UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL SOFTWARE

Trabajo Fin de Grado

Clasificación de imágenes médicas con técnicas de *Deep Learning*

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL SOFTWARE

Trabajo Fin de Grado

Clasificación de imágenes médicas con técnicas de *Deep Learning*

Autor: Juan José Carballo Pacheco

Tutor: Alberto Gómez Mancha

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En el campo de la medicina, así como en muchas otras disciplinas, se han realizado una gran cantidad de avances gracias a la Ingeniería Informática. En estos últimos años, han surgido numerosas nuevas técnicas que permiten mejorar los diagnósticos de diferentes maneras, así como también los tratamientos y el seguimiento de diferentes enfermedades. Este trabajo se centra en la mejora de la emisión de diagnósticos conseguida mediante el campo de la Inteligencia Artificial.

En algunas enfermedades, emitir un diagnóstico de forma rápida es crucial, siendo necesario esto para que sea tratada correctamente y con el menor número de secuelas posibles. Dentro de estas enfermedades se enmarca el cáncer de piel provocado por melanomas. Así, el proyecto que aquí se presenta pretende ayudar al diagnóstico temprano de dicha dolencia mediante la automatización del análisis de imágenes, además de intentar superar algunas de las dificultades que presenta este proceso, de forma que pueda suponer un apoyo a los profesionales del sector ahorrando tiempo y haciendo el proceso más eficiente. Para esta tarea se han desarrollado diferentes modelos de Inteligencia Artificial, basándose en *Deep Learning* y, dentro de este, en redes neuronales convolucionales.

Durante el desarrollo del proyecto se ha empleado una metodología de *Data Science* (Ciencia de Datos). Esta metodología es ampliamente utilizada en proyectos que conlleven realizar un modelo basado en Inteligencia Artificial. En términos generales, consiste en comprender el problema, extraer los datos que van a ser utilizados, procesarlos de forma que sean utilizables para el proyecto, y clasificarlos en función de la variable deseada. Una vez realizados dichos pasos, se crea un modelo al que se le dará como entrada dichos datos, y por último se evaluarán, analizarán y observarán los resultados obtenidos con dicho modelo. Esta metodología es iterativa, permitiendo volver a una fase anterior para ser mejorada en cualquier momento.

Por último, tras finalizar el mencionado análisis de resultados sobre los modelos finales, se sacarán las debidas conclusiones y se propondrán mejoras para realizarse en un trabajo futuro.

1. INTRODUCCIÓN

No hace falta ser ingeniero para haber escuchado hablar de Inteligencia Artificial. Tanto es así, que cada vez es más común encontrar soluciones basadas en Inteligencia Artificial para diferentes problemas a los que se enfrentan las personas en su día a día. Este proyecto intenta hacer una aportación dentro de la ayuda al diagnóstico médico.

Este trabajo consiste en clasificar imágenes con el fin de discernir diferentes tipos de lesiones de la piel mediante el uso de Inteligencia Artificial. En concreto, se busca conseguir distinguir los melanomas del resto de lesiones menos graves.

Para conseguir estos objetivos que se proponen se han utilizado datos públicos existentes en una competición organizada a nivel internacional como es el *ISIC Challenge*, del que se habla más profundamente en el capítulo *5.3.1. Obtención de los datos*.

La Inteligencia Artificial ha revolucionado por completo tanto la informática como muchos otros campos, ayudando también a desarrollar muchas otras disciplinas como la meteorología, la genética, la medicina o incluso el arte. Sin embargo, y aunque haya supuesto una gran revolución, al igual que otras tecnologías, ésta no está exenta de problemas, pero lo cierto es que actualmente se están encontrando variadas y eficaces soluciones a muchos problemas.

Como se ha mencionado anteriormente, la revolución a la que ha llevado esta tecnología se ha producido en campos muy variados. Este proyecto se enmarca en el campo de la medicina.

La Inteligencia Artificial en el campo de la medicina tiene un increíble potencial que se está viendo incrementado gracias a que cada vez se encuentra disponible más cantidad de información mediante diferentes análisis, sensores, métricas, etc. Y lo cierto es que cada vez existen más y más aplicaciones dentro de la medicina para esta nueva tecnología.

Si bien este proyecto se centra en la ayuda para el diagnóstico creando un clasificador, la Inteligencia Artificial también se está utilizando en la medicina para tareas tan variadas como mejorar la productividad mediante la automatización de determinadas acciones, personalizar los tratamientos para cada persona, monitorizar el estado de salud o los biomarcadores de quien lo necesite, acelerar el desarrollo de medicinas, etc.

La ayuda para el diagnóstico, donde se enmarca este proyecto, consiste en diseñar un modelo que pueda emitir diagnósticos de la forma más precisa posible, con el fin de ayudar a los médicos especialistas en el campo a diagnosticar correctamente de forma más rápida. Según algunos estudios como: "*Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists*" [1], un modelo bien diseñado puede alcanzar una tasa de aciertos del nivel de un especialista humano en la materia, por lo que contar con la ayuda de estos tipos de clasificadores podría incrementar enormemente la eficiencia del proceso de diagnóstico.

1.1. Planificación

Respecto a la planificación del proyecto, éste ha llevado en torno a 330 horas. La distribución de este tiempo es la que se muestra en la *Tabla 1* que se presenta a continuación.

Fase	Tarea	Tiempo (en horas)
Fase 1	Definición del proyecto	3
	Establecimiento de objetivos	3
	Planificación	5
	TOTAL FASE	**11**
Fase 2	Investigación sobre el proyecto	10
	Búsqueda del conjunto de datos	5
	Estudio del lenguaje de programación y librerías	5
	Planificación de las pruebas	3
	TOTAL FASE	**23**
Fase 3	Preprocesado del conjunto de datos	5
	Creación e implementación de modelos	10
	Entrenamiento y validación	100
	Corrección de problemas en los modelos	30
	Solución de problemas derivados de recursos (librerías o entornos de desarrollo)	45
	TOTAL FASE	**190**

Fase 4	Comparación de resultados	5
	Conclusiones	5
	Mejoras futuras	1
	TOTAL FASE	**11**
Fase 5	Documentación	90
	Presentación	5
	TOTAL FASE	**95**
	TOTAL	**330**

Tabla 1. Planificación del proyecto.

El proyecto fue principalmente realizado en verano de 2022, pero debido a que el autor comenzó a trabajar en agosto del mismo año, se alargó hasta noviembre por falta de tiempo.

1.2. Estructura

Una vez introducidos los motivos, el contexto en el que se desarrolla el proyecto y la planificación realizada, se procede a presentar la estructura de la memoria de este Trabajo de Fin de Grado. La división se ha realizado de la siguiente forma:

En el punto 1, en el que se encuentra este apartado, se introduce la memoria, se contextualiza el proyecto describiendo la situación actual en el mundo de la Inteligencia Artificial y se describen las motivaciones que han llevado al autor a realizar este estudio; también se comenta la estructura del documento que se presenta.

En el punto 2 se describen los objetivos a los que se plantea llegar con el desarrollo del proyecto. Dentro de estos, se habla tanto de los objetivos generales menos específicos de este proyecto concreto y más enfocadas al campo de estudio, como de algunas metas específicas que pretenden cumplirse mediante el desarrollo aquí propuesto.

En el apartado 3 se presenta el estado de la cuestión del marco teórico en el cual se desarrolla el proyecto, además de contextualizar y explicar los conceptos relevantes con los que se va a trabajar. Se describen muchas de las técnicas aplicadas así como también muchas que se han planteado aplicar siendo finalmente descartadas, tales como el *Transfer Learning*, el *dropout* y otras. Por último, también se habla de las

aplicaciones médicas de estas tecnologías, debido a que el proyecto se enmarca en dichas aplicaciones.

En el capítulo 4 se habla de las tecnologías que se han utilizado a lo largo del proyecto, principalmente el lenguaje de programación utilizado, sus ventajas e inconvenientes y la finalidad de su elección; las librerías con las que se ha contado y cuáles son sus funcionalidades en el proyecto, así como sus ventajas respecto a otras que pudieran realizar las mismas tareas; los entornos en los que se ha desarrollado el proyecto, teniendo en cuenta que se han utilizado dos diferenciados, comparando las razones por las que se ha optado por uno u otro en determinados momentos; el conjunto de datos, explicando brevemente de dónde se han obtenido estos datos además de algunas de sus características básicas; y cerrando con las limitaciones técnicas que se han encontrado durante el desarrollo así como las soluciones propuestas que se han adoptado.

En el capítulo 5 se expone el desarrollo que se ha seguido, cómo se han procesado los diferentes datos de entrada para normalizar los valores de las diferentes fuentes, cómo se han solucionado los problemas relativos a los datos ya que, al estar pensados para una competición, el conjunto de datos presenta problemas evidentes, la técnica que se ha seguido a la hora de plantear el ajuste de los hiperparámetros, y cómo se han planteado las diferentes pruebas que se han realizado con diferentes conjuntos de datos y empleando técnicas variadas para comprobar cuál de todas ellas podría dar mejores resultados.

En el apartado 6 se muestran los resultados de las pruebas realizadas durante todo el desarrollo y se comentan los valores de las métricas obtenidos. El objetivo de este punto será interpretar los resultados a partir de las métricas establecidas para obtener el mejor modelo entre los diferentes probados, y las mejores técnicas en el contexto de este conjunto de datos y este proyecto. Hay que destacar que no siempre la mejor métrica será la elegida por diferentes motivos explicados en dicho apartado.

Por último, en el punto 7, se comentan las conclusiones obtenidas en base a los resultados y las pruebas realizadas. Estas conclusiones se extraen a partir del desarrollo del proyecto, interpretando los resultados obtenidos con las pruebas realizadas anteriormente.

2. OBJETIVOS

Este proyecto tiene como objetivo principal desarrollar un modelo de aprendizaje profundo que permita predecir si una lesión de la piel es un melanoma mediante el análisis de una imagen del área afectada.

Para el desarrollo del proyecto se han empleado técnicas de *Deep Learning*. Concretamente se han utilizado redes neuronales convolucionales.

La decisión de realizar el desarrollo utilizando herramientas del campo del *Machine Learning* viene dada por la amplia variedad de sectores en los que está comenzando a utilizarse esta tecnología además de su potencial para dar resultados más que decentes.

Se pueden destacar también otros objetivos buscados con el desarrollo de este proyecto:

- **Conocimiento del campo:** Se busca ampliar los conocimientos en este campo, puesto que la ingeniería de datos en general es un campo en el que se requiere una gran cantidad de profesionales, y tremendamente multidisciplinar, ya que se está comenzando a aplicar en prácticamente cualquier campo dentro de la ciencia.
- **Profundizar en Python y sus bibliotecas:** Se busca también ampliar conocimientos en un lenguaje puntero como es Python, así como diversas librerías empleadas durante el desarrollo del proyecto. Python se ha establecido como uno de los lenguajes ampliamente utilizados para ciencia de datos, y cuenta con diversas librerías que se han convertido en una de las opciones preferidas de los ingenieros de datos.

3. ESTADO DE LA CUESTIÓN

Durante este apartado se trata el estado actual del contexto en el que se desarrolla el trabajo. En primer lugar, se explican algunos de los conceptos básicos más generales asociados a las redes neuronales. Seguidamente se describen algunos de los conceptos más específicos que han sido utilizados durante el desarrollo, así como se da contexto a la utilización de estos campos en el mundo de la medicina donde se enmarca este trabajo. Esto permitirá entender algunas de las bases que se aplican para desarrollar este proyecto, así como conceptos del área de trabajo en el que se enmarca el desarrollo de dicho proyecto.

3.1. Inteligencia Artificial

La Inteligencia Artificial (IA) ha sido definida de diferentes formas a lo largo de su historia. Una de ellas es la siguiente: La inteligencia artificial es la ciencia e ingeniería de hacer máquinas inteligentes, especialmente programas de ordenador inteligentes [2].

Aunque también se puede entender la Inteligencia Artificial como un campo que combina tanto ciencias de la computación, como conjuntos de datos robustos para alcanzar soluciones a problemas [3].

El origen de la Inteligencia Artificial se remonta a los años 50, donde se comenzaron a asentar los modelos teóricos de lo que posteriormente sería la IA, además de constituirse como campo académico oficialmente. Por otro lado, en esos años también se publicó el famoso artículo de Alan Turing "*Computing Machinery and Intelligence*" [4], introduciendo el Test de Turing, pensado para descubrir si una máquina puede actuar de forma indistinguible respecto a una forma de vida pensante.

Si bien está extendido que el concepto de la Inteligencia Artificial se basa en la humana, esto no es del todo cierto. Entre algunas de las diferencias que existen entre la Inteligencia Artificial y la humana se encuentran:

- **La arquitectura de ambas:** Mientras que en la artificial existen unas entradas y salidas fácilmente identificables, en la humana no se da el caso, siendo mucho más compleja que eso.

- **La importancia del contexto:** El cerebro es capaz de adaptarse fácilmente a cada situación, por muy única que sea, sin embargo, la Inteligencia Artificial responde a estímulos previamente determinados, aunque ya comienzan a verse las primeras Inteligencias Artificiales conscientes del contexto [5].
- **Los humanos somos más emocionales que racionales:** El pensamiento racional humano queda limitado a algunas situaciones concretas a lo largo del día, obviamente, las Inteligencias Artificiales trabajan a base de argumentos sólidos, apartando la posible carga emocional de estos [3].

La Inteligencia Artificial está compuesta de diferentes técnicas, si bien la clasificación de éstas depende ampliamente de la fuente. Aquí se tratará en profundidad el *Machine Learning* en concreto, y dentro de este, el campo del *Deep Learning*. Esta clasificación se encuentra explicada gráficamente en la *Figura 1*.

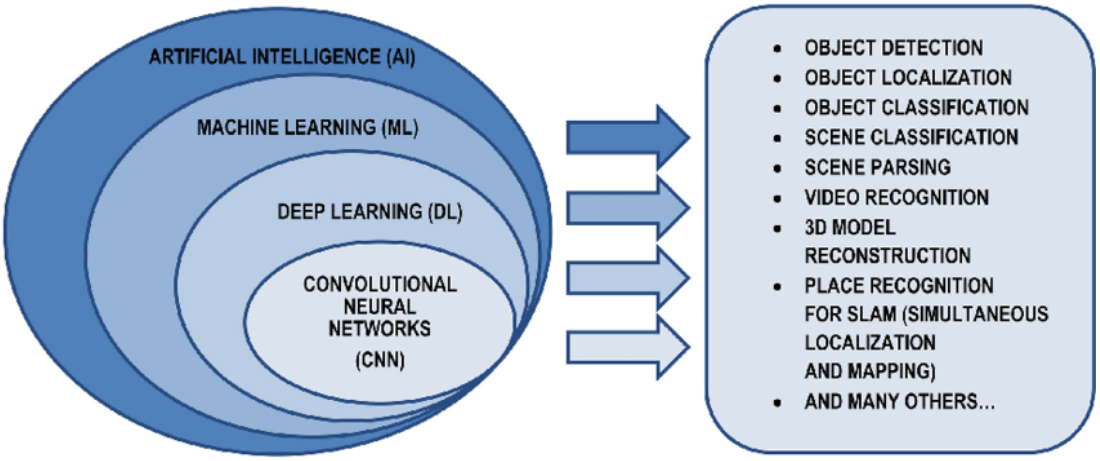

Figura 1. Clasificación de la Inteligencia Artificial [6]

Por último, cabe mencionar la existencia del concepto de Inteligencia Artificial General (AGI, por sus siglas en ingles) que sería un tipo de Inteligencia Artificial capaz de comprender o aprender cualquier tarea intelectual que un ser humano puede hacer. Por otro lado, hay gente que considera que alcanzar esta generalidad sería conseguir una Inteligencia Artificial consciente. Hay algunas polémicas respecto a si esto se podrá conseguir algún día [7].

3.2. *Machine Learning*

El *Machine Learning* es un campo de la Inteligencia Artificial que consiste en encontrar la capacidad de las máquinas de imitar la forma de aprender de los humanos, realizando tareas complejas mientras aprenden de la experiencia.

Dentro de la Inteligencia Artificial, y más concretamente del *Machine Learning*, existe una gran variedad de técnicas que se pueden aplicar a diferentes campos. A su vez, estas técnicas se agrupan en tres grandes campos que son los siguientes:

- **Aprendizaje supervisado:** Consiste en el uso de etiquetas que permitan cotejar los resultados del algoritmo utilizado con los reales, permitiendo a éste ajustar los pesos correspondientes en caso de acertar o fallar. Algunas de las técnicas que utilizan este tipo de aprendizaje son la regresión logística o las máquinas de soporte vectorial.
- **Aprendizaje no supervisado:** Este tipo de algoritmos no cuentan con etiquetas u otra forma de saber previamente qué datos son más cercanos a otros. Es por esto por lo que este tipo de aprendizaje aparece cuando se buscan patrones escondidos en los datos, permitiendo encontrar estos sin necesidad de intervención humana. Un ejemplo de este campo sería el *clustering.*
- **Aprendizaje semisupervisado (o aprendizaje reforzado):** Este campo mezcla las técnicas utilizadas en los dos anteriores, permitiendo resolver los posibles problemas que provoquen los datos etiquetados a medias. La idea es que el modelo utilice ejemplos etiquetados similares a los no etiquetados con el fin de poder extrapolar la información dada en los etiquetados, con los no etiquetados. Esto permite reducir el conjunto de datos etiquetados que es más costoso de conseguir.

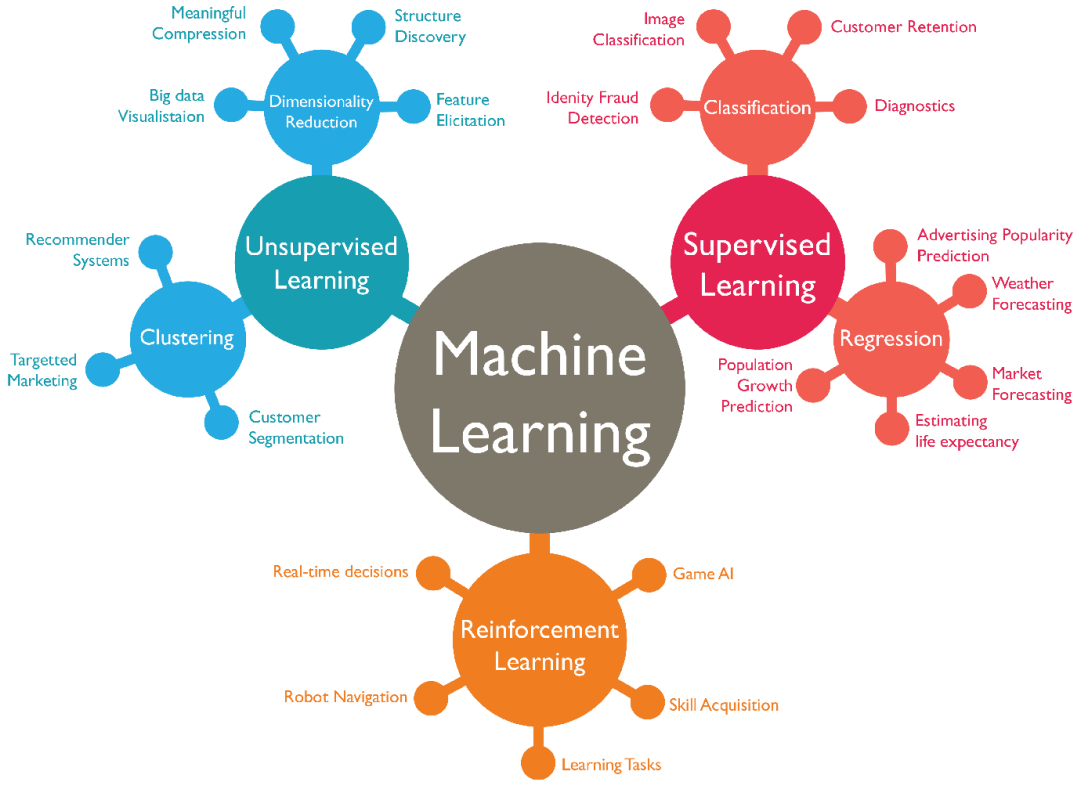

Figura 2. Clasificación del Machine Learning [8]

Con el fin de ilustrar la división por categorías, además de algunos ejemplos de uso de cada una de ellas se presenta la *Figura 2*. Las definiciones dadas anteriormente se han recogido de las siguientes fuentes [9]–[11].

Dentro del *Machine Learning* existe un conjunto de técnicas que se agrupan bajo el nombre de *Deep Learning*, y en las que se profundizará posteriormente.

Dentro del aprendizaje supervisado existen diversos conjuntos de algoritmos. Algunos de estos algoritmos son los utilizados durante este proyecto, sirviendo estos para clasificar.

3.2.1. Clasificación

La clasificación es una forma de reconocimiento de patrones para predecir una clase dados unos datos de entrada.

La idea de esto subyace en reconocer patrones comunes a todos los elementos de una clase, conociendo así dicha clase y pudiendo predecir si una nueva instancia pertenece

a ésta. Para ello, es posible recurrir a técnicas de *Machine Learning* o *Deep Learning* tales como las redes neuronales, las cuales se tratarán más adelante.

Cabe destacar que la clasificación puede dividirse en dos subtipos: binaria, donde la clasificación se realiza entre dos clases diferenciadas, y multiclase, donde la clasificación se realiza entre varias clases diferentes. Un ejemplo de clasificación, multiclase en este caso, se muestra en la *Figura 3* a continuación.

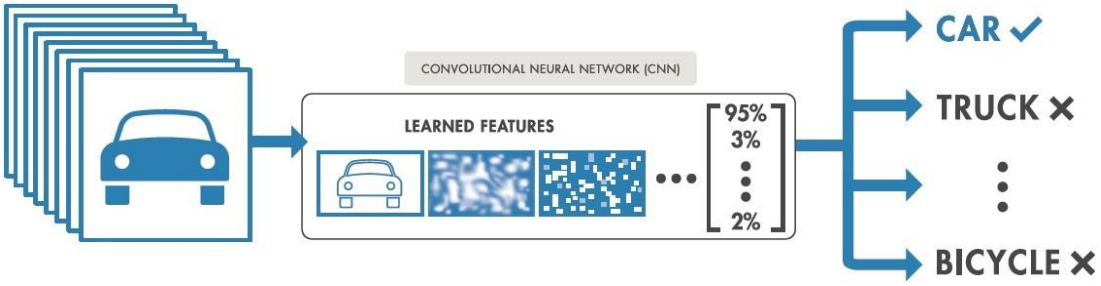

Figura 3. Ejemplo de una clasificación utilizando Deep Learning [12]

3.3. *Deep Learning*

El *Deep Learning* se suele enmarcar dentro del *Machine Learning.* El *Deep Learning* aglutina una serie de técnicas que utilizan como base mayoritaria el concepto de las redes neuronales artificiales o ANN por sus siglas en inglés, si bien también cuenta con otras técnicas.

Para entender lo que es una red neuronal artificial hay que comprender previamente lo que es una neurona artificial.

Las neuronas artificiales, también conocidas como perceptrones, serán funciones matemáticas cuyo funcionamiento se basa en la forma de funcionar de las neuronas humanas.

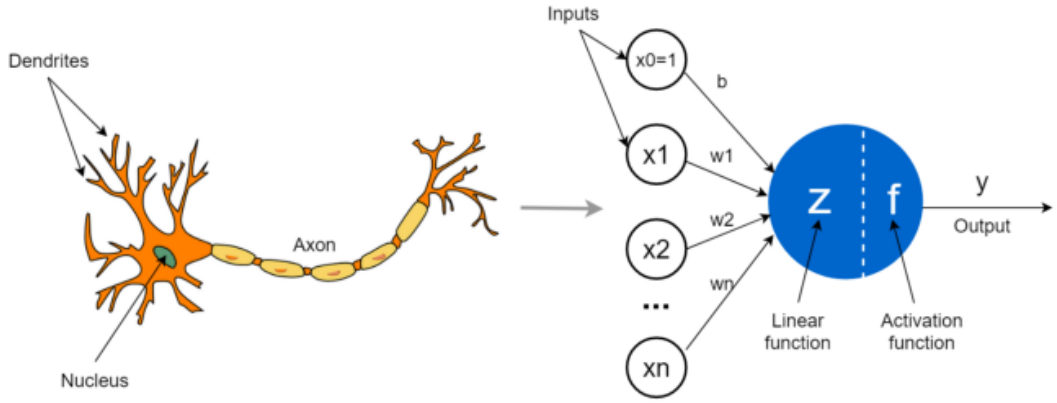

Figura 4. Comparativa entre una neurona y una neurona artificial [13].

Como se puede observar en la *Figura 4*, las neuronas artificiales tendrán una o varias señales de entrada (*inputs*), similares a las dendritas de una neurona; constarán también de la unidad de procesamiento en la que se encuentran las funciones lineales y de activación (en el capítulo *3.3.2. Funciones de activación* se trata en profundidad a estas últimas), realizando una función similar al núcleo de la neurona; y una señal de salida (*output*) que simula la función del axón.

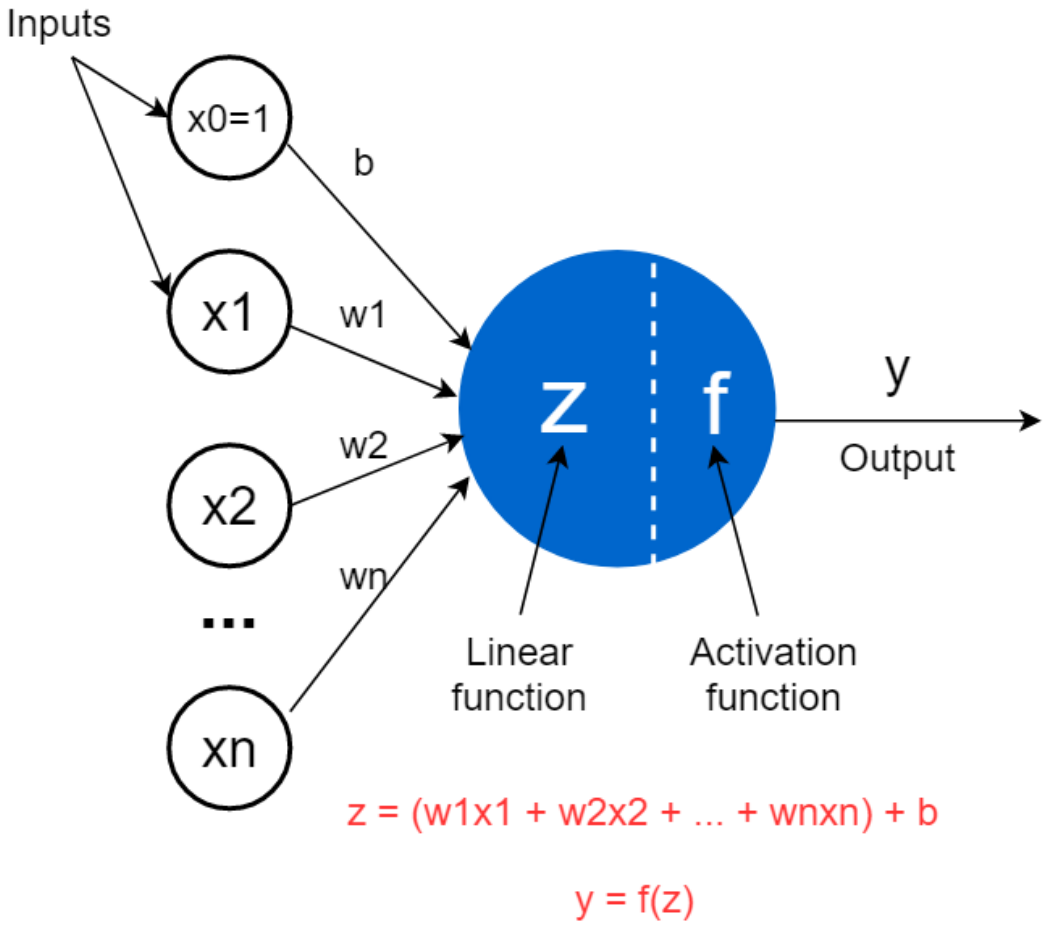

Figura 5. Imagen detallada de un perceptrón o neurona artificial [13].

En la *Figura 5* se presenta en profundidad un perceptrón. En primer lugar se pueden ver las diferentes señales de entrada; estas señales serán multiplicadas cada una por el peso que le corresponda. El peso será el valor que la red intentará optimizar de forma que cada entrada tenga la importancia que le corresponda. Una vez sumadas todas las componentes de entrada tras ser aplicados los pesos, se aplicará el *bias;* que actuará como el término independiente de la función, desplazándola.

Hasta ahora, todas las operaciones realizadas han sido lineales. Aquí entra la función de activación, que permitirá aportar la no linealidad aplicándose sobre la parte lineal ya calculada, si bien también podría aplicarse una función de activación lineal. Como se explica en el punto *3.3.2. Funciones de activación*, existen diversas funciones de activación.

En la siguiente *Figura 6* se puede comprobar como una red neuronal (convolucional, en este caso) identificará diferentes patrones presentes en la imagen de entrada a lo largo de las diferentes capas presentes.

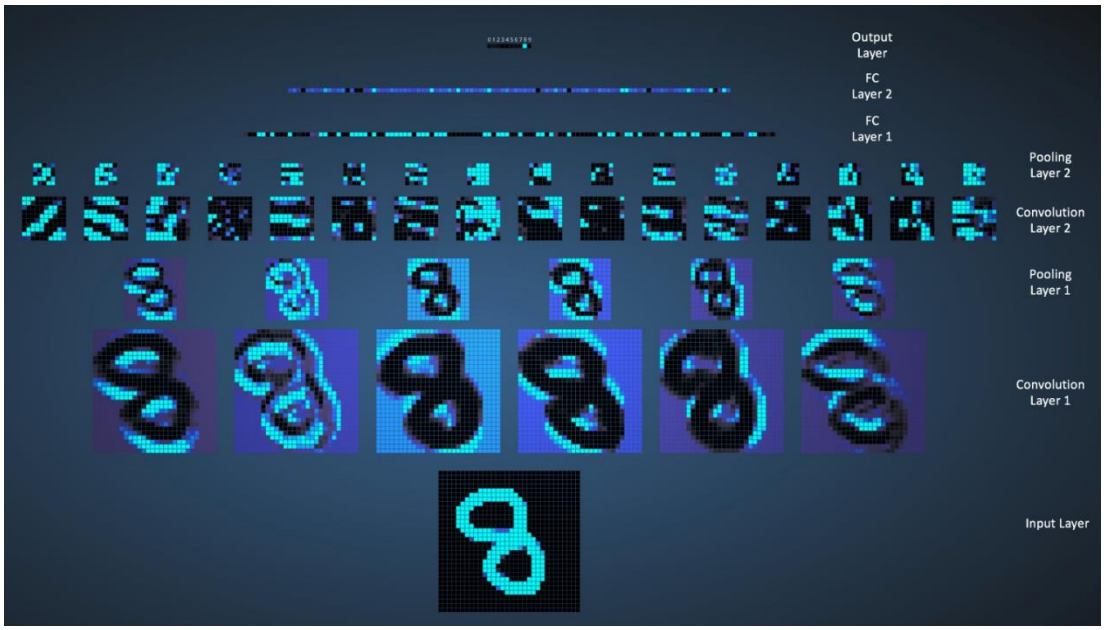

Figura 6. Ejemplo de salida de una red neuronal [14]

Dentro de las ANN existen grandes variantes que son utilizadas para tareas completamente diferentes entre sí. En este marco aparecen las redes neuronales convolucionales o CNN por sus siglas en inglés, explicadas a continuación.

3.3.1. Redes Neuronales Convolucionales (CNN)

Las CNN se diferencian de las ANN en que las segundas procesan cada píxel de manera independiente, perdiendo información textual por el camino. Esto provoca que las ANN no sean capaces de distinguir patrones en forma de bordes, contrastes, figuras geométricas y otro tipo de información textual disponible en las imágenes [15].

Los modelos de CNN sí que utilizan esta información contextual. Este tipo de red se basa en el funcionamiento de tres tipos distintos de capas: la capa convolucional, la capa de *pooling* y la capa totalmente conectada (*fully conected*). En la *Figura 7* se muestra un ejemplo de red neuronal convolucional.

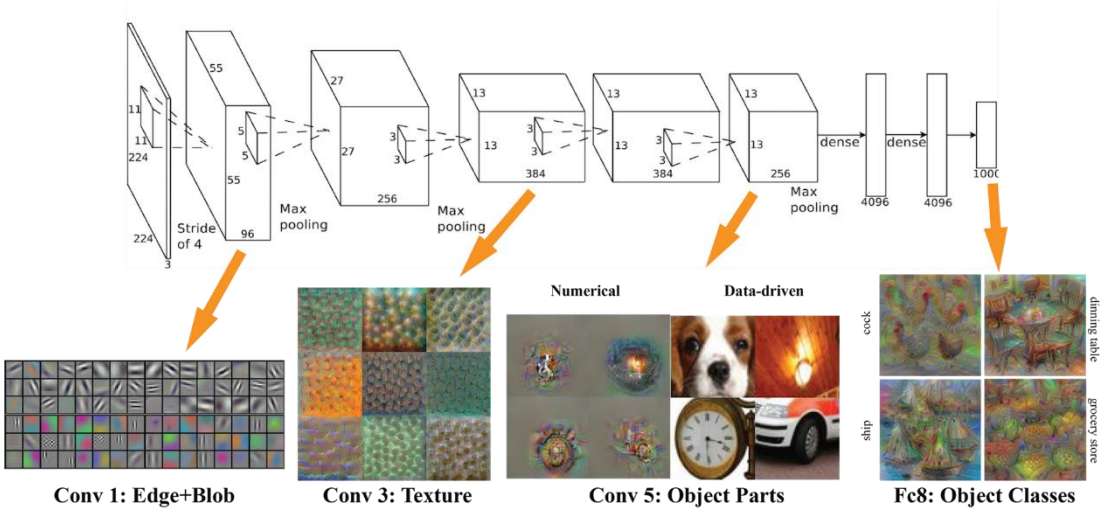

Figura 7. Ejemplo de funcionamiento de una CNN [16]

3.3.1.1. Capa convolucional

La primera capa de este tipo de red es la capa convolucional. Esta capa recibirá como entrada una imagen o un mapa de características, procesándolos y dando como salida otro mapa de características.

Un mapa de características será una imagen en la que se refleje la aplicación de un filtro concreto, explicado posteriormente.

Este proceso se realiza mediante los llamados filtros o *kernels*, donde se aplican de forma independiente en cada canal (una imagen en blanco y negro únicamente contará con 1, una en color RGB, con 3) y recibe el nombre de convolución.

Los filtros son matrices cuyo objetivo será detectar diferentes formas, bordes, cambios de contraste y cualquier otro tipo de patrón presentes en la imagen de entrada; con el fin de detectar estos patrones, la red ajusta los valores que componen el filtro a lo largo del entrenamiento, correspondiéndose cada conjunto de valores a un patrón diferente.

Aunque hay diferentes formas de inicializar dichos filtros, se suelen utilizar valores predefinidos obtenidos de una red ya entrenada, o valores aleatorios. Estos valores, como se muestra en la *Figura 8*, serán reducidos tras realizarse sobre ellos algunas operaciones, dependiendo también esta reducción del tamaño del *kernel* y el tamaño de los pasos dados por este último, pasos conocidos también como *stride*. La matriz reducida que resulta de este proceso es conocida como mapa de activación o *activation map*.

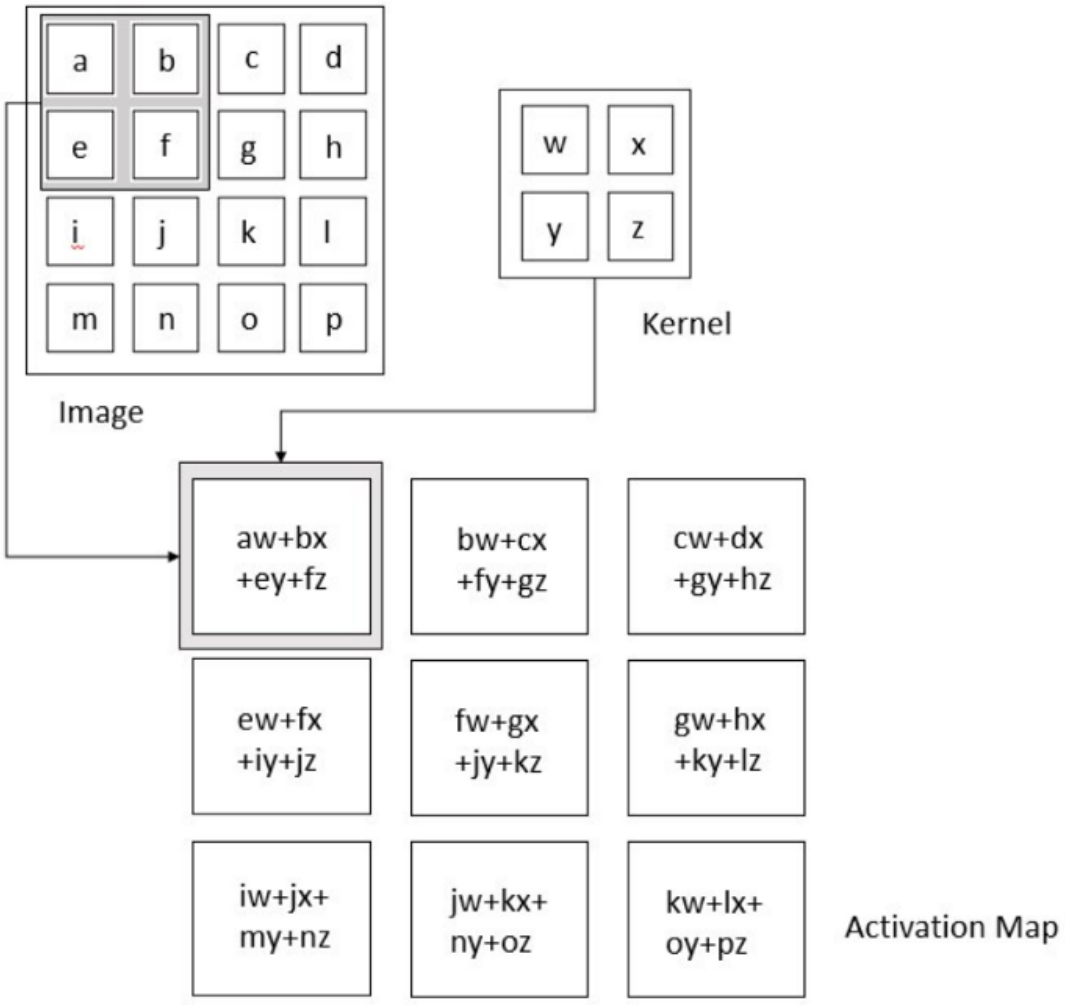

Figura 8. Ejemplo de funcionamiento de un filtro [17].

En la *Figura 9* se pueden observar las matrices de una imagen en blanco y negro (ya que sólo dispone de un canal) y de un filtro 3x3, donde los píxeles con valor 0 significarán apagado y los píxeles con valor 1 encendido.

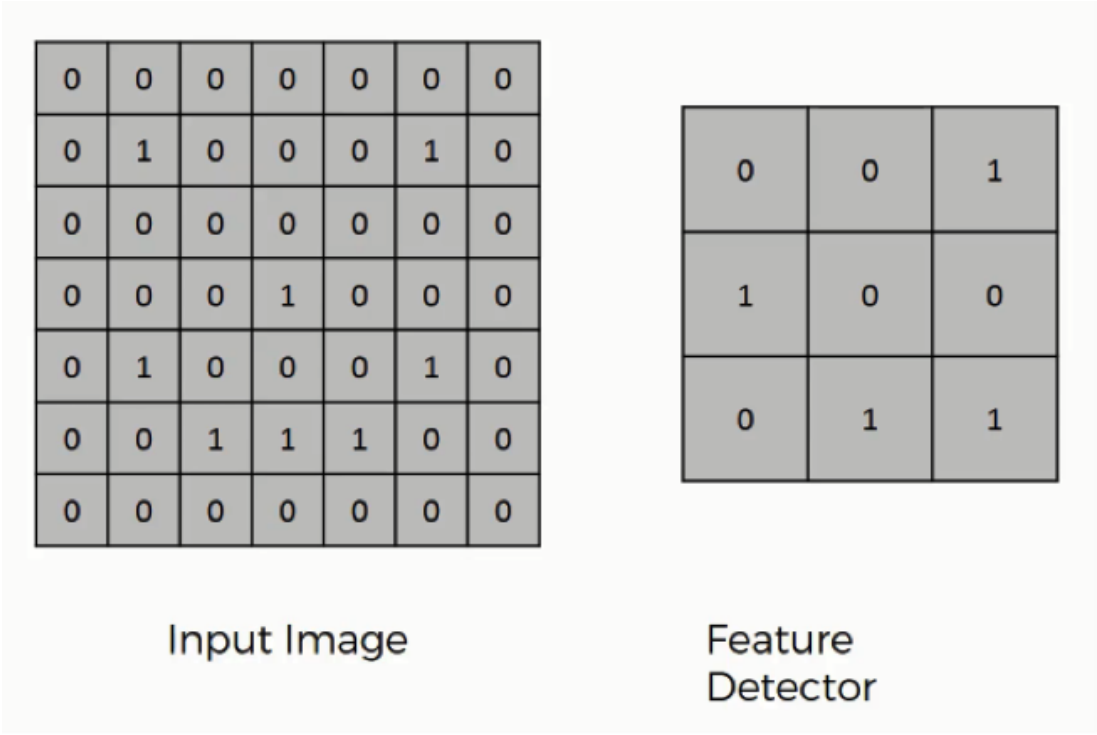

Figura 9. Matriz de píxeles de una imagen y un filtro [18]

Para realizar una convolución se sigue el proceso presentado anteriormente. El filtro recorrerá la imagen reconociendo si el patrón examinado existe área por área, realizándose una multiplicación entre sus valores y los de la propia imagen y dando lugar al mapa de características. Las características detectadas dependerán del filtro utilizado, como se puede ver en la *Figura 10*.

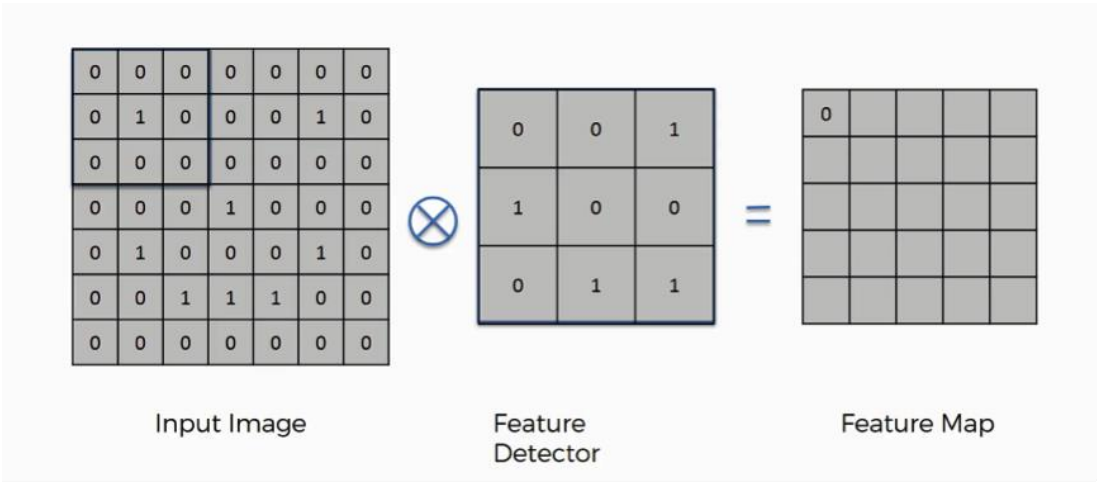

Figura 10. Ejemplo de procesado de un filtro [18]

Esto se repetirá sobre toda la matriz de la imagen de entrada comenzando por la esquina superior izquierda y acabando en la inferior derecha, realizando el movimiento en horizontal y vertical, respectivamente. Esto dará como resultado una nueva matriz como la mostrada en la *Figura 11*, representando el mapa de características completo.

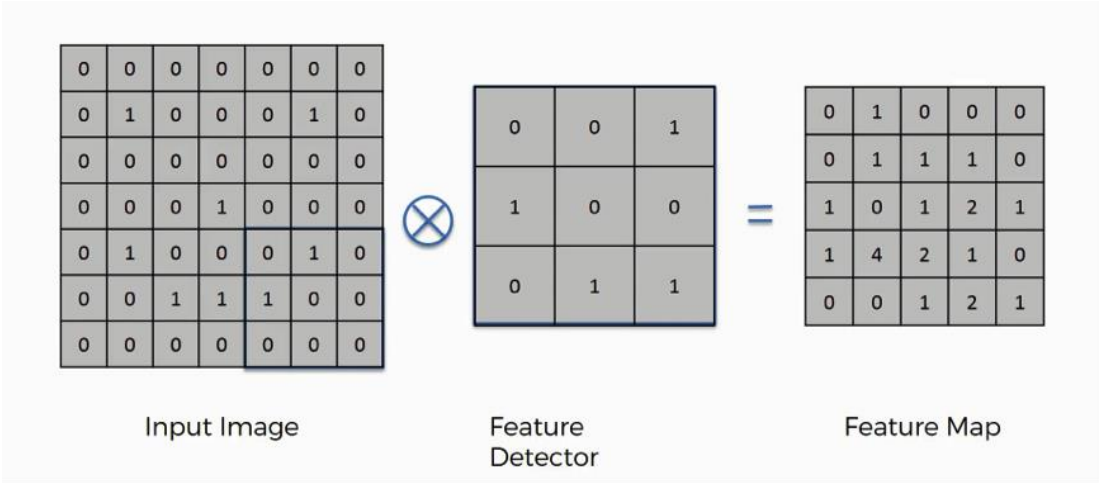

Figura 11. Imagen del procesado realizado por un filtro [18]

Según el filtro aplicado podrían darse diferentes mapas como salida; por ejemplo, un filtro que detecte los bordes dará como salida algo como lo que se puede ver a continuación en la *Figura 12*.

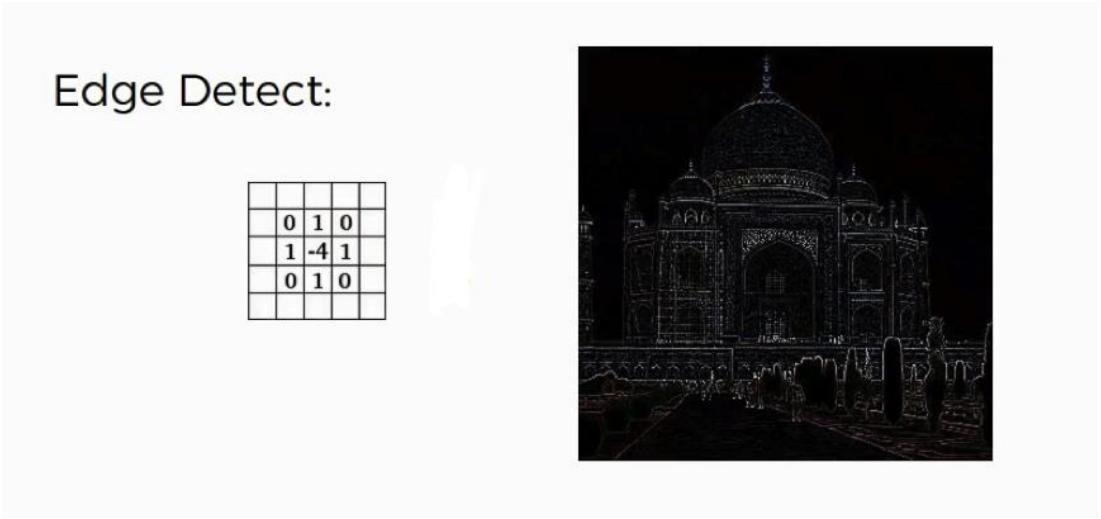

Figura 12. Aplicación de un filtro de detección de bordes [18]

3.3.1.2. Capa de Pooling

Esta capa se encarga de reducir la dimensionalidad de los mapas de características. Esto reduce el número de parámetros sobre los que se opera, haciendo la siguiente operación menos costosa. Además de esto, también resume las características presentes en cada uno de los mapas resultantes; es por esto por lo que se utiliza después de la convolución. Este resumen permitirá a la red ser más independiente de la posición en la que apareció la característica descubierta, impidiendo que la red asocie una posición a una característica concreta.

El *pooling* funciona de una forma similar a la operación de convolución [19]. Se pasa un filtro sobre cada uno de los mapas de características, siendo los más utilizados los filtros de 2x2. Los filtros realizarán las operaciones que correspondan sobre el llamado *receptive field* (esta será el área afectada por el filtro en una operación) según el tipo de *pooling* que se aplique, reduciendo su dimensionalidad.

Hay tres tipos diferentes tipos de *pooling:*

- ***Max pooling*:** Esta técnica consiste en, simplemente, seleccionar el mayor valor de todo el *receptive field* examinado, conservando las características más destacadas de la anterior capa. Es la técnica generalmente utilizada.
- ***Average pooling*:** Esta otra técnica realizará la media sobre los valores examinados por el filtro, conservando en este caso las características medias de la capa.
- ***Global pooling*:** Esta operación está pensada como sustitución de la capa completamente conectada en las redes convolucionales. Consiste en realizar una operación (ya sea el máximo, la media…) sobre un *receptive field* que ocupe todo el mapa de características, resultando en una única respuesta de tamaño 1x1 para todo el mapa.

El funcionamiento de esta capa se puede ilustrar con un ejemplo de la siguiente forma.

La operación se realiza de forma similar a la de convolución: se dividirá la entrada en áreas que se recorrerán en orden dando saltos del tamaño que indique el parámetro encargado de ello. Para cada área examinada se realizará la operación correspondiente dependiendo del tipo de *pooling* que se quiera aplicar.

Por un lado, si se aplica *max pooling*, este recuperará el mayor valor del área examinada, manteniendo la característica más prominente de toda el área. Por otro lado, el *average pooling*, realizará la media aritmética de los valores presentes, conservando la media de las características presentes en cada área. En la *Figura 13* se muestra un ejemplo de aplicación de distintos tipos de *pooling*.

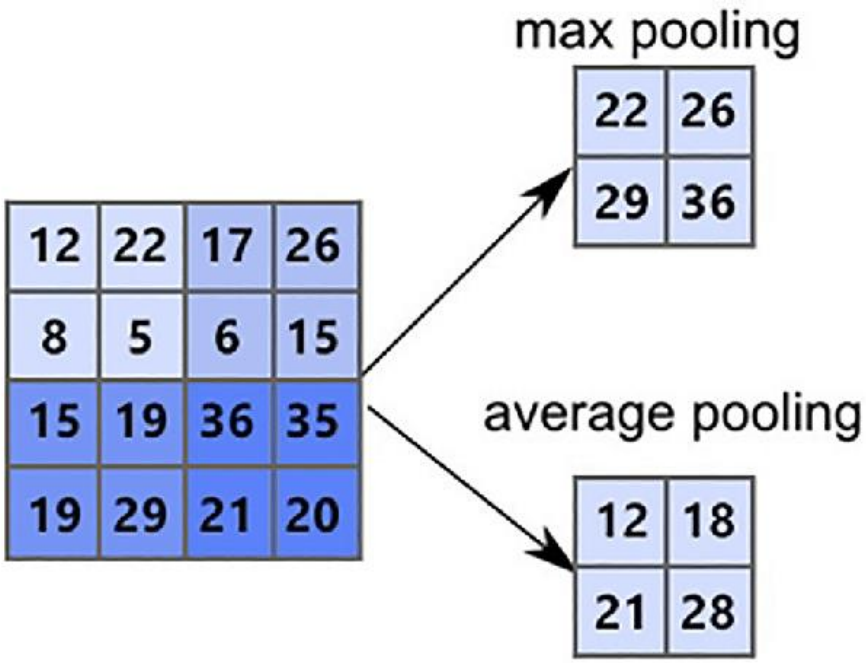

Figura 13. Aplicación de distintos tipos de Pooling [20]

3.3.1.3. Capa Flatten

El *flattening* es una técnica mediante la cual se convierten listas multidimensionales a listas de una sola dimensión. La capa de *flatten* utiliza esta técnica.

El *flatten* se aplica por diferentes razones, una de ellas es que las listas unidimensionales consumen menos memoria que las multidimensionales por lo tanto, en lo que a Inteligencia Artificial se refiere, esto conlleva un aumento en la velocidad además de la ya comentada optimización de la memoria.

Por otro lado, como se muestra en la *Figura 14*, en el caso de utilizar una capa totalmente conectada (explicada a continuación), dicha capa necesita una entrada unidimensional, siendo el *flatten* uno de los métodos más utilizados para llegar a esa lista unidimensional.

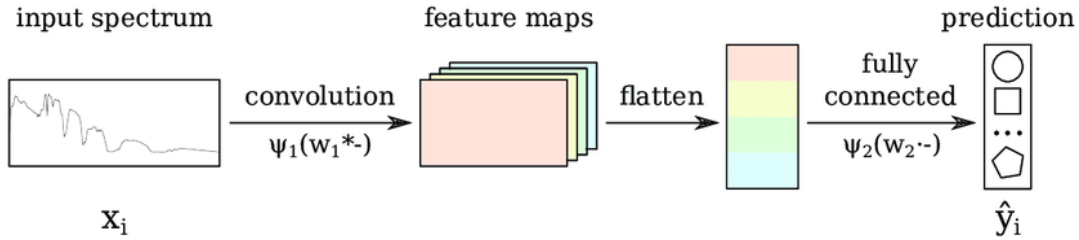

Figura 14. Ejemplo de funcionamiento de capa de flatten [21].

3.3.1.4. Capa Totalmente Conectada

Esta capa es similar a la homónima que aparece en las redes neuronales artificiales (ANN). Tendrá como entrada la salida proporcionada por las diferentes capas convolucionales y de *pooling*, mientras que dará como salida el cómputo final [15].

El objetivo de esta capa será hacer la predicción definitiva compilando los datos completos de la imagen extraídos por las capas anteriores, proporcionados gracias a la completa interconexión que existe en esta capa.

En la *Figura 15*, que se puede observar a continuación, se pueden ver las diferencias entre las capas convolucionales y totalmente conectadas.

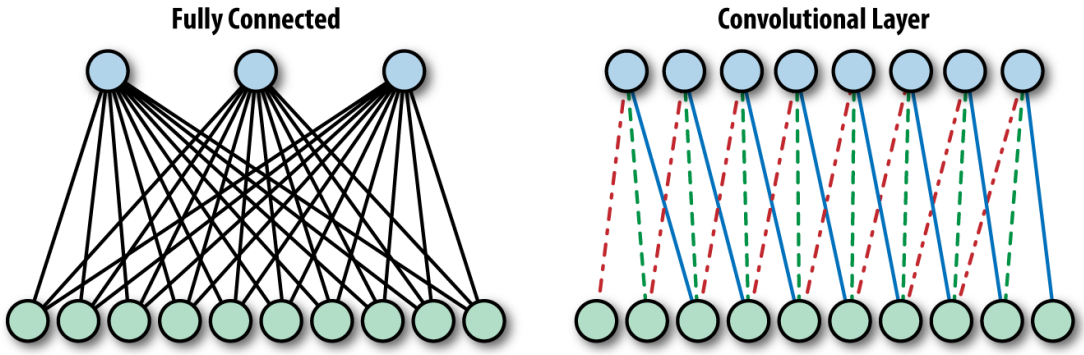

Figura 15. Diferencias entre capa convolucional y totalmente conectada [22]

3.3.1.5. Capa Dropout

La técnica conocida como *dropout* es una forma de evitar el sobre entrenamiento, más conocido como *overfitting*. Dicho problema aparece en el punto en el que la red es capaz de acertar todas las instancias de los datos de entrenamiento, pero no puede

generalizar. Esto ocurre debido a que la red ha memorizado de una forma tan profunda los datos de entrenamiento que es capaz de detectar cada mínimo detalle de estos, mientras que si le introduces un dato nuevo con patrones ligeramente diferentes a los aprendidos, no será capaz de inferir la clase del nuevo dato. Este comportamiento debe ser evitado ya que el objetivo final de un clasificador será el de tener la capacidad de clasificar nuevos datos no introducidos durante el entrenamiento con una precisión suficiente según la sensibilidad de los datos.

Sabiendo el problema que el *dropout* soluciona, hay que explicar cómo lo hace. Esta técnica, con el fin de evitar el problema ya mencionado, consistirá en eliminar algunas de las neuronas que actúan durante el proceso de entrenamiento durante un tiempo determinado [23]. Técnicamente, esto significa que, en lugar de realizarse un ajuste sobre todos los pesos en cada iteración, únicamente se realizará el ajuste de algunos (los que pertenezcan a las neuronas no afectadas por el *dropout*), ayudando a que no existan neuronas que dominen el proceso por tener pesos muy altos, así como eliminando la codependencia entre neuronas y ayudando a que la red aprenda las características más robustas.

Generalmente, el valor dado a este *dropout*, teniendo en cuenta que su rango puede variar entre 0 y 1, está por debajo de 0,5.

Con el fin de mostrar de forma visual el *dropout*, se muestra la siguiente *Figura 16*.

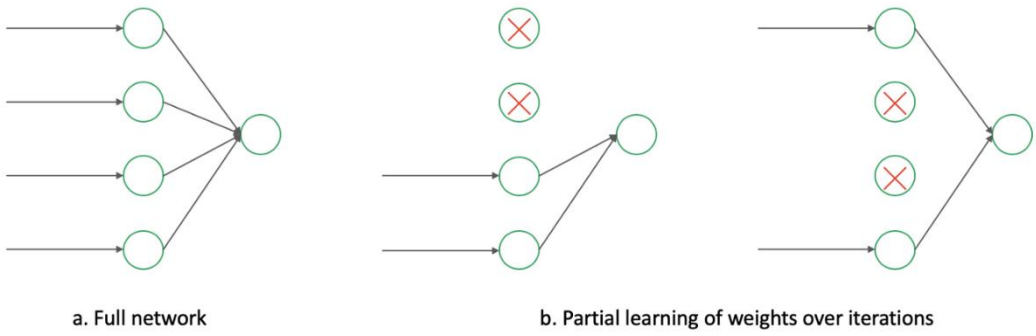

Figura 16. Ejemplo de dropout [24]

3.3.2. Funciones de activación

Dentro del funcionamiento de las redes neuronales, se han mencionado las funciones de activación. Aquí se comentará en profundidad de qué se encargan, así como algunos de los diferentes tipos que existen.

Las funciones de activación son, por lo general, el componente que permite a una red neuronal volverse un modelo no lineal, ayudando a que la red consiga converger y facilitando dicha convergencia. Además, la no linealidad permite que una red neuronal pueda establecer complejas relaciones entre las entradas y las salidas de cada neurona [25].

Como se ha mencionado varias veces, hay diferentes funciones de activación; aquí se tratarán algunas de ellas. Todas las fórmulas se representarán en función de x con el fin de facilitar su entendimiento.

- **Función lineal:** Como ya se dijo, es posible utilizar una red neuronal como un modelo lineal, para ello, simplemente se necesita contar con una función de activación lineal, si bien no es lo más común, es una posibilidad.

  La fórmula que se corresponde con dicha función se muestra a continuación, y se representa de forma gráfica en la *Figura 17*.

$$f(x) = \mathrm{x}$$

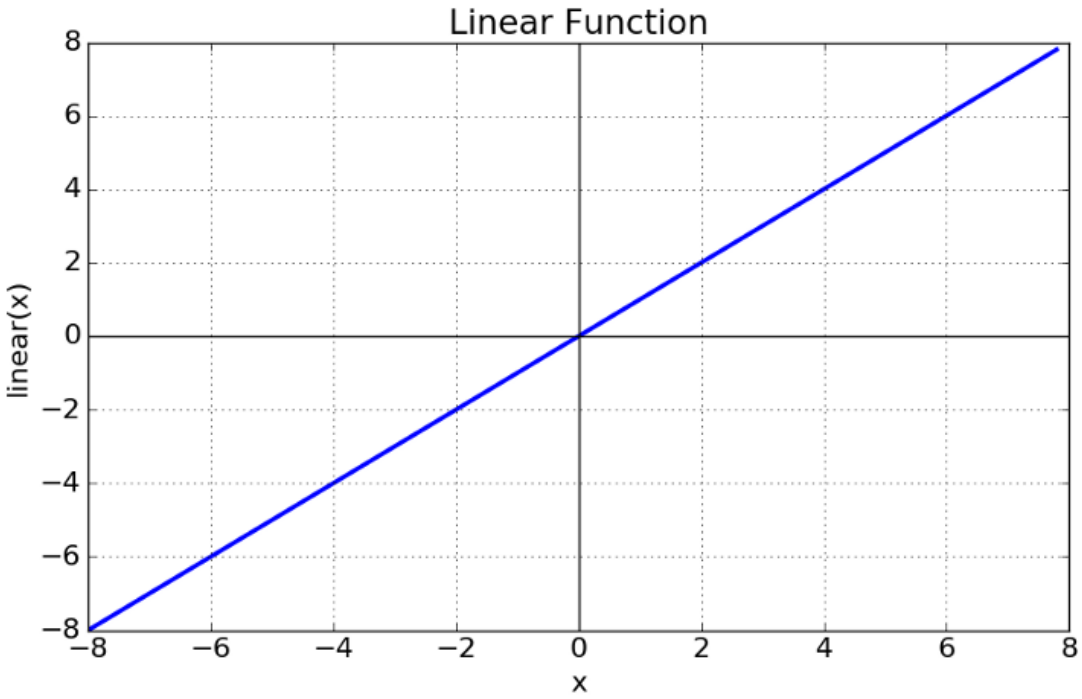

Figura 17. Ejemplo de función de activación lineal [26].

- **Funciones no lineales:** Las funciones que se suelen ver en la mayoría de los casos son las no lineales, estas transformarán las redes en modelos no lineales con gran capacidad de aprendizaje. Dentro de estas funciones existen diversos tipos, algunos de ellos son los siguientes.

    1. **Sigmoide:** Esta función transforma cualquier entrada en una salida entre 0 y 1, convirtiéndola en una función extremadamente útil en el caso de que se quiera como salida una probabilidad. Su fórmula será la siguiente.

$$f(x) = \frac{1}{1 + e^{-x}}$$

    2. ***Softmax*:** La función S*oftmax* será una generalización de la función sigmoide utilizada en los casos donde se necesite como salida una probabilidad, pero el problema sea multiclase. De la misma forma que la sigmoide, transformará la entrada a una salida entre 0 y 1. Su formulación es la siguiente.

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

En ambos casos, la gráfica que representa la función será la siguiente (*Figura 18*).

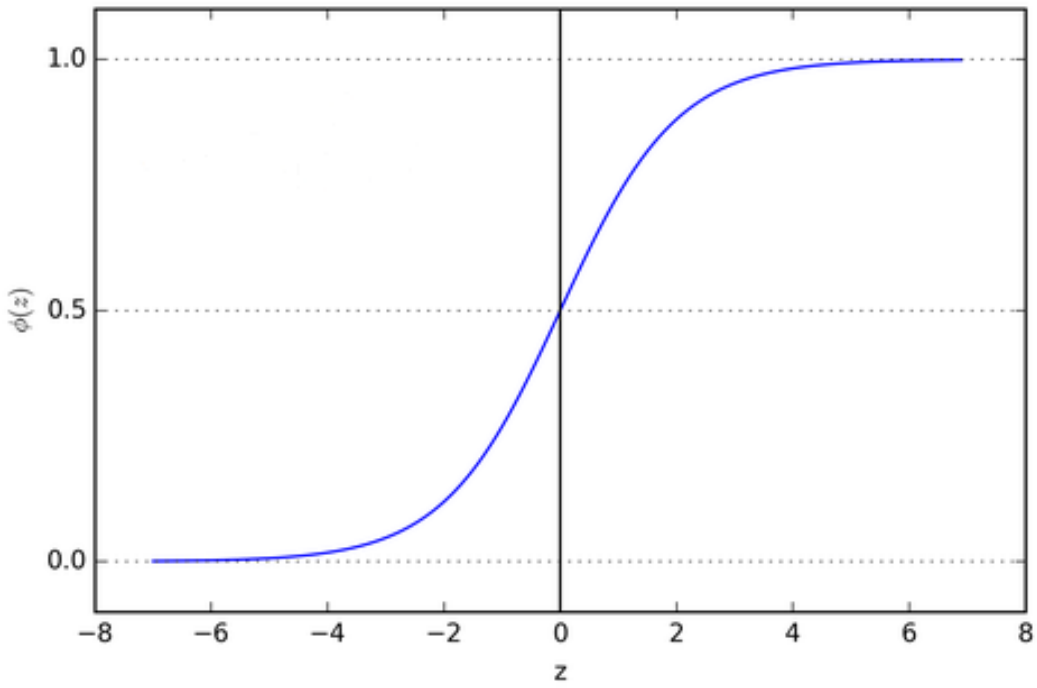

Figura 18. Ejemplo de función de activación sigmoide/softmax [26].

3. ***ReLU:*** La función R*eLU (Rectified Linear Unit)* suele ser la función más utilizada, principalmente porque se utiliza ampliamente en contextos como las redes neuronales convolucionales. Esta función actúa como una función lineal para las entradas positivas, actuando a su vez como no lineal para las negativas. Siendo una función simple, permite un procesado rápido sin necesidad de complejos cálculos; además, también permite tener como salida ceros. Esto último es un cambio respecto a las demás funciones de activación, que suelen obtener ceros relativos, esto es, un número prácticamente cero pero con una gran cantidad de pequeños decimales.

Es representada por la siguiente ecuación, y su gráfica se muestra en la *Figura 19*.

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

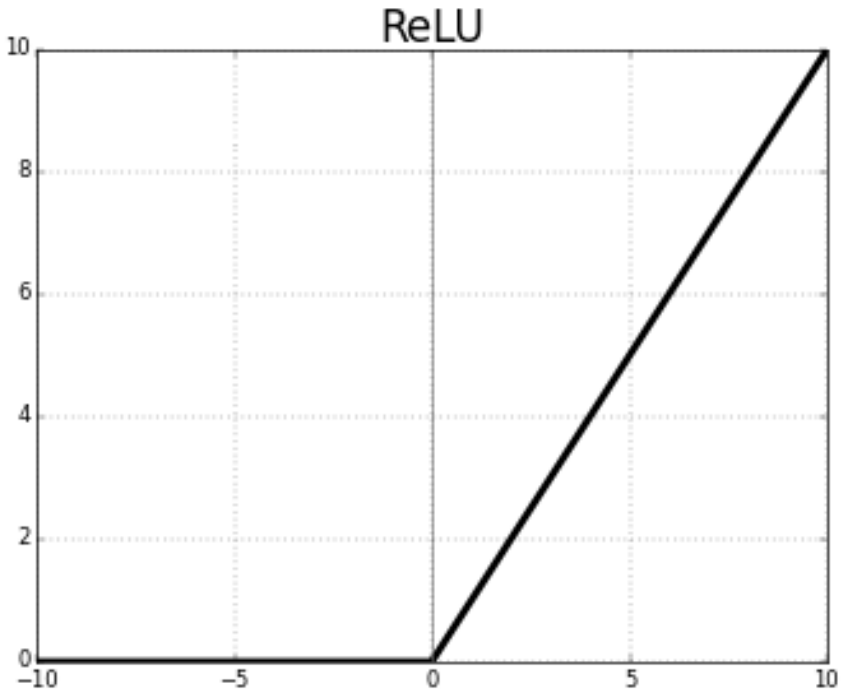

Figura 19. Ejemplo de función de activación ReLU [26].

3.3.3. *Transfer Learning*

Por otro lado, la técnica conocida como *Transfer Learning* es una de las tantas existentes bajo el amparo del *Machine Learning*, y suele ser utilizada con el fin de mejorar los entrenamientos, especialmente los realizados con recursos más limitados.

Esta técnica se diseñó para conseguir una alta precisión en modelos mediante la utilización de un conjunto de datos más limitado, ya sea en calidad o cantidad, así como para cuando el entrenamiento debe ser más corto que en una situación convencional, ya sea por limitaciones técnicas o por falta de tiempo.

Para conseguir unos buenos resultados teniendo en cuenta estas limitaciones, el *Transfer Learning* consiste en utilizar un modelo preentrenado en otra tarea similar a

la que se quiere realizar, volviendo a entrenar únicamente sus últimas capas afinándolo así para la nueva tarea.

Se ha llegado al punto en el que, por lo general, no se suelen desarrollar modelos desde cero para la clasificación de imágenes o procesamiento del lenguaje natural, sino que se utiliza esta técnica. Además, esto permite reducir el coste computacional del entrenamiento ya que hay muchos parámetros "congelados", significando esto que no se entrenarán más, reduciendo el número de parámetros que se deben entrenar, facilitando, como ya se ha mencionado, el entrenamiento en caso de limitaciones temporales.

En la *Figura 20* se ilustra lo aquí comentado.

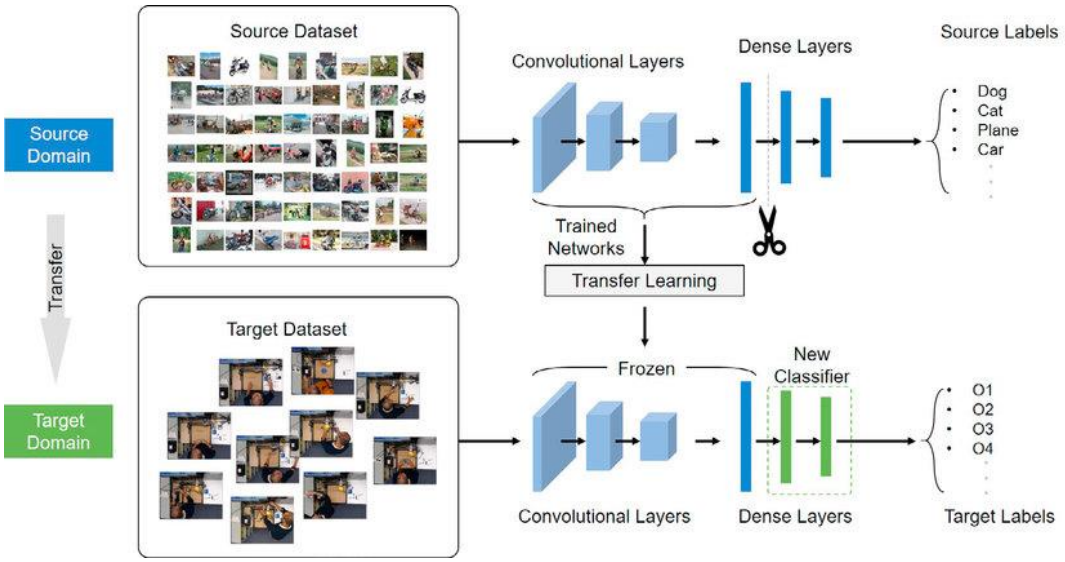

Figura 20. Ejemplo de funcionamiento del Transfer Learning [27].

3.4. Conceptos generales utilizados durante el desarrollo

En este primer punto se van a explicar algunos conceptos transversales a esta clase de tecnologías, necesarios para entender el contexto en el que se realiza el proyecto y utilizados durante el desarrollo del trabajo que aquí se presenta.

3.4.1. Conjunto de entrenamiento, validación y test

Dentro de determinados tipos de desarrollo en el mundo de la Inteligencia Artificial, se utilizan diferentes conjuntos de datos conocidos como conjuntos de entrenamiento, validación y test. Estos tres conjuntos serán necesarios para entrenar, evaluar, y comparar de forma precisa el rendimiento del modelo; además de impedir problemas como el *overfitting* tratados en profundidad más adelante. Cada uno de estos conjuntos cumplirá una de las funciones necesarias:

- **El conjunto de entrenamiento** está formado los datos que vayan a ser utilizados con el fin de crear y mejorar el modelo, así como enseñarle nuevas características y patrones, con el fin de que incremente su rendimiento. Por lo general suele ser el conjunto mayoritario.
- **El conjunto de validación** permitirá evaluar el modelo. Los datos que forman este conjunto son utilizados también para optimizar los hiperparámetros, por lo que el modelo podrá ver estos datos pero nunca aprenderá de ellos. Este conjunto también es utilizado para detectar el *overfitting* que podría ocurrir durante el entrenamiento del modelo. El tamaño de este conjunto será menor que el de entrenamiento.
- **El conjunto de test** suele ser el último utilizado, ya que se utiliza para evaluar el modelo una vez finalizado el entrenamiento completo. Normalmente cuenta con ejemplos en los que se den situaciones que podrían ocurrir en la vida real, con el fin de evaluar correctamente la capacidad de generalización del modelo entrenado. Este conjunto se utiliza también para comparar el rendimiento de diferentes modelos sobre una misma tarea, por ejemplo, en competiciones.

Mediante el uso de estos tres conjuntos se permitirá un correcto entrenamiento y una correcta validación del modelo.

3.4.2. Sesgo y varianza

Dentro de los clasificadores, generalmente se conocen diferentes tipos de error. Uno de ellos es el tipo irreducible, esto es, un error inherente a los modelos debido a variables desconocidas y que por tanto no podrán ser evaluados y reducidos; otro tipo de error es el reducible, estos errores se pueden reconocer y mejorar con el fin de que

el modelo rinda mejor. Tanto el sesgo como la varianza pertenecerán al segundo tipo de errores. Esto se puede ver en la *Figura 21*.

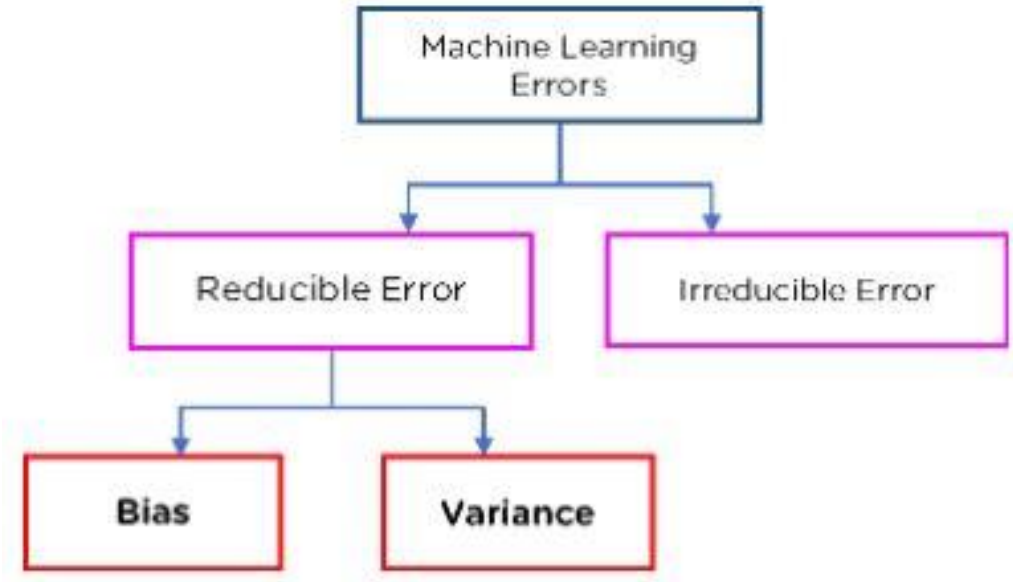

Figura 21. Tipos de error en Machine Learning [28]

Estos dos errores son inversamente proporcionales, es decir, si uno de ellos se incrementa, el otro decrementará. Ambos forman parte de los errores más comunes por lo que se explican en profundidad a continuación:

- **Sesgo**: Técnicamente, este error se suele definir como el error que existe entre los valores predichos y los valores reales [29]. Si se valora más en profundidad, este error aparece según las asunciones que el modelo haga sobre los datos de entrada. Siendo esto así, en el caso de que el sesgo sea muy alto se puede considerar que el modelo entrenado está asumiendo patrones muy simples, pasando por alto algunas de las características más específicas y que diferenciarán esta tarea de otras similares. Cuando esto ocurre se le denomina *underfitting*, haciendo referencia a que el modelo necesita un mejor entrenamiento. Un ejemplo de lo que es el *bias* y los efectos del *overfitting* se presentan en la *Figura 22*.

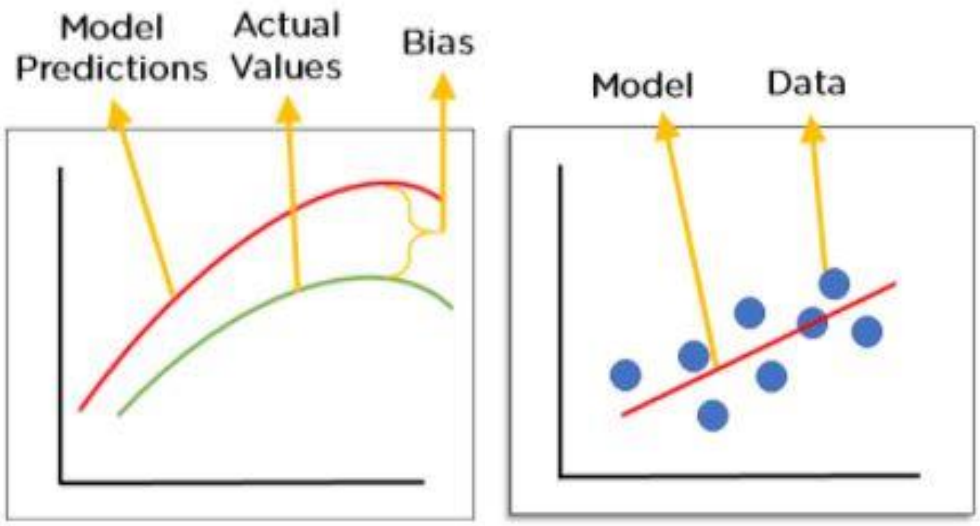

Figura 22. Ejemplo de bias (sesgo) y underfitting [28].

- **Varianza**: La varianza mide la variabilidad de las predicciones realizadas [29]. Dicho de otra forma, la varianza mide la diferencia entre el valor precedido y el real. Por un lado, en el caso de que la varianza sea baja, cualquier cambio en los datos podría suponer un cambio en el modelo mientras en el caso contrario, de varianza alta, sería muy difícil que el modelo cambie necesitándose un gran cambio en los datos para que esto ocurra.

  Existen modelos que por su forma de entrenar constan de mayor o menor varianza, de esta forma, los árboles de decisión o las máquinas de soporte vectorial constarán de una mayor varianza, siendo las regresiones ejemplo de baja varianza.

  Como se ha dicho, en el caso de que la varianza sea alta, es muy difícil que un modelo cambie. Esto puede provocar el problema conocido como *overfitting* o sobre entrenamiento, donde el modelo se adapta muy bien a los datos de entrenamiento pero falla en replicar lo aprendido con ejemplos fuera de estos como se puede ver en la *Figura 23*.

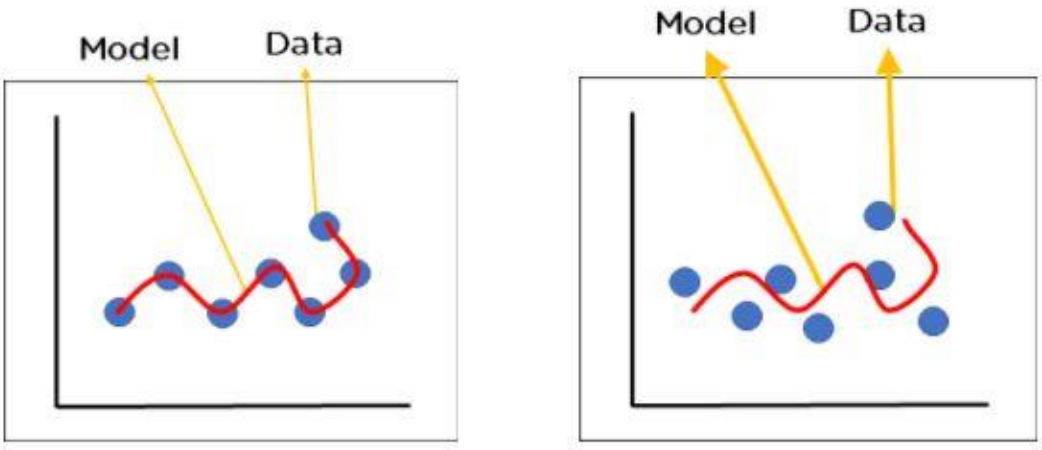

Figura 23. Ejemplo de overfitting en datos de entrenamiento (izquierda) y datos nuevos (derecha) [28].

3.4.3. Hiperparámetros

En el contexto en el que se sitúa este proyecto, los hiperparámetros son los parámetros existentes en los modelos que son establecidos por el usuario. Existen otros muchos parámetros que el propio modelo deberá inferir y ajustar durante el entrenamiento con el fin de obtener los mejores resultados. Así, los hiperparámetros serán definidos en el proceso de instanciación de un modelo, definidos explícitamente por el usuario antes de la creación de dicho modelo. Estos parámetros pueden ser pensados como las características que describen cómo se quiere que funcione el modelo creado [30].

3.4.3.1. Diferencia entre parámetros e hiperparámetros

Aunque en ocasiones se utilicen indistintamente, lo cierto es que existe una diferencia entre parámetros e hiperparámetros en el mundo del *Machine Learning*. En la *Tabla 2* se especifican algunas de las diferencias más importantes:

Parámetro	Hiperparámetro
Valor obtenido automáticamente durante el entrenamiento con los datos correspondientes.	Valor especificado manualmente de forma previa a la creación del modelo.
Es una parte del modelo.	Es externo al modelo.

El valor obtenido se guarda como parte del modelo entrenado.	Al no ser una parte del modelo no será necesario guardarlo junto a él más que por razones de trazabilidad.
Dependiente de los datos de entrenamiento.	Independiente de cualquier conjunto de datos.

Tabla 2. Diferencias entre parámetros e hiperparámetros [31].

3.4.3.2. Tipos de hiperparámetros

Los hiperparámetros además se dividen en dos clases diferenciadas. En primer lugar, existen los hiperparámetros para la optimización, conociéndose el resto de estos como hiperparámetros para modelos específicos. Si bien los nombres son descriptivos, se explicarán algunos de los más importantes así como su finalidad:

- **Hiperparámetros para la optimización:** Son los parámetros que, como indica su nombre, serán utilizados con el fin de optimizar el funcionamiento del modelo. Es sobre los que se aplicará el proceso de optimización de hiperparámetros del que se habla en *6.3. Ajuste de hiperparámetros*. Algunos de estos parámetros son los siguientes:
    - ***Learning rate* o tasa de aprendizaje:** Este parámetro se encarga de controlar la cantidad que se ajustarán los pesos de la red para reducir el error. Esto significa que en el caso de ser demasiado pequeño se podría tardar demasiado en llegar a un valor óptimo, así como en caso de ser demasiado grande se podría saltar el punto óptimo resultando en un ajuste malo. Es por ello por lo que se deben buscar valores intermedios mediante la optimización mencionada. A continuación, en la *Figura 24* se encuentra un ejemplo de un valor bajo, en la *Figura 25* el ejemplo de un valor demasiado alto, y en la *Figura 26* un valor intermedio y correcto:

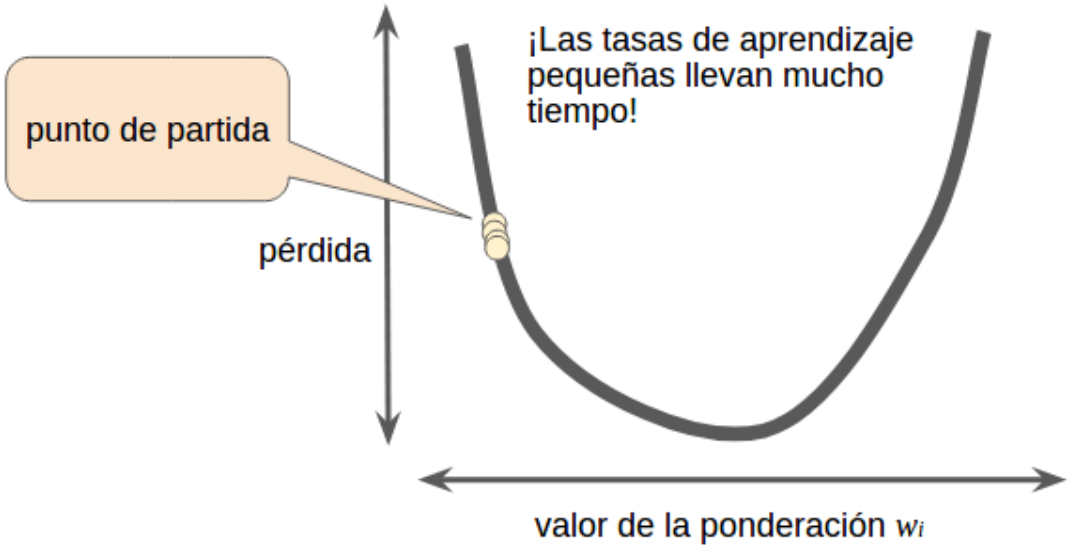

Figura 24. Ejemplo de learning rate pequeño [32].

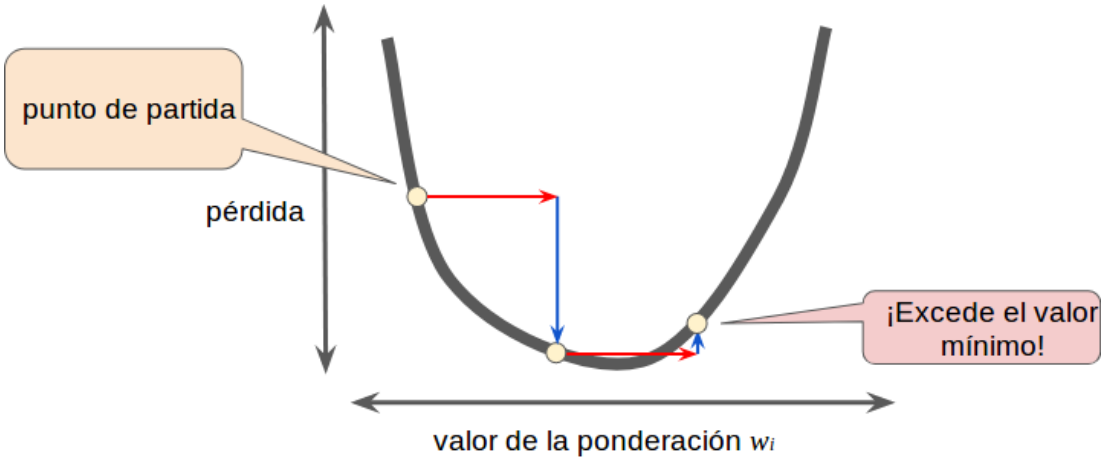

Figura 25. Ejemplo de learning rate grande [32].

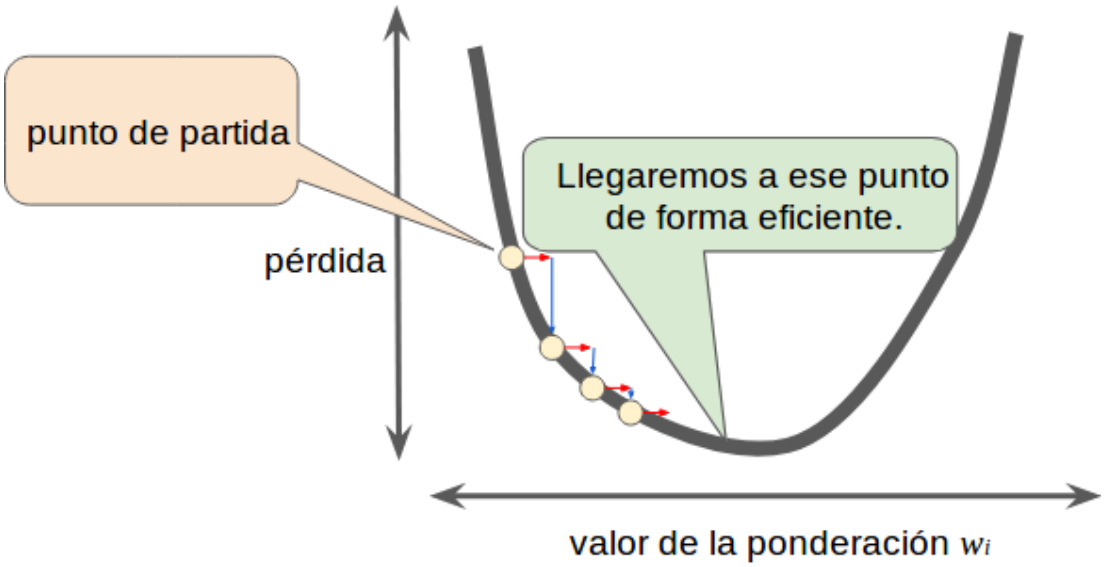

Figura 26. Valor intermedio de learning rate [32].

- ***Batch size* o tamaño del lote:** El valor de este parámetro definirá cuántos datos se pasarán al modelo a la vez. Si bien pensando de forma simple se llega a la conclusión de que cuanto mayor sea este número menos durará el entrenamiento, cosa que es cierta, tiene sus desventajas. El hecho de incrementar demasiado este valor puede llevar a que el entrenamiento se realice de forma imprecisa, provocando que el modelo no pueda generalizar correctamente lo que está aprendiendo. Es por ello por lo que continúa siendo importante encontrar un valor intermedio para este parámetro.
- **Número de *epochs* o número de épocas:** Por último, una *epoch* se define como un recorrido completo de todos los datos a través del modelo. Esto significa que cada *epoch* que sea completada, todos los datos de entrenamiento habrán sido aprendidos por el modelo una vez, repitiéndose este proceso hasta el número indicado en este hiperparámetro. Este número es importante, ya que en caso de ser demasiado bajo podría provocar que los datos no fueran aprendidos completamente, resultando en un modelo simplista; mientras que en caso de ser demasiado grande provocaría el ya comentado *overfitting*.

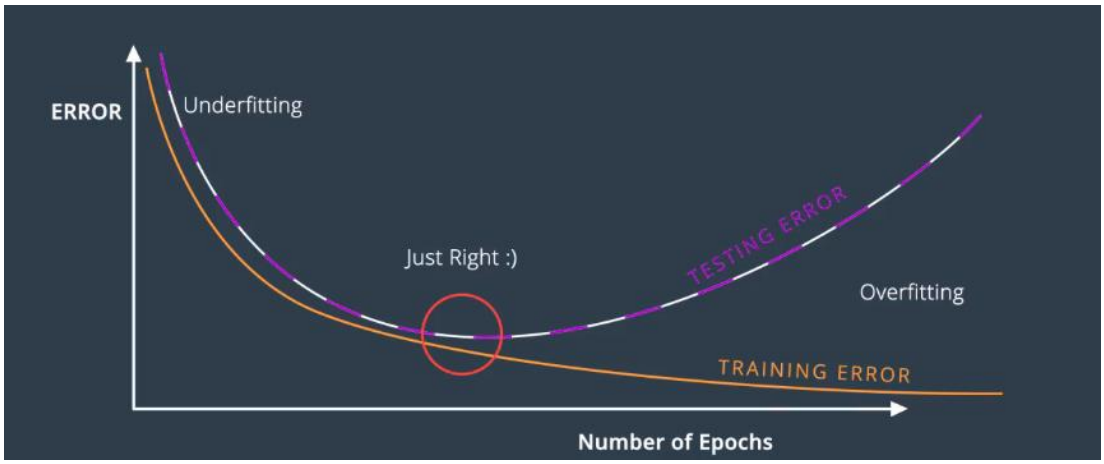

Figura 27. Ejemplo de un óptimo número de epochs [33].

En la *Figura 27* se puede ver lo mencionado anteriormente. Al comienzo del entrenamiento el error se va reduciendo con el tiempo, atravesando la fase de *underfitting* donde el modelo no habrá conseguido aprender todo lo que los datos pueden aportar. En el punto medio se produce el cambio de tendencia, indicando que ahí es el lugar donde se debería parar el entrenamiento, pasando a la fase de *overfitting* en la que el error de entrenamiento seguirá descendiendo mientras que el de test comenzará a ascender.

- **Hiperparámetros para modelos específicos**: Son los parámetros envueltos en la construcción de la estructura del modelo, es decir, que diferenciarán un modelo de otro. Algunos parámetros de esta clase son los siguientes:
    - **Número de capas y tipo de estas:** Cada una de las capas será un componente que, junto al resto de estos, formará el modelo completo. Es obvio que un modelo con más capas generalmente obtendrá mejores resultados, si bien no es lo único importante a la hora de desarrollar uno, pudiendo obtener peores resultados. Además, cabe tener en cuenta el hecho de que, a un mayor número de capas, el entrenamiento será más largo y costoso, pudiendo hacerse imposible según la disponibilidad técnica de cada usuario.
    - **Número de unidades ocultas:** Las unidades ocultas serán los componentes que se encuentren entre la entrada y salida de una unidad.

Esto definirá la capacidad del modelo. En lo que se va a aplicar durante el proyecto (desarrollado utilizando redes neuronales convolucionales), estas unidades serán los filtros utilizados durante las convoluciones.

3.4.3.3. Optimización de hiperparámetros

La optimización de hiperparámetros consiste en realizar una búsqueda entre diferentes valores con el fin de encontrar los óptimos para los hiperparámetros que mejoren el resultado. Esto puede marcar la diferencia entre un modelo decente y uno realmente bueno, por lo que es una parte clave en todo desarrollo. Con el fin de encontrar estos valores óptimos se pueden aplicar diferentes técnicas de búsqueda. Aquí se explican dos de las más importantes:

- ***Random search* (búsqueda aleatoria):** La búsqueda aleatoria, como su nombre indica, consistirá en escoger valores aleatoriamente realizando una prueba con cada conjunto escogido. Con esto se permite una gran flexibilidad a la hora de buscar los hiperparámetros óptimos.
- ***Grid search* (búsqueda en cuadrícula):** La búsqueda en cuadrícula se realiza estableciendo unas combinaciones fijas de hiperparámetros, probando cualquier combinación entre estos con el fin de descubrir cuál de ellas funciona mejor. Si bien es más organizada que la aleatoria, el usuario será el que establezca el conjunto de hiperparámetros que se prueba.

3.5. Técnicas de equilibrado de datos

En ocasiones, en los datos utilizados se podría no contar con suficientes ejemplos para realizar un entrenamiento que pueda arrojar buenos resultados, o podrían aparecer de forma poco balanceada, significando esto que la gran mayoría de ejemplos podrían pertenecer a la misma clase. Estos dos problemas son muy típicos en conjuntos públicos y pueden suponer grandes complicaciones a la hora de realizar clasificadores, ya que favorecen enormemente el *overfitting* y pueden sesgar el modelo favoreciendo a las clases mayoritarias.

Siendo problemas tan comunes, se han desarrollado soluciones variadas según el tipo de datos con el que se cuente, la cantidad, número de clases… En este caso, se ha

decidido hablar de tres aproximaciones concretas: el *oversampling*, el *undersampling* y data *augmentation*.

3.5.1. Oversampling y undersampling

En primer lugar, las soluciones más sencillas para obtener un conjunto de datos más equilibrado consisten en multiplicar las clases de las que hay menos datos o recortar las más numerosas.

El *oversampling* consiste justo en lo primero mencionado. Esta técnica consistirá en duplicar datos ya existentes en el conjunto original que pertenezcan a las clases menos numerosas con el fin de igualar o, al menos, acercarse al número de datos existentes para las clases mayoritarias. En el caso de tratar con datos en forma de imágenes esta práctica debe ser evitada ya que no aporta ventaja alguna, aunque sí que podría causar algún problema como el *overfitting*.

Por otro lado, el *undersampling* se utiliza ampliamente cuando se cuenta con imágenes o con un conjunto de datos especialmente numeroso; esto se debe a que esta técnica consistirá en reducir el número de datos de las clases más numerosas con el fin de que las clases minoritarias puedan tener un mayor peso, equilibrando el conjunto.

En la siguiente ilustración (*Figura 28*) se puede observar de una forma visual el uso de estas técnicas.

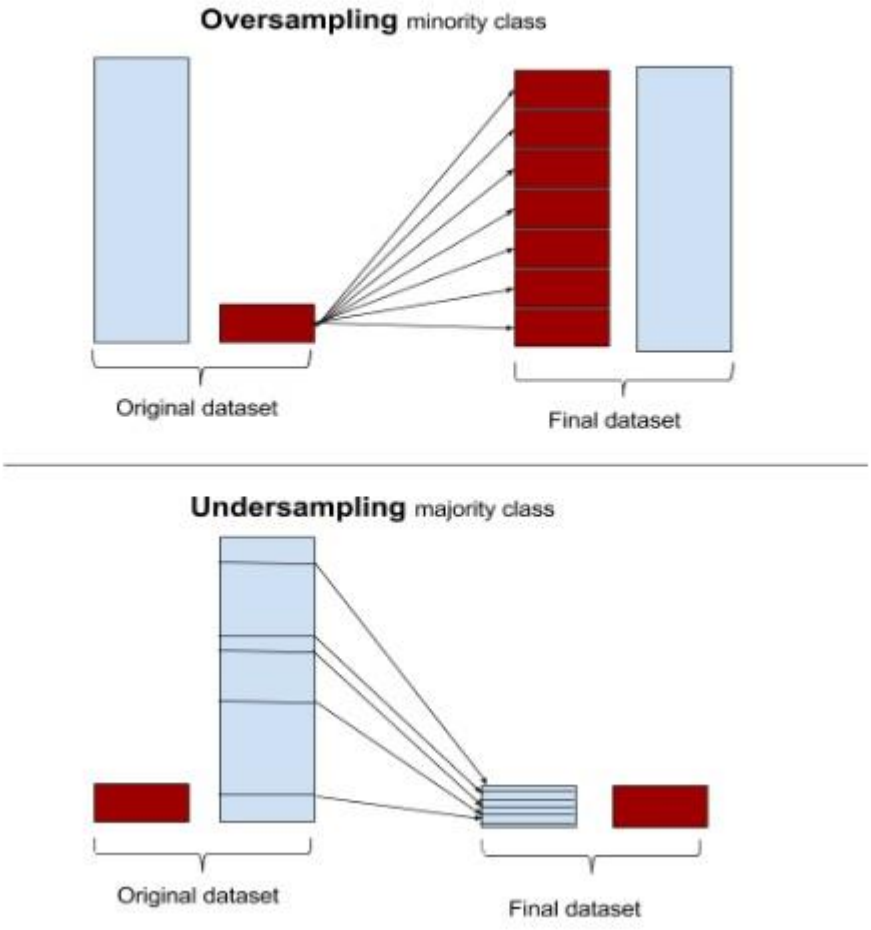

Figura 28. Ejemplo de oversampling y undersampling [34].

3.5.2. Data Augmentation

La ampliación de datos o data *augmentation* es una solución similar al ya tratado *oversampling.* La diferencia entre ambos radica en que esta técnica, en lugar de duplicar ejemplos existentes, creará nuevos datos a partir de los existentes.

Esta técnica consistirá en incrementar el número de muestras de las clases deseadas, generando nuevas muestras a partir de modificaciones realizadas sobre el conjunto original mediante el uso de diferentes técnicas de modificación de muestras. La idea de la generación de ejemplos nuevos será la de contar con un mayor número de datos, pudiendo utilizarse también esta técnica para incrementar clases minoritarias y permitir que la red generalice mejor.

Con respecto a las imágenes, el *data augmentation* sí que se utiliza, a diferencia del *oversampling*, ya que mediante el uso de esta herramienta se pueden generar nuevas imágenes utilizando diferentes técnicas como recortes, estiramientos, zoom, o filtros generales que, para una red neuronal, serán completamente nuevas, permitiendo una mejor generalización del modelo. A continuación, en la *Figura 29*, se puede ver un ejemplo de *data augmentation*.

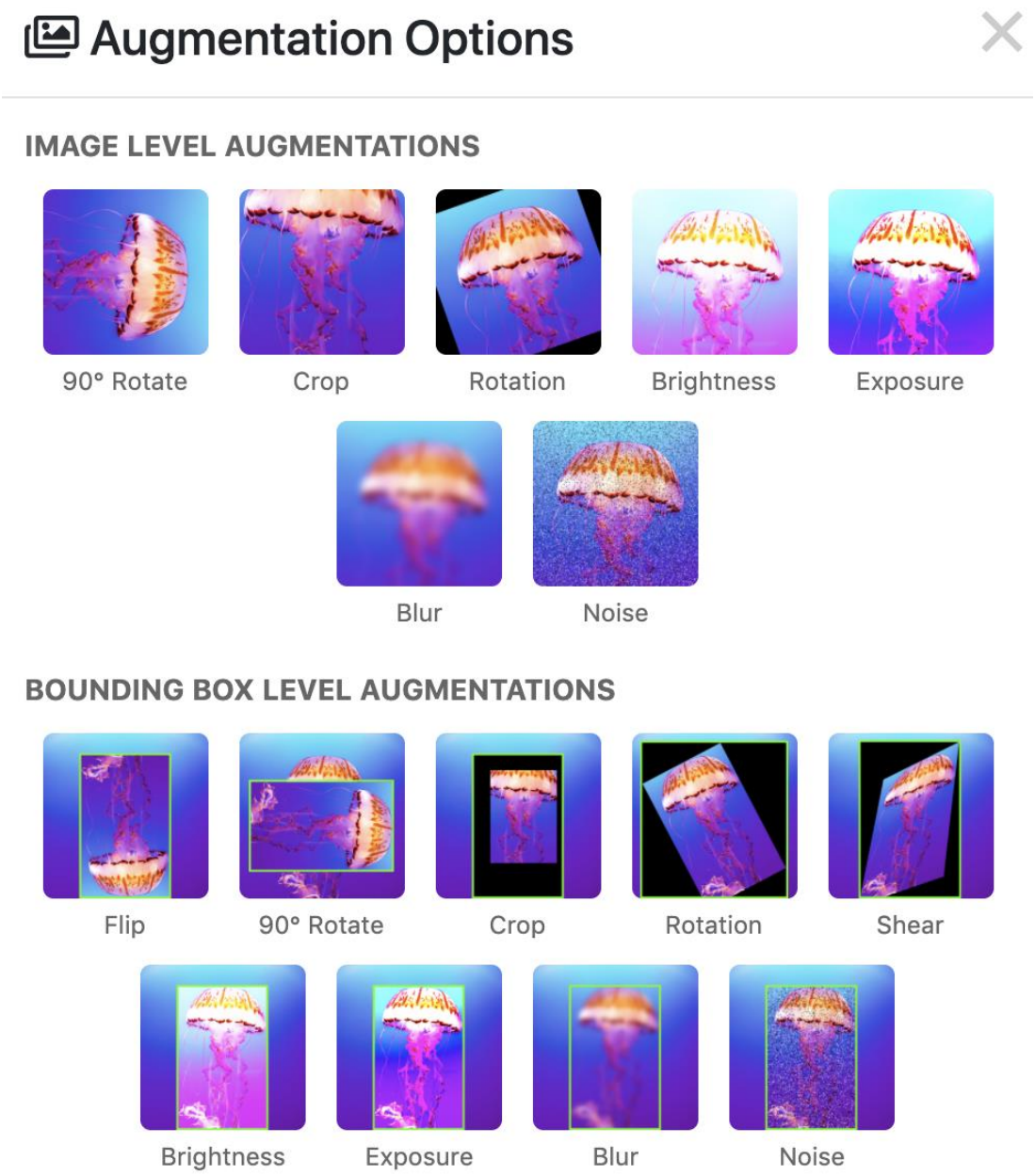

Figura 29. Ejemplo de uso de Data Augmentation [35].

3.6. Métricas

Para evaluar los resultados del modelo durante su entrenamiento utilizándose suelen utilizar la precisión (*accuracy*) y la función de pérdida (*loss*)

La precisión es la métrica que mide el porcentaje de aciertos sobre el total de las predicciones realizadas, por lo que a mayor precisión, mejor rendimiento del modelo. Es un porcentaje, por lo que tomará valores entre 0 y 1. Un ejemplo de su funcionamiento sería el siguiente; en caso de realizar 100 predicciones y acertar 80 de ellas, el cálculo de la precisión arrojaría un 0,8. De la misma forma, si de 150 se acertaran 125, el cálculo arrojaría la cifra de 0,83. Este cálculo sigue la siguiente fórmula:

$$Precisión = \frac{Número\ de\ predicciones\ acertadas}{Número\ de\ predicciones\ totales}$$

Por otro lado, la función de pérdida será una función encargada de calcular la diferencia que existe entre las predicciones realizadas por la red en comparación a los valores reales que deberían predecirse, lo cual quiere decir que a menor pérdida mejor será el modelo entrenado. Para ello, existe una gran cantidad de métodos diferentes. En este caso se ha utilizado el método de Entropía categórica cruzada, más conocida por su nombre en inglés *Categorical Cross Entropy*. Este método concreto funciona especialmente bien en tareas de clasificación debido a que, idealmente, se debe predecir una clase con probabilidad 1 a la par que el resto de las posibilidades deben ser predichas con probabilidad 0. En este caso, la matemática es algo más compleja siendo esta su ecuación:

$$-\frac{1}{N}\sum_{i=1}^{N}\log p_{model}\left[y_i \in C_{y_i}\right]$$

Tras el entrenamiento, se muestran también otras métricas como la matriz de confusión [36] o el reporte de la clasificación [37] que aporta la librería *scikit-learn*, para completar la información que se puede sacar de la red.

La matriz de confusión consiste en generar una matriz en la cual la diagonal principal constará con las predicciones que se han realizado correctamente. Para conseguir esto,

cada fila y columna corresponderán a las predicciones reales y las de la red, respectivamente. Esto consigue que, con un vistazo rápido, se puedan observar diferentes características de cada clase. En concreto, para cada clase, se puede observar el número de verdaderos positivos (TP en inglés), falsos positivos (FP en inglés), verdaderos negativos (TN en inglés) y falsos negativos (FN en inglés); estas métricas son muy útiles de cara a ver que está fallando en un modelo y si existe algún sesgo hacia o desde alguna clase concreta.

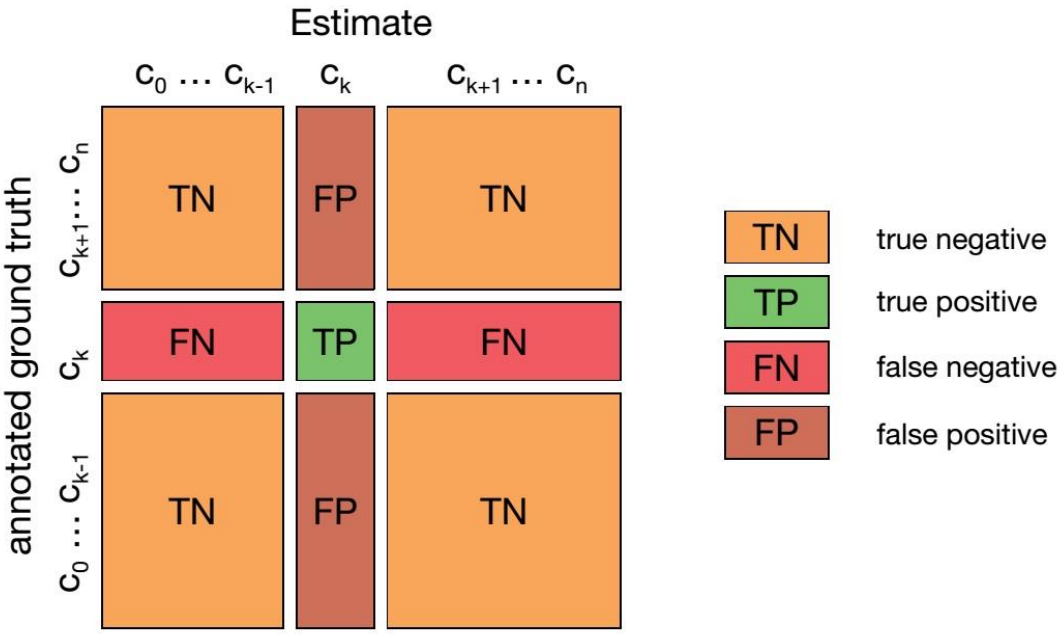

Figura 30. Ejemplo de matriz de confusión [38]

Por ejemplo, como se puede ver en la *Figura 30*, si se quisieran comprobar los datos correspondientes a la clase *k*, podríamos determinar los verdaderos positivos echando un vistazo a los datos presentes en la posición *k*, *k* de la diagonal principal. Del mismo modo para los falsos negativos y falsos positivos simplemente se deberían observar los datos de la fila k y la columna k respectivamente, a excepción de los presentes en la propia diagonal principal. Por último, el resto de los datos pertenecerían al conjunto de los falsos negativos.

En último lugar, cabe destacar el informe de la clasificación aportado por la librería *scikit-learn*. Este informe cuenta con diversidad de métricas calculadas

automáticamente y permite comprobar la verdadera precisión del modelo. Como resumen, a continuación, se comentan algunas de ellas:

- *Precision*: Cómo ya se ha comentado anteriormente, será el número de aciertos sobre el total de predicciones para una clase concreta teniendo en cuenta todos los datos predichos como positivos; se corresponde a la columna de la clase en la matriz de confusión.
- *Recall:* Será el número de aciertos sobre el total que realmente pertenece a una clase concreta teniendo en cuenta los datos predichos como positivos y los predichos como negativos. Esto se corresponderá con la fila en la matriz de confusión.
- *F1 score*: Esta medida combina ambas (*precision* y *recall*) en un solo valor. Y cuanto más cercano de 1 se encuentre este, mejor será el modelo de clasificación. Sigue la siguiente fórmula:

$$2 * \frac{precision * recall}{precision + recall}$$

- *Support:* Esta medida indicará la cantidad de datos utilizados para el cálculo del resto de medidas.

Además, el informe de clasificación cuenta también con una medida media que aglutina los resultados de todas las clases, realizando dicha media de diferentes maneras. Así, las formas mediante las cuales se realiza dicha media son las siguientes:

- *Macro average:* Se obtiene simplemente realizando la media aritmética entre las distintas clases evaluadas.
- *Weighted average:* Se obtiene teniendo en cuenta la medida *support,* o lo que es lo mismo, el aporte de cada clase. Una clase con menos datos aportará menos a la media que una con un mayor número de estos.
- *Micro average:* En el caso de esta medida, cabe destacar que es la misma tanto para *precision*, como para *recall* y *F1 score*. Esto hace que únicamente aparezca en una de las columnas, ya que no tiene sentido realizar dicho cálculo para cada medida, y sigue la siguiente fórmula tomando TP como verdaderos positivos, FP como falsos positivos y FN como falsos negativos:

$$\frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Como es obvio, cada una de estas medias es utilizada dependiendo del tipo de datos y situación en las que se esté trabajando, así como dependiendo de la importancia de cada una de sus clases.

3.7. Aplicaciones médicas

Si bien hoy en día se siguen produciendo impresionantes avances en éste y otros campos dentro de la Inteligencia Artificial, se puede decir que la tecnología ya está suficientemente madura como para poder marcar diferencias considerables. La aparición del *Machine Learning* y el *Deep Learning* ha permitido marcar un antes y un después en las aplicaciones médicas desarrolladas mediante Inteligencia Artificial.

El *Deep Learning* es ahora mismo una técnica utilizada en campos tan diferentes que sería imposible acotar su área de interés, aunque este proyecto se centrará en su aplicación al área de diagnósticos médicos y, más concretamente, al área de las lesiones de la piel.

Dentro de lo que supone este avance para la medicina, se pueden encontrar diversos estudios que avalan su utilización y confirman sus avances en dichos campos, llegando al extremo de conseguir reconocer rápidamente patrones que los médicos especialistas en el campo tardaban mayor tiempo en reconocer [39].

Por último, cabe destacar la gran capacidad de algunos de estos modelos, pues se ha conseguido superar la precisión a la hora de emitir un diagnóstico de médicos especializados en el campo concreto. Así ha sido el caso de un estudio publicado en la revista *ScienceDirect*[1]. En este estudio se ha comparado la precisión en el diagnóstico de melanomas entre la red neuronal de *Google*, *Inception v4 CNN*, y un conjunto de 58 dermatólogos con diferentes niveles de experiencia y especialización en el tema. Por primera vez se ha realizado una comparación de este estilo, resultado victorioso el

[1] Página web de ScienceDirect - ScienceDirect.com | Science, health and medical journals, full text articles and books.

modelo de *Deep Learning* en gran parte de los casos [1]. Hay que destacar que el objetivo de este estudio no es sustituir a los médicos, sino comparar la precisión respecto al modelo para validar este último como herramienta auxiliar para el diagnóstico.

4. METODOLOGÍA

La metodología seguida durante el desarrollo de este proyecto es una metodología de Ciencia de Datos (*Data Science*). Esta metodología sirve de guía para realizar un proyecto de datos de forma eficaz, eficiente y organizada. Existen múltiples adaptaciones de esta metodología según las necesidades y naturaleza del proyecto, pero en general, se sigue el esquema presentado en la *Figura 31*:

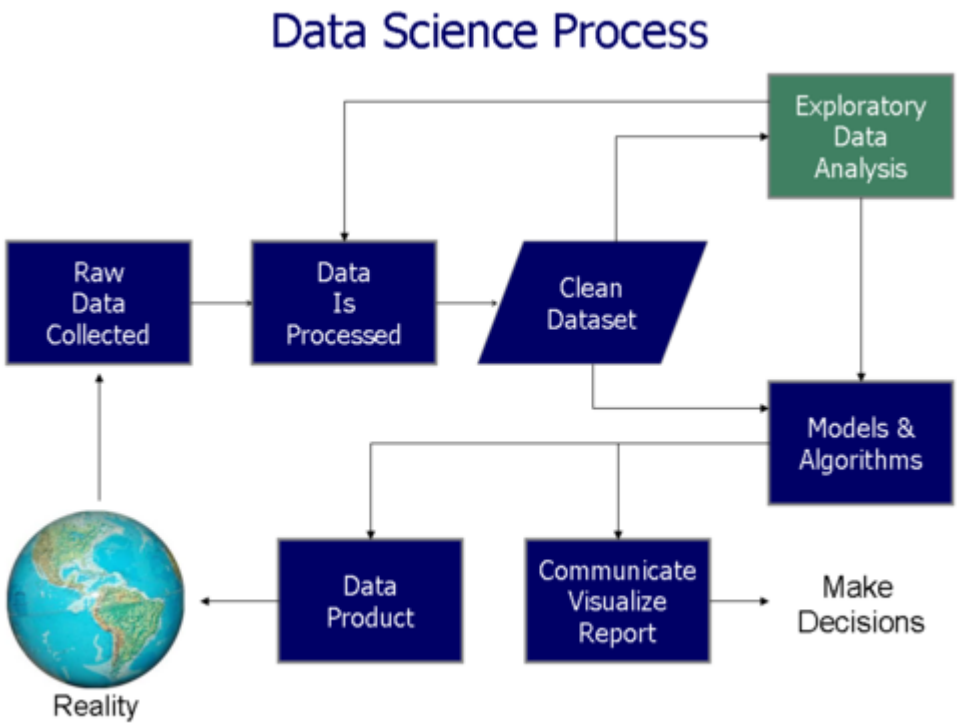

Figura 31. Metodología de Ciencia de Datos [40].

Cabe destacar que el proceso presentado es un proceso iterativo. La fase de toma de decisiones (*make decisions*) normalmente implica modificar el modelo con el fin de mejorar los resultados obtenidos mediante un ajuste, previo a la puesta en producción. Dicho esto, las fases que sigue esta metodología son las siguientes:

- **Fase 1: Definición del proyecto**

Durante la primera fase se debe definir qué es lo que se quiere conseguir. Se debe observar el mundo real con el fin de recabar la información necesaria para acotar el

problema que se quiere resolver de forma clara y concisa. También se deben evaluar los recursos necesarios para conseguirlo, así como planificar la hoja de ruta que seguirá el proyecto.

- **Fase 2: Obtención y preparación de datos**

Definido ya el problema, se pasa a la obtención y preparación de los datos que serán utilizados. En primer lugar, se recogerán los datos de una de las múltiples fuentes de las que estos se pueden conseguir, tales como API (interfaz de programación de aplicaciones, en castellano), bases de datos, creación desde cero… Además, se formatearán y limpiarán estos datos con el fin de que sean utilizables en el proyecto que se va a desarrollar.

- **Fase 3: Exploración y análisis de datos**

Teniendo los datos en un formato que pueda ser utilizado, se procede a su análisis. Este análisis consistirá en determinar qué datos influirán de forma significativa en el resultado que se intenta obtener, qué características deben ser utilizadas y cuáles descartadas por no ser suficientemente significativas. Durante esta fase también se deben comprobar los posibles problemas que conlleven los datos utilizados, tales como desequilibrio en algunas características, así como otros que puedan no haber sido eliminados durante la preparación por alguna razón.

- **Fase 4: Desarrollo del modelo**

Una vez conocidos los puntos fuertes y débiles del conjunto de datos final, se desarrollará el modelo preparado para obtener el resultado deseado. El desarrollo consistirá en aplicar los algoritmos que correspondan sobre los datos con el fin de llegar al resultado esperado. Según el tipo de problema para el que se modele, se aplicarán unos u otros algoritmos.

Este desarrollo se realizará de forma iterativa, creando modelos y mejorándolos con la nueva información que se aporte en las siguientes fases.

- **Fase 5: Comunicación de los resultados**

Durante esta fase se realiza una reunión con el cliente o usuario final y se recogen los comentarios que este tenga sobre el modelo desarrollado. Esta información será útil para iterar sobre las anteriores fases con el fin de mejorar los modelos desarrollados así como de ajustar la solución a lo que el cliente o usuario pretenda conseguir.

- **Fase 6: Toma de decisiones**

Por último, se realiza una toma de decisiones basadas en la información recabada a partir de los cometarios realizados por el cliente o usuario. Esta toma de decisiones puede implicar dar por finalizado el proyecto y entregar el modelo realizado, o realizar una nueva iteración realizando modificaciones sobre el proyecto con el fin de ajustarse a las propuestas realizadas por los usuarios finales.

5. TECNOLOGÍAS UTILIZADAS

En este apartado se procede a comentar las tecnologías utilizadas. De forma más específica se comentarán el lenguaje de programación utilizado, algunas de las librerías principales y más utilizadas a nivel de *Machine Learning*, el entorno de desarrollo en el que se enmarca el proyecto y un primer vistazo a la obtención de los datos con los que se ha trabajado.

5.1. Lenguaje de programación. Python.

El lenguaje de programación que se ha elegido ha sido Python[2]. Python es un lenguaje de programación de alto nivel, interpretado, y cuya sintaxis se centra en la legibilidad.

La elección de este lenguaje se debe a que, además de popularizarse ampliamente últimamente, ha tenido siempre una relación importante con el *Data Mining* y el *Machine Learning*, siendo uno de los más importantes en este paradigma junto al lenguaje R. Para lograr ser uno de los lenguajes más utilizados en estos campos, Python cuenta con diversos módulos que facilitan el trabajo haciendo que se pueda conseguir un código funcional con un esfuerzo bastante menor.

Algunas de las características que ofrece este lenguaje son las siguientes:

- Ser un lenguaje de código abierto, lo que permite la gran aportación de la comunidad
- Programación orientada a objetos, imperativa y funcional
- Es multiparadigma, de ahí su versatilidad
- Posibilidad de escribir sus módulos en otros lenguajes (C, C++…)

Los módulos con los que cuenta este lenguaje han sido claves para el correcto desarrollo y definición del proyecto. Aquí se comentan algunos de los principales.

[2] Página web de Python - Welcome to Python.org

5.1.1. TensorFlow

El módulo de TensorFlow[3] es un conocido de todo aquel que haya trabajado con Inteligencia Artificial en Python. Esta librería cuenta con diversas funciones que facilitan la creación de código dedicado al aprendizaje automático, especialmente con el relacionado a las redes neuronales.

La elección de TensorFlow por encima de alternativas tan potentes como PyTorch se debió principalmente a la facilidad que aportar TensorFlow a la hora de realizar el despliegue y desarrollo de la red. Esta facilidad permite que el proyecto se pueda centrar en mejorar los resultados pues el desarrollo y ajuste de la red implementada es sencillo y rápido.

Dentro de la plataforma de TensorFlow hay que mencionar la API (interfaz de programación de aplicaciones, en castellano) de Keras[4]. Keras es una API sobre *Deep Learning* escrita en Python, que corre sobre la plataforma ofrecida por TensorFlow. Uno de los objetivos de Keras, dicho por ellos mismos, es: "*Being able to go from idea to result as fast as possible is key to doing good research*" [41]. La traducción sería la siguiente: "Ser capaces de ir desde la idea hasta el resultado lo más rápido posible es la clave para realizar una buena investigación".

5.1.2. Scikit-learn

La librería Scikit-learn[5] es un módulo de código abierto que proporciona herramientas para el campo del *Machine Learning*. Entre otros, soporta preprocesamiento de datos, evaluación, selección y ajuste de modelos. Esta librería se construyó mediante el uso de los módulos de NumPy, SciPy y Matplotlib entre otros.

En el caso que nos ocupa, la librería ha sido utilizada para algunas métricas interesantes que puede aportar. Entre ellas, se han utilizado las conocidas como curva ROC y puntuación AUC (en inglés, *ROC curve* y *AUC Score*), el reporte de clasificación o *classification report* y la matriz de confusión o *confussion matrix*.

---

[3] Página web de la librería TensorFlow - https://www.tensorflow.org/

[4] Web de la API de Keras - https://keras.io/

[5] Web de Scikit-learn - https://scikit-learn.org/stable/index.html

Posteriormente, se explicarán en profundidad cada una de las métricas mencionadas así como su utilidad.

5.1.3. Matplotlib

Matplotlib[6] es una librería diseñada para realizar visualizaciones de datos y gráficas en Python. Una de las aportaciones que hace este módulo es proporcionar una alternativa libre a MATLAB.

En el proyecto, Matplotlib ha sido utilizada para crear diferentes visualizaciones de una gran variedad de datos, tales como gráficas para aportar información adicional (pérdida y precisión del modelo, distribución de las predicciones…); también ha sido utilizada para aportar visualización de los filtros utilizados por la red neuronal, así como los mapas de características generados por estos filtros.

5.2. Entorno de desarrollo

Para el desarrollo del proyecto se han utilizado el entorno de programación de *Google Colab*[7], y junto a él se ha utilizado también una máquina virtual existente en la Escuela Politécnica de Cáceres (EPCC). *Colab* es un entorno de desarrollo en la nube, creado por Google y realizado especialmente para gente del campo de la ciencia de datos, mientras que la máquina virtual será un equipo instalado en la EPCC accesible desde internet y que permite un trabajo de forma remota.

5.2.1. *Google Colab*

Algunas de las ventajas que aporta *Colab* son que no requiere configuración ninguna, te da acceso a tarjetas gráficas (GPU, por sus siglas en inglés) de forma gratuita, y mediante el uso de los *Jupyter Notebooks* [42] hace que sea muy sencillo compartir tu código con otras personas. Además, permite el uso de las recientemente incorporadas

[6] Página web de Matplotlib - https://matplotlib.org/

[7] Web de Google Colab - https://colab.research.google.com/

TPU, conocidas en español como unidades de procesamiento de tensores, especializadas para el trato con tensores.

Cabe destacar también que *Google Colab* te permite utilizar datos almacenados en la propia nube de Google, *Google Drive*, así como datos almacenados en tu propio disco duro o incluso descargados de un repositorio en el propio código; ya que también permite la inserción de comandos de Linux para la instalación de paquetes, clonado de repositorios o lo que sea necesario.

Por último, hay que destacar también alguna de las ventajas de los *notebooks*, tales como la posibilidad de dividir en bloques de código, incrementando la legibilidad del código y la posibilidad de una organización más clara, así como la inserción de títulos, subtítulos, comentarios… Que permitan conocer que es lo que se va a realizar a continuación sin necesidad de llenar el código de comentarios.

Colab permite, además, saltar imposibilidades técnicas debido a que el equipo con el que cuenta el autor es un equipo antiguo, imposibilitando el entrenamiento y validación de forma rápida de redes neuronales complejas, como algunas de las que aquí se han utilizado. *Colab* proporciona un equipo potente para evitar que una persona con recursos limitados pueda ver impedida la realización de su proyecto en un campo tan exigente como lo es la ciencia de datos.

5.2.2. Máquina virtual

El segundo entorno de desarrollo aporta otro tipo de ventajas que han resultado ser claves para el desarrollo del proyecto. Una de las principales es la disponibilidad absoluta de la máquina a lo largo de las horas. La disponibilidad ha permitido que continuamente, accediendo desde cualquier dispositivo, se pudieran realizar pruebas de forma que se aprovechara cualquier momento disponible para realizar experimentos.

La máquina ha hecho posible el correcto desarrollo del proyecto especialmente en pruebas largas y que requerían un tiempo especialmente extenso por alguno de sus hiperparámetros como se explicará más adelante en el punto *IMPLEMENTACIÓN Y DESARROLLO*.

5.3. Datos

5.3.1. Obtención de los datos

Los datos utilizados han sido extraídos de los concursos realizados por la Society for Imaging Informatics in Medicine (SIIM) en colaboración con el *International Skin Imaging Collaboration* (ISIC) en la plataforma Kaggle[8]. Este concurso fue el denominado como *Kaggle Machine Learning Challenge on Melanoma Classification*. El *dataset* que este concurso proporciona consta de una gran variedad de imágenes permitiendo una generalización de la lesión que se busca detectar, si bien, al ser un *dataset* pensado para un concurso, presenta algunas dificultades importantes de forma que sea un reto conseguir resultados con una precisión muy alta. Esto ha sido uno de los motivos por los que se decidió escoger este conjunto de datos, ya que puede aportar un mayor conocimiento si bien será más complejo obtener resultados significativos.

Los datos utilizados se encuentran disponibles de forma totalmente libre y con licencia *Creative Commons Attribution-Non Commercial 4.0 International License* (CC-BY-NC) en la página web oficial del concurso del ISIC [43]. La portada del concurso se muestra en la *Figura 32*.

Figura 32. Portada del concurso [44].

Si bien la edición del concurso que se tomó como base para este proyecto fue la realizada durante 2020, los datos pertenecen tanto a dicha edición como a la anterior (2019).

[8] Página web de Kaggle - https://www.kaggle.com/

En la página de dicho concurso existe un foro en el que se aportan problemas, soluciones y debate sobre los datos y la clasificación realizada. Además, en dicho foro aparecen descritas de forma extensa las tres soluciones con mejores resultados.

Cabe destacar la solución aportada por los primeros clasificados del concurso "All Data Are Ext", equipo compuesto por Qishen Ha, Bo Liu y Fuxu Liu. Hay que destacar que su solución contaba también con el uso de los datos textuales que se aportan, no sólo con las imágenes. Sin embargo, el vídeo en el que explican su solución [45] fue especialmente útil a la hora de afrontar la clasificación de imágenes, pues en él se explican diferentes técnicas y posibilidades.

Por otro lado, el hilo del foro en el que se expone su solución [46] también ayudó a la hora de desarrollar este proyecto, ya que algunas de las preguntas y respuestas que se daban resultaron de especial utilidad para resolver problemas y mejorar la precisión de los modelos.

5.3.2. Dataset

Los datos utilizados se han formado mediante la mezcla de datos de dos años diferentes, 2020 y 2019. Esto se debe a que el conjunto de datos del 2020 se encontraba claramente desbalanceado en contra de los melanomas, los cuáles representaban únicamente 581 imágenes (1,76%) de todas las existentes. En el caso de los datos de 2019, contaban con una representación de 4522 imágenes, un 17,85%; haciendo algo más viable el entrenamiento para su detección.

Como se puede observar a continuación, en el conjunto de datos de 2020 además el ya comentado desbalanceo, también hay una gran cantidad de diagnósticos desconocidos impidiendo utilizar esas imágenes en el caso de querer predecir por enfermedad. Esto también es solucionado gracias a la mezcla junto al otro conjunto mencionado.

Por último, también existen algunos diagnósticos que cuentan con un peso irrisorio ya que únicamente aparecen etiquetados en decenas de imágenes, significando que entre todos estos diagnósticos, no se acercan al 1% del total. Estos diagnósticos no han sido tenidos en cuenta durante el desarrollo al igual que los desconocidos.

A continuación, se muestra la distribución de diagnósticos en los datos del año 2020 en la *Figura 33*.

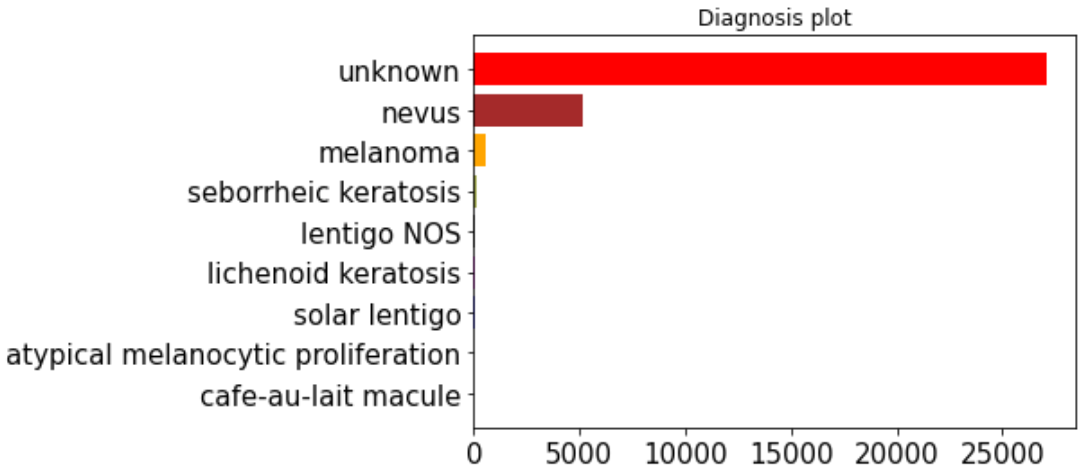

Figura 33. Distribución de clases en los datos de 2020.

Agrupando ambos conjuntos de datos, el resultado es bastante más equilibrado, consiguiendo hasta 5106 melanomas, pasando a representar hasta un 8,7% del total.

Es necesaria realizar la puntualización de que en la distribución comentada siguen existiendo los diagnósticos desconocidos, y debido a que como se ha mencionado anteriormente, las imágenes que cuenten con dicho diagnóstico no serán utilizadas, los melanomas representarán una mayor proporción. Sin embargo, el hecho de que los melanomas representen un número mayor se consigue reduciendo el total de ejemplos disponibles para entrenamiento, validación y test del modelo propuesto. Esta reducción puede acarrear consecuencias en forma de *underfitting*, ya que la reducción de ejemplos producida es significativa.

A continuación, se muestra la distribución existente en los datos llegados a este punto, en la *Figura 34*.

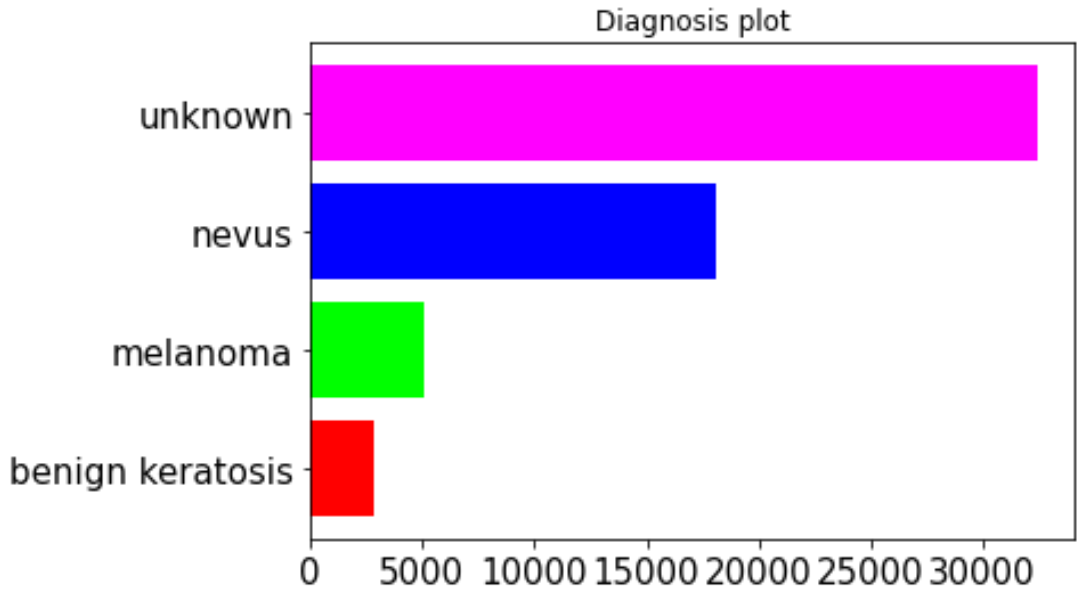

Figura 34. Distribución de clases en los datos mezclados sin eliminar desconocidos.

Después de realizar el último ajuste, consistente en la eliminación de los datos desconocidos, el *dataset* acaba equilibrándose algo más, contando con los siguientes datos que en la *Tabla 3* se presentan.

Diagnóstico	Número absoluto	Número relativo
Keratosis	2847	10,94%
Melanoma	5106	19,62%
Nevus	18068	69,44%
Total	26021	100%

Tabla 3. Cantidad de imágenes por diagnóstico.

Después de la realización de los cambios mencionados, los datos acaban simplificándose a la par que equilibrándose, facilitando algo más el trabajo que se debe realizar. A continuación, se aporta una gráfica en la *Figura 35* donde se puede ver mejor el resultado de los cambios realizados.

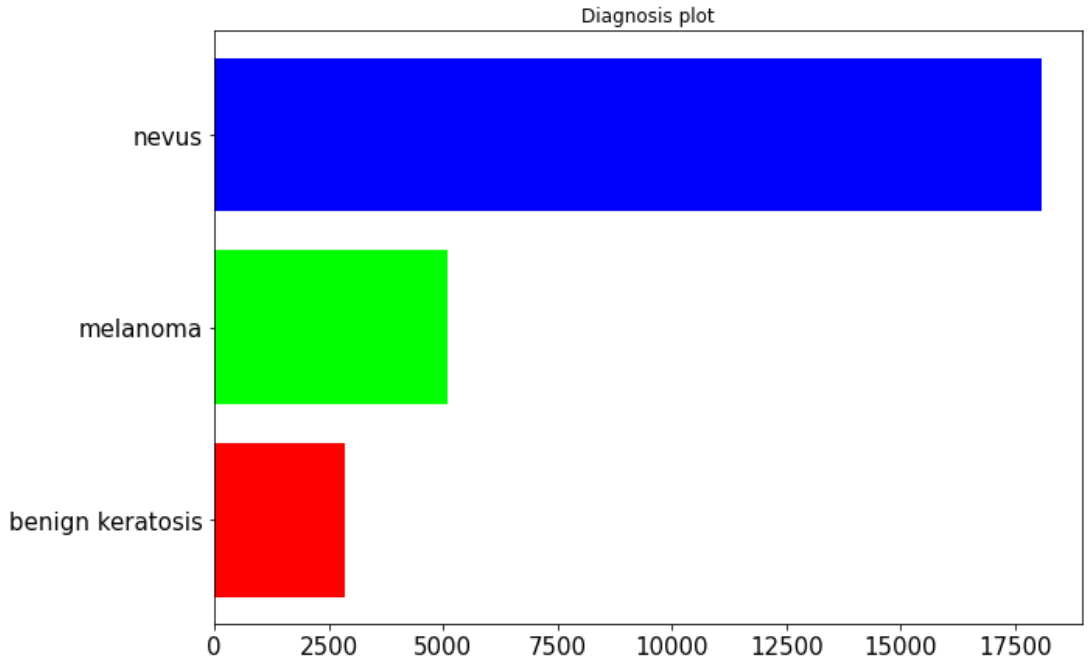

Figura 35. Distribución de datos final.

Observando el *dataset* y obviando los problemas mencionados anteriormente, cabe destacar que todas las imágenes pertenecen a personas con piel clara, sesgando el conjunto de datos si se quisiera clasificar una imagen perteneciente a una persona de piel más oscura. Este tipo de sesgos aparecen comúnmente en los datos de entrenamiento, extrapolándose a los modelos entrenados con dichos datos y son una de las complicaciones que se siguen investigando hoy en día, por lo común de estos, el impacto negativo en el rendimiento en esos casos concretos y los problemas éticos y morales que plantean.

5.4. Limitaciones técnicas

En el transcurso del desarrollo del proyecto, se han encontrado diversos problemas relacionados a las limitaciones técnicas que existían. Dichas limitaciones comienzan por contar con dos ordenadores demasiado antiguos como para soportar un entrenamiento como el necesario, siendo demasiado lentos además de no contar con disponibilidad absoluta debido a la necesidad de realizar otras tareas en paralelo. Dichos equipos no cuentan tampoco con librerías como CUDA, que podrían acelerar bastante todos los procesos involucrados.

Siguiendo por esa línea llegó la elección de Google Colab, que si bien es una buena alternativa debido a lo ya comentado en el apartado *5.2. Entorno de desarrollo*, tiene sus propias limitaciones que impedían llegar al potencial deseado para el desarrollo del proyecto, tales como el límite temporal o la necesidad de estar interactuando constantemente (a partir de cierto tiempo) para evitar una desconexión.

Por último, se solicitó a la Escuela Politécnica de Cáceres (EPCC) una máquina virtual que permitiera acceder a un equipo existente en la Escuela con el fin de, aunque no conste con librerías como CUDA, mantener entrenando la máquina constantemente consiguiendo así subsanar las flaquezas técnicas con mayor tiempo de entrenamiento.

6. IMPLEMENTACIÓN Y DESARROLLO

A lo largo de este apartado se explica en la implementación realizada para conseguir la clasificación de los melanomas. Dentro de esta implementación se encuentran cada uno de los pasos realizados hasta llegar a los resultados de las diferentes pruebas. Así, algunos de estos pasos que aquí se tratan en profundidad incluyen el desarrollo de los modelos utilizados, los hiperparámetros definidos para ellos, y por qué se están utilizando dichos modelos, el preprocesado realizado sobre los datos en general y para cada una de las pruebas individualmente, las diversas técnicas utilizadas para diferencias unas pruebas de otras, o el ajuste de hiperparámetros con el fin de encontrar los óptimos para el trabajo que aquí se presenta.

6.1. Preparación de modelos

La primera tarea a la hora de comenzar con el desarrollo del proyecto ha sido crear dos modelos con los que se realizan las pruebas propuestas. Así, la diferencia entre estas dos redes consiste en que la primera será una desarrollada por el autor, un modelo simple con un número de capas bajo pero que podría obtener buenos resultados en una tarea como la que se realiza en este contexto. El segundo por otro lado será el modelo *EfficientNetB6*, siendo este una red convolucional preentrenada con los pesos obtenidos de un entrenamiento a partir del famoso conjunto de datos *ImageNet*[9]. Esta red será entrenada con la técnica de *Transfer Learning*, añadiéndole algunas capas encima de las originales preentrenadas para ser adaptado a la nueva tarea que se le está asignando.

La elección de esta red sobre otros modelos se debe a los buenos resultados obtenidos con ella durante el concurso *Kaggle Machine Learning Challenge on Melanoma Classification*[10].

El primer modelo de los dos, creado por el autor, se ha creado desde cero a base de introducir capas.

---

[9] Página web de ImageNet - https://image-net.org/index.php

[10] Página del concurso - https://www.kaggle.com/competitions/siim-isic-melanoma-classification/overview

La red final consiste en 4 capas convolucionales de forma que desde la primera en adelante se van duplicando los filtros, que comienzan en 16; también se especifica que para cada capa el tamaño del *kernel* será de 3x3. Cada una de las capas convolucionales cuenta a su vez con una asociada de *max pooling* y una de *dropout* con un valor de 0.2. Tras estas 4, se encuentra la capa de *flatten* seguida por la primera de las capas *fully connected*, que cuenta con 256 neuronas y una nueva capa asociada de *dropout* con un valor de 0.3. Por último, se coloca la capa totalmente conectada con las 3 neuronas asociadas a las 3 clases que deben clasificarse.

Un diagrama que representa la arquitectura de la red es presentado en la *Figura 36*.

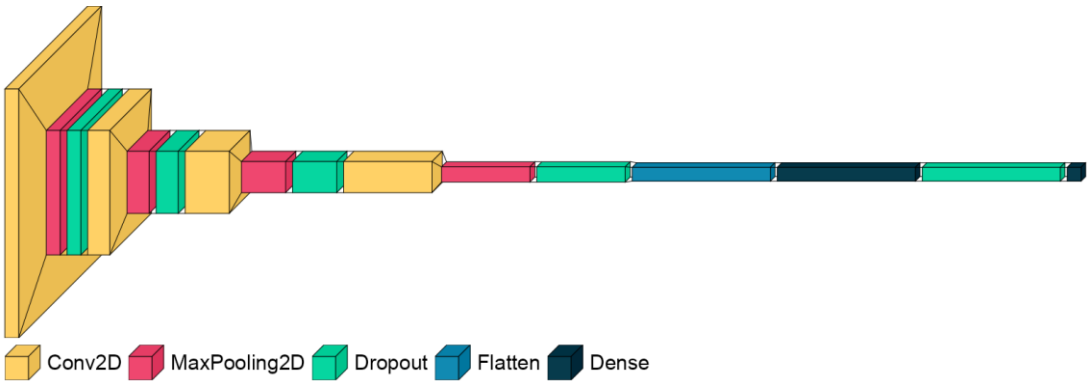

Figura 36. Diagrama de la red creada desde cero.

Mientras que la red basada en EfficientNetB6 estará estructurada de la siguiente manera.

En primer lugar y como base, se ha utilizado el modelo EfficientNetB6 existente en la propia librería *Keras*. Una vez colocada la base del modelo, se suma una capa de *global average pooling* previo a introducir la correspondiente capa de *flatten* necesaria. A continuación existe una capa totalmente conectada contando con 512 neuronas, junto a una capa de *dropout* con un valor de 0.5. Por último, se añade la capa totalmente conectada consistente en las 3 neuronas que identificarán cada una de las clases.

Para entender mejor la arquitectura de la red, se presenta el resumido diagrama de la *Figura 37*. Hay que destacar que el fragmento amarillo cuenta con la parte de la arquitectura correspondiente a la convolución de la red EfficientNetB6. La razón por la que reducir la estructura de dicho modelo a un fragmento es por su gran complejidad

y tamaño, resultando en un diagrama ilegible en estas condiciones si se mostrara de forma completa.

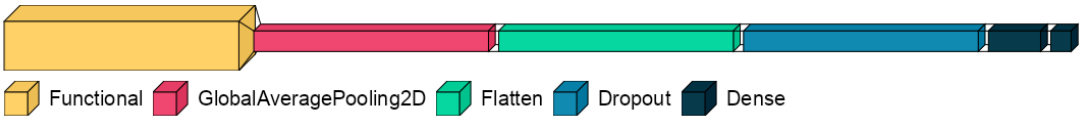

Figura 37. Diagrama del modelo EfficientNetB6.

6.2. Preprocesado de datos

Durante el desarrollo del proyecto uno de los principales problemas que se han descubierto es el gran desbalanceo que presentan los datos con los que se van a trabajar. A lo largo de este apartado se explicarán las diferentes aproximaciones que se han realizado para conseguir equilibrar dicho conjunto.

En primer lugar hay que destacar, como se mencionó en el apartado *5.3.2. Dataset*, que únicamente un 1,76% de las imágenes del conjunto de datos original pertenecen a melanomas, complicando mucho la clasificación de estos. Además, obviando el hecho de que el *dataset* se encuentra tan sumamente desequilibrado, existe otro problema ya que ese porcentaje representa a, únicamente, 581 imágenes, complicando más el trabajo debido al bajo número de muestras con las que trabajar.

La primera solución implementada ha sido utilizar junto a los datos del concurso del año 2020 los datos pertenecientes al concurso del año anterior (2019). Con ello, se han conseguido más muestras de melanomas, en concreto 4522 imágenes nuevas, sumando también que en dicho *dataset* el porcentaje que representan esas 4522 imágenes es un 17,85%, ayudando al equilibrio de los datos que se utilizarán a lo largo del proyecto. En conjunto y con los ajustes comentados en el apartado referido a datos, ambos *dataset* unidos presentarán un 8,73% de melanomas, siendo esto 5106 imágenes, los porcentajes de imágenes se pueden ver de forma más visual en la *Figura 38*.

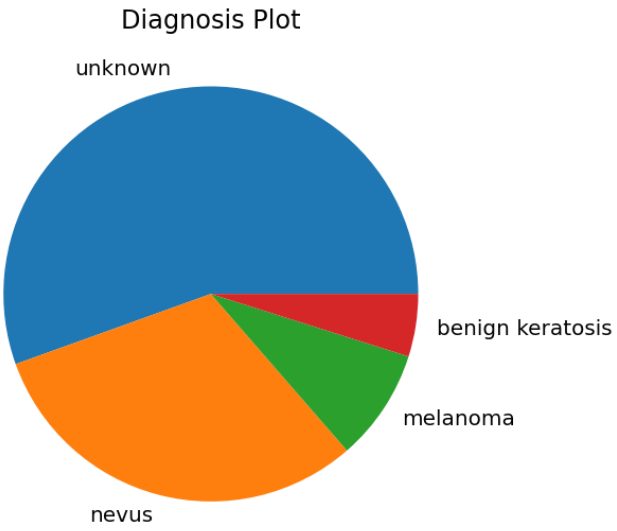

Figura 38. Muestra de los datos tras la unión.

Con el fin de realizar dicha mezcla se han debido adaptar los datos del año 2019, pues diferían de los de 2020 en diferentes características. Para ello se ha comenzado renombrando la columna del nombre de las imágenes de *image* a *image_name*. Con ello, ambas columnas correspondientes al nombre de la imagen se fusionarán.

También se ha transformado la forma en la que se codificaban los datos correspondientes a los diagnósticos. Esto se debe a que en los datos de 2019 las enfermedades se codificaban mediante valores binarios por columnas, por ejemplo, si una fila corresponde a un melanoma, la columna correspondiente a los melanomas llevará un 1, valorando el resto de las columnas como 0.

Para transformar los diagnósticos, se ha comprobado en que columna se encontraba el 1 iterando sobre las filas transformando ese 1 en la columna a el nombre del diagnóstico en la fila introducida en los datos originales de 2020.

Una vez realizados estos cambios, se introducen los datos modificados de 2019 sobre los existentes de 2020.

Como apunte adicional, también se han renombrado ciertos diagnósticos existentes debido al bajo número de muestras que aportaban como es el caso de los diagnósticos *café-au-lait-macule* y *atypical melanocytic proliferation*, que se han eliminado; al igual que se han renombrado todos los tipos de keratosis para formar una única clase conjunta.

Antes de continuar, se ha añadido una columna con un identificador marcando el año al que pertenece la fila indicada por trazabilidad, resultando los datos con el siguiente formato presentado en la *Figura 39*.

image_name	patient_id	lesion_id	diagnosis	benign_malignant	year
ISIC_0029561	nan	nan	nevus	nan	2019
ISIC_0026161	nan	nan	nevus	nan	2019
ISIC_0434532	IP_6342052	IL_7194395	unknown	benign	2020
593_downsampled	nan	nan	nevus	nan	2019
ISIC_0061852	nan	nan	nevus	nan	2019
ISIC_7323300	IP_7829228	IL_1025707	unknown	benign	2020
ISIC_8603493	IP_8442345	IL_6567891	unknown	benign	2020
ISIC_4536709	IP_9721483	IL_1354938	unknown	benign	2020
ISIC_0009895	nan	nan	nevus	nan	2019
ISIC_0028822	nan	nan	benign keratosis	nan	2019
ISIC_4434803	IP_8663649	IL_0206306	unknown	benign	2020
ISIC_2504207	IP_5177184	IL_8320108	nevus	benign	2020
ISIC_0068658	nan	nan	nevus	nan	2019
ISIC_3604975	IP_3240837	IL_5708840	unknown	benign	2020
ISIC_8204142	IP_1222917	IL_9187287	nevus	benign	2020

Figura 39. Extracto de los datos de trabajo.

Hay que destacar que las columnas en la Figura 39 han sido reducidas con el fin de una presentación más clara; sin embargo, en los datos utilizados para la realización de la prueba no se ha aplicado ningún tipo de reducción de columnas.

Para continuar, se ha decidido realizar una clasificación basada en diagnóstico en lugar de simplemente si la lesión es benigna o maligna como estaba pensado en un principio. Esto se debe a que gracias a separar en más clases los datos de entrada, se consigue mayor granularidad que permite equilibrar más los datos. Como se ha mencionado anteriormente, la clasificación binaria consistía en maligno (melanomas) o benigno (resto de lesiones), provocando un desequilibrio mayor que si se diversifica la clasificación benigna en diferentes diagnósticos.

Para esto, se ha modificado la red neuronal que se tenía pensada para realizar las pruebas, de forma que en lugar de actuar como un clasificador binario actúe como un

multiclase añadiendo las neuronas necesarias a la última capa además de cambiar la función de activación de *sigmoid* a *softmax.*

De esta forma, finalmente la red deberá distinguir entre 3 clases diferenciadas, siendo dichas clases nevus, melanoma y keratosis. Como se puede observar, la división realizada sobre los datos originales consistirá en sustituir los ejemplos benignos con las clases keratosis y nevus, renombrando también los ejemplos malignos a melanoma.

A partir de este punto, se ha buscado la mejor forma de entrada de los datos basándose en experimentos que utilizaban los valores de los hiperparámetros descubiertos a través de la optimización de estos. Por ello se han realizado pruebas con tres conjuntos de datos diferenciados.

Hay que destacar que las pruebas se han realizado sobre los modelos comentados en el anterior apartado, realizándose una prueba independiente para cada conjunto de datos diferenciado sobre cada modelo. Cada prueba se realizará de forma independiente tomando siempre como base los datos completos y sin modificación alguna, es decir, antes de cada prueba se revertirán los cambios realizados sobre el *dataset* para partir de los mismos datos originales en cada prueba.

En primer lugar, se ha realizado la técnica conocida como *undersampling*, explicada anteriormente, la cual consistía en reducir el número de imágenes de las clases mayoritarias para equilibrar estas con las clases minoritarias. Realizando dicha técnica, se ha procedido a entrenar los dos modelos con un *dataset* equilibrado consistente en 2847 imágenes de cada clase, haciendo esto un total de 8561 imágenes, siendo un 32,82% del total de imágenes disponibles. Para el conjunto de entrenamiento fueron utilizadas 2277 imágenes de cada clase, quedando el resto para validación, haciendo un total de 6831 imágenes utilizadas durante el entrenamiento.

Para la realización de esta prueba se ha desarrollado un algoritmo que en primer lugar, separará el conjunto de datos en 3 *dataset* diferenciados, cada uno representando a una de las clases existentes, aleatorizando el orden de sus filas. A continuación se detecta cual de estos conjuntos contiene un menor número de datos, estableciéndolo como el número límite de imágenes. Una vez realizado esto, se recorren cada uno de los 3 *dataset* por separado, leyendo el diagnóstico y moviendo la imagen que corresponda a la fila leída hasta su carpeta correspondiente (en función del diagnóstico) hasta que se

llegue al tope de imágenes impuesto. Este conjunto de imágenes organizado en las carpetas será el utilizado posteriormente.

A continuación, se ha realizado una prueba escogiendo imágenes aleatoriamente entre todo el conjunto de datos existente. Para esta prueba se han tomado 26021 imágenes, la totalidad de imágenes disponibles a excepción de las que representan una lesión desconocida, con el fin de que los datos de la prueba sean representativamente diferentes respecto a los de la prueba anterior.

El algoritmo diseñado consiste en, simplemente, aleatorizar el conjunto de datos, recorriéndolo posteriormente con el fin de leer tantas filas como imágenes se haya propuesto utilizar; de cada fila leída se obtendrá el diagnóstico que le corresponda y se enviará la imagen referenciada en ella a la carpeta homónima. Con este *dataset* de imágenes aleatorias se han obtenido las siguientes fracciones para cada una de las clases:

- Melanoma: 5106 imágenes, un 19,62%
- Nevus: 18068 imágenes, un 69,44%
- Keratosis: 2847 imágenes, un 10,94%

Se puede observar cómo los datos escogidos aleatoriamente presentan un gran desequilibrio, derivado del desequilibrio existente en el propio conjunto de datos inicial; si bien se puede observar cómo ha afectado positivamente el introducir los datos pertenecientes al año 2019 en el porcentaje ocupado por los melanomas.

Durante la tercera prueba se ha propuesto un algoritmo similar a este con la ventaja de utilizar la técnica de *data augmentation*. Esta técnica consistirá en crear nuevas imágenes a partir de las ya existentes, rotándolas, recortándolas, haciendo zoom sobre ellas, y otras tantas modificaciones. Con esto no se consigue únicamente equilibrar el conjunto de datos, sino que también se conseguirá mejorar el modelo pues generalizará mejor, es por esta y otras ventajas por las que el *data augmentation* se ha convertido en una técnica utilizada ampliamente con todo tipo de conjuntos de datos, incluyendo muchos en los que ni siquiera sería necesario para obtener buenos resultados.

Esta prueba se basa en un algoritmo que escogerá aleatoriamente 26021 imágenes de nuevo y sin importar su clase, del conjunto de datos original. Para ello se seguirá el mismo proceso explicado en la prueba anterior; aleatorizar el conjunto de datos,

recorrer las filas observando el diagnóstico, y transportar las imágenes que correspondan a las carpetas indicadas. Posteriormente sobre las 26021 imágenes desbalanceadas, se aplicará la técnica conocida como *data augmentation.*

A la hora de realizar esta técnica, se comenzará contando cuantas imágenes existen en el conjunto de trabajo de cada una de las clases. Es importante destacar que este conjunto será el formado por las 20815 imágenes elegidas, no el original, ya que estas serán las que conformen el conjunto de entrenamiento. Una vez contadas se tomará como referencia el número de la clase mayoritaria, pues será este número el que marque la cantidad objetivo para el resto de las clases del conjunto. Con el fin de equilibrar en la medida de lo posible, se realizará la división entre el número obtenido como mayor y la cantidad de imágenes de cada clase. En el caso de que la división tenga como resultado o se obtenga tras aplicar un redondeo 1 no se tendrá en cuenta dicha clase, pues será la mayoritaria o contendrá un número de datos similar al de esta. La razón por la que es necesario el redondeo es que no se puede generar media imagen, necesitando, por lo tanto, un número entero de generaciones. Además, se debe tener en cuenta que el máximo de copias que se realizará sobre cada imagen será de 3, con el fin de no introducir un sesgo en caso de que algunas de las imágenes resultantes puedan parecerse a la original en exceso.

Obtenidas las proporciones en las que se debe aumentar cada clase, se procede a pasar cada imagen de las clases que deban ser aumentadas por el generador, que copiará dichas imágenes tantas veces como le haya sido indicado realizando modificaciones aleatorias dentro del conjunto indicado, aplicándole una de las *augmentations* permitidas sobre los rangos especificados.

El conjunto de *augmentations* permitidas sobre los datos será el siguiente:

- Volteo horizontal
- Volteo vertical
- Desplazamiento horizontal
- Desplazamiento vertical
- Rotación
- Zoom

Una vez finalice dicho proceso, el conjunto se encontrará más equilibrado gracias a las nuevas imágenes generadas.

Así, se puede realizar una comparativa del equilibrio del conjunto de entrenamiento antes y después de aplicar la técnica como la realizada en la *Tabla 4*:

Clase	Imágenes antes	Porcentaje antes	Imágenes después	Porcentaje después
Melanoma	4084	19,62 %	12252	36,53%
Nevus	14454	69,44 %	14454	43,10 %
Keratosis	2277	10,94 %	6831	20,37 %
Total	**20815**	**100%**	**33537**	**100%**

Tabla 4. Comparación de datos aplicando data augmentation.

Cómo se puede observar en la anterior tabla, se consigue un mucho mejor equilibrio mediante el uso de *data augmentation*. Gracias a los nuevos ejemplos creados con este método se podría eliminar el sesgo desarrollado por la red entrenada hacia las clases mayoritarias.

Para terminar este apartado, se preparó una cuarta prueba que debido principalmente a la falta de tiempo no se llegó a realizar debidamente y, por ello, no se menciona en profundidad. Esta prueba consistía en añadir al *learning rate* una estrategia de *cosine annealing*. Esta estrategia consiste en establecer una tasa de aprendizaje relativamente alta, que será rápidamente reducida hasta un mínimo impuesto tras un número concreto de *epochs*, también preestablecido, para ser reiniciado. Los reinicios se realizarán cada vez que se llegue al número de *epochs* preestablecido.

Debido a que en Keras no existe implementación alguna de esta tasa de aprendizaje, se pretendía utilizar una implementación desarrollada en GitHub, como es la siguiente: Implementación en Keras de Cosine Annealing.

6.3. Ajuste de hiperparámetros

En este punto se realizará un estudio sobre los distintos hiperparámetros, con el fin de intentar encontrar el óptimo para el proyecto sobre el que se está trabajando, tal como se comentó en la sección *3.4.3.3. Optimización de hiperparámetros*. Esto se realiza con el fin de obtener los mejores resultados posibles sin descuidar la eficiencia, más importante aún en un proyecto limitado como el que aquí se presenta.

Se buscará realizar modificaciones sobre parámetros como el *learning rate* o tasa de aprendizaje y el *batch size* o tamaño del lote. Para conseguir esto se realizará la estrategia conocida como *grid search*; esta estrategia consiste en realizar pruebas con cada combinación posible del conjunto de valores escogidos previamente. Es decir, se escogerán diferentes valores para, en este caso, el *learning rate* y el *batch size*, y se realizará una prueba independiente con cada combinación posible entre ellos, prevaleciendo el que mejores resultados consiga. Para observar mejor los resultados obtenidos y el conjunto de valores sobre el que se realiza la prueba, se ilustrarán mediante sendas tablas; una presentando el conjunto de valores sobre los que se realizan las pruebas, y otra con los resultados de cada combinación posible.

Por otro lado, el número de *epochs* sobre el que se realizarán las pruebas será de veinte con el fin de unificar este número para diferentes valores de los hiperparámetros.

Cabe destacar que, si bien estos hiperparámetros deben ser ajustados para cada red y conjunto de datos por separado, por falta de recursos, como se menciona en el apartado *5.4. Limitaciones técnicas*, se generalizarán los obtenidos como óptimos en estas pruebas para el resto de las redes y conjuntos de datos utilizados más adelante.

BÚSQUEDA DE HIPERPARÁMETROS	
Learning rate	0.01, 0.001, 0.0001
Batch size	16, 32, 64, 128
Técnica base	Modelo creado

Tabla 5. Valores para búsqueda de hiperparámetros.

Mediante el ensayo realizado con cada una de las combinaciones entre los valores presentes en la *Tabla 5*, se han obtenido diferentes resultados que se muestran a continuación.

Índice	**Hiperparámetros**		**Precisión Máxima Alcanzada**	
N.º de Prueba	Tamaño del batch	Learning Rate	Entrenamiento	Validación
1	16	0.01	0.2536	0.25
2	**16**	**0.001**	**0.6872**	**0.5811**
3	16	0.0001	0.6393	0.4561
4	32	0.01	0.2527	0.2500
5	32	0.001	0.6623	0.5579
6	32	0.0001	0.6167	0.4605
7	64	0.01	0.2527	0.2500
8	64	0.001	0.6501	0.5189
9	64	0.0001	0.5908	0.5566
10	128	0.01	0.2540	0.25
11	128	0.001	0.6383	0.4890
12	128	0.0001	0.5632	0.3952

Tabla 6. Pruebas realizadas para la optimización de parámetros.

Observando la *Tabla 6* se puede comprobar el resultado individual de cada prueba, en concreto, se pueden comprobar tanto la mejor precisión alcanzada, una mayor precisión significará un mejor modelo; como la mejor pérdida obtenida, menor pérdida significará mejor modelo. Si bien se podrían mostrar más métricas, normalmente estas dos son las más importantes, y las utilizadas para realizar una elección sobre los valores de los hiperparámetros. Los resultados obtenidos son sobre el número de *epochs* previamente establecido.

De nuevo, hay que destacar que únicamente se realiza este proceso con un solo modelo por razones de falta de tiempo y recursos; además, por las mismas razones se ha empleado una división concreta entre datos de entrenamiento y validación, siendo estos un 80% y 20% respectivamente. Esta práctica no debe realizarse sobre un modelo para extrapolarlo a otros, ya que el ajuste de hiperparámetros es dependiente del modelo y por lo tanto un modelo diferente podría arrojar resultados completamente distintos.

7. RESULTADOS Y DISCUSIÓN

En este punto se comentan los resultados obtenidos sobre el mejor modelo encontrado en el anterior apartado. Para maximizar aún más las posibilidades de obtener buenos resultados, se ha realizado una prueba con cada técnica comentada en el punto anterior, prevaleciendo el que mejores resultados obtenga.

Una vez encontrados los hiperparámetros óptimos, que como ya se ha comentado se utilizan los mismos para todos los modelos, se ha realizado un entrenamiento con el fin de obtener buenos resultados.

Para determinar si un modelo es mejor que otro, se van a tener en cuenta diversas métricas. Además de la precisión y la pérdida, también se aportará la matriz de confusión generada, y el reporte de clasificación, que aportará diversas métricas. Con los valores proporcionados por todas estas métricas, se interpretarán los resultados y se determinará cuan bueno es el modelo.

A continuación, se aporta una leyenda común a todas las tablas que se van a presentar de ahora en adelante:

Color de resaltado	Significado
	Máxima precisión alcanzada en entrenamiento
	Máxima precisión alcanzada en validación
	Mínima pérdida alcanzada en entrenamiento
	Mínima pérdida alcanzada en validación

7.1. Resultados sobre redes preentrenadas

En primer lugar, aquí se exponen los resultados que se han obtenido sobre el modelo basado en la red preentrenada *EfficientNetB6*. Este modelo ha sido entrenando mediante la técnica de T*ransfer Learning* pues ya ha sido entrenado previamente con el conjunto de imágenes de acceso púbico *ImageNet*.

7.1.1. *Dataset* equilibrado mediante *undersampling*

La primera de las pruebas realizadas consiste en la selección de datos equilibrada utilizando el número de la clase minoritaria como referencia. Las redes neuronales preentrenadas suelen utilizarse en contextos en los que los datos no son suficientes para desarrollar un modelo desde cero, entre otros, por lo que se podría esperar que respondiera bien a este tipo de pruebas. A continuación se presentan los resultados obtenidos a partir de dicha prueba:

Epoch	**Precisión del entrenamiento**	**Precisión de la validación**
1	0,3326	0,3333
2	0,3394	0,3333
3	0,3336	0,3350
4	0,3469	0,3333
5	0,3397	0,3333
6	0,3399	0,3333
7	0,3327	0,3409
8	**0,3399**	**0,3971**
9	0,3411	0,3333
10	0,3414	0,3316
11	0,3443	0,3550
12	0,3426	0,3333
13	0,3420	0,3602
14	0,3364	0,3725
15	0,3389	0,3339

Epoch	**Pérdida del entrenamiento**	**Pérdida de la validación**
1	1,1148	1,1032
2	1,0992	1,0991
3	1,1005	1,0986
4	1,0986	1,1097
5	1,0998	1,0986
6	1,0982	1,0983
7	1,0985	1,0983
8	1,0988	1,0981
9	1,0985	1,0983
10	1,0981	1,0980
11	1,0985	1,0979
12	1,0980	1,0981
13	1,0986	1,0978
14	1,0982	1,0978
15	1,0980	1,0985

16	0,3427	0,3392
17	**0,3484**	**0,3400**
18	0,3475	0,3363
19	0,3366	0,3801
20	0,3414	0,3415

16	1,0986	1,0980
17	1,0980	1,0984
18	**1,0977**	**1,0980**
19	**1,0978**	**1,0975**
20	1,0985	1,0977

Tabla 7. Precisión y pérdida de la primera prueba en la red preentrenada.

.

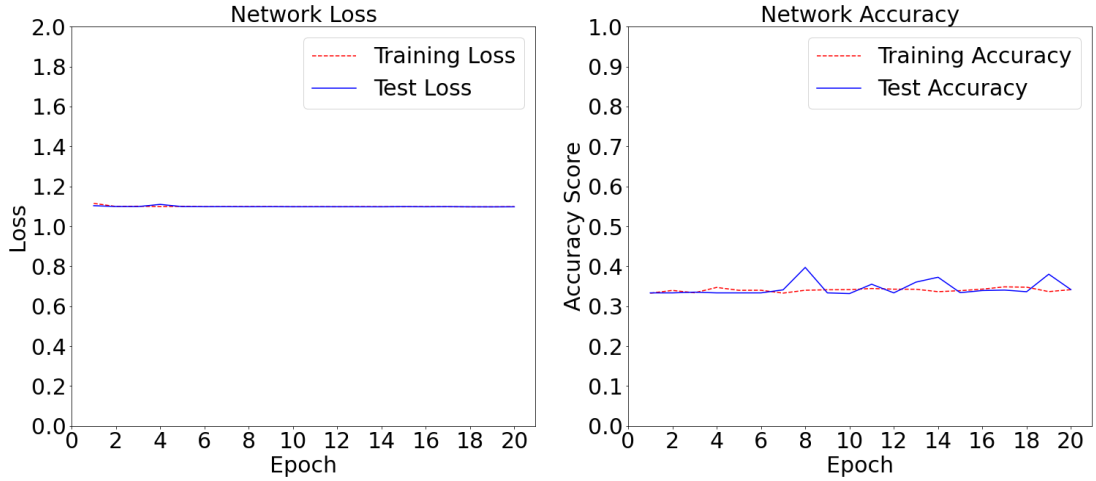

Figura 40. Precisión y pérdida de la primera prueba en la red preentrenada.

Si se observan los resultados presentados en la *Tabla 7* así como en la *Figura 40*, se ve como esta prueba arroja unos resultados pésimos. Se puede ver de forma clara como la pérdida se mantiene prácticamente fija, a la vez que la precisión es incapaz de variar más que unas pocas centésimas. En este caso el número de imágenes era muy reducido, pudiendo favorecer esto los malos resultados del entrenamiento realizado. Además, hay que destacar el problema del ajuste de hiperparámetros, ya que los utilizados no tienen por qué ser los óptimos para este modelo concreto. Entre unos y otros problemas, el resultado acaba siendo el que aquí se puede comprobar, una pérdida fija y una precisión pésima y con una variación minúscula.

Por último, se destaca el cambio de escala en la *Figura 40*. Para el resto de las pruebas se toman como escalas verticales escalas que comienzan en cero y acaban en uno, en este caso se ha ampliado la escala hasta el dos, esto se debe por los altos valores de la

pérdida, superando constantemente la barrera de la unidad. Por motivos de mejor visualización se ha tomado esta decisión.

Con el fin de optimizar los resultados obtenidos, se ha utilizado el modelo de red entrenado con 19 *epochs.*

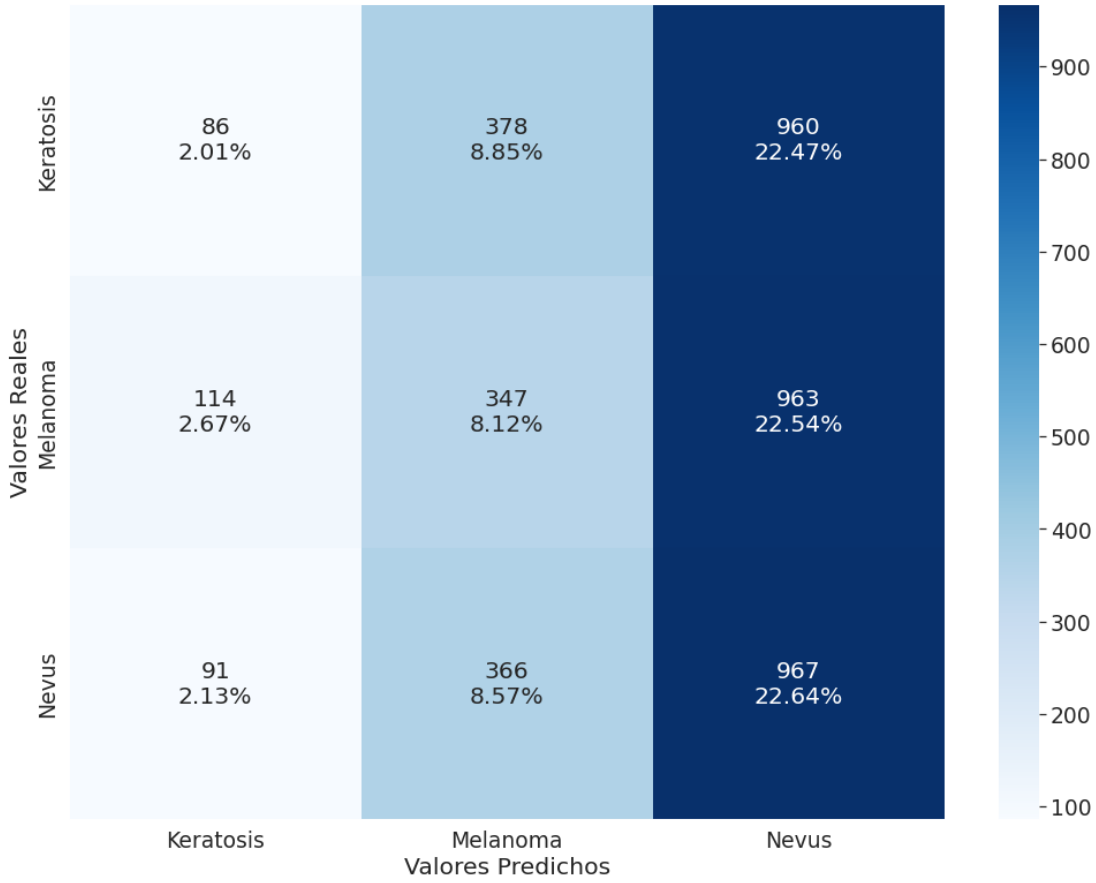

Figura 41. Matriz de confusión de la primera prueba sobre redes preentrenadas.

En la *Figura 41* se puede observar la matriz de confusión resultante de esta prueba. En ella se puede ver claramente y de una forma visual y simple, como la red tiende a clasificar las imágenes con una distribución similar indistintamente de la clase a la que pertenezcan. Probablemente esto viene provocado por la falta de aprendizaje de la red, no siendo capaz de distinguir unas clases de otras. Hay que recordar que esta prueba contaba con pocos ejemplos, pudiendo esto ser la causa del *underfitting* observado.

7.1.2. *Dataset* completo y desequilibrado

Para la segunda prueba se ha utilizado la estrategia basada en obtener 26021 imágenes de forma completamente aleatoria, sin intervenir de ninguna forma en su equilibrio más allá de la mezcla de datos de 2020 y 2019. Si bien en este caso existen más imágenes que en la primera de las pruebas, se espera que la robustez adoptada por las capas preentrenadas con *ImageNet* pueda ayudar a evitar sesgos por el desbalanceo de los datos. Así, los resultados han sido los siguientes:

Epoch	**Precisión del entrenamiento**	**Precisión de la validación**
1	0,6940	0,6942
2	0,6944	0,6942
3	0,6944	0,6942
4	0,6944	0,6942
5	0,6944	0,6942
6	0,6944	0,6942
7	0,6944	0,6942
8	0,6944	0,6942
9	0,6944	0,6942
10	0,6944	0,6942
11	0,6944	0,6942
12	0,6944	0,6942
13	0,6944	0,6942
14	**0,6944**	**0,6942**
15	0,6944	0,6942

Epoch	**Pérdida del entrenamiento**	**Pérdida de la validación**
1	0,8247	0,8172
2	0,8181	0,8161
3	0,8170	0,8156
4	0,8171	0,8154
5	0,8171	0,8309
6	0,8174	0,8160
7	0,8165	0,8152
8	0,8171	0,8167
9	0,8163	0,8145
10	0,8159	0,8166
11	0,8158	0,8159
12	0,8160	0,8151
13	0,8161	0,8145
14	**0,8152**	**0,8143**
15	0,8160	0,8145

16	0,6944	0,6942	16	0,8156	0,8159
17	0,6944	0,6942	17	0,8149	0,8147
18	0,6944	0,6942	18	0,8166	0,8157
19	**0,6944**	**0,6942**	**19**	**0,8148**	**0,8149**
20	0,6944	0,6942	20	0,8152	0,8172

Tabla 8. Precisión y pérdida de la segunda prueba en la red preentrenada.

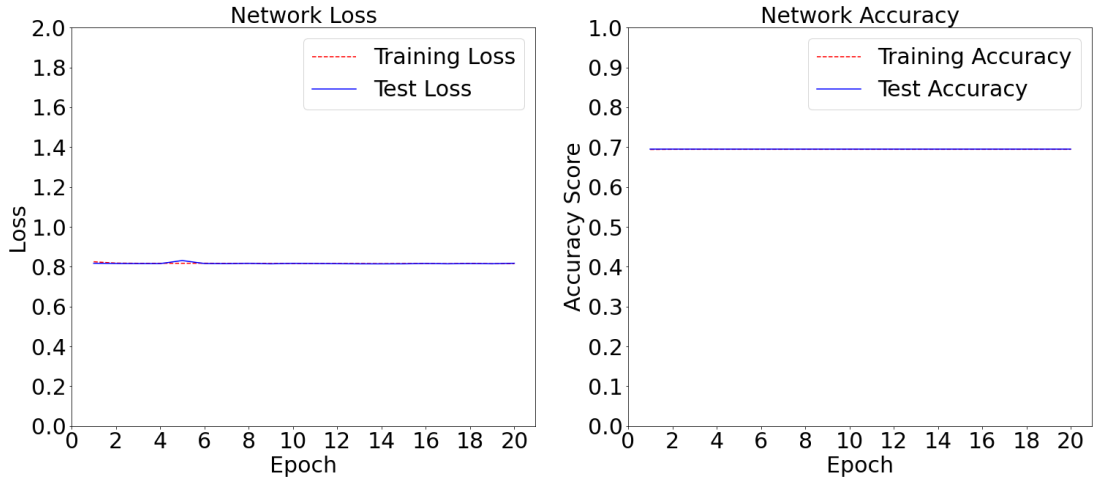

Figura 42. Precisión y pérdida de la segunda prueba en la red preentrenada.

Para comenzar con el análisis de la segunda prueba realizada sobre redes preentrenadas, hay que destacar el hecho de que la red se ha mantenido completamente estable durante todo el entrenamiento. Esto se puede observar fácilmente en la *Figura 42* donde tanto la precisión como la pérdida aparecen como líneas continuas horizontales. Realmente no es completamente estable, sino que en la pérdida existe una mínima variación de, por lo general, unas pocas centésimas, si bien esto no es suficiente como para que sea realmente relevante. Estas pequeñas variaciones se pueden ver de forma desglosada en la *Tabla 8*. Esto muestra que la red no aprende nada nuevo durante el entrenamiento al que se le somete, manteniéndose los pesos con valores muy cercanos a los que existían en la red preentrenada.

La escala utilizada durante esta prueba para la pérdida se mantiene con un rango entre cero y dos debido a que la pérdida es demasiado grande para visualizarlo correctamente en caso de mantener la escala entre cero y uno originalmente pensada.

Debido a la casi absoluta estabilidad del sistema, se ha utilizado el modelo basado en un entrenamiento de 20 *epochs*.

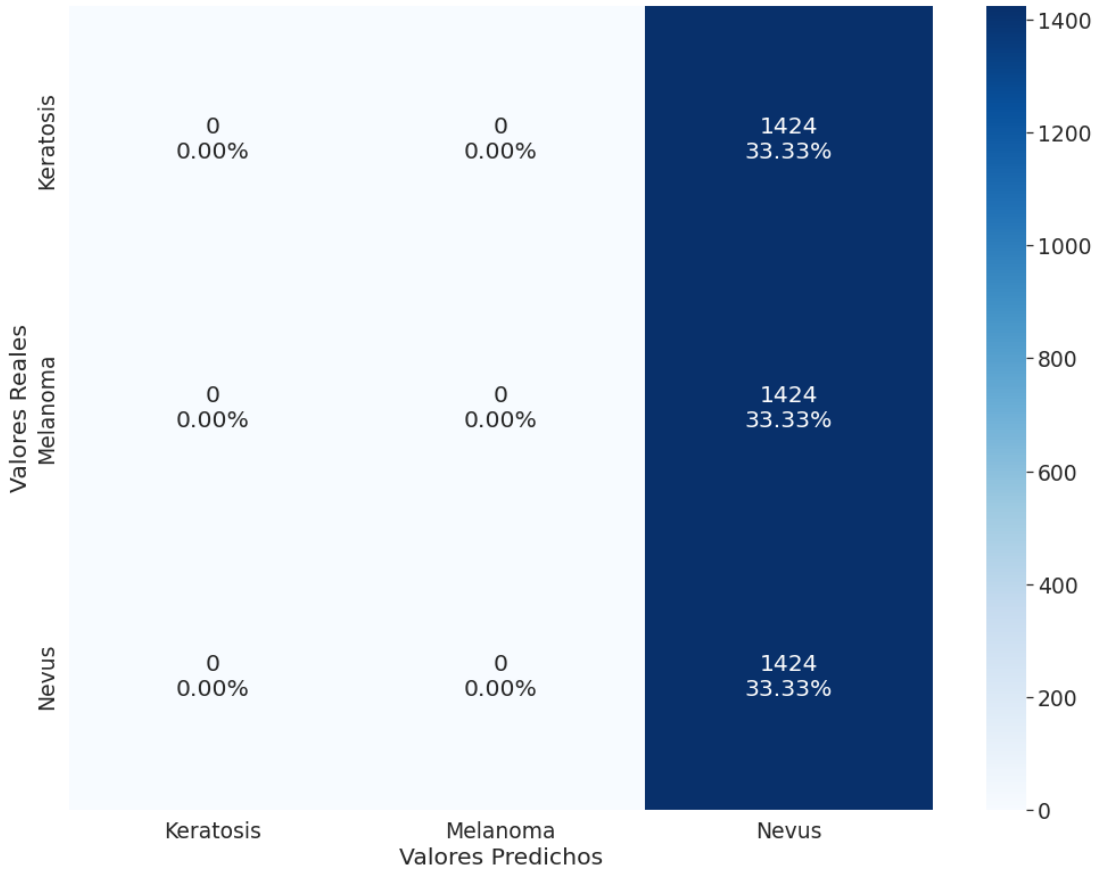

Figura 43. Matriz de confusión de la segunda prueba sobre redes preentrenadas.

La matriz de confusión presentada en la *Figura 43* no requiere de una extensa explicación, pues se puede comprobar a simple vista como la red clasifica la totalidad de los ejemplos proporcionados como nevus. Es decir, tiene una tasa de aciertos similar a la que tendría un clasificador aleatorio, siendo este resultado muy malo.

7.1.3. *Dataset* completo equilibrado mediante *Data Augmentation*

Por último, se ha realizado también una prueba sobre la red preentrenada contando con el mismo número de imágenes con el que se realizó la anterior prueba, diferenciándose de ésta en que se han aplicado técnicas de *data augmentation* con el fin de equilibrar el conjunto de datos. Este es el caso en el que el modelo preentrenado únicamente cuenta con la ventaja de reconocer algunos patrones presentes en los datos desde el

principio, ya que el número de imágenes con el que se cuenta es grande (33537 imágenes) y el conjunto se encuentra más equilibrado.

A continuación se presentan los resultados de esta última prueba:

Epoch	**Precisión del entrenamiento**	**Precisión de la validación**
1	0,4261	0,1963
2	0,4277	0,1963
3	0,4299	0,1984
4	0,4282	0,1963
5	0,4281	0,1963
6	0,4286	0,1963
7	0,4307	0,1963
8	0,4297	0,1963
9	0,4315	0,1963
10	0,4312	0,1963
11	0,4316	0,2050
12	0,4326	0,1932
13	0,4320	0,1963
14	0,4317	0,1961
15	0,4329	0,1963
16	0,4330	0,1965
17	0,4320	0,1963
18	**0,4337**	**0,1963**

Epoch	**Pérdida del entrenamiento**	**Pérdida de la validación**
1	1,0649	1,3747
2	1,0588	1,1797
3	1,0569	1,1227
4	1,0559	1,1983
5	1,0540	1,1618
6	1,0534	1,2435
7	1,0537	1,2073
8	1,0530	1,1951
9	1,0512	1,2207
10	1,0513	1,2483
11	1,0513	1,1223
12	1,0512	1,2273
13	1,0501	1,2250
14	1,0515	1,1686
15	1,0502	1,1828
16	1,0505	1,1489
17	1,0500	1,2953
18	1,0500	1,2324

19	**0,4330**	**0,4985**	**19**	**1,0499**	**1,1052**
20	0,4333	0,1990	20	1,0507	1,1499

Tabla 9. Precisión y pérdida de la tercera prueba en la red preentrenada.

Figura 44. Precisión y pérdida de la tercera prueba en la red preentrenada.

Como se puede observar en los datos presentados anteriormente tanto en la *Tabla 9* como en la *Figura 44*, la precisión tanto de entrenamiento como de validación se mantiene prácticamente constante, apenas avanzando alguna décima en el caso del entrenamiento, e incluso únicamente avanzando centésimas en los datos correspondientes a la validación. Esto significa que la red no está consiguiendo aprender nada nuevo a partir de los datos introducidos.

Hay que destacar un pico que existe cuando se acercan las últimas *epochs* en el entrenamiento, en concreto en la número diecinueve. En vista del resto de datos del que se disponen, el pico parece completamente puntual, volviendo a valores anteriores en la siguiente *epoch*.

Con respecto a la pérdida, se mantiene en consonancia con los datos de la precisión, de una forma bastante estable. Aunque sí que es destacable que la pérdida de validación varía de una forma un poco más destacable, no varía lo suficiente como para ser algo determinante.

En base a los datos comentados, se ha elegido el modelo que contaba con un entrenamiento de 19 *epochs*.

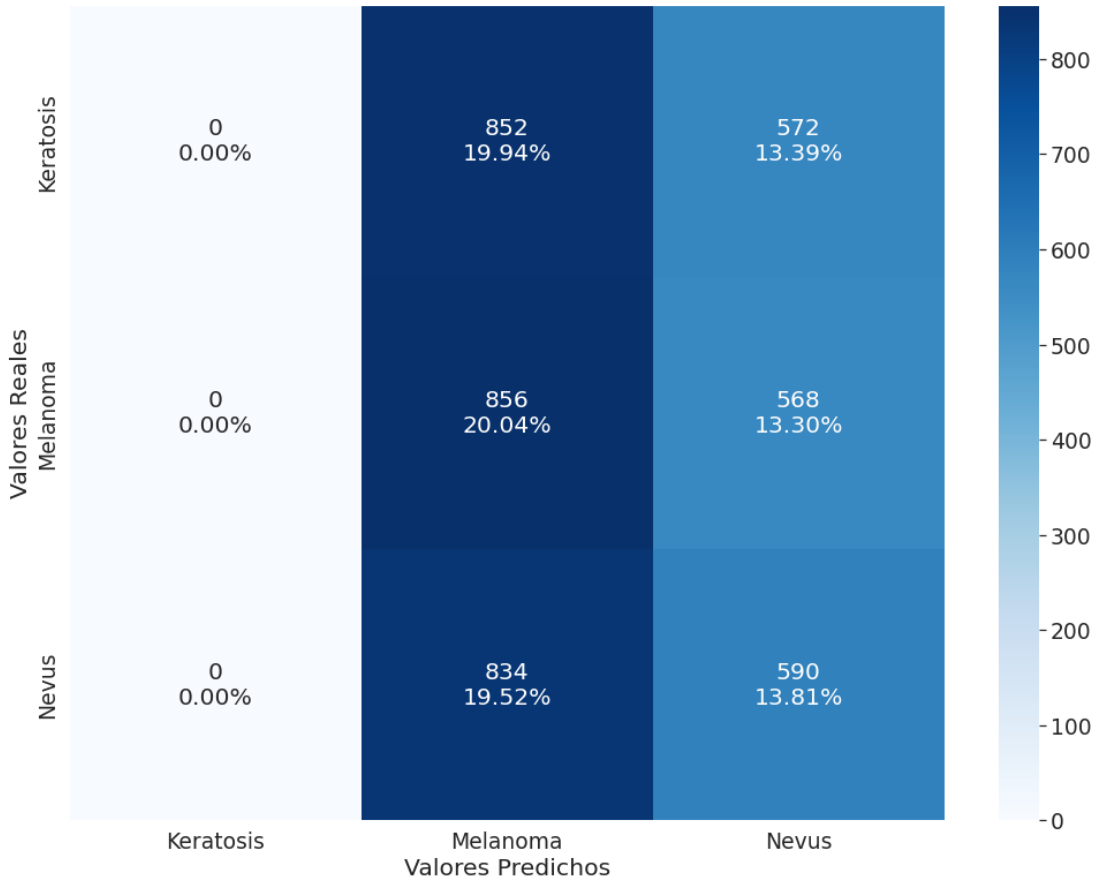

Figura 45. Matriz de confusión de la tercera prueba sobre redes preentrenadas.

En la matriz de confusión (*Figura 45*) asociada a la prueba utilizando *data augmentation* sobre redes preentrenadas, se puede ver claramente como el modelo presenta un sesgo hacia los melanomas. Si bien clasifica correctamente un 20% de imágenes como melanoma, del 80% restante únicamente acierta en clasificar un 13% más como nevus. Todas las imágenes restantes quedarán mal clasificadas, destacando el hecho de que la red no detecta ninguna que presente keratosis, cuando en realidad la predicción se ha realizado con un *dataset* equilibrado de imágenes.

7.2. Resultados sobre redes creadas

Este otro modelo ha sido creado por el usuario, significando que no cuenta con ningún tipo de entrenamiento previo, a diferencia del anterior. Esto conlleva que se deban aprender todos los patrones que existan en el conjunto de imágenes de entrenamiento desde cero, pues no existe base previa.

7.2.1. *Dataset* equilibrado mediante undersampling

Para comenzar, se ha realizado la prueba consistente en la aplicación del *undersampling*. Esto significa que se tomará como referencia el número de datos existentes en la clase más pequeña, reduciendo el número de datos del resto hasta dicha cantidad. El hecho de que la clase minoritaria sea tan reducida como ya se ha comentado, implica que este entrenamiento probablemente cuente con unos resultados malos. Para evitar estos malos resultados se ideó el *Transfer Learning*, utilizado aquí para el modelo preentrenado. Los resultados de esta primera prueba se muestran a continuación:

Epoch	Precisión del entrenamiento	Precisión de la validación
1	0,4295	0,5708
2	0,5504	0,5807
3	0,5832	0,5702
4	0,5875	0,5807
5	0,6094	0,5860
6	0,6081	0,5708
7	0,6189	0,5820
8	0,6359	0,6210
9	0,6441	0,6246
10	0,6607	0,5228
11	0,6686	0,6269
12	0,6937	0,6216
13	0,7036	0,6298
14	0,7333	0,6339

Epoch	Pérdida del entrenamiento	Pérdida de la validación
1	1,1690	0,9341
2	0,9277	0,8902
3	0,8862	0,8990
4	0,8707	0,8899
5	0,8413	0,8824
6	0,8372	0,9204
7	0,8172	0,8833
8	0,7900	0,8358
9	0,7710	0,8412
10	0,7536	0,9542
11	0,7393	0,8201
12	**0,6993**	**0,8153**
13	0,6697	0,8226
14	0,6235	0,8385

15	**0,7577**	**0,6421**	15	0,5646	0,8200
16	0,7836	0,5936	16	0,5165	0,8999
17	0,8075	0,6357	17	0,4716	0,8859
18	0,8315	0,6222	18	0,4160	0,9579
19	0,8464	0,6181	19	0,3934	1,0180
20	**0,8668**	**0,6175**	**20**	**0,3336**	**1,1310**

Tabla 10. Precisión y pérdida de la primera prueba en la red propia.

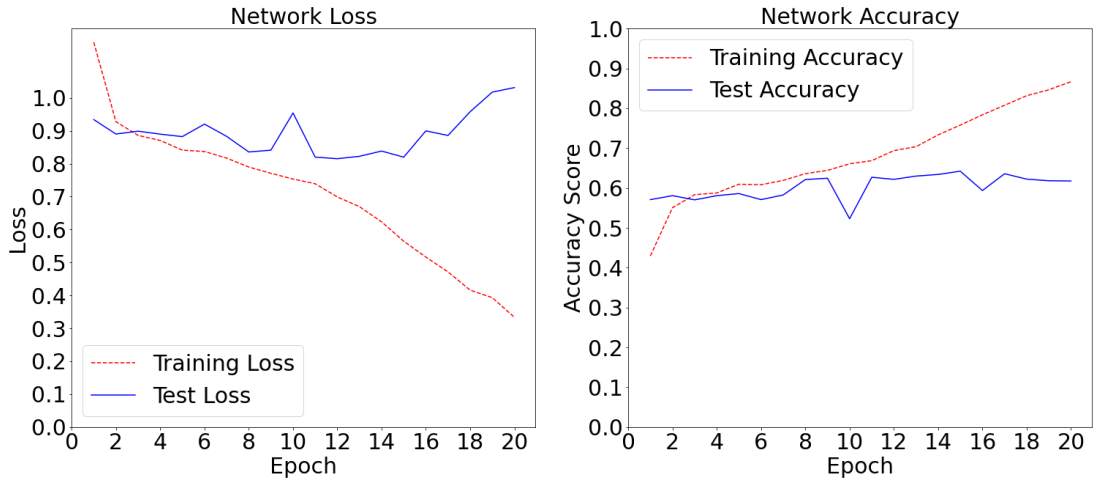

Figura 46. Precisión y pérdida de la primera prueba en la red propia.

Los resultados referidos a la primera prueba realizada sobre la red creada se presentan tanto en la *Tabla 10* como en la *Figura 46*. Observando dichos resultados se pueden comprobar diferentes cosas. En primer lugar, se comprueba una evolución natural mientras desciende la pérdida y aumenta la precisión, ambas muy poco a poco y especialmente las de validación. El cambio es mínimo, sin embargo, llegados a las *epochs* doce y quince para la pérdida y precisión respectivamente, se ve una disrupción donde la red empieza a variar ligeramente y comienza una tendencia más o menos negativa. Esto significa que la red ha alcanzado la convergencia, esto es un mínimo de su función de pérdida, especialmente cuando un par de *epochs* más tarde la pérdida comienza a incrementarse de una forma más o menos uniforme, alcanzando el efecto denominado *overfitting*.

En este caso los resultados tanto de precisión como de pérdida son bastante malos. Esto se puede deber a los pocos datos con los que se contaba, ya que las redes entrenadas desde cero son bastante sensibles en lo que a número de datos se refiere. Además, la red creada es sencilla, lo cual puede influir de cara a aprender un gran número de patrones que puedan ser útiles en esta tarea de clasificación.

En esta ocasión se ha escogido el modelo que cuenta con 17 *epochs* de entrenamiento ya que, además de contar con valores de precisión y pérdida cercanos a los mejores alcanzados en este entrenamiento, en este punto se encuentra el último descenso de la pérdida durante todo el entrenamiento.

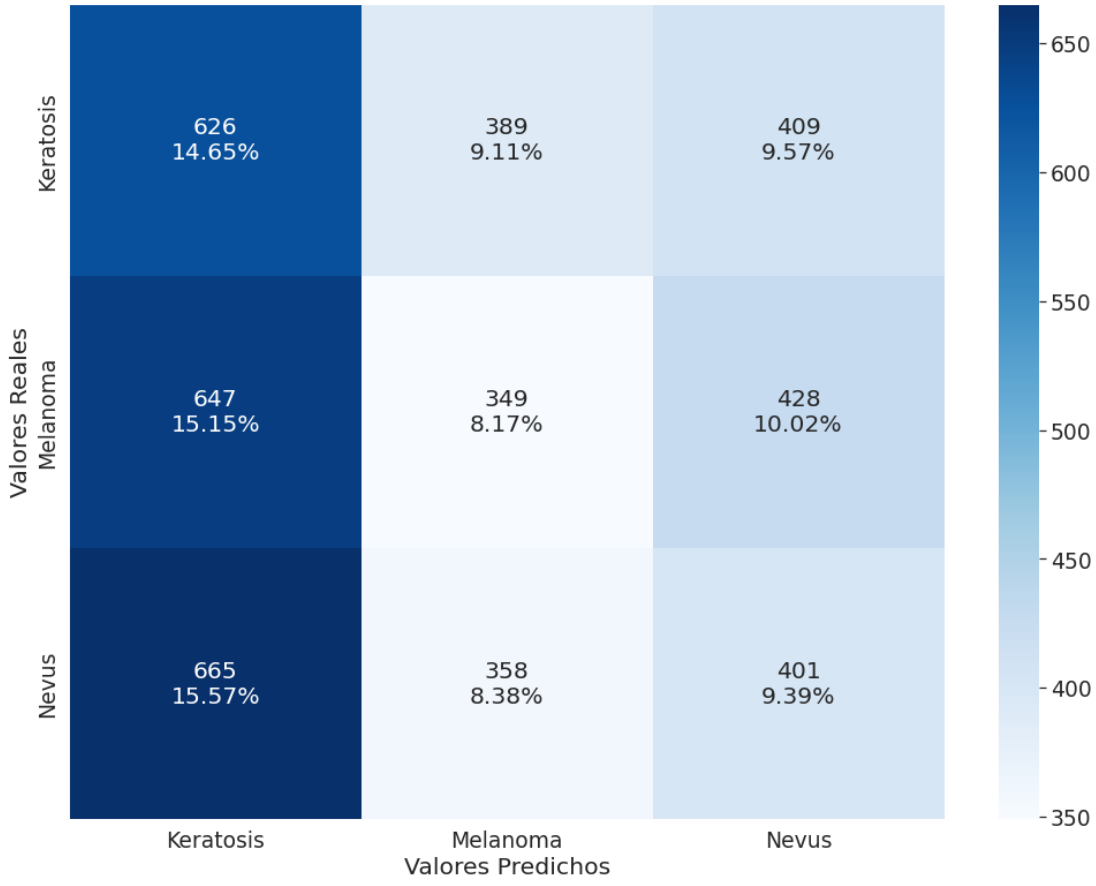

Figura 47. Matriz de confusión de la primera prueba sobre red propia.

Durante la primera prueba realizada sobre la red desarrollada desde cero, se han obtenido los datos presentados en la *Figura 47*. En esta ocasión se puede ver que la red se equivoca favoreciendo enormemente a la clase de Keratosis. Además, se puede comprobar como en ninguna de las tres clases la red acierta más de lo que falla.

Estos malos resultados tienen una causa clara. Al ser una red creada desde cero, se necesitan muchos más datos para afinar sus predicciones que si fuera una preentrenada. Tanto es así, que en esta primera prueba con muy pocas imágenes los resultados son nefastos.

7.2.2. *Dataset* completo desequilibrado

En segundo lugar se ha realizado otro tipo de prueba. En esta ocasión se han tomado 26021 imágenes de forma aleatoria, con el fin de incrementar los datos disponibles ya que, como se ha comentado, con una cantidad pobre de datos es muy difícil obtener buenos resultados entrenando desde cero un modelo. Si bien se ha contado con un número mayor de datos, hay que destacar que en esta ocasión los datos se encontraban claramente desbalanceados, con los problemas que eso puede suponer para este modelo. Los resultados se presentan a continuación:

Epoch	**Precisión del entrenamiento**	**Precisión de la validación**
1	0,6964	0,5154
2	0,7190	0,3367
3	0,7260	0,3584
4	0,7326	0,6915
5	0,7432	0,7134
6	0,7479	0,7520
7	0,7552	0,7549
8	0,7624	0,7572
9	0,7662	0,7520
10	0,7737	0,7589
11	0,7814	0,7662

Epoch	**Pérdida del entrenamiento**	**Pérdida de la validación**
1	0,8439	0,9023
2	0,6509	0,1071
3	0,6368	1,0877
4	0,6223	0,7108
5	0,6025	0,7540
6	0,5939	0,6170
7	0,5801	0,6407
8	0,0566	0,6073
9	0,0557	0,6049
10	0,5380	0,5857
11	**0,5248**	**0,5715**

12	0,7887	0,7697	12	0,5151	0,5731
13	0,7961	0,7649	13	0,4928	0,5914
14	0,8010	0,7622	14	0,4786	0,5761
15	0,8199	0,7595	15	4,4329	0,5815
16	0,8288	0,7683	16	0,4224	0,5977
17	**0,8408**	**0,7743**	17	0,3974	0,5800
18	0,8554	0,7678	18	0,3630	0,6154
19	0,8618	0,7718	19	0,3428	0,6157
20	**0,8758**	**0,7678**	**20**	**0,3139**	**0,6457**

Tabla 11. Precisión y pérdida de la segunda prueba en la red propia.

Figura 48. Precisión y pérdida de la segunda prueba en la red propia.

Como se puede observar en los datos presentados anteriormente, tanto en la *Tabla 11* como en la *Figura 48*, se pueden distinguir claramente tres fases diferenciadas. En primer lugar, el comienzo, donde los pesos son inicializados. En esta etapa hay una gran variación de las métricas debido a la falta de optimización de los pesos y el cambio continuo de estos, además la pérdida en la validación llega a superar la unidad. Esto es algo que suele ocurrir durante las primeras iteraciones a la hora de entrenar

una red neuronal. Por último, hay que destacar que poco a poco la red se va estabilizando tras el caos inicial, llegando a la segunda etapa diferenciada.

Durante la segunda fase, que contiene las *epochs* entre la octava y la decimocuarta, se puede observar una red mucho más estable y que progresa poco a poco, si bien tan poco a poco que parece que la convergencia ha sido alcanzada en alguna de esas epochs.

En la tercera fase, aunque la precisión se mantiene bastante estable en general, se puede observar como la red consigue mejores resultados sobre el conjunto de entrenamiento que sobre el de validación. Esto significa que la red ha alcanzado su óptimo con estos hiperparámetros y se está comenzando a producir *overfitting*.

Es por este fenómeno producido en la última fase por el que se ha decidido dejar la prueba en únicamente las 20 *epochs* propuestas como base, ya que no se cree que se pueda progresar más con esta configuración.

Los motivos por los que la red puede haber alcanzado su óptimo tan pronto y con una precisión relativamente baja en el conjunto de validación pueden ser muy diversos, pero es bastante probable que el desbalanceo que existe en el conjunto de datos haya influido, sesgando el modelo en favor de las clases mayoritarias.

Para evaluar el modelo se ha escogido el que ha sido entrenado con 17 *epochs.* La elección en este caso viene respaldada por el hecho de que justamente en ese punto se alcanzó la máxima precisión de validación, además de contar con valores decentes en el resto de las métricas observadas.

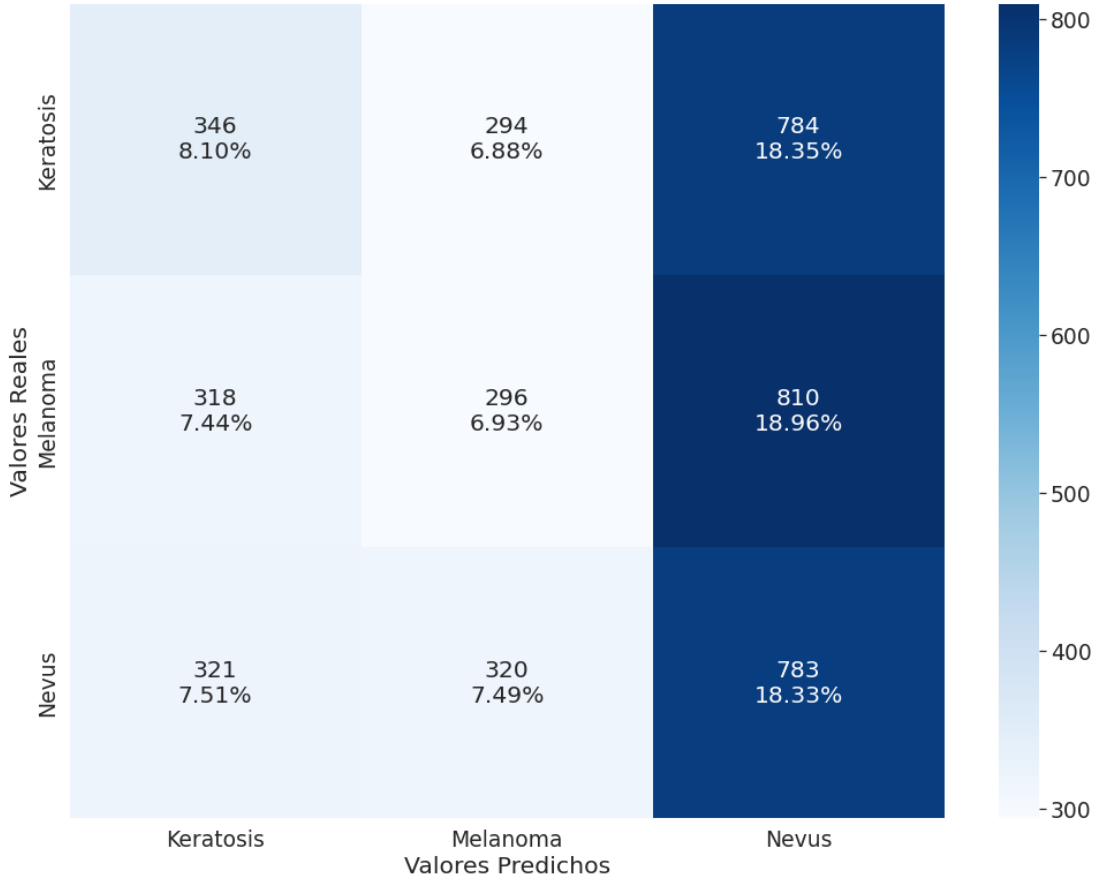

Figura 49. Matriz de confusión de la segunda prueba sobre red propia.

En la *Figura 49* se muestra la matriz de confusión correspondiente a la segunda prueba realizada sobre la red propia. En esta ocasión vemos como el sesgo favorece a la clase Nevus, debido a que durante esta prueba los datos se encuentran desequilibrados muy a favor de dicha clase. Cabe destacar que a pesar de favorecer enormemente a la clase mencionada, en los intentos de clasificar la clase Keratosis es capaz de predecir más ejemplos como Keratosis que como alguna de las otras dos clases; si bien no llega a conseguir clasificar bien la mayoría de los ejemplos de dicha clase.

7.2.3. *Dataset* completo equilibrado mediante *Data Augmentation*

En tercer lugar se ha realizado una prueba combinando las dos anteriores. Esta prueba consistía en recuperar 26021 imágenes aleatorias tal y como se realizó durante la segunda prueba, aumentando después los conjuntos minoritarios mediante el uso de *data augmentation*, resultando en un dataset equilibrado como en la primera prueba realizada. Esto ha permitido equilibrar los datos, diferenciando esta prueba de las

demás realizadas y permitiendo evitar una gran cantidad de los problemas que podría generar un conjunto desbalanceado. A continuación se presentan los resultados:

Epoch	**Precisión del entrenamiento**	**Precisión de la validación**
1	0,6009	0,7061
2	0,6474	0,7216
3	0,6627	0,7126
4	0,6731	0,7201
5	0,6842	0,7301
6	0,6955	0,7397
7	0,7100	0,7330
8	0,7247	0,7518
9	0,7435	0,7499
10	0,7621	0,7462
11	0,7838	0,7520
12	0,8046	0,7466
13	0,8247	0,7451
14	0,8417	0,7439
15	**0,8559**	**0,7536**
16	0,8675	0,7506
17	0,8809	0,7456
18	0,8910	0,7491
19	0,8981	0,7408

Epoch	**Pérdida del entrenamiento**	**Pérdida de la validación**
1	0,8397	0,7286
2	0,7126	0,6700
3	0,6809	0,6797
4	0,6643	0,6367
5	0,6434	0,6234
6	0,6280	0,6184
7	0,6081	0,6190
8	**0,5820**	**0,6085**
9	0,5512	0,6112
10	0,5178	0,6187
11	0,4820	0,6277
12	0,4423	0,6704
13	0,4066	0,6976
14	0,3702	0,7500
15	0,3413	0,7511
16	0,3187	0,8323
17	0,2918	0,8474
18	0,2724	0,7976
19	0,2518	0,8790

20	**0,9043**	**0,7433**	**20**	**0,2400**	**0,8930**

Tabla 12. Precisión y pérdida de la tercera prueba en la red propia.

Figura 50. Precisión y pérdida de la tercera prueba en la red propia.

Durante la tercera prueba realizada sobre la red creada desde cero, se puede observar tanto en la *Tabla 12* como en la *Figura 50* que existen dos corrientes diferenciadas. Por un lado, tanto la precisión como la pérdida del entrenamiento siguen una mejora lineal y constante durante cada una de las *epochs*. Por otro lado, aunque la precisión de la validación sigue esta misma corriente, llegados a la decimoquinta *epoch* esta experimenta un ligero descenso que da a entender que ha llegado a su punto de estabilización. La pérdida de validación por otro lado alcanza su punto mínimo mucho antes, durante la *epoch* número ocho; a partir de dicho punto la pérdida comienza a subir hasta el punto de que consigue superar su anterior máximo.

Sí que se puede comprobar que dados los números que se manejan durante estas pruebas, los resultados obtenidos en esta específicamente son buenos, superando los tres cuartos de precisión.

Durante esta tercera prueba, se ha escogido el modelo con 15 *epochs* para evaluar cuán óptima ha resultado ser esta prueba concreta. Una vez más, se escoge ese modelo en concreto por tener la mayor precisión de validación; teniendo en cuenta también que la pérdida se encuentra en ascenso, si bien en ese punto aún se encuentra en un valor decente.

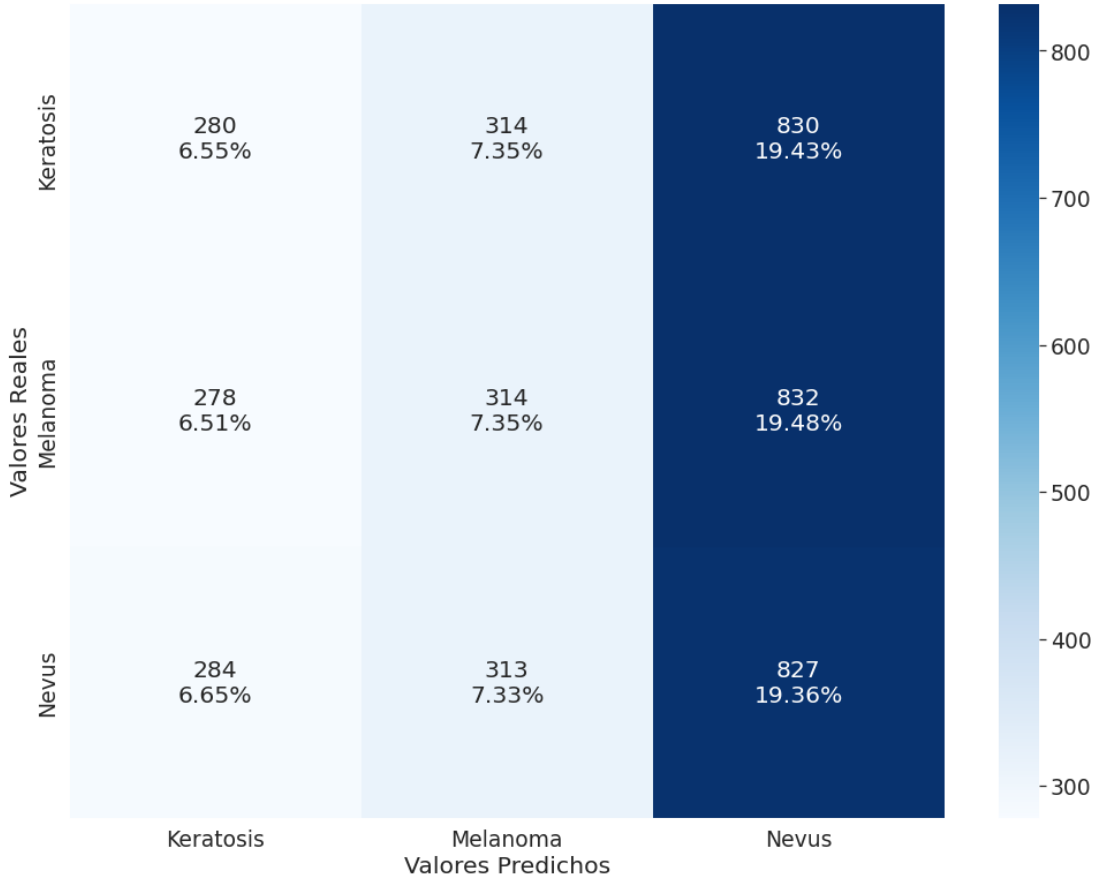

Figura 51. Matriz de confusión de la tercera prueba sobre red propia.

En la tercera prueba realizada sobre la red convolucional propia, se consiguen los resultados mostrados en la *Figura 51*, donde se puede observar que de nuevo se tiende a clasificar la mayoría de los ejemplos como Nevus. En este caso el resto de las clases quedan reducidas a una gran minoría, hasta el punto de que ni sumando todos los ejemplos clasificados como Melanoma o Keratosis se llega al número de imágenes clasificadas como Nevus.

8. CONCLUSIONES

8.1. Comparativa

Con el fin de aportar más información sobre el tema, mejorar el análisis de resultados y debido a que comparten un objetivo común, se van a comparar los resultados obtenidos en el proyecto aquí presentado con los obtenidos en el trabajo de fin de grado "Detección precoz de cáncer de piel en imágenes basado en redes convolucionales" desarrollado por Cristina Pérez Lorenzo en la Universidad Autónoma de Madrid [47].

	Proyecto propio	**Proyecto Cristina Pérez Lorenzo**
Tamaño del batch	**16**	**64**
Learning Rate	**0.001**	**0.001**
Epochs	**20**	**30**

Tabla 13. Comparativa de hiperparámetros entre proyectos.

En primer lugar, en la *Tabla 13* se presenta el ajuste de hiperparámetros realizado en cada uno de los proyectos comparados, esto se hace con el fin de poder entender las diferencias visibles en los resultados presentados.

Con el fin de obtener los mejores resultados posibles de esta comparativa, y debido a la existencia de un gran número de pruebas sumando los existentes en los dos proyectos comparados, se comparará únicamente la red que mejor resultado haya obtenido en cada uno de los dos trabajos.

	AlexNet (Cristina Pérez Lorenzo)	**Red propia (Autor)**
Precisión de entrenamiento	**0,7736**	**0,8408**
Precisión de validación	**0,7411**	**0,7743**

Tabla 14. Comparativa de precisión entre los dos proyectos.

Teniendo esto en cuenta, se compara el resultado de AlexNet presentado en la *Tabla 14*, con el resultado de la segunda prueba realizada sobre la red propia que se puede leer en la *Tabla 11*.

En la comparativa, se puede comprobar que la red original del autor, entrenada con las 26021 imágenes sin *Data Augmentation*, resulta más precisa que la que mejor resultados ha arrojado (AlexNet) en el estudio realizado por Cristina Pérez Lorenzo. Si bien la precisión es una métrica importante, sería interesante disponer de otras como la pérdida o la matriz de confusión con el fin de poder sacar conclusiones más específicas y comparar de forma menos genérica los resultados de ambos proyectos.

8.2. Conclusiones

A lo largo de este proyecto se han adquirido conocimientos sobre el mundo de la Inteligencia Artificial así como de algunas de sus ramas tales como el *Machine Learning* o el *Deep Learning.* Además, debido a que el desarrollo se ha centrado en redes neuronales y más concretamente en redes convolucionales, se ha aprendido mucho sobre dichos sistemas. Al haberse realizado sobre Python, mediante el uso de Jupyter, se ha conocido más en profundidad el lenguaje Python así como las ventajas e inconvenientes de utilizar *notebooks*. Por último, puesto que se ha realizado en diferentes plataformas como son una máquina virtual remota y Google Colab, se ha desarrollado un conocimiento sobre este tipo de plataformas.

Con respecto a los objetivos establecidos durante el capítulo *OBJETIVOS*, se han cumplido ampliamente. Los objetivos que aquí se planteaban, se centraban en ampliar el conocimiento tanto en los campos de la Inteligencia Artificial, Machine Learning y Deep Learning; cómo en ampliar el conocimiento de los módulos y las librerías utilizadas dentro del lenguaje Python para este tipo de proyectos. Dichos objetivos, como se ha mencionado en el párrafo anterior, han sido cumplidos.

En concreto, el estudio realizado durante el desarrollo del capítulo *ESTADO DE LA CUESTIÓN* ha ayudado mucho a realizar dichos objetivos, ya que durante la investigación necesaria para conocer bien todo lo que hay detrás de las disciplinas utilizadas a lo largo del trabajo, se ha conseguido recabar una gran cantidad de información y conocer mucho más en detalle dichas disciplinas. Otra cuestión que

también ha ayudado a cumplir dichos objetivos es lo relatado durante el punto *TECNOLOGÍAS UTILIZADAS*, durante dicho punto se explican en profundidad las tecnologías y módulos utilizados para desarrollar el proyecto, habiéndose necesitado más investigación con el fin de conocer bien todas estas tecnologías y ayudando al cumplimiento de los objetivos.

Por otro lado, se pueden extraer diversas conclusiones de los resultados obtenidos en el capítulo *IMPLEMENTACIÓN Y DESARROLLO* y la implementación que ha llevado a dichos resultados explicada en el punto *RESULTADOS Y DISCUSIÓN*.

Los resultados extraídos de los modelos entrenados son, por lo general, malos. Si bien las métricas de precisión y pérdida llegan a ser decentes e incluso buenas, al observar la matriz de confusión resultante de los modelos con los mejores resultados obtenidos, así como sus reportes de clasificación, los resultados que se pueden observar son bastante malos. Estos malos resultados no se deben a una situación de *overfitting* ya que en *epochs* anteriores los resultados empeoran aún más; por otro lado sí que podrían ser malos debido a *underfitting*, ya que como se ha mencionado en varias ocasiones, el número de imágenes de melanomas era inferior al que se encuentra para el resto de las clases.

Estos resultados se dan en gran parte por la falta de recursos que permitan pruebas realmente potentes es un tiempo aceptable, contribuye a esta falta de buenos resultados. Entre otros, la falta de imágenes de melanomas, de velocidad a la hora de realizar el entrenamiento y la falta de potencia para realizar correctamente la técnica de *Data Augmentation* han contribuido especialmente a los pobres resultados.

Por último, teniendo en cuenta el objetivo médico de una red como la aquí realizada, los resultados obtenidos se consideran realmente insuficientes para ser utilizada como herramienta de apoyo al diagnóstico. El problema aquí tratado es especialmente sensible debido al campo al que pertenece y a las implicaciones que podría tener un error en el diagnóstico de un melanoma. Se espera de este proyecto que pueda ayudar a otras investigaciones realizadas en el campo de la Inteligencia Artificial en la medicina.

8.3. Trabajo Futuro

En vista de los resultados obtenidos en este proyecto, aquí se proponen algunas mejoras que podrían suponer una gran diferencia en los resultados pero que no han podido ser exploradas por falta de herramientas, tiempo u otras razones:

- Mezclar los datos existentes con otros conjuntos de datos, ya sean los de la propia competición pero de otros años u otros conjuntos externos, con el fin de equilibrar de forma natural el *dataset* y conseguir un mayor número de datos con los que trabajar.
- Si bien se ha utilizado como ejemplo de las redes preentrenadas el modelo EfficientNetB6, lo cierto es que existen muchos y muy variados, por lo que puede que alguno de los modelos no utilizados consiguiera mucho mejores resultados que los aquí obtenidos.
- En el terreno de *Data Augmentation,* en caso de las imágenes que contienen piel y por lo tanto pueden contener pelos, se suele realizar un aumento de dicho pelo. Este aumento está pensado para que los modelos sean capaces de reconocerlo como algo que no aporta información, no teniendo en cuenta el pelo a la hora de clasificar la imagen en las distintas categorías.
- En lo referente a imágenes, cabría esperar mejores resultados si se utilizaran las imágenes a tamaño completo. La razón por la que no se ha realizado esto se por el hecho de que una imágenes tan grandes requerirían una gran potencia computacional de la que no se dispone.
- Se podrían mejorar los resultados trabajando más en la optimación de hiperparámetros, la cual es muy superficial debido a la falta de potencia de cálculo.
- La exploración de otro tipo de técnicas de generalización como el *k-fold cross validation* podrían ayudar a mejorar los resultados obtenidos en las pruebas realizadas.
- La introducción de los metadatos presentes en los datos correspondientes a la competición de 2020 podría mejorar las predicciones del modelo resultante, si bien se necesita, en caso de utilizar datos externos, una forma de etiquetar previamente los datos pertenecientes a competiciones de otros años.

REFERENCIAS BIBLIOGRÁFICAS

[1] H. A. Haenssle *et al.*, «Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists», *Annals of Oncology*, vol. 29, n.º 8, pp. 1836-1842, ago. 2018, doi: 10.1093/annonc/mdy166.

[2] «McCarthy - WHAT IS ARTIFICIAL INTELLIGENCE.pdf». Accedido: 19 de junio de 2022. [En línea]. Disponible en: https://borghese.di.unimi.it/Teaching/AdvancedIntelligentSystems/Old/IntelligentSystems_2008_2009/Old/IntelligentSystems_2005_2006/Documents/Symbolic/04_McCarthy_whatisai.pdf

[3] «What is Artificial Intelligence (AI)?», 16 de septiembre de 2021. https://www.ibm.com/cloud/learn/what-is-artificial-intelligence (accedido 18 de junio de 2022).

[4] A. M. TURING, «I.—COMPUTING MACHINERY AND INTELLIGENCE», *Mind*, vol. LIX, n.º 236, pp. 433-460, oct. 1950, doi: 10.1093/mind/LIX.236.433.

[5] A. Ghozia, G. Attiya, E. Adly, y N. El-Fishawy, «Intelligence Is beyond Learning: A Context-Aware Artificial Intelligent System for Video Understanding», *Comput Intell Neurosci*, vol. 2020, p. 8813089, 2020, doi: 10.1155/2020/8813089.

[6] Z. Borysiuk, M. Konieczny, K. Kręcisz, y P. Pakosz, «Application of sEMG and Posturography as Tools in the Analysis of Biosignals of Aging Process of Subjects in the Post-production Age», 2018, pp. 23-29. doi: 10.1007/978-3-319-75025-5_3.

[7] R. Fjelland, «Why general artificial intelligence will not be realized», *Humanit Soc Sci Commun*, vol. 7, n.º 1, Art. n.º 1, jun. 2020, doi: 10.1057/s41599-020-0494-4.

[8] «Machine Learning: ¿qué es y cuál es su relación con la IA?» https://otech.uaeh.edu.mx/noti/index.php/machine-learning/machine-learning-que-es-y-cual-es-su-relacion-con-la-ia/ (accedido 1 de julio de 2022).

[9] C. Janiesch, P. Zschech, y K. Heinrich, «Machine learning and deep learning», *Electron Markets*, vol. 31, n.º 3, pp. 685-695, sep. 2021, doi: 10.1007/s12525-021-00475-2.

[10] L. Pierson y J. Porway, *Data science*, 2nd edition. Hoboken, NJ: John Wiley and Sons, Inc, 2017.

[11] «What is Machine Learning?», 5 de noviembre de 2021. https://www.ibm.com/cloud/learn/machine-learning (accedido 20 de junio de 2022).

[12] «Transfer Learning: cómo entrenar modelos de Deep Learning de manera asequible | datos.gob.es». https://datos.gob.es/es/blog/transfer-learning-como-entrenar-modelos-de-deep-learning-de-manera-asequible (accedido 1 de julio de 2022).

[13] R. Pramoditha, «The Concept of Artificial Neurons (Perceptrons) in Neural Networks», *Medium*, 29 de diciembre de 2021. https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc (accedido 6 de septiembre de 2022).

[14] «Introduction to Convolutional Neural Networks with Weights & Biases on Weights & Biases». http://wandb.ai/site/articles/intro-to-cnns-with-wandb (accedido 1 de julio de 2022).

[15] S. Indolia, A. K. Goswami, S. P. Mishra, y P. Asopa, «Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach», *Procedia Computer Science*, vol. 132, pp. 679-688, ene. 2018, doi: 10.1016/j.procs.2018.05.069.
[16] «Construir una red neuronal desde cero (5) Red neuronal convolucional (CNN) - programador clic». https://programmerclick.com/article/377994238/ (accedido 1 de julio de 2022).
[17] «Convolutional Neural Networks, Explained | by Mayank Mishra | Towards Data Science». https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 (accedido 10 de septiembre de 2022).
[18] «The Ultimate Guide to Convolutional Neural Networks (CNN) - Blogs - SuperDataScience | Machine Learning | AI | Data Science Career | Analytics | Success». https://www.superdatascience.com/blogs/the-ultimate-guide-to-convolutional-neural-networks-cnn (accedido 1 de julio de 2022).
[19] Seb, «What is Pooling in a Convolutional Neural Network (CNN): Pooling Layers Explained - Programmathically», 5 de diciembre de 2021. https://programmathically.com/what-is-pooling-in-a-convolutional-neural-network-cnn-pooling-layers-explained/, https://programmathically.com/what-is-pooling-in-a-convolutional-neural-network-cnn-pooling-layers-explained/ (accedido 21 de junio de 2022).
[20] A. Almryad y H. Kutucu, «Automatic identification for field butterflies by convolutional neural networks», *Engineering Science and Technology, an International Journal*, vol. 23, feb. 2020, doi: 10.1016/j.jestch.2020.01.006.
[21] J. Acquarelli, E. Marchiori, L. Buydens, T. Tran, y T. van Laarhoven, «Spectral-Spatial Classification of Hyperspectral Images: Three Tricks and a New Learning Setting», *Remote Sensing*, vol. 10, p. 1156, jul. 2018, doi: 10.3390/rs10071156.
[22] «4. Convolutional Neural Networks - Learning TensorFlow [Book]». https://www.oreilly.com/library/view/learning-tensorflow/9781491978504/ch04.html (accedido 1 de julio de 2022).
[23] «Dropout In (Deep) Machine learning: A Simple Overview (2021)», 9 de marzo de 2021. https://www.jigsawacademy.com/blogs/ai-ml/dropout (accedido 19 de julio de 2022).
[24] C. Ranjan, «Simplified Math behind Dropout in Deep Learning», *Medium*, 17 de febrero de 2022. https://towardsdatascience.com/simplified-math-behind-dropout-in-deep-learning-6d50f3f47275 (accedido 19 de julio de 2022).
[25] S. Yıldırım, «Activation Functions in Neural Networks», *Medium*, 20 de agosto de 2020. https://towardsdatascience.com/activation-functions-in-neural-networks-eb8c1ba565f8 (accedido 12 de octubre de 2022).
[26] S. SHARMA, «Activation Functions in Neural Networks», *Medium*, 4 de julio de 2021. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (accedido 12 de octubre de 2022).
[27] «The architecture of our transfer learning model. | Download Scientific Diagram». https://www.researchgate.net/figure/The-architecture-of-our-transfer-learning-model_fig4_342400905 (accedido 2 de julio de 2022).
[28] «Bias and Variance in Machine Learning: An In Depth Explanation», *Simplilearn.com*. https://www.simplilearn.com/tutorials/machine-learning-tutorial/bias-and-variance (accedido 8 de agosto de 2022).

[29] S. Wickramasinghe, «Bias & Variance in Machine Learning: Concepts & Tutorials», *BMC Blogs*. https://www.bmc.com/blogs/bias-variance-machine-learning/ (accedido 8 de agosto de 2022).
[30] «¿Qué son los hiperparametros y para qué sirven en "machine learning"?», *Quora*. https://es.quora.com/Qué-son-los-hiperparametros-y-para-qué-sirven-en-machine-learning (accedido 8 de agosto de 2022).
[31] «Hyperparameter vs. Parameter: Difference Between The Two», *HitechNectar*. https://www.hitechnectar.com/blogs/hyperparameter-vs-parameter/ (accedido 8 de agosto de 2022).
[32] «Reducción de la pérdida: Tasa de aprendizaje». https://willarevalo.github.io/machine%20learning/2018/05/16/reducci%C3%B3n-de-la-p%C3%A9rdida-tasa-de-aprendizaje.html (accedido 9 de agosto de 2022).
[33] U. Vijay, «What is epoch and How to choose the correct number of epoch», *Medium*, 26 de septiembre de 2019. https://medium.com/@upendravijay2/what-is-epoch-and-how-to-choose-the-correct-number-of-epoch-d170656adaaf (accedido 9 de agosto de 2022).
[34] «Learning from Imbalanced Classes», *KDnuggets*. https://www.kdnuggets.com/learning-from-imbalanced-classes.html/2 (accedido 13 de octubre de 2022).
[35] J. S. JUN 24 y 2020 9 Min Read □ □ □ □, «Why and How to Implement Random Rotate Data Augmentation», *Roboflow Blog*, 24 de junio de 2020. https://blog.roboflow.com/why-and-how-to-implement-random-rotate-data-augmentation/ (accedido 13 de octubre de 2022).
[36] «sklearn.metrics.confusion_matrix», *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.metrics.confusion_matrix.html (accedido 17 de julio de 2022).
[37] «sklearn.metrics.classification_report», *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.metrics.classification_report.html (accedido 17 de julio de 2022).
[38] arvindpdmn Pawan_Dubey, «Confusion Matrix», *Devopedia*, 20 de agosto de 2019. https://devopedia.org/confusion-matrix (accedido 18 de julio de 2022).
[39] S.-C. B. Lo, H.-P. Chan, J.-S. Lin, H. Li, M. T. Freedman, y S. K. Mun, «Artificial convolution neural network for medical image pattern recognition», *Neural Networks*, vol. 8, n.º 7, pp. 1201-1214, ene. 1995, doi: 10.1016/0893-6080(95)00061-5.
[40] «The Data Science Process», *KDnuggets*. https://www.kdnuggets.com/the-data-science-process-2.html (accedido 12 de noviembre de 2022).
[41] K. Team, «Keras documentation: About Keras». https://keras.io/about/ (accedido 1 de julio de 2022).
[42] «Project Jupyter». https://jupyter.org (accedido 26 de junio de 2022).
[43] «ISIC Challenge». https://challenge.isic-archive.com/data/ (accedido 24 de julio de 2022).
[44] «The Melanoma Classification Challenge - Society for Imaging Informatics in Medicine», *SIIM.org*. https://siim.org/page/melanoma_classification_challenge (accedido 3 de noviembre de 2022).
[45] *SIIM-ISIC Melanoma Classification Challenge 1st Place Winner Solution*, (2 de septiembre de 2020). Accedido: 7 de noviembre de 2022. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=wXEDf0TFatA

[46] «SIIM-ISIC Melanoma Classification 1st Place Discussion». https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/175412 (accedido 7 de noviembre de 2022).
[47] C. Pérez Lorenzo, «Detección precoz de cáncer de piel en imágenes basado en redes convolucionales», jun. 2019, Accedido: 12 de septiembre de 2022. [En línea]. Disponible en: https://repositorio.uam.es/handle/10486/688935