG intenta aportar una solución automática y empírica que solvete dichas carencias.

Este proyecto presenta un método automático, basado en técnicas de DL, que estima la edad legal de un individuo a partir de una OPG. Se trata de una solución novedosa. No es la primera vez que se aborda el problema de la edad de esta forma, pero sí es la primera vez que se intenta resolver para individuos sudafricanos negros. A pesar de que la raza y la región de origen han demostrado ser factores diferenciales en esta tarea, no se encontró ninguna publicación previa de esta índole.

Los resultados de este proyecto demuestran que, de los candidatos explorados, los modelos basados en DL (*ResNet18*, *ResNet50*) son mejores que aquellos basados en técnicas clásicas de ML (*LR*, *SVM* y *Random Forest*). Sin embargo, ni siquiera el mejor candidato, un modelo basado en *ResNet50* (con $ACC \approx 0.74$, $SEN \approx 0.76$ y $SPE \approx 0.72$), alcanza el rendimiento de expertos humanos para el mismo grupo poblacional ($ACC \approx 0.90$, $SEN \approx 0.81$ y $SPE \approx 0.96$) [77] o de soluciones similares propuestas para otros grupos poblacionales. En [89], se alcanza una $ACC \approx 0.92$, $SEN \approx 0.93$ y $SPE \approx 0.92$ para una muestra de origen chino y en [90] se alcanza una $ACC \approx 0.87$, $SEN \approx 0.88$ y $SPE \approx 0.87$ para una muestra de origen mayoritariamente brasileño. No obstante, estas soluciones parten con una ventaja bastante significativa, utilizan una muestra de datos mucho

mayor y mucho más equilibrada o balanceada (no hay diferencias significativas entre el número de mayores y menores de edad).

El alto grado de desbalanceo de los datos disponibles fue el factor que más limitó el rendimiento de los modelos. Se contaba con 1.741 imágenes, de las cuales sólo 301 (17.28 %) pertenecían a individuos mayores de edad. Este fenómeno demostró ser muy perjudicial para el rendimiento de los modelos. Para paliar sus efectos, el sobremuestreo resultó ser una técnica vital. Sin aplicarlo, todos los modelos explorados, desde el más sencillo (*LR*) hasta el más complejo (*ResNet50*), aprendían a predecir siempre que la imagen procesada pertenecía a un menor de edad, interiorizaban el sesgo de los datos. Resulta sorprendente que algo tan sencillo como duplicar las imágenes de los menores de edad hasta tener tantas como mayores de edad, aportase las mejoras que se observaron. Durante la experimentación con modelos DL, todas las técnicas de aumento de datos exploradas (rotaciones, traslaciones, *MixUp* y combinaciones de estas) demostraron ser un complemento efectivo para seguir combatiendo el desbalanceo y encontrar un mejor modelo. No obstante, la mejora no era tan significativa como en el caso de aplicar sobremuestreo o no. Por desgracia, ni siquiera la acción conjunta de ambas técnicas logró evitar el sobreajuste de la clase minoritaria, dado el alto grado de sobreajuste y, por ende, su falta de representatividad. Añadir que también se exploró la posibilidad de aplicar *Fine-Tuning*, pero no aportó ningún tipo de beneficio. Lo único que conseguía era adelantar y aumentar el sobreajuste de la clase minoritaria.

Gracias a una versión adaptada del método *GradCAM*, el clasificador propuesto no se comporta como una “caja negra”. Además de la predicción, el modelo puede generar un mapa de activación que permite localizar las regiones de una OPG que más afectan a la edad legal que el modelo predice para ella. De esta forma, un experto forense puede valorar si la predicción es fiable o incluso aprender de las nuevas características que el modelo ha utilizado. En nuestro caso, fue útil para identificar posibles sesgos en los datos, varias predicciones utilizaban anotaciones escritas sobre las imágenes para realizar su predicción.

Para futuras investigaciones, se propone: pre-procesar los datos para evitar sesgos como los identificados gracias a *GradCAM*, explorar el uso de otras arquitecturas CNN que se emplean en los métodos del Estado del Arte para otros grupos poblacionales (*DenseNet*, *ResNet101* y *EfficientNet*) y entrenar los modelos únicamente con sectores de la imagen, como también se hace en uno de los métodos de referencia. Los resultados de estas dos últimas propuestas confirmarían o desmentirían que el desbalanceo de los datos es lo que limita a los clasificadores de este proyecto, ya que no existirían más diferencias a nivel metodológico entre este TFG y los métodos del Estado del Arte. En caso de confirmarlo, existirían dos posibilidades para continuar

a futuro: o recopilar más datos o reenfocar el problema. En lugar de intentar resolverlo como un problema de clasificación, plantearlo como una detección de anomalías.

Recapitulando, los objetivos del proyecto se lograron de forma satisfactoria. Se revisó la literatura relacionada con el problema de forma intensiva para encontrar un marco comparativo que, además, inspirase la forma de resolverlo. Se demostró que, para esta tarea, los modelos basados en DL son mejores que los modelos basados en técnicas clásicas de ML. Se descubrió que el sobremuestreo y el aumento de datos son técnicas útiles para combatir el desbalanceo de los datos y obtener mejores clasificadores. Se implementó un marco software experimental que permitió la realización de todos los experimentos de forma eficiente (puede encontrarse en <https://github.com/Palenchuekla/TFG>). Finalmente, se presentó un método automático, basado en DL, para determinar la edad legal de un individuo a partir de una OPG, que además justifica su respuesta. Puede que el modelo no exhiba el rendimiento de los métodos del Estado del Arte, pero las posibles fuentes del limitado rendimiento están localizadas, y sus predicciones pueden ser evaluadas por un experto.

Capítulo 8

Apéndice

Para facilitar la lectura y comprensión del capítulo dedicado a la experimentación (6), algunos cuadros de resultados se desplazaron a esta sección complementaria. El pie de cada uno informa del experimento al que corresponden.

α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	<i>M_{CV}</i>	<i>O_{CV}</i>
0.4	0.0100	0.5072	0.7647	0.6772	0.7734	0.0962	0.1688
	0.1000	0.5222	0.7618	0.6929	0.7687	0.0758	0.1396
	1.0000	0.5663	0.7418	0.7012	0.7459	0.0447	0.0984
	10.0000	0.6118	0.6880	0.7243	0.6843	0.0400	0.0090
	100.0000	0.6073	0.6959	0.6929	0.6961	0.0032	0.0116
0.5	0.0100	0.5533	0.7260	0.7326	0.7254	0.0072	0.1239
	0.1000	0.5453	0.7396	0.7249	0.7412	0.0162	0.0908
	1.0000	0.5864	0.7181	0.7332	0.7168	0.0165	0.0430
	10.0000	0.6443	0.6456	0.7877	0.6314	0.1563	-0.0493
	100.0000	0.6315	0.6586	0.7560	0.6488	0.1072	-0.0405
0.6	0.0100	0.5533	0.7260	0.7326	0.7254	0.0072	0.1239
	0.1000	0.5499	0.7310	0.7329	0.7309	0.0020	0.1025
	1.0000	0.5864	0.7181	0.7332	0.7168	0.0165	0.0430
	10.0000	0.6637	0.6155	0.8185	0.5950	0.2234	-0.0546
	100.0000	0.6545	0.6228	0.7868	0.6061	0.1806	-0.0553

Cuadro 8.1: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* con distintos valores para α_{MU} . Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{CV}
0.4	0.0100	0.5169	0.7676	0.7166	0.7727	0.0561	0.0435
	0.1000	0.5240	0.7654	0.7092	0.7711	0.0619	0.0502
	1.0000	0.5540	0.7446	0.7246	0.7467	0.0221	0.0225
	10.0000	0.6206	0.6650	0.7803	0.6536	0.1267	-0.0422
	100.0000	0.6248	0.6593	0.7723	0.6480	0.1243	-0.0679
0.5	0.0100	0.5346	0.7468	0.7486	0.7467	0.0019	0.0184
	0.1000	0.5432	0.7468	0.7403	0.7474	0.0071	0.0288
	1.0000	0.5613	0.7382	0.7326	0.7388	0.0062	0.0084
	10.0000	0.6302	0.6543	0.7957	0.6401	0.1556	-0.0533
	100.0000	0.6286	0.6543	0.7803	0.6417	0.1386	-0.0377
0.6	0.0100	0.5489	0.7288	0.7646	0.7254	0.0393	0.0166
	0.1000	0.5432	0.7468	0.7403	0.7474	0.0071	0.0288
	1.0000	0.5846	0.7066	0.7637	0.7009	0.0628	0.0021
	10.0000	0.6302	0.6543	0.7957	0.6401	0.1556	-0.0533
	100.0000	0.6459	0.6277	0.8034	0.6101	0.1933	-0.0319

Cuadro 8.2: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* y M^* con distintos valores para α_{MU} . $M^* = R$ or T con $l = \{0, 1\}$ y $p = 0.5$. Primero se aplica M^* y después *MixUp*. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{CV}
0.4	0.0100	0.5233	0.7582	0.7172	0.7624	0.0452	0.0379
	0.1000	0.5289	0.7597	0.7246	0.7632	0.0386	0.0411
	1.0000	0.5577	0.7554	0.7009	0.7609	0.0599	0.0105
	10.0000	0.6217	0.6787	0.7560	0.6709	0.0851	-0.0489
	100.0000	0.6226	0.6801	0.7246	0.6757	0.0490	-0.0291
0.5	0.0100	0.5447	0.7417	0.7403	0.7419	0.0016	0.0281
	0.1000	0.5289	0.7597	0.7246	0.7632	0.0386	0.0411
	1.0000	0.5630	0.7468	0.7163	0.7498	0.0335	0.0070
	10.0000	0.6362	0.6579	0.7880	0.6449	0.1431	-0.0642
	100.0000	0.6457	0.6298	0.8040	0.6125	0.1915	-0.0770
0.6	0.0100	0.5645	0.7188	0.7643	0.7143	0.0500	-0.0011
	0.1000	0.5468	0.7432	0.7406	0.7435	0.0029	0.0239
	1.0000	0.6124	0.6600	0.7963	0.6466	0.1498	-0.0400
	10.0000	0.6466	0.6356	0.8120	0.6181	0.1939	-0.0817
	100.0000	0.6486	0.6205	0.8117	0.6015	0.2102	-0.0733

Cuadro 8.3: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet18* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* y M^* con distintos valores para α_{MU} . $M^* = R$ or T con $l = \{0, 1\}$ y $p = 0.5$. Primero se aplica *MixUp* y después M^* . Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{CV}
0.4	0.4765	0.8070	0.6618	0.8216	0.1598	0.1922
0.5	0.4883	0.7927	0.6852	0.8035	0.1182	0.2142
0.6	0.4883	0.7927	0.6852	0.8035	0.1182	0.2142

Cuadro 8.4: Resultados cuantitativos en validación cruzada de una *ResNet50* que entrena con un conjunto balanceado vía sobremuestreo de la clase minoritaria. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$).

α_M	l	p	$Loss$	ACC	SEN	SPE	M_{CV}	O_{SEN}
0.4	{1}	0.5	0.4450	0.8328	0.6148	0.8547	0.2400	0.2343
		0.9	0.4355	0.8543	0.3308	0.9068	0.5760	0.5885
	{0, 1}	0.5	0.5234	0.7870	0.7400	0.7916	0.0516	0.0558
		0.9	0.4591	0.8300	0.6135	0.8516	0.2381	0.1669
0.5	{1}	0.5	0.4450	0.8328	0.6148	0.8547	0.2400	0.2343
		0.9	0.4355	0.8543	0.3308	0.9068	0.5760	0.5885
	{0, 1}	0.5	0.5234	0.7870	0.7400	0.7916	0.0516	0.0558
		0.9	0.5071	0.7884	0.6769	0.7995	0.1226	0.1418
0.6	{1}	0.5	0.4611	0.8242	0.6225	0.8445	0.2220	0.2371
		0.9	0.4597	0.8465	0.3388	0.8973	0.5586	0.5766
	{0, 1}	0.5	0.5407	0.7590	0.7640	0.7585	0.0055	0.0857
		0.9	0.5071	0.7884	0.6769	0.7995	0.1226	0.1418

Cuadro 8.5: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía rotaciones de 90° , 180° , o 270° . La red se entrenó transformando sólo las imágenes de la clase positiva ($l = \{0\}$) o las imágenes de ambas clases ($l = \{0, 1\}$) con una frecuencia de $p = \{0.5, 0.9\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	l	p	$Loss$	ACC	SEN	SPE	M_{CV}	O_{SEN}
0.4	{0, 1}	0.5	0.4959	0.7934	0.7166	0.8011	0.0845	0.0920
		0.9	0.4967	0.7920	0.7243	0.7987	0.0744	0.0675
0.5	{0, 1}	0.5	0.4959	0.7934	0.7166	0.8011	0.0845	0.0920
		0.9	0.5065	0.7877	0.7323	0.7932	0.0609	0.0540
0.6	{0, 1}	0.5	0.5141	0.7748	0.7320	0.7790	0.0470	0.1011
		0.9	0.5098	0.7805	0.7403	0.7846	0.0442	0.0677

Cuadro 8.6: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía traslaciones de hasta un 20 % del tamaño de la imagen original. La red se entrenó transformando imágenes de ambas clases ($l = \{0, 1\}$) con una frecuencia de $p = \{0.5, 0.9\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	l	p	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	<i>M_{CV}</i>	<i>O_{SEN}</i>
0.4	{0, 1}	0.5	0.4979	0.7956	0.7012	0.8051	0.1039	0.1188
		0.9	0.5030	0.7977	0.6923	0.8082	0.1159	0.0843
0.5	{0, 1}	0.5	0.5412	0.7611	0.7637	0.7608	0.0029	0.0944
		0.9	0.5214	0.7776	0.7243	0.7830	0.0587	0.0833
0.6	{0, 1}	0.5	0.5412	0.7611	0.7637	0.7608	0.0029	0.0944
		0.9	0.5322	0.7639	0.7403	0.7664	0.0261	0.0659

Cuadro 8.7: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía rotaciones de 90°, 180° o 270° y traslaciones de hasta el 20 %, con distinta p . Para cada imagen transformada, se escoge entre rotación o traslación (R or T). Proporción de imágenes transformadas $p = \{0.5, 0.9\}$. Se aumentaron ambas clases $l = \{0, 1\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	l	p	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	<i>M_{CV}</i>	<i>O_{SEN}</i>
0.4	{0, 1}	0.5	0.5155	0.7941	0.7243	0.8011	0.0768	0.0005
		0.9	0.4929	0.8142	0.6372	0.8319	0.1946	0.0697
0.5	{0, 1}	0.5	0.5320	0.7841	0.7400	0.7885	0.0485	0.0128
		0.9	0.5305	0.7776	0.6852	0.7868	0.1016	0.0797
0.6	{0, 1}	0.5	0.5430	0.7647	0.7560	0.7656	0.0096	0.0453
		0.9	0.5491	0.7374	0.7252	0.7388	0.0136	0.0810

Cuadro 8.8: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía rotaciones de 90°, 180° o 270° y traslaciones de hasta el 20 %, con distinta p . Para cada imagen transformada se aplica una rotación y una traslación (R and T). Proporción de imágenes transformadas $p = \{0.5, 0.9\}$. Se aumentaron ambas clases $l = \{0, 1\}$. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{CV}
0.4	0.0100	0.4638	0.8300	0.6142	0.8516	0.2375	0.1630
	0.1000	0.5248	0.7884	0.7160	0.7955	0.0795	0.1112
	1.0000	0.5065	0.8092	0.6705	0.8232	0.1528	0.0331
	10.0000	0.5722	0.7640	0.7262	0.7679	0.0418	0.0336
	100.0000	0.5404	0.8286	0.6083	0.8509	0.2426	0.0514
0.5	0.0100	0.5163	0.7826	0.6849	0.7924	0.1075	0.1859
	0.1000	0.5162	0.7834	0.7240	0.7892	0.0652	0.1553
	1.0000	0.5332	0.7877	0.7092	0.7956	0.0863	0.0470
	10.0000	0.5931	0.7331	0.7726	0.7292	0.0434	0.0278
	100.0000	0.6076	0.7073	0.8114	0.6968	0.1145	0.0124
0.6	0.0100	0.5163	0.7826	0.6849	0.7924	0.1075	0.1859
	0.1000	0.5371	0.7554	0.7557	0.7553	0.0004	0.1550
	1.0000	0.5705	0.7267	0.7720	0.7221	0.0499	0.1032
	10.0000	0.6367	0.6629	0.8428	0.6447	0.1980	0.0163
	100.0000	0.6515	0.6514	0.8738	0.6290	0.2449	-0.0181

Cuadro 8.9: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* con distintos valores para α_{MU} . Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	<i>M_{CV}</i>	<i>O_{CV}</i>
0.4	0.0100	0.5109	0.7862	0.7320	0.7916	0.0596	0.0900
	0.1000	0.5042	0.8028	0.7003	0.8129	0.1126	0.0511
	1.0000	0.5400	0.7834	0.7326	0.7885	0.0559	0.0281
	10.0000	0.5491	0.7977	0.6785	0.8098	0.1313	0.0222
	100.0000	0.5551	0.7920	0.6942	0.8019	0.1078	0.0103
0.5	0.0100	0.5109	0.7862	0.7320	0.7916	0.0596	0.0900
	0.1000	0.5400	0.7640	0.7637	0.7640	0.0003	0.0899
	1.0000	0.5609	0.7640	0.7634	0.7640	0.0006	0.0221
	10.0000	0.5895	0.7203	0.7957	0.7127	0.0830	-0.0074
	100.0000	0.5744	0.7518	0.7560	0.7513	0.0047	-0.0046
0.6	0.0100	0.5426	0.7690	0.7480	0.7711	0.0231	0.0270
	0.1000	0.5479	0.7539	0.7717	0.7522	0.0195	0.0804
	1.0000	0.5727	0.7403	0.7871	0.7355	0.0516	0.0301
	10.0000	0.6136	0.6923	0.8268	0.6788	0.1480	-0.0232
	100.0000	0.6072	0.7038	0.8105	0.6929	0.1175	-0.0293

Cuadro 8.10: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* y M* con distintos valores para α_{MU} . M* = R or T con $l = \{0, 1\}$ y $p = 0.5$. Primero se aplica *MixUp* y después M*. Resultado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

α_M	α_{MU}	<i>Loss</i>	<i>ACC</i>	<i>SEN</i>	<i>SPE</i>	M_{CV}	O_{CV}
0.4	0.0100	0.5215	0.7762	0.7400	0.7798	0.0398	0.1013
	0.1000	0.5226	0.7891	0.7403	0.7940	0.0537	0.0656
	1.0000	0.5218	0.7848	0.7086	0.7924	0.0838	0.0590
	10.0000	0.5397	0.7905	0.6868	0.8012	0.1144	0.0198
	100.0000	0.5648	0.7676	0.7252	0.7719	0.0467	0.0134
0.5	0.0100	0.5289	0.7712	0.7480	0.7735	0.0255	0.0969
	0.1000	0.5226	0.7891	0.7403	0.7940	0.0537	0.0656
	1.0000	0.5557	0.7540	0.7554	0.7537	0.0017	0.0324
	10.0000	0.5733	0.7482	0.7486	0.7482	0.0004	0.0183
	100.0000	0.5793	0.7439	0.7572	0.7428	0.0144	0.0030
0.6	0.0100	0.5593	0.7489	0.7714	0.7466	0.0248	0.0677
	0.1000	0.5271	0.7827	0.7480	0.7861	0.0381	0.0578
	1.0000	0.5877	0.7130	0.7954	0.7048	0.0906	0.0566
	10.0000	0.6389	0.6514	0.8428	0.6322	0.2106	0.0043
	100.0000	0.6197	0.6858	0.8194	0.6724	0.1470	-0.0060

Cuadro 8.11: Comparativa cuantitativa del resultado en validación cruzada de una misma *ResNet50* que entrena con un conjunto de entrenamiento aumentado vía *MixUp* y M^* con distintos valores para α_{MU} . $M^* = R$ or T con $l = \{0, 1\}$ y $p = 0.5$. Primero se aplica M^* y después *MixUp*. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

r	C	ACC	SEN	SPE	M_{CV}	O_{SEN}
-	-	0.8996	0.1655	0.9732	0.8076	0.8345
-	-	0.8773	0.2277	0.9424	0.7147	0.7723
L_1	0.0001	0.9089	0.0000	1.0000	1.0000	0.0000
	0.0010	0.9089	0.0000	1.0000	1.0000	0.0000
	0.0100	0.7447	0.2000	0.8000	0.6000	0.0000
	0.1000	0.7912	0.4486	0.8256	0.3770	0.2288
	1.0000	0.8680	0.3532	0.9195	0.5662	0.6468
	10.0000	0.8773	0.2588	0.9392	0.6804	0.7412
L_2	0.0001	0.8063	0.4477	0.8422	0.3945	0.0407
	0.0010	0.8085	0.4554	0.8437	0.3883	0.1178
	0.0100	0.8214	0.4409	0.8595	0.4186	0.4436
	0.1000	0.8601	0.3846	0.9076	0.5230	0.6154
	1.0000	0.8737	0.2665	0.9345	0.6680	0.7335
	10.0000	0.8773	0.2354	0.9416	0.7062	0.7646

Cuadro 8.12: Comparativa cuantitativa del resultado en validación cruzada de un regresor lineal. La subtabla con una sola entrada, corresponde al candidato que entrena con el conjunto de datos sin balancear. El resto de candidatos utiliza un conjunto de datos balanceado vía sobremuestreo de la clase minoritaria (positiva). Los dos primeros candidatos no utilizan ningún tipo de regularización. Resultado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$ y en entre los mejores 5 candidatos con mejor $\frac{SEN + SPE}{2}$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

c	max_f	n_{leaf}	n	ACC	SEN	SPE	M_{CV}	O_{SEN}
Gini	$\sqrt{n_f}$	2.0000	100	0.9103	0.0625	0.9953	0.9328	0.8587
Gini	$\sqrt{n_f}$	2.0000	50	0.9082	0.1812	0.9811	0.7998	0.8188
		2.0000	100	0.9067	0.1735	0.9803	0.8067	0.8265
		0.1000	50	0.8472	0.4089	0.8911	0.4822	0.2886
		0.1000	100	0.8501	0.3775	0.8974	0.5199	0.3450
		0.2000	50	0.7841	0.4723	0.8153	0.3430	0.1400
		0.2000	100	0.7784	0.4566	0.8106	0.3540	0.1835
		0.3000	50	0.7274	0.4883	0.7514	0.2631	0.1322
	$\log_2(n_f)$	0.3000	100	0.7281	0.5037	0.7506	0.2469	0.1204
		2.0000	50	0.9110	0.2132	0.9811	0.7678	0.7868
		2.0000	100	0.9110	0.2286	0.9795	0.7509	0.7714
		0.1000	50	0.8321	0.4012	0.8753	0.4740	0.2071
		0.1000	100	0.8479	0.4092	0.8919	0.4826	0.2153
		0.2000	50	0.7640	0.4800	0.7924	0.3124	0.1046
		0.2000	100	0.7583	0.5037	0.7838	0.2801	0.0890
Entropy	$\sqrt{n_f}$	0.3000	50	0.7425	0.5514	0.7617	0.2103	0.0587
		0.3000	100	0.7267	0.5434	0.7451	0.2017	0.0492
		2.0000	50	0.9067	0.1649	0.9811	0.8162	0.8351
		2.0000	100	0.9089	0.1575	0.9842	0.8267	0.8425
		0.1000	50	0.8450	0.3855	0.8911	0.5055	0.2999
		0.1000	100	0.8508	0.3778	0.8982	0.5203	0.3430
		0.2000	50	0.7848	0.4646	0.8169	0.3523	0.1437
	$\log_2(n_f)$	0.2000	100	0.7848	0.4566	0.8177	0.3611	0.1855
		0.3000	50	0.7274	0.4883	0.7514	0.2631	0.1322
		0.3000	100	0.7281	0.5037	0.7506	0.2469	0.1204
		2.0000	50	0.9075	0.1889	0.9795	0.7906	0.8111
		2.0000	100	0.9096	0.2052	0.9803	0.7750	0.7948
		0.1000	50	0.8393	0.3935	0.8840	0.4904	0.2170
		0.1000	100	0.8479	0.4092	0.8919	0.4826	0.2036

Cuadro 8.13: Comparativa cuantitativa del resultado en validación cruzada para distintas combinaciones de hiperparámetros de un *Random Forest*. La entrada superior con una sola entrada, corresponde al candidato que entraña con el conjunto de datos sin balancear y con los hiperparámetros por defecto. El resto de candidatos utiliza un conjunto de datos balanceado vía sobremuestreo de la clase minoritaria (positiva). n_f = Número de características por instancia del conjunto de datos. c = Criterio de pureza. n_{leaf} = Número de instancias mínimas para no considerar a un nodo hoja. n = Número de árboles. max_f = Número de características evaluadas en cada nodo. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$ y en entre los mejores 5 candidatos con mejor $\frac{SEN + SPE}{2}$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

k	d	C	γ	ACC	SEN	SPE	M_{CV}	O_{SEN}
g	-	1.0	-	0.9075	0.0080	0.9976	0.9896	0.1000
p	10	10.0000	auto	0.6384	0.6206	0.6401	0.0195	0.2716
s	-	10.0000	scale	0.6636	0.6369	0.6661	0.0292	0.0860
p	10	1.0000	auto	0.6779	0.5431	0.6914	0.1483	0.2999
g	-	10.0000	auto	0.7927	0.4717	0.8248	0.3531	0.4243
l	-	0.0100	-	0.8149	0.4868	0.8476	0.3609	0.4957
s	-	1.0000	scale	0.8006	0.4637	0.8342	0.3705	0.0979
s	-	10.0000	auto	0.7991	0.4560	0.8335	0.3775	0.3023
p	5	0.0010	scale	0.7977	0.4243	0.8350	0.4107	0.3361
p	2	0.1000	scale	0.8372	0.4483	0.8761	0.4278	0.5438
g	-	0.1000	scale	0.8329	0.3852	0.8777	0.4924	0.3829
p	5	10.0000	auto	0.8357	0.3622	0.8832	0.5211	0.0867
p	5	0.1000	auto	0.8357	0.3622	0.8832	0.5211	0.0867
p	5	0.0100	auto	0.8357	0.3622	0.8832	0.5211	0.0867
p	5	0.0010	auto	0.8357	0.3622	0.8832	0.5211	0.0867
p	5	1.0000	auto	0.8357	0.3622	0.8832	0.5211	0.0867
l	-	0.0010	-	0.8709	0.3462	0.9234	0.5773	0.0615
p	2	0.0100	scale	0.8766	0.3382	0.9306	0.5924	0.0616
p	2	10.0000	auto	0.8866	0.2911	0.9463	0.6553	0.0006
p	2	1.0000	auto	0.8866	0.2911	0.9463	0.6553	0.0006
p	2	0.0010	scale	0.8866	0.2911	0.9463	0.6553	0.0006

Cuadro 8.14: Comparativa cuantitativa del resultado en validación cruzada para distintas combinaciones de hiperparámetros de un *SVM*. Se incluyen sólo los 20 candidatos con mejor M_{CV} . La entrada superior con una sola entrada, corresponde al candidato que entrena con el conjunto de datos sin balancear y con los hiperparámetros por defecto. El resto de candidatos utiliza un conjunto de datos balanceado vía sobremuestreo de la clase minoritaria (positiva). $auto = \{\frac{1}{n_f}\}$. $scale = \frac{1}{n_f * Var(X_{train})}$. El resto de acrónimos puede consultarse en la Subsección 6.2.3. Resaltado en color el mejor candidato (menor $M_{CV} \equiv |SEN - SPE|$) y en entre los mejores 5 candidatos con mejor $\frac{SEN + SPE}{2}$). Por simplicidad, $O_{SEN} = O_{CV}(SEN)$.

Bibliografía

- [1] Douglas H Ubelaker. Forensic anthropology: methodology and diversity of applications. *Biological anthropology of the human skeleton*, pages 41–69, 2008.
- [2] Gobierno de España. Protocolo Marco sobre determinadas actuaciones en relación con los Menores Extranjeros No Acompañados. BOE, Num. 251, pages 83894—919, 2014.
- [3] Daja Wenke. Age assessment: Council of europe member states' policies, procedures and practices respectful of children's rights in the context of migration, 2017.
- [4] Pedro M Garamendi González, Rafael Bañón González, Amadeo Pujoł Robinat, Fernando F Aguado Bustos, María Irene Landa Tabuyo, José Luis Prieto Carrero, and Fernando Serrulla Rech. Recomendaciones sobre métodos de estimación forense de la edad de los menores extranjeros no acompañados: Documento de consenso de buenas prácticas entre los institutos de medicina legal de españa (2010). *Revista Española de Medicina Legal*, 37(1):22–29, 2011.
- [5] European Asylum Support Office. EASO practical guide on age assessment, 2018.
- [6] Andreas Schmeling, Reinhard Dettmeyer, Ernst Rudolf, Volker Vieth, and Gunther Geserick. Forensic age estimation: methods, certainty, and the law. *Deutsches Ärzteblatt International*, 113(4):44, 2016.
- [7] Defensor del Pueblo de España. ¿Menores o Adultos? Procedimientos para la determinación de la edad., 2012.
- [8] Maisy Lossois and Eric Baccino. Forensic age estimation in migrants: Where do we stand? *Wiley Interdisciplinary Reviews: Forensic Science*, 3(4):e1408, 2021.
- [9] MI Landa. Parámetros de maduración con la edad en ortopantomografías digitales. *Universidad de Granada, Granada, Spain*, 2007.

- [10] I Garamendi González. Conclusiones de la jornada de trabajo sobre determinación forense de la edad de los menores extranjeros no acompañados. documento de consenso de buenas prácticas entre los institutos de medicina legal de españa. *Revista Espanola de Medicina Legal*, 37(1):5–6, 2011.
- [11] Pablo Mesejo, Rubén Martos, Óscar Ibáñez, Jorge Novo, and Marcos Ortega. A survey on artificial intelligence techniques for biomedical image analysis in skeleton-based forensic human identification. *Applied Sciences*, 10(14):4703, 2020.
- [12] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.
- [13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [14] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12, 2016.
- [15] Andre Esteva, Katherine Chou, Serena Yeung, Nikhil Naik, Ali Madani, Ali Mottaghi, Yun Liu, Eric Topol, Jeff Dean, and Richard Socher. Deep learning-enabled medical computer vision. *NPJ digital medicine*, 4(1):5, 2021.
- [16] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.
- [17] William Walter Greulich and S Idell Pyle. Radiographic atlas of skeletal development of the hand and wrist. *The American Journal of the Medical Sciences*, 238(3):393, 1959.
- [18] Arto Demirjian, Harvey Goldstein, and James M Tanner. A new system of dental age assessment. *Human biology*, pages 211–227, 1973.
- [19] Manuel Kellinghaus, Ronald Schulz, Volker Vieth, Sven Schmidt, and Andreas Schmeling. Forensic age estimation in living subjects based on

- the ossification status of the medial clavicular epiphysis as revealed by thin-slice multidetector computed tomography. *International journal of legal medicine*, 124:149–154, 2010.
- [20] International Organization for Migration. *World Migration Report 2022*. 2021.
 - [21] Eurostat. Online Data Codes: (migr_asyappctza), (migr_asydcfsta), (migr_asydcfina) and (migr_asypenctzm). Extracted: 2023-10.
 - [22] Dreamstime. Kirsty Pargeter. https://www.dreamstime.com/kjpargeter_info. Accessed: 2023-10.
 - [23] PyTorch. <https://pytorch.org/>. Accessed: 2023-10.
 - [24] Scikit Learn. <https://scikit-learn.org/stable/index.html>. Accessed: 2023-10.
 - [25] YAML. <https://captum.ai/>. Accessed: 2023-10.
 - [26] fastai. <https://docs.fast.ai/>. Accessed: 2023-10.
 - [27] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, 2012.
 - [28] Tom M Mitchell. Machine learning, 1997.
 - [29] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.
 - [30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
 - [31] Scott Krig and Scott Krig. Interest point detector and feature descriptor survey. *Computer Vision Metrics: Textbook Edition*, pages 187–246, 2016.
 - [32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
 - [33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
 - [34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

- [35] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [37] CS231n: Deep Learning for Computer Vision. Standford Free Course. Module 1: Neural Networks. Neural Networks Part 1: Setting up the Architecture. <https://cs231n.github.io/neural-networks-1/>. Accessed: 2023-10.
- [38] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [39] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.
- [40] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.
- [41] Yoni Nazarathy Benoit Liquet, Sarat Moka. *The Mathematical Engineering of Deep Learning*. 2021. <https://deeplearningmath.org/tricks-of-the-trade.html>. Accessed: 2023-10.
- [42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [43] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [44] CS231n: Deep Learning for Computer Vision. Standford Free Course. Module 1: Neural Networks. Backpropagation, Intuitions. <https://cs231n.github.io/optimization-2/>. Accessed: 2023-10.
- [45] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.
- [46] CS231n: Deep Learning for Computer Vision. Standford Free Course. Module 2: Convolutional Neural Networks. <https://cs231n.github.io/>. Accessed: 2023-10.

- [47] Soumenbondhu. Easiest Way to Understand CNN (Convolution Neural Network). <https://medium.com/@soumenbondhu/easiest-way-to-understand-cnn-convolution-neural-network-f6dfe5cc11a8>. Accessed: 2023-10.
- [48] Muhamad Yani, Si MT Budhi Irawan, S, and MT Casi Setiningsih, ST. Application of transfer learning using convolutional neural network method for early detection of terry's nail. In *Journal of Physics: Conference Series*, volume 1201, page 012052. IOP Publishing, 2019.
- [49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [50] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.
- [51] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [52] Li Liu, Jie Chen, Paul Fieguth, Guoying Zhao, Rama Chellappa, and Matti Pietikäinen. From bow to cnn: Two decades of texture representation for texture classification. *International Journal of Computer Vision*, 127:74–109, 2019.
- [53] 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.
- [54] M Yaşar Işcan, Susan R Loth, and Ronald K Wright. Metamorphosis at the sternal rib end: a new method to estimate age at death in white males. *American journal of physical anthropology*, 65(2):147–156, 1984.
- [55] JC Prieto, S Mihaila, A Hilaire, L Fanton, C Odet, and C Revol-Muller. Age estimation from 3d x-ray ct images of human fourth ribs. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2012.

- [56] Richard S Meindl and C Owen Lovejoy. Ectocranial suture closure: A revised method for the determination of skeletal age at death based on the lateral-anterior sutures. *American journal of physical anthropology*, 68(1):57–66, 1985.
- [57] Robert W Mann, Richard L Jantz, William M Bass, and Patrick S Willey. Maxillary suture obliteration: a visual method for estimating skeletal age. *Journal of Forensic Sciences*, 36(3):781–791, 1991.
- [58] T Wingate Todd. Age changes in the pubic bone. *American journal of physical anthropology*, 4(1):1–70, 1921.
- [59] Sheilagh Brooks and Judy M Suchey. Skeletal age determination based on the os pubis: a comparison of the acsádi-nemeskéri and suchey-brooks methods. *Human evolution*, 5:227–238, 1990.
- [60] C Owen Lovejoy, Richard S Meindl, Thomas R Pryzbeck, and Robert P Mensforth. Chronological metamorphosis of the auricular surface of the ilium: a new method for the determination of adult skeletal age at death. *American journal of physical anthropology*, 68(1):15–28, 1985.
- [61] Joanne L Buckberry and Andrew T Chamberlain. Age estimation from the auricular surface of the ilium: a revised method. *American Journal of Physical Anthropology: The Official Publication of the American Association of Physical Anthropologists*, 119(3):231–239, 2002.
- [62] Harry H Mincer, Edward F Harris, Hugh E Berryman, et al. *The ABFO study of third molar development and its use as an estimator of chronological age*. ASTM International, 1993.
- [63] Roberto Cameriere, Luigi Ferrante, and Mariano Cingolani. Age estimation in children by measurement of open apices in teeth. *International journal of legal medicine*, 120:49–52, 2006.
- [64] José L Prieto, Elena Barbería, Ricardo Ortega, and Concepción Maña. Evaluation of chronological age based on third molar development in the spanish population. *International journal of legal medicine*, 119:349–354, 2005.
- [65] Kaan Orhan, L Ozer, AI Orhan, Sand Dogan, and CS Paksoy. Radiographic evaluation of third molar development in relation to chronological age among turkish children and youth. *Forensic science international*, 165(1):46–51, 2007.
- [66] A Meindl, S Tangl, C Huber, B Maurer, and G Watzek. The chronology of third molar mineralization in the austrian population—a con-

- tribution to forensic age estimation. *Forensic science international*, 169(2-3):161–167, 2007.
- [67] Andreas Olze, Andreas Schmeling, Mari Taniguchi, Hitoshi Maeda, Piet van Niekerk, Klaus-Dieter Wernecke, and Gunther Geserick. Forensic age estimation in living subjects: the ethnic factor in wisdom tooth mineralization. *International Journal of Legal Medicine*, 118:170–173, 2004.
- [68] Iris Cadenas, César Celis, and Alejandro Hidalgo. Método de demirjian para estimación de edad dentaria en base a estudios de mineralización. *Anu. Soc. Radiol. Oral Máximo Facial de Chile*, 13:17–23, 2010.
- [69] R Cameriere, L Ferrante, D De Angelis, F Scarpino, and F Galli. The comparison between measurement of open apices of third molars and demirjian stages to test chronological age of over 18 year olds in living subjects. *International journal of legal medicine*, 122:493–497, 2008.
- [70] Roberto Cameriere, Valeria Santoro, Roberta Roca, Piercarlo Lozito, Francesco Intronà, Mariano Cingolani, Ivan Galić, and Luigi Ferrante. Assessment of legal adult age of 18 by measurement of open apices of the third molars: study on the albanian sample. *Forensic science international*, 245:205–e1, 2014.
- [71] Roselhy Juliana Quispe Lizarbe, Christian Solís Adriánzén, Milushka Miroslava Quezada-Márquez, Ivan Galić, and Roberto Cameriere. Demirjian's stages and cameriere's third molar maturity index to estimate legal adult age in peruvian population. *Legal Medicine*, 25:59–65, 2017.
- [72] Sharma Preeti, Vijay Wadhwan, and Neeraj Sharma. Reliability of determining the age of majority: a comparison between measurement of open apices of third molars and demirjian stages. *The Journal of Forensic Odonto-stomatology*, 36(2):2, 2018.
- [73] Parul Khare, Jiang Li, Luz Andrea Velandia Palacio, Ivan Galić, Luigi Ferrante, and Roberto Cameriere. Validation of the third molar maturity index cut-off value of 0.08 for indicating legal age of 18 years in eastern chinese region. *Legal Medicine*, 42:101645, 2020.
- [74] Maria Melo, Fadi Ata-Ali, Javier Ata-Ali, José María Martínez González, and Teresa Cobo. Demirjian and cameriere methods for age estimation in a spanish sample of 1386 living subjects. *Scientific reports*, 12(1):2838, 2022.
- [75] R Cameriere, Stefano De Luca, and L Ferrante. Study of the ethnicity's influence on the third molar maturity index (I_{3M}) for estimating age

- of majority in living juveniles and young adults. *International Journal of Legal Medicine*, 135:1945–1952, 2021.
- [76] Francesco De Micco, Federica Martino, Luz Andrea Velandia Palacio, Mariano Cingolani, and Carlo Pietro Campobasso. Third molar maturity index and legal age in different ethnic populations: Accuracy of cameriere's method. *Medicine, Science and the Law*, 61(1_suppl):105–112, 2021.
- [77] N Angelakopoulos, S De Luca, LA Velandia Palacio, E Coccia, L Ferrante, and R Cameriere. Third molar maturity index (i 3m) for assessing age of majority: study of a black south african sample. *International journal of legal medicine*, 132:1457–1464, 2018.
- [78] David B Larson, Matthew C Chen, Matthew P Lungren, Safwan S Halabi, Nicholas V Stence, and Curtis P Langlotz. Performance of a deep-learning neural network model in assessing skeletal maturity on pediatric hand radiographs. *Radiology*, 287(1):313–322, 2018.
- [79] Hyunkwang Lee, Shahein Tajmir, Jenny Lee, Maurice Zissen, Bethel Ayele Yeshiwash, Tarik K Alkasab, Garry Choy, and Synho Do. Fully automated deep learning system for bone age assessment. *Journal of digital imaging*, 30:427–441, 2017.
- [80] Concetto Spampinato, Simone Palazzo, Daniela Giordano, Marco Aldinucci, and Rosalia Leonardi. Deep learning for automated skeletal bone age assessment in x-ray images. *Medical image analysis*, 36:41–51, 2017.
- [81] Jeong Rye Kim, Woo Hyun Shim, Hee Mang Yoon, Sang Hyup Hong, Jin Seong Lee, Young Ah Cho, and Sangki Kim. Computerized bone age estimation using deep learning based program: evaluation of the accuracy and efficiency. *American Journal of Roentgenology*, 209(6):1374–1380, 2017.
- [82] Markus Auf der Mauer, Eilin Jopp-van Well, Jochen Herrmann, Michael Groth, Michael M Morlock, Rainer Maas, and Dennis Säring. Automated age estimation of young individuals based on 3d knee mri using deep learning. *International Journal of Legal Medicine*, 135:649–663, 2021.
- [83] Cuong Van Pham, Su-Jin Lee, So-Yeon Kim, Sookyoung Lee, Soo-Hyung Kim, and Hyung-Seok Kim. Age estimation based on 3d post-mortem computed tomography images of mandible and femur using convolutional neural networks. *PLoS One*, 16(5):e0251388, 2021.

- [84] Yuan Li, Zhizhong Huang, Xiaoai Dong, Weibo Liang, Hui Xue, Lin Zhang, Yi Zhang, and Zhenhua Deng. Forensic age estimation for pelvic x-ray images using deep learning. *European radiology*, 29:2322–2329, 2019.
- [85] Paul H Yi, Jinchi Wei, Tae Kyung Kim, Jiwon Shin, Haris I Sair, Ferdinand K Hui, Gregory D Hager, and Cheng Ting Lin. Radiology “forensics”: determination of age and sex from chest radiographs using deep learning. *Emergency Radiology*, 28:949–954, 2021.
- [86] Denis Milošević, Marin Vodanović, Ivan Galić, and Marko Subašić. Automated estimation of chronological age from panoramic dental x-ray images using deep learning. *Expert Systems with Applications*, 189:116038, 2022.
- [87] Noor Mualla, Essam H Houssein, and MR Hassan. Dental age estimation based on x-ray images. *Computers, Materials & Continua*, 62(2), 2020.
- [88] Nicolas Vila-Blanco, Maria J Carreira, Paulina Varas-Quintana, Carlos Balsa-Castro, and Inmaculada Tomas. Deep neural networks for chronological age estimation from opg images. *IEEE transactions on medical imaging*, 39(7):2374–2384, 2020.
- [89] Yu-Cheng Guo, Mengqi Han, Yuting Chi, Hong Long, Dong Zhang, Jing Yang, Yang Yang, Teng Chen, and Shaoyi Du. Accurate age classification using manual method and deep convolutional neural network based on orthopantomogram images. *International journal of legal medicine*, 135:1589–1597, 2021.
- [90] Jared Murray, D Heng, A Lygate, L Porto, A Abade, S Manica, and A Franco. Applying artificial intelligence to determination of legal age of majority from radiographic. *Morphologie*, 108(360):100723, 2024.
- [91] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- [92] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [93] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

- [94] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.
- [95] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- [96] Distribución Beta. Wikipedia. https://es.wikipedia.org/wiki/Distribuci%C3%B3n_beta. Accessed: 2023-10.
- [97] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Jun-suk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.
- [98] Pablo Mesejo, Daniel Pizarro, Armand Abergel, Olivier Rouquette, Sylvain Beorchia, Laurent Poincloux, and Adrien Bartoli. Computer-aided classification of gastrointestinal lesions in regular colonoscopy. *IEEE transactions on medical imaging*, 35(9):2051–2063, 2016.
- [99] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [100] PyTorch. Docs. Models and pre-trained weights. ResNet. <https://pytorch.org/vision/main/models/resnet.html>. Accessed: 2023-10.
- [101] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [102] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.
- [103] Rachel Lea Draelos and Lawrence Carin. Use hirescam instead of grad-cam for faithful explanations of convolutional neural networks. *arXiv preprint arXiv:2011.08891*, 2020.
- [104] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*, volume 1, pages 886–893. Ieee, 2005.

- [105] Histogram of Oriented Gradients explained using OpenCV. Satya Mallick. <https://learnopencv.com/histogram-of-oriented-gradients/>. Accessed: 2023-10.
- [106] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.
- [107] The intuition behind kernel methods. HashPi. <https://www.hashpi.com/the-intuition-behind-kernel-methods>. Accessed: 2023-10.
- [108] Support Vector Machines (SVM). Satya Mallick. <https://learnopencv.com/support-vector-machines-svm/>. Accessed: 2023-10.
- [109] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
- [110] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.
- [111] Alina Munteanu. Computational models to investigate binding mechanisms of regulatory proteins. 2018.
- [112] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.
- [113] Python. <https://www.python.org/>. Accessed: 2023-10.
- [114] Pandas. <https://pandas.pydata.org/pandas-docs/stable/index.html>. Accessed: 2023-10.
- [115] NumPy. <https://numpy.org/>. Accessed: 2023-10.
- [116] Visual Studio Code. <https://code.visualstudio.com/>. Accessed: 2023-10.
- [117] Project Jupyter. <https://jupyter.org/>. Accessed: 2023-10.
- [118] Conda. <https://docs.conda.io/en/latest/#>. Accessed: 2023-10.
- [119] YAML. <https://yaml.org/>. Accessed: 2023-10.
- [120] GridSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed: 2023-10.
- [121] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems*, 27, 2014.

- [122] Criterion for Tree Classification. Scikit Learn. <https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation>. Accessed: 2023-10.

