Proyecto Fin de Carrera
Grado en Ingeniería de las Tecnologías de Telecomunicación

Clasificación de imágenes mediante algoritmos de Deep Learning: Mascarillas de COVID-19.

Autor: Marta Pérez Ortiz de Landaluce
Tutor: Susana Hornillo Mellado

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

Sevilla, 2021

Proyecto Fin de Carrera
Grado en Ingeniería de las Tecnologías de Telecomunicación

Clasificación de imágenes mediante algoritmos de Deep Learning: Mascarillas de COVID-19.

Autor:
Marta Pérez Ortiz de Landaluce

Tutor:
Susana Hornillo Mellado
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Carrera: Clasificación de imágenes mediante algoritmos de Deep Learning: Mascarillas de COVID-19.

Autor: Marta Pérez Ortiz de Landaluce

Tutor: Susana Hornillo Mellado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Después de estos años, se me hace complicado resumir en pocos párrafos cómo agradecer a quienes han estado conmigo en este camino tan bonito.

Mi familia, lo primero siempre. Gracias a ellos me he convertido en la persona que soy hoy y he logrado aquello que me he propuesto, porque con ellos nada me falta y son los únicos que nunca se van. Mamá, papá, Tita y Rafa, sois, sin duda, las personas más importantes de mi vida.

A mis abuelos, porque hacer que os sintáis orgullosos de vuestra nieta es la mayor motivación que tengo. Por cuidarme y quererme tanto, por devolveros el cariño que me dais. A los cuatro, os quiero muchísimo.

A mis amigas. Puedo escribir durante horas sobre ellas, pero me conformo con decir que tener un apoyo incondicional tan sincero es un tesoro que nunca quiero perder. Por creer en mí y empujarme hacia mis metas, habéis hecho que el camino sea fácil hasta cuando todo parecía hundirse; quedo en deuda con vosotras por haberme ayudado tanto.

A Samuel, por haberme cuidado estos años y los que quedan por venir.

Gracias a mi tutora Susana Hornillo por su comprensión y amabilidad. Por saber trasladarme sus conocimientos y mostrarse disponible en todo momento. Ha sido todo un acierto hacer este trabajo con ella. Al resto de profesores, gracias por indicarme el camino hacia la meta y dedicar tiempo a vuestros alumnos con tanto interés, me llevo muy buen recuerdo de todos.

Estas son sólo algunas de las personas que me han acompañado en esta inolvidable etapa, pero hay muchas más. Soy una afortunada de estar tan bien rodeada y espero, por difícil que parezca, estar a vuestra altura para haceros lo feliz que me hacéis a mí.

Por último, aunque a veces se me olvide, me gustaría recordarme a mí misma el esfuerzo que he hecho. Porque me sirva para no volver a dudar de mí y por aspirar a ser la mejor versión de mí misma cada día.

Marta Pérez Ortiz de Landaluce

Sevilla, 2021

Resumen

La pandemia mundial causada por la Covid-19 ha provocado un antes y un después en nuestras vidas, tanto, que ahora llevar mascarilla con el fin de frenar su contagio es algo primordial e impensable en determinadas ocasiones. A raíz de la desesperación originada por este virus se ha incrementado el interés en métodos científicos que puedan ayudar a estabilizar y controlar la situación. Este proyecto gira en torno a este tema tan actual, ya que persigue alcanzar una eficiente clasificación de imágenes según se lleve mascarilla o no, así como diferenciando también si se lleva de forma incorrecta. Para desarrollarlo, se han empleado redes neuronales convolucionales basadas en *Deep Learning*, algunos populares paquetes básicos de aprendizaje automático como es Keras o TensorFlow y el lenguaje de programación Python 3.6. Los resultados obtenidos en este experimento, usando las herramientas presentadas y trabajando para lograr un ajuste de parámetros que optimice el resultado, terminan con una precisión del algoritmo máxima de un 95.31 % para el diseño final seleccionado.

Abstract

The global pandemic caused by Covid-19 has caused a before and after in our lives, so much that now wearing a mask in order to stop its contagion is essential and unthinkable in certain occasions. As a result of the desperation caused by this virus, there has been an increased interest in scientific methods that could help stabilize and control the situation. This project revolves around this very current topic, since it seeks to achieve an efficient images classification depending on whether a mask is worn or not, as well as differentiating whether it is worn incorrectly. To develop it, convolutional neural networks based on Deep Learning, some popular basic machine learning packages such as Keras or TensorFlow and the Python 3.6 programming language have been used. The results obtained in this experiment, using the tools presented and working to achieve a parameter adjustment that optimizes the result, ends with a maximum algorithm precision of 95.31%.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

Notación

ANN	Red Neuronal Artificial (*Artificial Neural Network*)
CNN	Red Neuronal Convolucional (*Convolutional Neural Network*)
LR	Tasa de aprendizaje *(Learning Rate)*
CF	Función de coste *(Cost Function)*
ML	Aprendizaje Máquina *(Machine Learning)*
IA	Inteligencia Artificial
DL	Aprendizaje Profundo *(Deep Learning)*
MLP	Perceptrón multicapa *(Multi Layer Perceptron)*
BP	Propagación hacia atrás *(Back Propagation)*
FP	Propagación hacia delante *(Forward Propagation)*
BBDD	Base de datos
API	Interfaz de Programación de Aplicación (*Aplication Programming Interface*)
IDE	Entorno de Desarrollo Integrado (*Integrated Development Environment*)

1 INTRODUCCIÓN

"Algunas personas llaman a esto Inteligencia Artificial, pero la realidad es que esta tecnología nos mejorará. Entonces, en lugar de Inteligencia Artificial, creo que aumentaremos nuestra inteligencia".

- Virginia Rometty -

1.1. Motivación del proyecto.

La COVID-19 es una enfermedad infecciosa internacional provocada por una cepa de coronavirus nunca antes identificada, la cual amenaza a la población desde finales del año 2019, cuando todo este caos dio comienzo en un brote ocurrido en la ciudad china de Wuhan. Los síntomas de su contagio abarcan desde un leve resfriado hasta enfermedades respiratorias peligrosas, como graves neumonías o síndromes respiratorios agudos. El nuevo coronavirus ha dejado datos trágicos, con más de 4.5 millones de muertes y 219 millones de infectados.

Por su facilidad para expandirse, se declaró una emergencia sanitaria a nivel mundial el 11 de marzo de 2020 y es de extrema importancia el uso de medidas preventivas para frenar su evolución. Algunos ejemplos de estas medidas son lavarse las manos con frecuencia, mantener el distanciamiento social mínimo de 1.5 metros, el confinamiento de los infectados o, el epicentro de este trabajo, el uso de mascarillas faciales [1].

El virus se propaga mayoritariamente mediante gotículas creadas cuando un infectado las expulsa a través de tos, estornudos o por la propia respiración alrededor de un radio de dos metros desde su emisión. Debido a ello, las mascarillas son una excelente protección individual, puesto que estas gotas quedan contenidas en su interior, filtrando estas partículas aéreas en mayor o menor cantidad según sus características. Se debe recordar que la mayoría no son mascarillas reutilizables y su uso se puede prolongar de 4 a 8 horas según sus propiedades.

La mascarilla quirúrgica o convencional alcanza una eficacia de filtración en torno al 80%. Las más recomendadas por los expertos dada su alta seguridad son las mascarillas respiratorias N95 o FFP2, cuyo filtrado mejora alcanzando el 95%, al estar compuestas de múltiples capas de polipropileno [2] [3].

Figura 1-1. Mascarilla quirúrgica, FPP2 y KN95 (de izquierda a derecha).

1.2. Objetivos del proyecto.

Es una situación cotidiana que alguien tome la temperatura a los clientes y compruebe si se acude con mascarilla a la entrada de establecimientos, pero ¿Y si se desarrolla un algoritmo de aprendizaje profundo que automatice esta distinción entre llevar o no la mascarilla? Esta tarea es el principal objetivo para alcanzar en el trabajo expuesto, además de profundizar y estudiar los conceptos teóricos de las redes neuronales convolucionales que se van a implementar y las diferentes disciplinas científicas que se relacionan con ellas. Multitud de imágenes han de ser clasificadas con la mayor precisión posible, según se lleve mascarilla o no, o bien, se lleve de forma incorrecta sin cubrir nariz y boca.

Para completar la memoria, se van a proponer y analizar diferentes variantes de la red seleccionada, puesto que surgen diferentes conflictos y dudas con las implementaciones expuestas. La finalidad fundamental es, sin causar un mal ajuste que provoque una precisión no aceptable de la clasificación, optimizar los parámetros del modelo secuencial que rige a la red neuronal convolucional para obtener una clasificación satisfactoria. También se incorpora un estudio desde diferentes puntos de vista de la predicción final a raíz de la matriz de confusión, para conocer más sobre la variedad de métricas y sus utilidades.

Desde este enfoque de la Inteligencia Artificial se pueden diseñar aplicaciones de vigilancia y detecciones de alto rendimiento, que minimizan la necesidad de que alguien deba de estar regulando una situación parecida gracias a la automatización de los sistemas. Estas tareas pueden ser de gran utilidad en el panorama actual, sobre todo en el contexto de regulación del uso de mascarillas como se expone en esta memoria.

1.3. Estructura de la memoria.

A continuación, se presenta la arquitectura tomada para organizar la memoria acompañada de un breve resumen de los puntos a tratar. Los capítulos en los que se estructura son los siguientes:

- **Capítulo 1. Introducción**. En este primer capítulo, se argumenta la motivación para realizar el experimento. Además, se presentan los objetivos de este.
- **Capítulo 2. Introducción a la Inteligencia Artificial.** El segundo capítulo comienza con una breve descripción histórica y tecnológica de las diferentes etapas de la Inteligencia Artificial. También engloba los conocimientos técnicos a dominar sobre *Machine Learning* y *Deep Learning* para poder alcanzar los resultados esperados, señalando las diferencias entre ambas disciplinas.
- **Capítulo 3. Las redes neuronales artificiales:** Consiste en un estudio del algoritmo de *Deep Learning* para clasificación de imágenes, el núcleo principal del documento. Tras decidir aplicar técnicas de aprendizaje profundo por las prestaciones que tiene para esta tarea, se procede a realizar una completa descripción teórica de los conceptos que abarca la disciplina científica que se implementa posteriormente, sobre todo se detalla acerca de las redes neuronales artificiales y las redes neuronales convolucionales.
- **Capítulo 4. Implementación del proyecto**. Ahora, sabiendo qué se esconde bajo la codificación del proyecto, es hora de ejecutarlo. Se acompaña de una explicación sobre las herramientas requeridas, el entorno de trabajo, el lenguaje de programación y las librerías escogidas, el conjunto de datos utilizado y el propio código de entrenamiento de la red neuronal final.
- **Capítulo 5**. **Evaluación de los resultados y cambios del experimento**. Una vez ejecutado el proceso y logrados los resultados, es el momento de estudiarlos y realizar un análisis de ellos para evaluar el rendimiento del sistema. Se presentan fragmentos del código que pertenece al fichero empleado para la predicción y para obtener otras métricas interesantes, como la matriz de confusión. No obstante, se agregan al capítulo algunas modificaciones que resultan interesantes para ampliar los conocimientos sobre la red neuronal convolucional y los pasos previos dados para alcanzar el modelo final. Se introducen las redes pre-entrenadas.

- **Capítulo 6. Conclusiones**. En este último capítulo de la memoria, se sintetizan las conclusiones en una tabla y se recapitulan los aspectos más importantes que han sido expuestos en el transcurso del documento. Para finalizar, se plantean futuros trabajos orientados a mejorar el método utilizado.

2 INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

En este apartado se estudian en detalle las disciplinas de Inteligencia Artificial, Aprendizaje Máquina y Aprendizaje Profundo, siendo esta última la herramienta más adecuada para llevar a cabo una eficiente clasificación de imágenes de la forma que la haría un cerebro humano o incluso más avanzada. Se describen todos los conceptos requeridos para lograr comprender el proyecto desarrollado en los siguientes capítulos.

Se hace énfasis en el aprendizaje profundo, pues es principal punto de interés del trabajo ya que ha sido escogido por su flexibilidad e inmensa capacidad para cumplir con los requisitos del proyecto.

2.1. Introducción a la Inteligencia Artificial.

Se entiende por Inteligencia Artificial (IA) a la rama de la ingeniería que persigue que las máquinas actúen de forma análoga o superior a la inteligencia cognitiva humana. Debido a ello, las máquinas pueden presentar habilidades, razonar, ser creativas o incluso mostrar emociones de acuerdo con el comportamiento humano, mediante algortimos matemáticos que están constantemente en investigación para mejorar las capacidades de aprendizaje

Para poder entender de mejor manera los apartados que vienen a continuación, lo mejor es hacer una breve introducción de qué es el aprendizaje máquina, el aprendizaje profundo y las redes neuronales, a pesar de que después se expliquen en detalle. El aprendizaje máquina es una disciplina secundaria de la IA, que se basa en el aprendizaje automático y en dotar a las máquinas simular un aprendizaje cognitivo. Al adentrarse en el aprendizaje máquina, se encuentra el aprendizaje profundo, un grupo de algoritmos con mayor nivel de abstracción y menor supervisión humana para realizar estas tareas inteligentes. Pero aquello que permite obtener la profundidad son las redes neuronales; sistemas computacionales formados por tantas capas interconectadas entre ellas hasta alcanzar esta 'profundidad' deseada y constituidas por neuronas artificiales que emulan los sistemas biológicos, a través de las cuales viajan las señales de información.

2.1.1. Contexto histórico y etapas de la IA.

Para desarrollar el contexto histórico en el que se desarrolló la Inteligencia Artificial, se van a diferenciar tres etapas de su proceso evolutivo. En capítulos posteriores, se explican rigurosamente los conceptos que se introducen ahora, siendo el objetivo situar temporalmente las tecnologías a modo de resumen.

La primera etapa comenzó en los años 60 y se basó en la ingeniería del conocimiento experto, donde se crearon programas informáticos con reglas lógicas acerca de ciertos dominios de aplicación muy restringidos, que emulaban la capacidad humana de toma de decisiones.

Como inconveniente, estos sistemas no cuentan con capacidad de aprendizaje y no generalizan bien las situaciones desconocidas. La mayor ventaja de estos sistemas de IA es su transparencia de cara a realizar el razonamiento lógico. A pesar de ser programas limitados, no daban lugar a incertidumbres y esto sigue siendo favorable en determinadas ocasiones.

Durante la década de 1970 y principios de la de 1980, el enfoque de sistema experto para el reconocimiento de voz, un gran desafío, se hizo bastante popular. Sin embargo, la falta de habilidades generales para aprender a partir de datos y para manejar la incertidumbre en el razonamiento cognitivo fue muy reconocida por los investigadores.

La segunda generación de IA aterrizó en la década de 1980 tras una evidente necesidad de mejorar los sistemas expertos anteriores para lograr la capacidad de aprendizaje solicitada en modelos complejos, con control de incertidumbre y generalización, pues los sistemas previos fueron basados sólo en conocimiento.

En esta etapa, la IA se centra en *Machine Learning* muy superficial, donde no hay que especificar reglas estrictas o guías de pasos como antes. Esta vez se enfocan en redes neuronales simples y después ajustan los parámetros de forma automática a partir de los datos de entrenamiento.

El paso a la tercera etapa tuvo lugar cuando estas redes adquirieron una mayor complejidad no asumible respecto a los algoritmos, métodos e infraestructuras.

La tercera ola de IA se inició una década atrás y continúa en la actualidad. El nivel de desempeño logra ser similar al humano gracias al aprendizaje profundo. En este paradigma, no se requiere tanta supervisión humana como antes, actividad que provocaba un efecto de cuello de botella a la hora de extraer características manualmente.

De este modo, esta generación gira en torno al *Deep Learning* a raíz del aumento de capas en las redes neuronales, que hizo resurgir el interés por las redes.

Algunas de las aplicaciones de IA han florecido a causa del éxito de las mejoras son el reconocimiento de voz (2010), la visión por computadora y la traducción automática. Así como otras más pertenecientes al mundo real, por ejemplo, análisis de imágenes médicas o los coches autónomos.

Adicionalmente, este progreso se ha combinado con un mejor software y herramientas para una implementación menos problemáticas y más rápida [4].

2.2. Introducción al Aprendizaje Máquina.

El aprendizaje máquina consiste en imitar comportamientos cognitivos artificiales a partir de emulaciones del entorno adyacente. El *Machine Learning* (ML) es considerado una subdisciplina dentro de la Inteligencia Artificial, que se basa en el aprendizaje automático y se centra en dotar a las máquinas de esta capacidad de aprendizaje. Para entender cómo funciona, es necesario saber diferenciar entre programar una máquina para que haga una acción y programarla para que, por ella misma de forma autónoma, aprenda a hacer algo; un concepto sustancial en ML, pero no imprescindible en la Inteligencia Artificial [5].

A pesar de que el aprendizaje máquina se ha convertido en un elemento dominante en el ámbito de la IA, cuenta con ciertos inconvenientes. En primer lugar, consume demasiado tiempo y recursos, a lo que se le añade posibles dificultades para la obtención de los datos. Asimismo, todavía no se considera una medida real de la inteligencia de la máquina, ya que se sirve de la participación humana para proponer las características que le permiten aprender e interpretar los resultados [6].

En la figura 2-1 se visualiza este grupo de sistemas científicos, desembocando en el maravilloso mundo de *Deep Learning* que roba la atención de la IA actual. Cada tecnología es una subcategoría interna de la que le rodea.

Una aplicación característica del aprendizaje máquina es la clasificación de imágenes, un estudio que puede tener distintos grados de supervisión humana según el método experimentado. Los métodos se inclinan hacia posiciones con menor intervención de usuarios, ya que un etiquetado manual puede consumir mucho tiempo y tener un coste elevado, además de estar sujetos a probables actualizaciones de los datos en un futuro. Se dividen en tres subgrupos generales: el aprendizaje supervisado, el aprendizaje débilmente supervisado y el aprendizaje no supervisado.

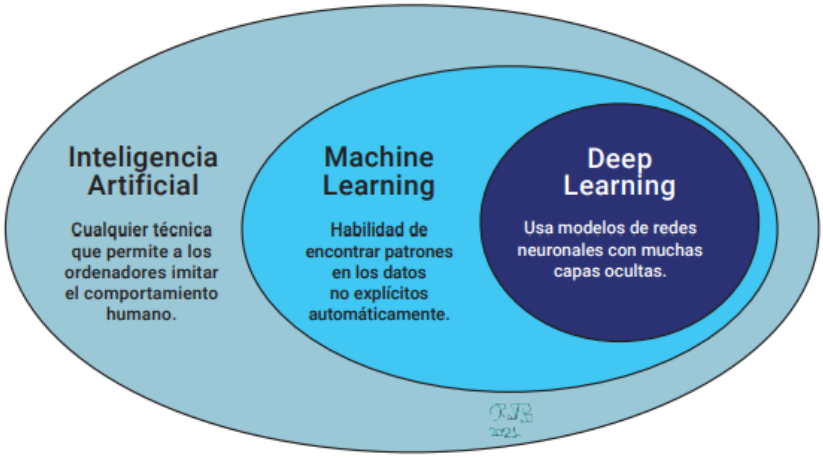

Figura 2-1. Evolución de la IA, ML y DL [7].

2.2.1. El aprendizaje supervisado.

El **aprendizaje supervisado** es aquel que descubre las relaciones existentes entre unas variables de entrada y otras de salida, debido a que se le muestran multitud de ejemplos anteriormente etiquetados o clasificados para el entrenamiento, de forma que el algoritmo aprenda a actuar en base a estas soluciones. Al mostrarle los resultados deseados, el ser humano participa en este algoritmo supervisándolo. El sistema se pone a prueba y si no acierta, se ajustan los parámetros de la máquina en la siguiente vuelta para aumentar la precisión y mejorar el rendimiento.

Mientras más cantidad y calidad de datos presente el sistema, más éxito debe tener. Por tanto, se recomienda tener bases de datos extensas [8] [9].

2.2.2. El aprendizaje no supervisado.

El **aprendizaje no supervisado** logra un resultado sin explicar al sistema qué se desea obtener, pues usa únicamente los datos ofrecidos a la entrada. No cuentan con la necesidad de que nadie supervise la respuesta ni ningún ejemplo etiquetado para comprobar que se está haciendo la clasificación pertinente, ya que no hay intervención humana directa. Consiste en un algoritmo que busca patrones de similitud en los datos de entrada, tratando de descubrir la estructura interna de ellos y de sus posibles variantes.

Las bases de datos no supervisadas son mucho más sencillas y básicas. A pesar de aspirar a tener un futuro prometedor, estas redes no están completamente implementadas todavía y siguen bajo estudio [8] [9].

2.2.3. El aprendizaje débilmente supervisado.

El **aprendizaje débilmente supervisado** utiliza otra información útil de distinta naturaleza, no usa las etiquetas, para derivar desde ella alguna información adicional sobre la disparidad entre las distintas categorías, sin usar el conocimiento fundamental como lo hace el lenguaje supervisado. Algunos ejemplos de etiquetado sencillo a nivel de imágenes son etiquetas de texto, frases o metadatos geográficos, como es simplemente indicar la ausencia o presencia de un objeto en la fotografía [10].

En el ejemplo de la figura 2-2, se muestra una clasificación a través del método supervisado y no supervisado. Con la supervisión, el resultado del problema es enseñar e indicar las cuatro clases según su forma geométrica y el color, que se identifica como la luz emitida por los píxeles. La red aprende a hacer la clasificación que el usuario le ha indicado previamente con etiquetas, por esa razón conoce el nombre de los elementos.

Por otro lado, si se emplea aprendizaje no supervisado, se tienen las imágenes de entrada que se quieren ordenar en categorías sin más información. Para solucionarlo, el algoritmo separa según formas y colores, pues ha ido

detectando similitudes, aunque esta vez desconoce el nombre de los elementos a pesar de detectar diferentes patrones entre ellos.

Las redes neuronales y, por tanto, el *Deep Learning* están fundamentados en el lenguaje supervisado, aunque en un grado más débil que otros algoritmos de *Machine Learning*. Como norma general, mientras más variedad de estrategias de aprendizaje se usen, mejor precisión de estimación se obtiene en una predicción [11].

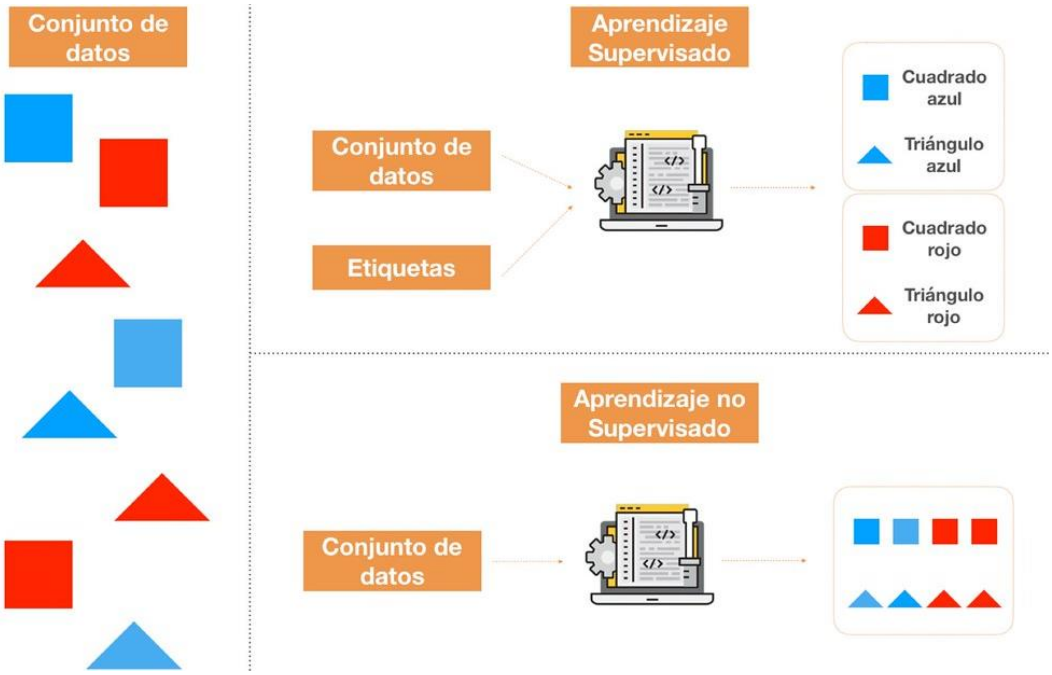

Figura 2-2. Clasificación con aprendizaje supervisado y aprendizaje no supervisado.

2.3. Introducción a *Deep Learning*.

Para entender qué es el aprendizaje profundo, hay que aclarar sus distinciones con respecto a otras disciplinas que pertenecen al campo de la Inteligencia Artificial. Como rasgo distintivo de los procedimientos, en el aprendizaje máquina el usuario es el encargado de extraer las características de los datos de entrada y esta experiencia implica a un operador humano que ayude a la máquina a aprender, proporcionándole cientos o miles de ejemplos de entrenamiento y corrigiendo a mano sus errores, actividad que dispara el tiempo que se le dedica [12].

La técnica de *Deep Learning* (DL) se define como un subaprendizaje de *Machine Learning*, cuyos modelos ya incluyen como parte de su procedimiento una extracción de características. Para la tarea en cuestión de clasificar imágenes, únicamente se le ofrece a la red una figura con una etiqueta previa que indica la clase a la que pertenece, donde cada entrada de la red profunda es un píxel, siendo el propio algoritmo el encargado de encontrar en qué se diferencian los patrones y las peculiaridades de las categoría sin más participación humana.

En lugar de estar basado en la lógica lineal, el aprendizaje profundo se sustenta en teorías acerca de cómo funciona el cerebro humano. No hay una serie de instrucciones indicando que, si una característica es detectada, se ha de realizar otro paso siguiente. El programa está constituido por capas anidadas de nodos interconectados entre ellos. Después de cada nuevo ciclo, la red aprende reajustando las conexiones o pesos entre las neuronas para minimizar el error.

El aprendizaje profundo tiene potencial para aplicaciones desde, por ejemplo, reconocer objetos en fotografías o clasificación de imágenes hasta otras prácticas más abstractas como la creación de técnicas capaces de determinar emociones o eventos que aparecen en un texto incluso sin ser citados explícitamente. Con su progreso, incluso es viable la realización de predicciones acerca del posible futuro comportamiento de las personas [13].

3 LAS REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales, también denominadas ANNs, constituyen un caso particular de algoritmos de aprendizaje máquina y quedan estrechamente relacionada con el aprendizaje profundo, siendo el núcleo de su funcionamiento.

3.1. Introducción

Las redes neuronales se describen como el conjunto de algoritmos de *Machine Learning* más popular, ahora en auge por la mejora tecnológica que sustenta la gran intensidad de operaciones aritméticas que involucran. Esta mejora se debe en gran medida al uso de GPUs o unidades de procesamiento gráfico, las cuales son unas plataformas encargadas de optimizar los procesamientos en paralelo, que provoca mayor facilidad, rapidez y menor coste [14].

A principio de los 50, surgió cierto interés en simular funciones características de los sistemas nerviosos de los animales. Sin embargo, las primeras redes neuronales artificiales creadas no podían detectar patrones complejos ya que sólo interactuaban una cantidad limitad de neuronas simultáneamente. Por ello, pasaron a un segundo plano de la Inteligencia Artificial. En la década de los 80, resurgió el interés en esta disciplina al descubrir los modelos de aprendizaje profundo como consecuencia del aumento de capas ocultas intermedias y hoy en día desempeñan un papel fundamental en este campo [15].

3.2. Conceptos básicos

Las redes neuronales artificiales son capaces de aprender de forma jerarquizada y adaptativa. En las primeras capas se aprende a procesar los conceptos más primitivos y en las posteriores, a partir de las salidas adecuadas de la capa previa que se consideran las entradas de la siguiente capa conectada, se obtiene información más compleja.

En las ANNs no hay un límite establecido de capas máximas a añadir y es este concepto el que da lugar a la profundidad de la red. Por lo tanto, aumenta la dificultad de las tareas, debido a multitud de iteraciones para el reconocimiento progresivo de objetos [16].

La figura 3-1 muestra una red con cuatro capas: la capa de entrada y la capa de salida siempre están presentes. Entre ambas, se insertan dos capas escondidas, con conexiones unidireccionales hacia delante.

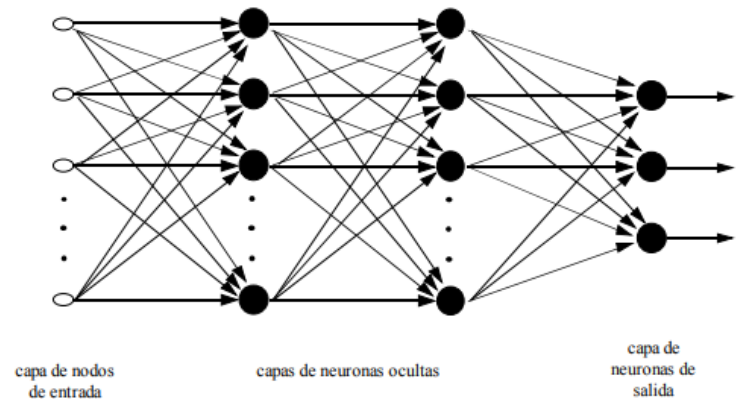

Figura 3-1. Red neuronal artificial multicapa [17].

3.2.1 Descripción de una neurona biológica.

Se comienza con una explicación sintetizada acerca del funcionamiento estándar de una neurona biológica, el elemento más importante del sistema nervioso humano, para después asociarla a la neurona artificial.

Una neurona está compuesta por tronco o soma, unido a múltiples extensiones denominadas dendritas, que es el lugar por donde se reciben las entradas o impulsos. Al otro lado del soma, tiene una extensión final de salida llamada axón. La sinapsis se encarga de conectar neuronas con otras, relacionando el axón de una y las dendritas de la contigua. En la figura 3-2 se señalan las partes de la neurona.

Como redes neuronales que forman, tienen una estructura basada en capas interconectadas y estas conexiones, por donde mandan señales, se asocian a pesos sinápticos [18] [19].

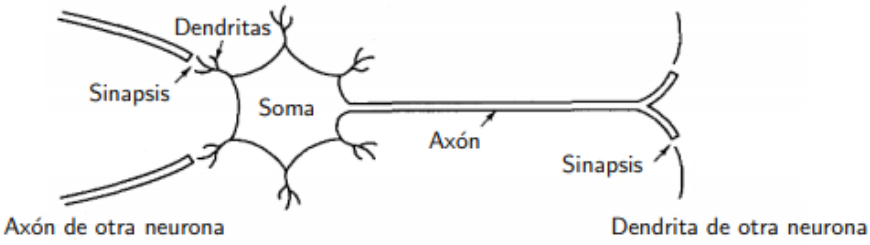

Figura 3-2. Partes de una neurona biológica [20].

Ahora, se entiende con más facilidad el comportamiento de una neurona en la red neuronal artificial. Esta neurona es la unidad básica de procesamiento, la cual tiene una conexión de entrada por la que recibe estímulos, con los que se activa la neurona a través de una función de activación y se ejecuta un cálculo interno. Una vez terminado, se entregan unos valores de salida que podrán transmitirse a otra neurona interconectada.

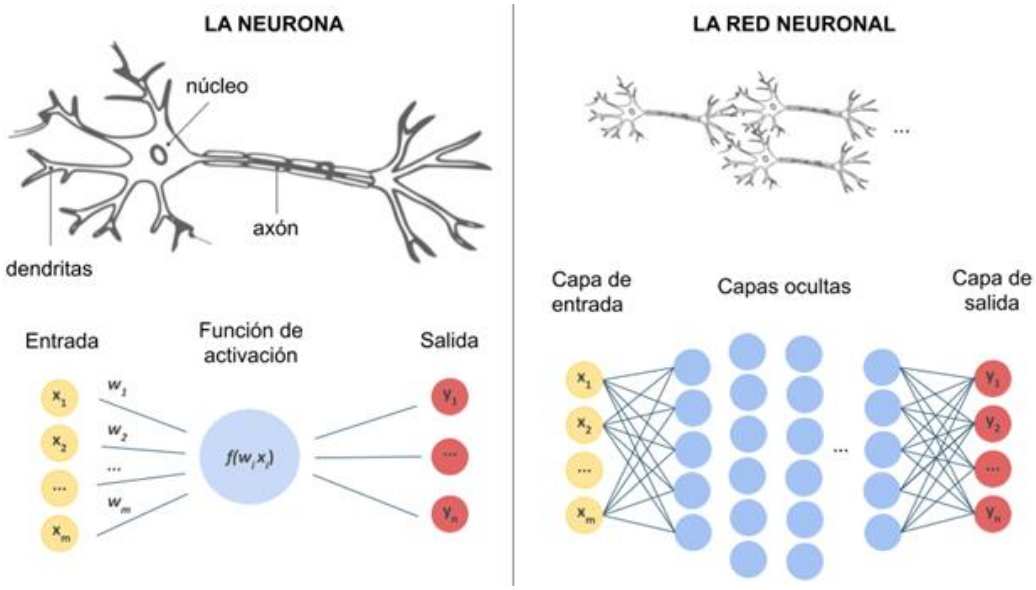

Figura 3-3. La arquitectura de la neurona y la red neuronal artificial y biológica.

3.2.2 Perceptrón

El perceptrón es la red neuronal más básica y se corresponde con el modelo matemático de una única neurona biológica, por lo tanto, está bastante limitada. Sólo es capaz de resolver problemas booleanos lineales, puesto

que para otros modelos más complejos hay que aumentar el número de neuronas y apilar más capas intermedias, concepto que da lugar al perceptrón multicapa (MLP).

En la figura 3-3 se expone un esquemático de la arquitectura de una simple neurona o perceptrón y la red resultante al unir varias de ellas, así como una comparación con su estructura artificial.

Pesos y sesgo

El cálculo intrínseco que realiza el perceptrón usa toda la información que recibe para realizar una ponderación de la suma de las entradas, determinada por los **pesos**. Los pesos son los parámetros del modelo que definen cuanto afecta la variable de entrada a la neurona según su valor y se van ajustando progresivamente al iterar para lograr una clasificación con el menor error posible entre el resultado esperado y el obtenido.

La neurona posee un **sesgo** o **bias**, una constante que representa otra conexión donde la variable que multiplica al peso es unitaria, para evitar multiplicaciones por cero. El sesgo es un componente del sumatorio que se controla únicamente variando su valor, dado que la entrada no repercute para nada. En una neurona con salida binaria, hay que evaluar el resultado numérico continúo extraído de la regresión lineal y según supere o no un cierto umbral, se le asigna un valor a la salida u otro. El sesgo se usa para igualar estas expresiones a cero, es decir, es equivalente a deshacer el efecto del umbral real creando un umbral artificial en cero; se le conoce también como **desplazamiento** o **umbral de activación** [21].

Las variables de entrada al modelo pueden ser desde binarias hasta un vector de muchas muestras y es la primera capa la que tiene la dimensión del tamaño inicial de la red.

En la figura 3-4 se muestra la estructura del perceptrón. La neurona recibe p valores de entrada (x_p) y cada uno de ellos se ve afectado por un peso (w_{kp}). Después, se hace un sumatorio de todos estos productos y se le suma el sesgo (θ_k). La expresión que resulta a cada salida tras fijar el umbral de la suma ponderada de las entradas se rige por la ecuación 3-1, aunque no está teniendo en cuenta la función de activación, que usa esta expresión como argumento [22].

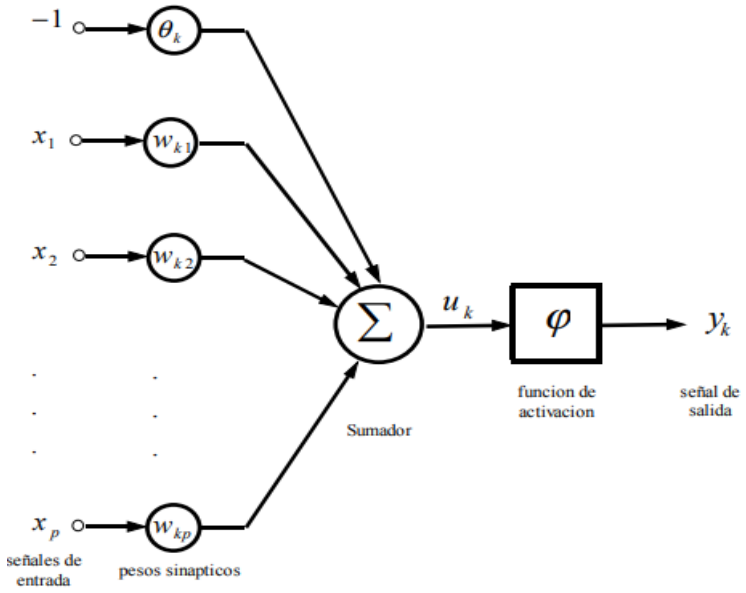

Figura 3-4. Estructura matemática de un perceptrón [17].

$$S_i = \theta_i + \sum_{j=1}^{p} x_j w_{i,j}$$

Ecuación 3-1. Salida del perceptrón.

3.2.3 Ajuste de parámetros para la clasificación.

Se puede entender el ajuste de parámetros desde otro punto de vista, como la forma de lograr una recta que sirva de frontera para lograr la clasificación buscada separando las categorías. Como ejemplo, en caso de que sea la recta deseada sea lineal, los problemas se pueden solucionar con una única neurona. Pero si la separación es más compleja, se aumenta el número de neuronas y capas, una técnica equivalente a usar más rectas para lograr la clasificación. Algunos ejemplos se ven en la figura 3-5 para ser explicados a continuación.

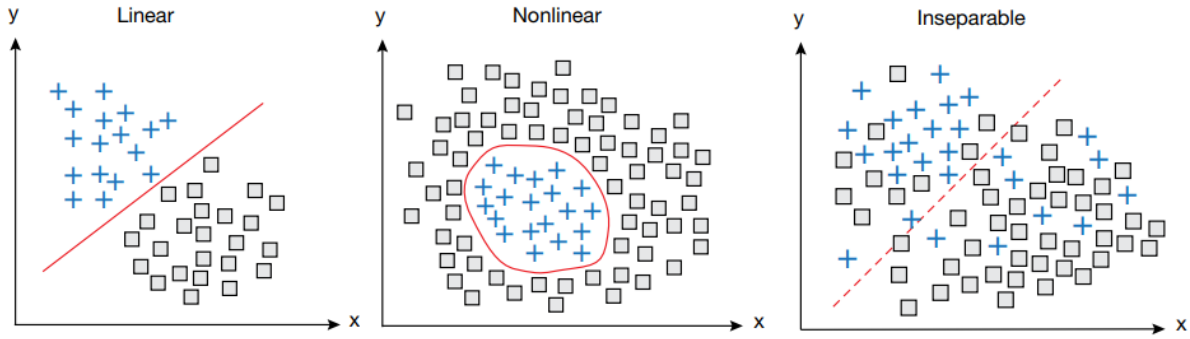

Figura 3-5. Diferentes casos de clasificación.

En el primer caso, se toman dos nubes de puntos gaussianas y estas se pueden separar linealmente: es un ejemplo sencillo, que se soluciona ajustando los parámetros de un simple perceptrón. Es decir, este sistema genera la clasificación sin necesidad de neuronas ni capas adicionales, se separan las nubes de puntos sin no linealidades.

El siguiente problema es más complejo, consiste en tener una nube de puntos exterior que rodea a otra interior y se desea lograr la frontera con una circunferencia. Si se usa una distinción como la de antes, con una línea, no se logra separar correctamente. Consecuentemente, hay que añadir más neuronas al problema que tengan funciones de activación que vayan añadiendo no linealidades al plano para encontrar esa superficie donde se separan los puntos. Si, por ejemplo, se añade una capa oculta con una única neurona no se logra nada, debido a que desde esa neurona hasta la de salida no hay ninguna manipulación. Insertando dos neuronas, añadiendo cada una su respectiva no linealidad, sí se consigue resolver.

El último ejemplo se puede resolver por difícil que parezca, de nuevo aumentando considerablemente el número de capas y neuronas en ellas. Y así para todos los casos que se puedan imaginar.

Como se ha manifestado, una red neuronal avanzada ya puede ajustar sus propios parámetros para aprender una representación interna de la información que le llega. Este proceso se desarrolla mediante el algoritmo de ***back propagation*** (BP). Gracias al método del descenso del gradiente se obtiene un vector de direcciones que indica pendiente de la función hacia donde el error se incrementa, conocido como vector gradiente. Por tanto, al extraer información de él e iterando se puede reducir el error del modelo.

3.2.4 Descenso del gradiente.

El conocido algoritmo de *Machine Learning* de descenso del gradiente es un método iterativo en el que el fin es minimizar una función de coste (CF) al hallar la derivada parcial de la función que representa la pendiente o gradiente. A partir de un comienzo con valores asignados de forma aleatoria, los coeficientes se vuelven a calcular en cada repetición. Así, se reducen en cada iteración, tomando el negativo de la derivada y siguiendo el ritmo que marca una tasa de aprendizaje (*learning rate,* LR), puesto que indica el tamaño del paso. Después, se multiplican por la derivada para que los mínimos locales se alcancen tras algunas iteraciones. Entonces, el proceso, correspondiente al de la figura 3-6, se detiene si converge al valor mínimo de la función y no se puede

decrementar más: se ha alcanzado el óptimo.

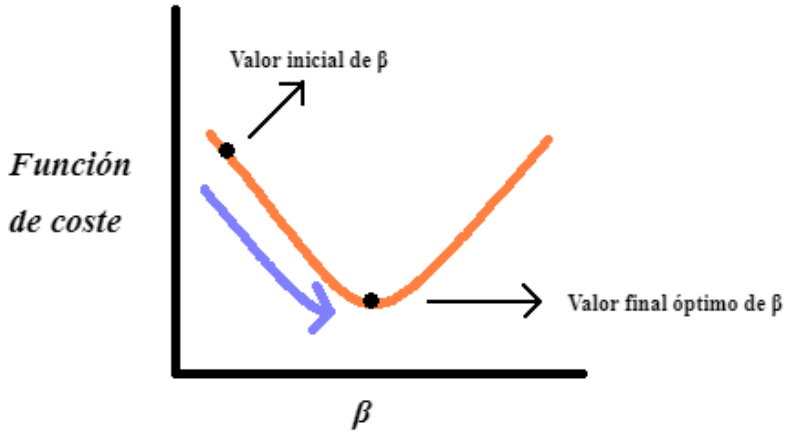

Figura 3-6. Método del descenso del gradiente.

La técnica de descenso del gradiente tiene un inconveniente y es que, si la LR indicada es demasiado rápida, va a pasar por alto el verdadero mínimo local para optimizar el tiempo. Pero por otro lado, si se escoge una tasa demasiado lenta, es posible que la función no llegue a converger porque se encuentra en bucle tratando de encontrar el mínimo local exacto. Se muestran las diferentes posibilidades de elección de la tasa y cómo afectan al calcular la derivada parcial en la figura 3-7.

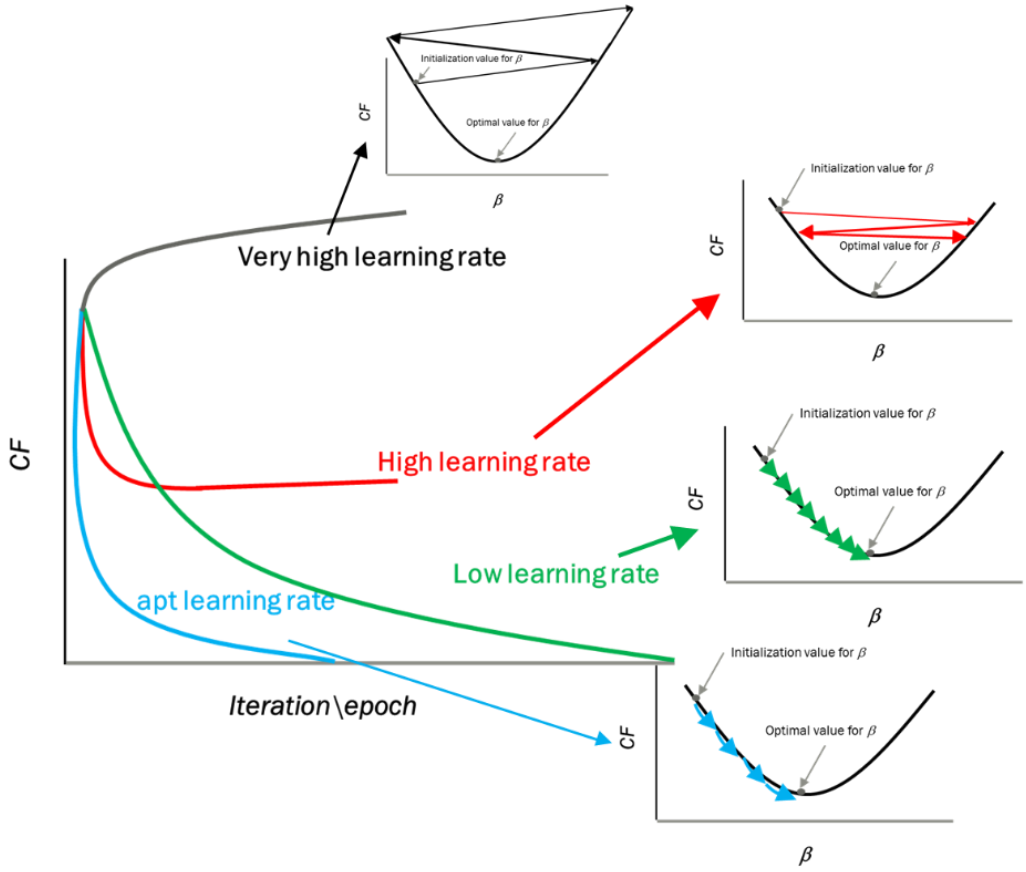

Figura 3-7. Situaciones de descenso de gradiente según la tasa de aprendizaje.

La tasa de aprendizaje afecta directamente al valor del mínimo se alcanza y qué tan rápido se alcanza. La práctica recomendada es basa en ir alterando este valor hasta encontrar uno que no ralentice demasiado el proceso y el error se vea disminuido lo suficiente [23] [24].

Actualmente, existen herramientas para calcular la tasa de aprendizaje de forma dinámica según convenga. Se rige por un sencillo principio de funcionamiento: cuando la pendiente de la función de coste es empinada, se adelante que el mínimo no es un punto cercano y para avanzar rápido, la tasa toma un valor alto. A medida que la curva se va suavizando, los pasos son más pequeños, pues el mínimo queda cerca. La figura 3-8 expuesta a continuación representa ambos comportamientos y cómo el valor del LR depende del gradiente, haciendo que el aprendizaje acelere y decelere [25]. A pesar de ser un mecanismo muy interesante, queda excluido del alcance de este proyecto, que emplea una tasa de aprendizaje fija.

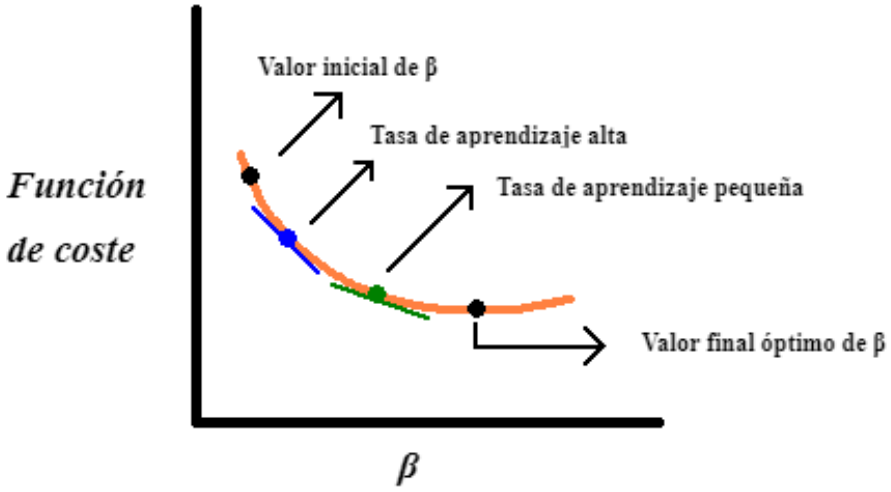

Figura 3-8. Descenso del gradiente con una tasa de aprendizaje dinámica.

3.2.5 Algoritmo de *Forward Propagation* y *Back Propagation*

Se asocia a un proceso de "ida y venida", es decir, antes de poder realizar una propagación hacia atrás hay que hacer una hacia adelante, llamada *forward propagation* (FP). En términos generales, la FP se focaliza en predecir imágenes dadas como entradas y calcular la función de error (*loss*), mientras que el BP actualiza las neuronas y sus pesos con derivadas parciales de una función de coste [26].

Para aplicar la retro propagación, se requiere tener el gradiente y combinarlo con regresión lineal; pero con redes neuronales, ¿Cómo varía el coste de la función cuando se cambia un parámetro de la red? La cuestión se resuelve al calcular la derivada parcial de la función de coste respecto a cada variable. No obstante, ya es más complejo debido a la multitud de conexiones con el resto de las neuronas de la misma capa y de las capas posteriores. Es decir, un cambio en un parámetro afecta mucho más que a una red básica y el algoritmo esconde una base matemática complicada, por lo que no se entrará en mayor detalle sobre ella al no ser el enfoque tomado para el proyecto.

Como se ha mencionado, el algoritmo de *back propagation* calcula el vector de gradientes dentro de la compleja red neuronal. Es una análisis hecho hacia atrás, tiene sentido hacerlo así porque en una red neuronal el error de las capas depende de forma directa del error de las capas anteriores y cada neurona tiene cierto porcentaje variable o peso en un error final. Con la retro propagación de errores se trabaja de forma eficiente, empezando por la última capa e indicándole a las neuronas anteriores a esta última cuánto afectan en el error. Una vez completado el reparto del error es fácil modificar los parámetros. Después, se va operando de forma recursiva capa tras capa moviendo el error hacia atrás una única vez y también se calcula la responsabilidad de cada neurona en el resultado.

Este proceso evita las muchísimas combinaciones que había que hacer hacia delante, siendo muy enrevesado entrenar a la red y gestionarla por la elevada carga computacional. Ahora, solo se hace un único cálculo de regresión lineal, ya que se desea que aprendan por ellas solas en el entreno [27] [28].

Cabe recordar que este algoritmo optimiza la red haciendo uso del descenso de gradiente, que minimiza el coste de la red, puesto que la BP lo habilita calculando las derivadas parciales del vector gradiente.

En la figura 3-9 se muestra un esquema visual de los pasos que sigue el entrenamiento neuronal.

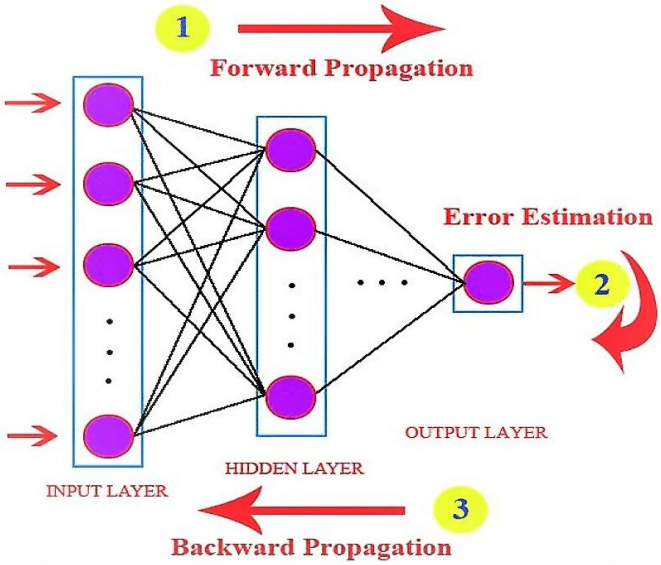

Figura 3-9. Algoritmo de *Forward Propagation* y *Back Propagation* [29].

Como se introdujo con anterioridad, las neuronas de la red se organizan en capas neuronales de tres tipos: capa de entrada, capas ocultas y capa de salida. Se pueden colocar neuronas en la misma capa, recibiendo la misma información de entrada de la capa anterior y dándole a la siguiente la misma salida que sus adyacentes. Si las neuronas se colocan secuencialmente, cada una de ellas recibe la información que ha procesado la anterior, haciendo referencia al comportamiento jerárquico mencionado. En el proceso de *Deep Learning* se conectan muchas neuronas secuencialmente con capas ocultas y cada una de ellas realiza un problema de regresión lineal, pero matemáticamente, sumar muchas líneas rectas da lugar a otra línea recta, la cual se puede lograr con una única neurona u operación. Esta redundancia se puede evitar con distorsiones o no linealidades en las rectas, a través de las **funciones de activación**.

3.2.6 Funciones de activación.

Las funciones de activación son descritas como aquellas con grado mayor de uno y alguna curvatura o distorsión en su trazado. En la neurona, en vez de obtener sólo como salida una suma ponderada que provenga de los datos de entrada, este cálculo pasa a ser el argumento de una función de activación que añade deformaciones no lineales, cuyo mapeo entre entradas y salidas depende de la función de activación seleccionada. Las funciones de activación se dividen en dos grupos: funciones acotadas y funciones no acotadas [30].

- Las **funciones acotadas** sólo activan la neurona si el valor de entrada pertenece a un rango de valores determinado.
- Las **funciones no acotadas** usan modelos donde no se establecen límites de estos valores para activar la neurona.

Al realizar el diseño de la red neuronal, se ha de indicar cómo son estos parámetros de activación y escoger previamente, según la finalidad y el tipo de datos, entre una de las posibles funciones de activación expuestas a continuación. La distorsión se aplica en último lugar del cálculo interno neuronal. En resumen, los tipos de funciones más comunes son los siguientes: *linear*, *sigmoide*, *tangente hiperbólica*, *unidad rectificada lineal* y *softmax*.

La función **linear** actúa de manera directamente proporcional a la señal de entrada, donde se mantiene una razón escogida por el usuario. Al ser una operación lineal, no permite resolver patrones complejos y aunque su gradiente no es nulo, es constante e independiente de la entrada. Entonces, con el paso de la retro propagación los pesos y los sesgos se actualizan, aunque el factor de actualización como tal sea invariable. Así pues, se recurre a ella para ofrecer mera interpretabilidad o tareas simples.

Se suele usar la función **sigmoide** a la salida si se tiene clasificaciones binarias, donde los valores mayores saturan en uno y los más bajos en cero. Debido a esto, es muy útil para representar probabilidades. No es una función simétrica respecto a cero y el signo de sus valores de salida es positivo, aunque siempre queda la opción de reescalar y personalizar la función.

La función **tangente hiperbólica** (conocida como *tanh*) es similar a la sigmoide pero el rango va desde una unidad negativa hasta una unidad positiva, quedando simétrico respecto a cero. Se prefiere ante la activación sigmoide ya que sus gradientes no se restringen a variar en una determinada dirección.

La función **unidad rectificada lineal** (conocida como *ReLU*), se corresponde con una función lineal cuando es positiva y un valor cero constante cuando el valor es negativo. Su mayor ventaja es que no se activan todas las neuronas simultáneamente, las que tienen salida nula están desactivadas. En ocasiones, el gradiente tiene un valor nulo y dada esta circunstancia, no se actualizan los pesos ni el sesgo en la retro propagación.

Adicionalmente, hay otras funciones variantes de la ReLU original, creadas para subsanar algunas desventajas de la función inicial. Por ejemplo, la función ***Leaky ReLU*** en lugar de valer cero constante para entradas negativas, se define como un escalado por valor muy pequeño (en torno a 0.01). Otro caso es ***LU Exponencial*** (ELU), donde las señales recibidas negativas se multiplican por una curva logarítmica para obtener una transición más suave de los valores negativos a los positivos.

En la figura 3-10 se presentan de forma gráfica algunas de las funciones descritas, de izquierda a derecha en el mismo orden en el que han sido expuestas antes.

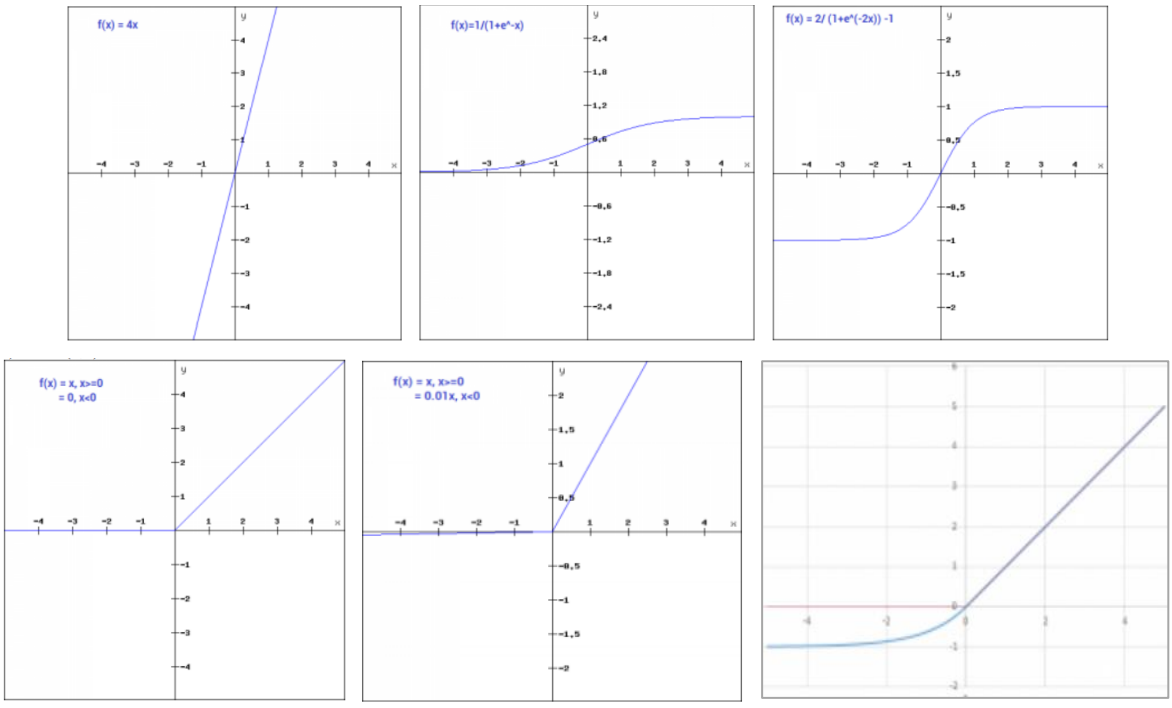

Figura 3-10. Funciones de activación comunes [31].

Por último, para clasificaciones de múltiples clases, la opción óptima trata de combinar varias funciones sigmoides. Para este cálculo, se llama a la función de activación **softmax**, muy frecuente su uso en algoritmos de *Deep Learning* basados en clasificación.

La salida de cada una de las neuronas es una probabilidad, por tanto, queda comprimida entre 0 y 1, de la categoría a la que se estima que pertenece. Proviene de una función de distribución de variables aleatorias discretas que ha sido normalizada, razón por la que al sumar todos los valores posibles de salida el resultado debe ser una unidad. En la figura 3-11 se representa de forma visual su funcionamiento.

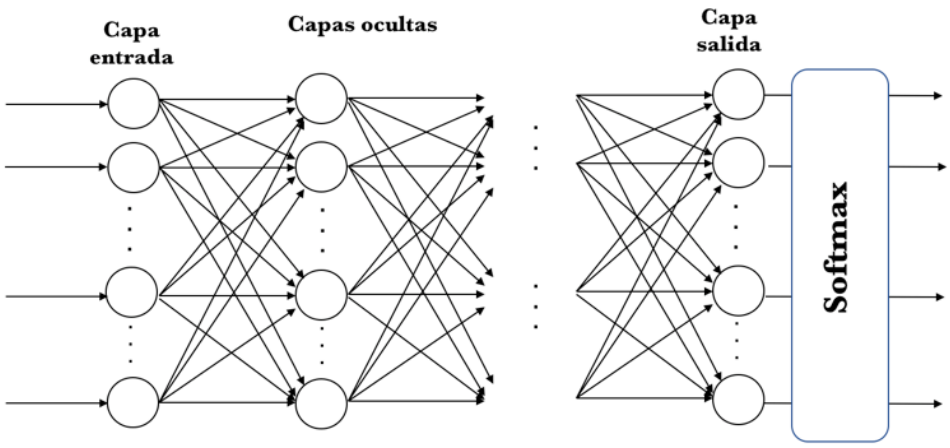

Figura 3-11. Función de activación *softmax*.

Se debe mencionar que, al construir una red de estas propiedades, la capa de salida ha de tener el mismo número de neuronas que categorías. Finalmente, se determina que la predicción correcta es la de mayor porcentaje como valor de salida, el más cercano a 1 [32] [33].

De cara a las operaciones matemáticas, esta capa actúa siguiendo la ecuación 3-2. En ella, *Y* se corresponde con la salida de una de las n capas ocultas anteriores y *K* es el número de categorías del modelo. Se adjunta en la figura 3-12 un ejemplo que ayuda a comprender mejor el proceso, donde la categoría de la entrada sería la que ofrece un porcentaje mayor, de 31.98% [34].

$$p_n = \frac{e^{Y_n}}{\sum_{k=1}^{K}(e^{Y_k})}$$

Ecuación 3-2. Modelo matemático función softmax.

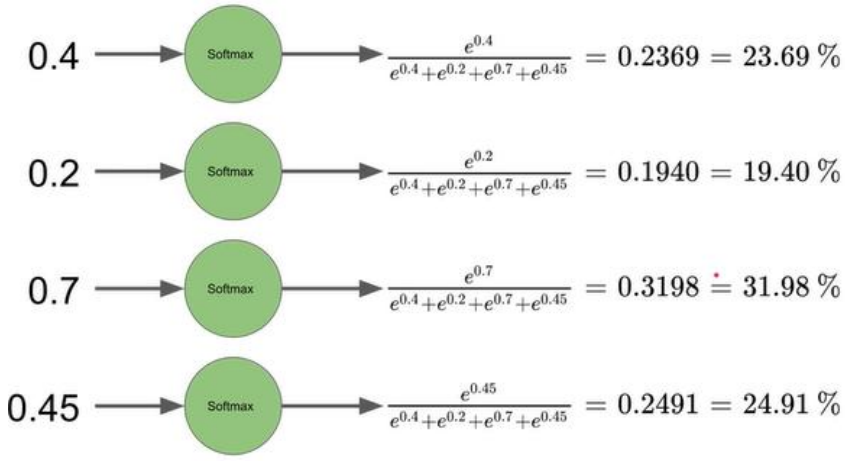

Figura 3-12. Ejemplo de salidas de la función softmax.

Con estas funciones se alcanza la no linealidad y otros beneficios según el momento en el que se empleen, pero ¿Cómo acertar con la función de activación seleccionada? No hay ninguna regla que estandarice el proceso, pero sí que algunas prácticas son recurrentes y por lo general, son consejos que actúan de la forma esperada.

- En primer lugar, para clasificaciones, se debe emplear siempre en la salida una combinación de funciones sigmoide (softmax) o una sola en caso de que sea una clasificación binaria.
- Por otra parte, la función ReLU es muy popular y suele ser apropiada en ocasiones muy versátiles, pero sólo para las capas ocultas. Además, el procesamiento que aplica es rápido.
- Cabe recordar que, si hay neuronas muertas en la red, es mejor sustituir la anterior por la función Leaky ReLu. Esta función esquiva uno de los problemas de la función ReLu original, pues al tener un exceso de carga se pierden valores y algunos nodos "mueren" por inactividad.
- Asimismo, es habitual que las funciones sigmoide y tangente hiperbólica causen problemas relacionados con el gradiente y a veces, no son correctas para las capas ocultas. Si se puede, evitarlas.

La opción más acertada para actuar es usar estas recomendaciones como punto de partida y si los resultados no son satisfactorios, ir cambiando el modo de activación hasta dar con la configuración final.

Aun así, la frontera de estas funciones descritas sigue siendo es una línea recta, pero con los parámetros presentes se pueden variar ajustes como es la orientación. El objetivo consiste en agrupar en una misma capa varias neuronas, cambiando entre ellas, por ejemplo, la inclinación y con una nueva neurona colocada secuencialmente en la siguiente capa, ya se puede obtener una solución que ofrece una frontera circular. De este modo, se codifican conocimientos más complejos, únicamente agregando capas y neuronas en ellas [31].

3.3. Etapas de la Red Neuronal Artificial.

Aunque en el capítulo 4 se estudia en detalle una red neuronal por partes y se analiza el código que la compone, en este apartado se van a explicar algunos conceptos teóricos que hacen entender mejor su arquitectura, así como errores frecuentes en los ajustes y como se identifican para poder evitarlos.

Una ANN está compuesta por dos etapas principales: la fase de creación o **entrenamiento** y la fase de ejecución o **predicción**.

3.3.1 Etapa de entrenamiento.

La primera etapa tiene como objetivo fundamental dejar la red ya preparada para predecir, con los parámetros ajustados para obtener la solución óptima deseada y desempeñar la función requerida con el menor error posible.

El entrenamiento se subdivide en tres tareas: el diseño de la estructura de la red, el entrenamiento de la red y la validación.

El diseño de su estructura engloba la toma de varias decisiones como es el conjuntos de datos empleado, el número de capas y de neuronas que compone cada una de ellas o las funciones de activación usadas. Consiste en describir su topología para lograr una buena eficiencia y como es de esperar, es difícil alcanzar la meta a la primera, ya que no siempre existen reglas establecidas. Lo práctico es entrenar la red con distintas arquitecturas y configuraciones midiendo su error, para escoger finalmente el ensayo más satisfactorio para el usuario.

El entrenamiento no es más que la modificación de los pesos sinápticos a través de la retro propagación para que la red aprenda de manera similar a la cognitiva. La fase de validación se usa en combinación del entrenamiento para evitar caer en el sobre entrenamiento, como se explica en el apartado 3.3.3 posterior [35].

3.3.2 Etapa de predicción

La etapa final es la de predicción, donde se ejecuta el sistema previo pero esta vez totalmente concluida su configuración. En líneas generales, se le solicita a la red que resuelva el mismo problema que al entrenar, pero esta vez con entradas nunca vistas.

Si la red es correcta, debe haber aprendido y no memorizado, por lo que generaliza el método y sabe cómo actuar con datos desconocidos. No obstante, no depende únicamente de la red, puesto que la calidad del *dataset* empleado juega un importante papel en el entrenamiento. Una colección ideal de datos es amplia y variada, pero no debe desviar la atención en el empleo de patrones innecesarios. Los datos deben albergar todos los casos que cubre la red, sin gastar carga computacional y tiempo en aprender conceptos irrelevantes [36].

La figura 3-13 resume la forma de actuar de ambas fases.

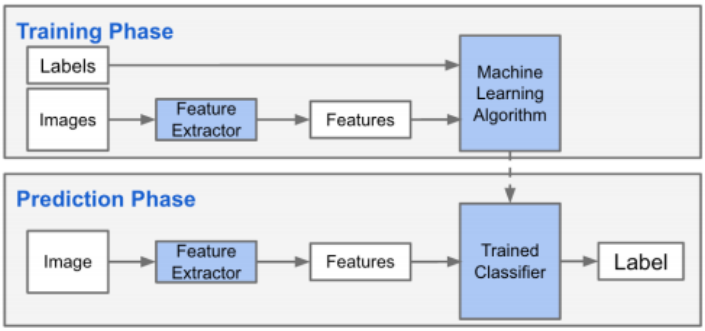

Figura 3-13. Fase de entrenamiento y fase de predicción.

Si no se cumplen algunos de estos criterios, aparecen conflictos con el ajuste de los pesos de interconexión, llamados *underfitting* y *overfitting*.

3.3.3 *Underfitting* y *overfitting*

El ***underfitting*** o **subentrenamiento** sucede cuando el modelo carece de flexibilidad suficiente para ajustarse a la nube de datos, pero se puede adaptar mejor aumentando el número del grado del polinomio. Mientras mayor sea el grado, más flexible es la curva que sigue los datos; si no, el modelo es tan rígido que no se puede ajustar bien a los datos, de forma que aumenta el error.

Figura 3-14. Sobreentrenamiento, subentrenamiento y comportamiento óptimo del modelo [37].

El ***overfitting*** o **sobreentrenamiento** es fácil de identificar cuando la precisión de entrenamiento roza la perfección, mientras que la de validación queda notablemente por debajo. El modelo es muy bueno en lo que ha visto con anterioridad, la curva cada vez es más flexible, pero no crea un patrón real. Lo que ocurre es que

modela el ruido de nuestros datos al entrenar en vez de discriminarlo, entonces las predicciones para los nuevos valores son peores, ya que la red es incapaz realizar una abstracción adecuada. Se puede evitar este error con el aumento de imágenes u otros recursos que mejoren los datos o cambiando la estructura, haciendo énfasis en el número de iteraciones y la complejidad [38].

La figura 3-14 expone ambas situaciones incorrectas y un caso donde la predicción es óptima.

Para identificar pronto el *overfitting* y no confundirlo con una buena generalización, hay que priorizar que con nuevos datos también estará a la altura prevista. La solución es dividir nuestros datos en dos grupos: de entrenamiento y de validación, para evaluar mejor el rendimiento. Durante el entrenamiento, el error de validación ha de ser mayor que el error resultante al entrenar. Se confirma que no existe *overfitting* si el error de datos de validación va disminuyendo. Por el contrario, al crear este conjunto de datos de validación se tienen menos datos para el propio entrenamiento y hay que tener precaución para no sobreentrenarlos [39]. En la imagen 3-15 se comparan las dos situaciones presentadas y se comprueba cómo detectarlas en base a los errores.

Un concepto clave relativo a las bases de datos es que la naturaleza de ambos grupos (entrenamiento y validación) de datos han de ser iguales. Es una práctica aconsejable desordenarlos de forma aleatoria, para validar que hayan sido distribuidos desinteresadamente en ambos grupos y hacer que no resulten datos casi replicados quen no filtren el *overfitting* de entrenamiento en la validación.

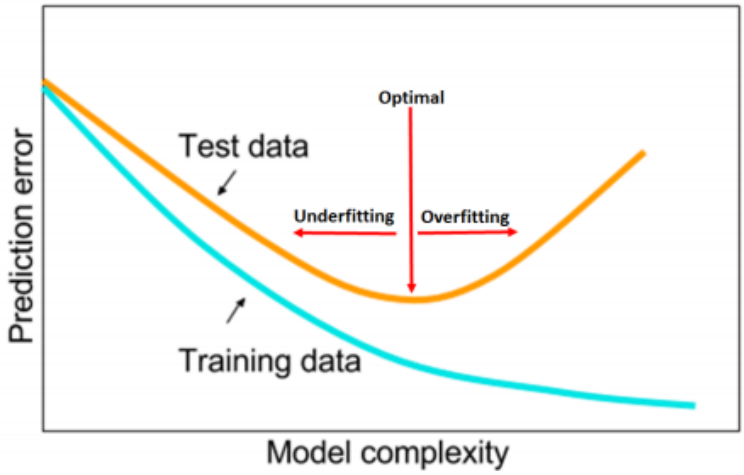

Figura 3-15. Situación de sobreentrenamiento y subentrenamiento respecto al error [40].

3.4. Las Redes Neuronales Convolucionales.

Las redes neuronales convolucionales (CNNs) conforman un subconjunto de las redes neuronales artificiales que nacen con la finalidad principal de descifrar patrones complejos en bases de datos de grandes dimensiones, pues destacan en aplicaciones como el procesado de imágenes o reconocimiento de voz. De nuevo, se trata de simular esta acción de acuerdo con el reconocimiento humano pero con un grado de complejidad inabordable por las ANNs clásicas. A continuación, se enfoca la explicación desde el punto de mira del proyecto: la clasificación de imágenes.

El pensamiento cognitivo provoca que las personas detecten, hasta inconscientemente, patrones en las imágenes. En la red esta detección se procesa en cascada y permite así clasificar elementos más abstractos a raíz de identificar características de otros básicos.

La red neuronal convolucional aprovecha las propiedades de la estructura espacial que presenta una imagen. La MLP trata cada píxel como una variable independiente, así, se procesa una imagen con un determinado número de píxeles, los cuales se pasan a un vector plano unidimensional y se separan uno a uno, como sucede en la figura 3-16, que cuenta con 9x12 píxeles de entrada [41]. En cambio, la CNN aprovecha la posición de éstos,

puesto que el valor que contenga cada píxel se encuentra ligado al de sus vecinos y es un factor que beneficia la detección de elementos [42].

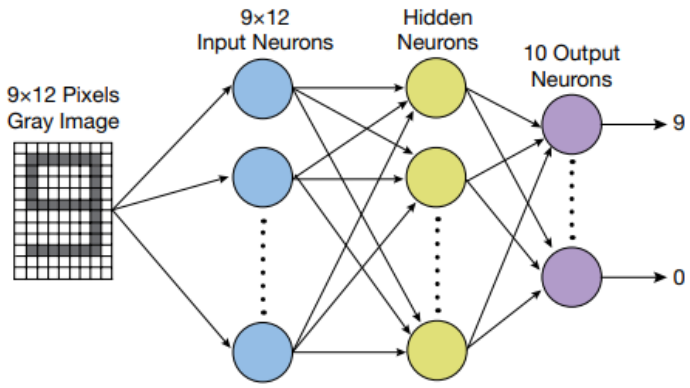

Figura 3-16. MLP con 9x12 píxeles independientes de entrada [42].

El nombre de estas redes lo determina la operación de convolución que se le aplica a la imagen en cada capa. Mediante ella, cada nuevo píxel se calcula con un filtro que actúa sobre la imagen original, con operaciones ariméticas en los píxeles adyacentes. El filtro va progresivamente recorriendo la imagen entrante hasta completar su paso por ella [43].

En la figura 3-17 se presenta una construcción básica de una CNN. Además de aplicar convoluciones con filtros, se completa con técnicas de *pooling* y *dropout*.

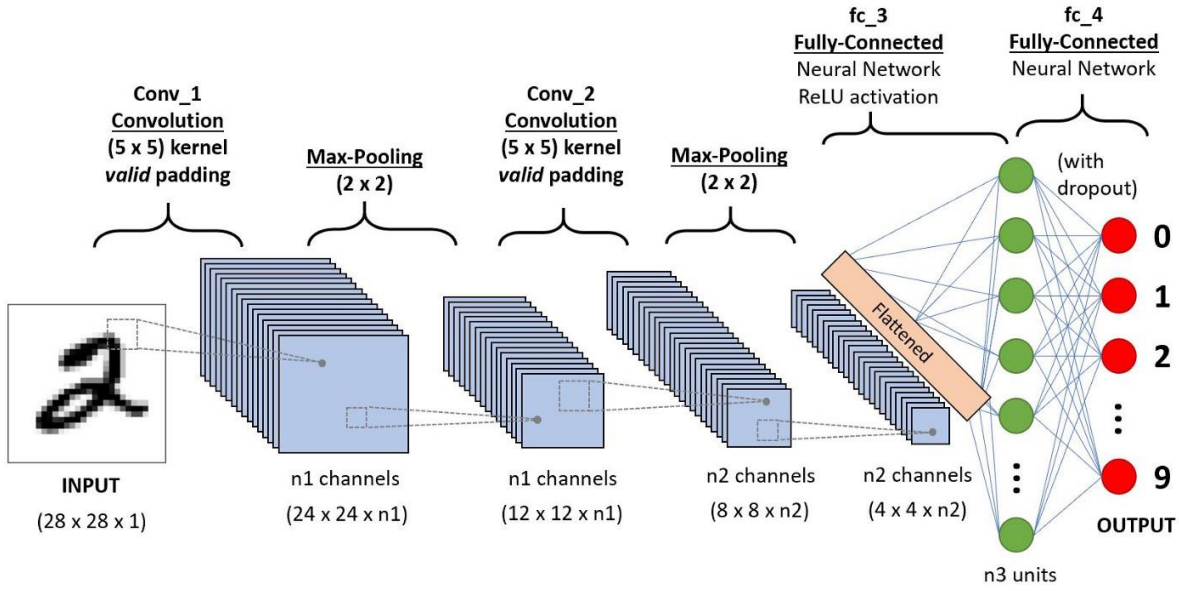

Figura 3-17. Estructura CNN [44].

3.4.1. Filtros.

Los filtros o *kelner* sólo son matrices formadas por números multiplicadores, pero ¿qué valores se le debe dar al filtro? Esta pregunta carece de respuesta inmediata ya que se ve sujeta a la situación y a las características que se deseen obtener. La CNN es la encargada de ir estableciendo los valores para poder hacer la tarea en cuestión de forma adecuada, así la red aprende estas configuraciones para detectar patrones [45].

En la figura 3-18 se desglosa un cálculo estándar de un nuevo pixel a raíz de un filtro, que se irá desplazando con una ventana deslizante hasta obtener la salida.

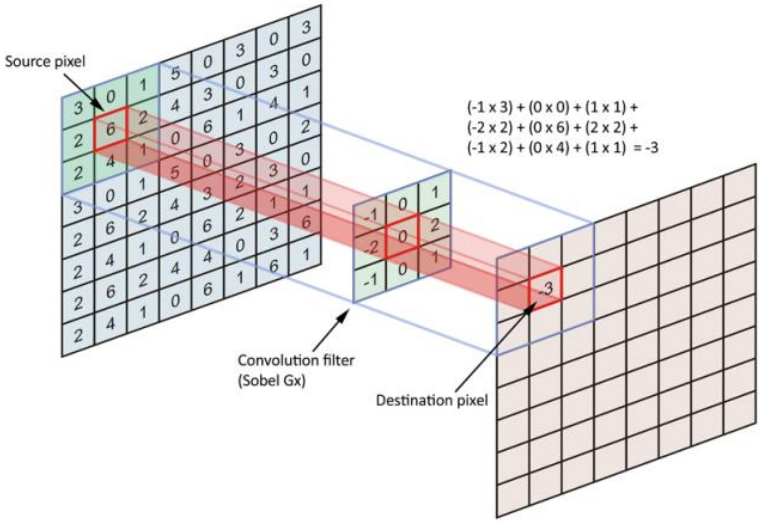

Figura 3-18. Ejemplo de cálculo matemático de un filtro o *kelner*.

Cada imagen obtenida tras convolucionar con un filtro a su salida se denomina mapa de características, debido a que en él se activan los elementos encontrados de acuerdo con el filtro aplicado, que se centra en buscar una cualidad concreta. Estos mapas, a su vez, pasan a ser las entradas de otro filtro, actuando secuencialmente. Las capas convolucionales pueden estar hechas de multiples filtros que ofrecen, cada uno, un mapa diferente para una misma imagen de entrada. Las convoluciones van adquiriendo más potencia a través de la profundidad y dificultad de la arquitectura del algoritmo de *Deep Learning*.

Ejemplificando, a una región de 9 píxeles se le puede aplicar un filtro 3x3 con el que convoluciona, sacando un único pixel de información basado en el original de esa posición y los otros 8 vecinos (véase figura 3-18 anterior). Como resultado de hacer este cálculo varias veces, la resolución de los mapas decrece notablemente y la cantidad de éstos aumenta, provocando una anchura mayor.

La red inicializa los filtros de forma aleatoria y después aprende de forma autónoma su configuración, pues selecciona la máxima activación del mapa de características a su salida. Se puede entender como una función de optimización.

Por último, como ya se ha mencionado, en las capas más bajas la red aprende patrones genéricos y sencillos, como puede ser un borde o una esquina. A medida que el número de capas se incrementa, los patrones son mas avanzados y se pueden identificar objetos más abstractos. Un ejemplo básico es el de la figura 3-19, donde para detectar que el elemento es un gato, primero se identifican curvas y bordes, después otros patrones como ojos o nariz y, por ultimo, se compone la cara del gato. Se conoce como la visualización de características a la técnica que analiza qué peculiaridad es la que más activa a un mapa de características dentro de la red [46].

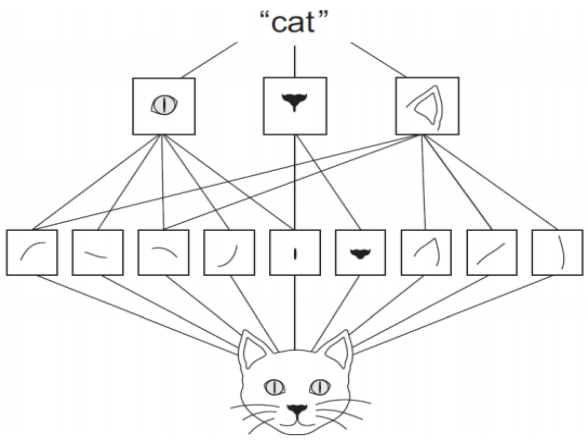

Figura 3-19. Jerarquía de detección del modelo.

En las capas ocultas de la red, además de convolucionar, se emplea un recurso denominado *pooling*. Esta técnica es una agrupación que reduce, aun más, las dimensiones de la imagen de forma que resulte menos pesada para ser cargada sin perder información.

3.4.2. Pooling

El denominado *pooling* es una técnica que consiste en agrupar los píxeles de una imagen y filtrarlos en subconjuntos. El *pooling* hace la imagen menos pesada y, además, ayuda a no sobreajustar el modelo. No es más que disminuir el número de muestras de una imagen, pero conservando aquellas características necesarias para la clasificación mediante invariantes de traslación y rotación. Aumenta la robustez de la invariación espacial y reduce en gran medida el coste del cálculo, pues el procesamiento es más rápido. El efecto que provoca en una imagen se aprecia en la figura 3-20 [47].

Figura 3-20. Ejemplo de una imagen al aplicar *pooling*.

Existen varios mecanismos de agrupación. Entre ellos, se encuentra el *MaxPooling*, donde se pasa un filtro de cierta altura y longitud sobre el modelo, identifica el número mayor de un cierto parche y ese es el único valor que continúa, descartando el resto. Otro método es el *AveragePooling*, que funciona igual, pero hallando el valor promedio. Las operaciones que siguen son las mostradas en la figura 3-21 [48] [49].

La capa de *pooling* contiene tantos filtros de agrupación como filtros convolucionales, pues se aplica a cada uno de ellos por separado. Por ello sólo disminuye la resolución, pero sin aumentar el ancho de los mapas.

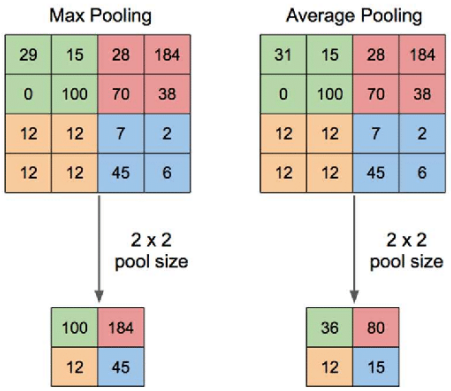

Figura 3-21. Técnicas de *MaxPooling* y *AveragePooling* [50].

Actualmente, gran variedad de técnicas para la regulación de las redes neuronales se van introduciendo en la comunidad del *Deep Learning*. Entre ellas, se sitúa otro recurso llamado *dropout* que será usado en el modelo.

3.4.3. Dropout

Si la red es demasiado grande en comparación con el tamaño del conjunto de ejemplos que se tiene para entrenarla, puede haber problemas de sobre ajuste. El *dropout* o pérdida de peso se encarga de desactivar un porcentaje de neuronas indicado para evitar errores de entrenamiento [51].

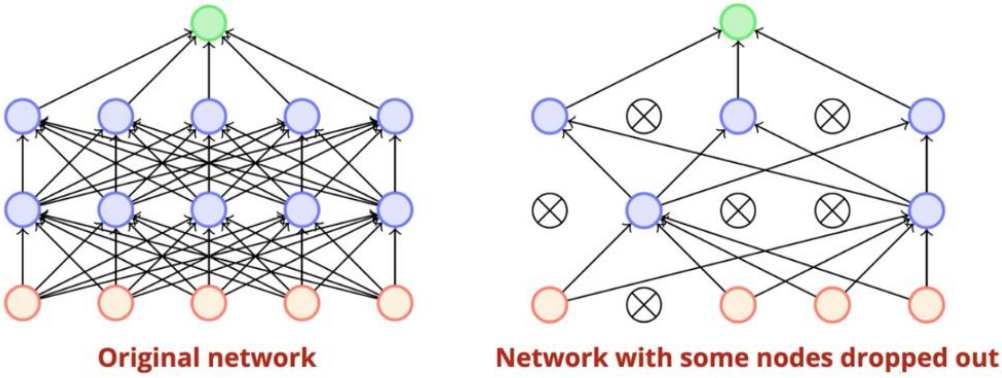

Figura 3-22. Funcionamiento de la técnica de *dropout*.

En el método sugerido, una colección de capas interconectadas utiliza una máscara de *dropout*, donde los pesos entre nodos se eliminan aleatoriamente en el proceso de entrenamiento, debido a que se ignoran algunas unidades de la red en cada ciclo regidas por un comportamiento probabilístico. Esta actuación ofrece mayor precisión de rendimiento puesto que disminuye la posibilidad de perder capacidad para generalizar y evita adaptaciones excesivamente perfectas, pero, por otro lado, ralentiza el entrenamiento porque escoge diferentes neuronas para apagar a cada paso [52]. Actúa de acuerdo con lo mostrado en la figura 3-22.

Finalmente, esta técnica es una alternativa muy buena para simular diferentes arquitecturas de entrenamiento en la red, puesto que si se necesitase crear diferentes modelos desde cero sería una práctica insostenible [53].

Llegado a un punto, existen suficientes mapas independientes como para que se utilicen de entrada de una red neuronal básica, por lo que se aplanan en un vector de una dimensión y se introducen en la ANN. Es una opción mucho más efectiva que tener píxeles iniciales como entrada, ya que no sacan partido del espacio de la imagen. En caso de que se introdujese la imagen directamente en una ANN, el número de conexiones entre capas sería demasiado elevado y puede dar lugar a problemas de detección en la red.

La arquitectura típica de red neuronal convolucional para procesar los datos y dejarlos preparados a su salida para que sirvan de entrada de la red neuronal básica de clasificación es la visible en la figura 3-23.

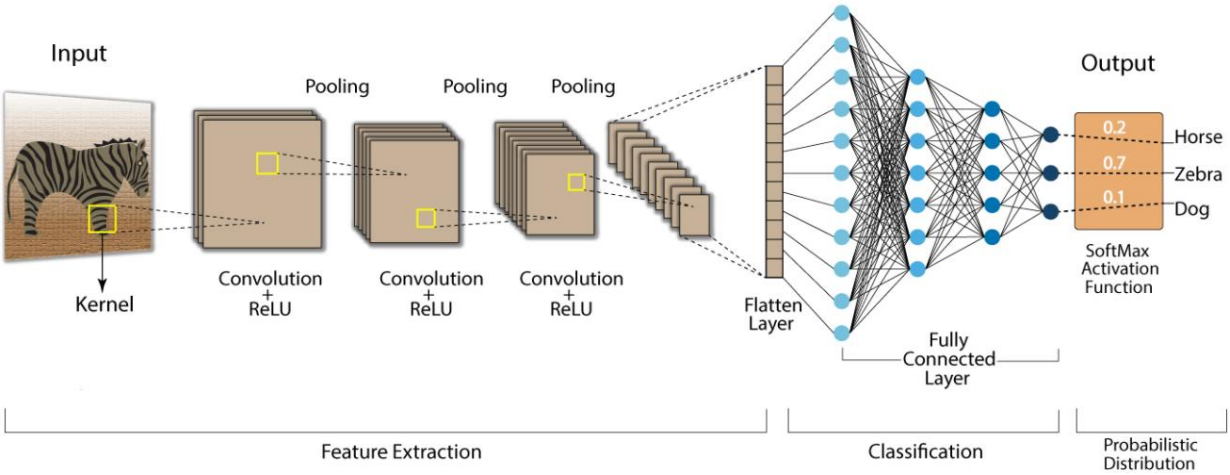

Figura 3-23. Red neuronal convolucional para la clasificación de imágenes [54].

4 IMPLEMENTACIÓN DEL PROYECTO

Después de estudiar en profundidad la teoría y algunos algoritmos que alberga este inmenso campo del *Deep Learning*, se presenta una propuesta de proyecto para mostrar desde un puesto de vista práctico los conceptos expuestos.

En primer lugar, se va a realizar una descripción de las herramientas, así como del entorno escogido para la familiarización con las redes neuronales. Se expone el conjunto de datos empleado y sus características, el código propuesto para simular el proyecto y se introducen algunas definiciones para comprender los parámetros internos de esta red.

4.1. Entorno

En este apartado, se muestra el entorno software empleado en el desarrollo del proyecto, acompañado de breves explicaciones acerca de su funcionamiento y sus ventajas.

En primer lugar, se ha creado un espacio virtual para poder trabajar en Anaconda. Anaconda es un software gratuito multiplataforma que proporciona gran variedad de herramientas, diseñado para investigar. Se ha instalado Anaconda porque permite acceso a distintos entornos que le permiten programar en conocidos lenguajes como Python o R. Estos entornos son conocidos como entornos de desarrollo integrados (IDEs), son aplicaciones que facilitan el desarrollo de código [55].

Figura 4-1. Anaconda.

El entorno donde se ha trabajado con esta programación científica ha sido Spyder. Spyder es un IDE que incluye, entre otras herramientas, un editor para codificar *scripts*. También permite inspeccionar y depurar código, perfecto para analizarlo línea a línea o identificar errores rápidamente.

Figura 4-2. Spyder.

A continuación, ha sido el lenguaje de programación de nivel medio Python 3.6 el escogido para implementar el código. Cabe mencionar que esta versión de Python no es la más nueva, puesto que está actualizado hasta la actual 3.9.6, pero se compagina muy bien con otras librerías utilizadas y no es necesario emplear un lenguaje con capacidades mayores para el proyecto en cuestión.

La elección sobre usar Python se basa en distintas razones, entre otras, es un lenguaje interpretado y orientado a objetos, es fácil de aprender y suficientemente versátil como para abordar una gran variedad de tareas. Además, es un código abierto y su popularidad ha crecido exponencialmente desde sus inicios en el año 1991. Esto es debido a las facilidades que les ofrece a los usuarios, como ser compatible con distintos sistemas (Linux, Windows, Mac…) o no necesitar de equipos de altas capacidades para su codificación [56].

Figura 4-3. Python.

No sólo tiene éxito en el ámbito de la Inteligencia Artificial, donde su uso se ha visto disparado, si no que alberga otras ciencias como el desarrollo de aplicaciones y la investigación científica.

Ambos se han compaginado con un Sistema Operativo Windows 10.

De acuerdo con referencias fiables como es PYPL o IEEE, Python se sitúa entre los lenguajes más populares de los últimos años. Además, según la figura 4-4, Python se encuentra es segunda posición respecto a su popularidad sólo por debajo de Java. No obstante, su crecimiento es mucho mayor, del 10% en los últimos años, restándole protagonismo a otras tecnologías como es PHP, que ha decrecido un 5%. A pesar de ser una gráfica extraída de un informe de PYPL de 2018, actualmente Python ha adelantado a Java en popularidad, situándose en primera posición frente a otros lenguajes como C++ o R [57].

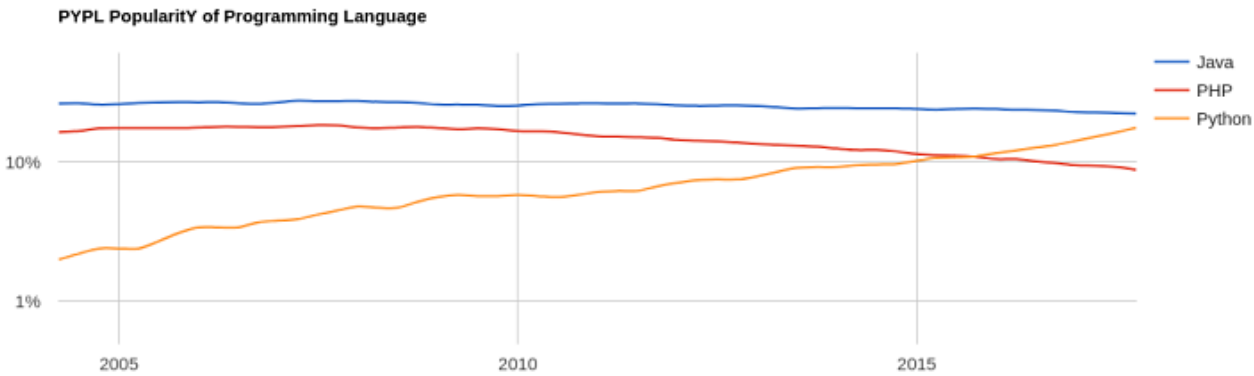

Figura 4-4. Popularidad de los lenguajes de programación (2018).

Después, para la implementación de las redes neuronales se ha hecho uso de los servicios que ofertan las distintas librerías y paquetes para optimizar los métodos científicos y de ingeniería, como es *Deep Learning*. En este trabajo, se exprimen los recursos de las siguientes APIs para el análisis de datos y la computación:

- El módulo **os** ofrece acceso a funciones que permiten interactuar con el Sistema Operativo. Es muy útil para gestionar directorios, por ejemplo, para abrir una carpeta, manipular una ruta o mostrar una lista de su contenido. Este aspecto es importante en el proyecto puesto que se debe acceder a diferentes carpetas de forma interactiva, ya que en ellas se encuentran los datos etiquetados de entrenamiento y predicción [58].
- El módulo **sys** concede acceso a variables y provee de funcionalidades, todas ellas se encuentran fuertemente unidas al intérprete, que las usa o las mantiene [59].
- El módulo **numpy** se utiliza para la creación de vectores o de matrices de elevadas dimensiones y dota al sistema de funciones de alto nivel para realizar cálculos algebraicos y operaciones matemáticas y estadísticas. Soporta grandes cantidades de datos numéricos [60].

Para desplegar el núcleo del proyecto, las CNNs, se han usado dos *frameworks* específicos: Keras y Tensor Flow, cuyo éxito se debe a la sencillez y velocidad que aportan a la creación de los modelos. Son plataformas muy intuitivas, con síntaxis e interfaz simples.

La biblioteca **TensorFlow** facilita un software de código abierto gratuito desde noviembre del 2015 y sobre ella, puede correr otro *framework* de alto nivel llamado **Keras**, encargado de realizar tarea de aprendizaje en Python. Ambas combinan a la perfección y se usan para desarrollar redes neuronales, tanto la construcción como el entrenamiento. Permiten identificar patrones análogos a los cognitivos [61].

No obstante, estas no son las únicas opciones de librerías especializadas en *Deep Learning*, aunque sí las que reinan entre sus competidoras, las más conocidas Theano y Caffe.

En último lugar, se ha instalado la librería Scikit-learn, la biblioteca más robusta para analizar modelos de ML con Python. Incluye una gran selección de herramientas útiles para clasificaciones, regresión, cambio de dimensiones y otras tareas. Para este proyecto, se importa la matriz de validación, ideal para estudiar los resultados obtenidos [62].

Figura 4-5. TensorFlow y Keras.

4.2. Base de datos.

El conjunto de datos empleado para entrenar la red neuronal fue, en principio, tomado de Kaggle [63], aunque después se ha visto modificado. La razón de este cambio es que se quiso insertar una tercera clase que no estaba incluida antes: mascarillas colocadas de forma incorrecta. Se buscaron imágenes en Google Imágenes para completarla de forma manual. Por otro lado, había una inmensa cantidad de imágenes en esta BBDD y como se va a comprobar a continuación, no es necesario para lograr resultados satisfactorios y ralentizaban la ejecución de las tareas debido a un tiempo de computación enorme, por lo que se ha descartado un porcentaje de estas.

En la tabla 4-1 queda resumida la estructura que siguen los datos. Se han separado en tres directorios diferentes, cuyos nombres se corresponden con los de las categorías de las imágenes para hacer así el etiquetado.

Etiqueta de las imágenes	Entrenamiento	Validación	Prueba
Con mascarilla	400	90	10
Sin mascarilla	400	100	10
Mascarilla Incorrecta	160	46	10

Tabla 4–1. Volumen de la base de datos.

Finalmente, la base de datos está compuesta de 1226 imágenes: 960 para entrenamiento, 236 para validación y 30 para probar. Aunque las cantidades no sean exactas, la separación gira en torno a un 80% de las imágenes adquiridas para el entrenamiento y el 20% restante para la validación. Para predecir (fase de prueba) se toman varias imágenes nuevas para la red [64].

Es oportuno aclarar ciertos aspectos particulares de este trabajo para evitar confusiones. En primer lugar, se considera que una mascarilla está puesta de forma incorrecta si no cubre nariz y boca, quedando colocada en la barbilla, y bajo este criterio se han seleccionado las imágenes de su respectiva clase. Por otro lado, no se especifica en ningún momento qué tipo de mascarilla se usa ni otros aspectos como es su color, el sistema debe saber clasificar una mascarilla genérica, sean cuales sean sus propiedades.

En las siguientes figuras se muestran ejemplos de la carpeta de entrenamiento de los tipos a diferenciar.

Figura 4-6. Muestras de entrenamiento: categoría ‘con mascarilla’.

Figura 4-7. Muestras de entrenamiento: categoría ‘mascarilla incorrecta’.

Figura 4-8. Muestras de entrenamiento: categoría ‘sin mascarilla’.

4.3. Implementación.

Después de explicar toda la teoría que esconde el inmenso mundo del *Deep Learning* y presentar el conjunto de datos que se va a usar para entrenar la red, así como el entorno sobre el que se ha desarrollado el trabajo, es momento de tratar los ficheros que se han codificado en Python para implementar la red neuronal convolucional. Antes de nada, se van a introducir los hiperparámetros que se han decidido emplear, estos son las variables externas de la red que se ajustan independientemente de los valores de entrada. Acto seguido, se comentará la estructura que define la CNN y la division en diferentes ficheros realizada, siendo uno para entrenar y otro para predecir la clasificación de imágenes hasta entonces desconocidas para el sistema. En el próximo capítulo, se estudian y analizan los resultados conseguidos, que lógicamente son totalmente dependientes de esta implementación que da forma a la red.

4.3.1. Estructura de directorios.

Para comenzar, es importante tener un esquema de cómo se ha organizado la información y así comprender como procesa el sistema los datos recopilados.

Inicialmente, se tienen tres carpetas, una llamada *entrenamiento*, otra *validación* y una última de *prueba*. Cada una de ellas contiene otras tres subcarpetas que representan las posibles categorías de la clasificación: *conmascarilla*, *mascarillaincorrecta* y *sinmascarilla*, excepto la carpeta de prueba, que no tiene ningún tipo de etiqueta, ya que este es justo el trabajo del que se encarga la red.

4.3.2. Hiperparámetros

Un rasgo relevante de las CNNs son unas conocidas variables externas, denominadas hiperparámetros. Las redes son parametrizadas por estos valores que el usuario es el único responsable de ajustar y son independientes de los datos, es decir, su valor no se puede estimar a partir de ellos. Su objetivo es maximizar la función de aprendizaje correspondiente, en esta ocasión, una clasificación de imágenes con el menor error posible. Los hiperparámetros configuran la forma de actuar del algoritmo antes de que empice a funcionar y un cambio en ellos puede repercutir notablemente a los resultados y al rendimiento de la red [65].

No hay reglas a seguir para establecerlos, por ello es típico iterar de forma manual hasta encontrar la solución esperada. No obstante, también existen cuadrículas predefinidas que guían el modelo y reglas empíricas para regir su comportamiento. Aun así, estas técnicas no son del todo eficientes, sobre todo cuando el número de parámetros es elevado y consume mucho tiempo. Por este motivo, está tomando popularidad la búsqueda automatizada de hiperparámetros o simplemente, tomar valores estándar que no optimizan el aprendizaje pero sí el tiempo y suelen defender bien el comportamiento del sistema ante distintos escenarios.

Es importante dominar el nivel de complejidad de la red. Un claro caso de mala configuración se da con el

sobreajuste, al determiner un modelo demasiado complejo que memorice en vez de aprender, y con el subajuste que causa un modelo de bajo nivel que no pueda capturer los datos. Por lo tanto, los hiperparámetros pueden inhibir gravemente la calidad del algoritmo [66]. Se va a hacer hincapié en tres de ellos: *learning rate*, *epoch* y *batch*.

- ***Learning rate*:** Este hiperparámetro ya ha sido presentado en un apartado previo (véase el apartado 3.2.4 de descenso del gradiente). En resumen, la tasa de aprendizaje marca el tamaño de paso en cada nueva iteración durante la búsqueda del mínimo de la función de error [67].
- ***Epoch*:** El número de épocas es un hiperparámetro entero de descenso del gradiente que controla el número de vueltas completas que se le da al conjunto de datos de entrenamiento, ó sea, el número de veces que todos los datos de entrenamiento pasan por la red. Cada *epoch* se compone de uno o más *batches*. A pesar de que su valor depende del modelo concreto, es una buena idea aumentar el número de épocas hasta que el error que pertenece a los datos de validación comience a aumentar.
- ***Batch size*:** Esta otra variable entera también relativa al descenso del gradiente, controla el número de muestras de entrenamiento que trabajan antes de actualizar los parámetros internos del modelo. Es decir, los datos de entrenamiento que pasar por la red se pueden divider en lotes menores. Esta partición sirve para marcar la próxima actualización del gradiente, puesto que ocurre al completar el tamaño del *batch* indicado. Cuando se termina de recorrer las muestras pertenecientes al batch y se tiene una predicción, esta se compara con la salida estimada y se calcula la función de pérdida [68].

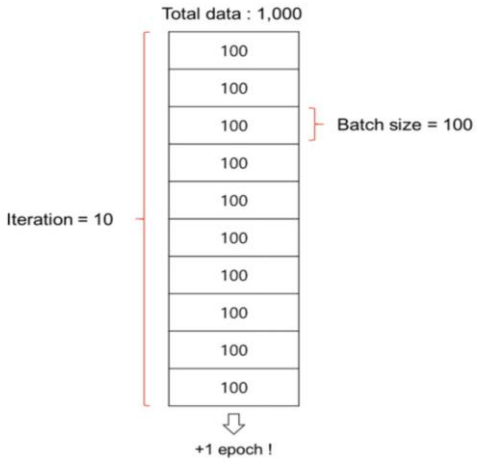

Figura 4-9. Una época dividida en 'lotes'.

De nuevo, existen métodos muy novedosos para cambiar según las necesidades del momento el tamaño de estos lotes, conocido como *batch* adaptativo. Para un tamaño de batch establecido menor, se converge en un resultado satisfactorio con menos épocas, mientras que para un tamaño de batch mayor, hacen falta más épocas pero no se sobre carga computacionalmente el equipo. No obstante, en este trabajo no se llega a tal nivel de adaptatividad por que es un trabajo con fin introductorio a *Deep Learning* y no alcanza un nivel experto [69].

4.3.3. Preprocesado de imágenes: Data Augmentation.

Los problemas de sobreajuste de las redes se pueden solucionar gracias al *Big Data*, pero el problema es que esta gran cantidad de datos suele ser pesada y aparecen conflictos computacionales. Además, en multitud de ocasiones no es fácil obtener tanta diversidad de datos puesto que son limitados.

La técnica de Data Augmentation se emplea en este proyecto para obtener, a partir de un conjunto de imágenes originales, multitud de imágenes como consecuencia de usar efectos sencillos que varían la rotación, orientación,

acercamiento o direccionalidad entre otras alteraciones [70]. Además de solucionar los problemas presentados, ayuda a encontrar patrones, pues no se trabaja solo el mismo tipo de fotografía "bien tomada".

Para este trabajo se realizan modificaciones geométricas tradicionales que son las primeras que probaron la efectividad de esta técnica, las más conocidas son el desplazar, acercar o alejar, girar, voltear, distorsionar o sombrear de algún tono la imagen. Un caso estándar de Data Augmentation es el de la figura 4-10, donde de una imagen inicial se obtienen otras 15 imágenes que la red detecta como nuevas al haber variado sus características, aunque todas pertenecen a la misma categoría. [71].

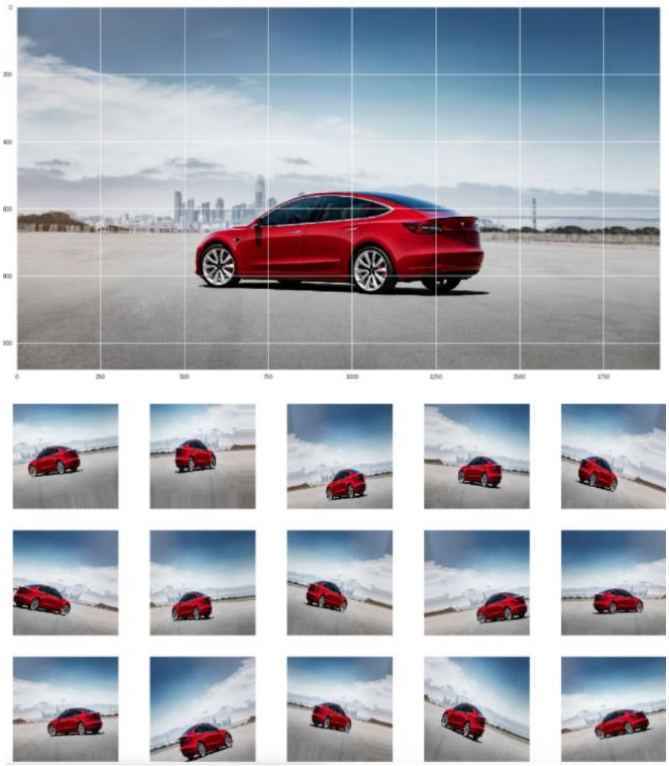

Figura 4-10. Ejemplo de Data Augmentation.

4.3.4. Fichero de entrenamiento: *entrenamiento.py*.

Llegado a este punto, se procede a predecir la clase de datos nuevos. De forma sintetizada, los pasos a seguir para lograr un buen modelo que ofrezca una buena clasificación para el problema son los siguientes:

- Importación de las librerías y módulos que facilitan el desarrollo del trabajo.
- Preprocesamiento de las imágenes del conjunto de datos recopilado, tanto de entrenamiento como de validación.
- Diseño del modelo de capas secuencial: de la parte convolucional y de la red clásica para clasificar respectivamente.
- Determinar la configuración del aprendizaje: se establece la función de pérdidas, el optimizador y la métrica.
- Ejecución del entrenamiento y la validación.

Aunque no sea un paso obligatorio, siempre cabe mencionar que, si el resultado obtenido no es el deseado, el origen de este desacuerdo suele residir en la construcción de la red neuronal o bien, en la base de datos. Por

tanto, es vital dominar ambos aspectos y corregir aquello que no permite tener la precisión requerida, pues sólo probando se descubre qué es lo mejor para la red.

La lista de librerías que se usan en el fichero de entrenamiento son las importadas a continuación:

```
import os
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D
from tensorflow.python.keras import backend as k
```

Código 4-1. Librerías importadas en el fichero de entrenamiento.

A continuación, se establecen las rutas de los datos de entrenamiento y de validación. Hay dos opciones para separar estos datos: hacer un reparto manual con un porcentaje en torno al 80-20% respectivamente o de otra forma, crear un código que de forma aleatoria divida el conjunto de datos [72].

```
datos_entrenamiento = './datos/entrenamiento'
datos_validacion = './datos/validacion'
```

Código 4-2. Rutas de los datos de entrenamiento y validación.

En esta ocasión se ha optado por la primera opción, pues así se ha descargado de Kaggle [63] y, además, la cantidad de imágenes en cada categoría para cada división no es exactamente la misma, por lo que se ha preferido no automatizar el reparto con una macro. No obstante, sí es una buena opción cuando el *dataset* es demasiado extenso y se tienen todas las imágenes mezcladas en un único grupo.

Después, se han establecido los valores de los hiperparámetros y de otros datos [73].

```
#Establecer hiperparámetros
epochs = 75
altura = 64
longitud = 64
batch_size = 32
pasos = (518/batch_size)
pasos_validacion = (38/batch_size)
```

Código 4-3. Datos del modelo final.

Se van a tratar las modificaciones de Data Augmentation a partir de los *Generators* de Keras. Los generadores sirven para preprocesar las imágenes a utilizar en la red. La creación del set de imágenes que se van a emplear en la red se hace con la función *ImageDataGenerator* importada, tanto en entrenamiento como en validación.

Como argumento de esta función se especifican los cambios que se desean hacer a las imágenes de entrada para una mayor robustez del modelo. A los datos de entrenamiento se les han realizado las siguientes modificaciones:

- *Rescale*: se utiliza para reescalar los valores del brillo de cada color de los píxeles de la imagen, que acostumbrados a abarcar un rango entre 0 y 255 y ahora van desde 0 hasta 1.
- *Shear_range*: se utiliza para variar algunas zonas de la imagen gracias a un mecanismo que varía su orientación. Así, el sistema entrena con imágenes con distintas inclinaciones y no aprende a trabajar sólo con imágenes rectas.

- *Zoom_range*: se utiliza para aplicar un zoom y que la red sepa como trabajar con imágenes recortadas.
- *Horizontal_flip*: se utiliza para obtener imágenes con distintas direcciones, ya que las invierte.

Para el conjunto de imágenes de validación se ha decidido no modificar su naturaleza, ya que simulan como actuaría la red ante imágenes desconocidas y estas no sufren de alteraciones previas. Se le entregan las imágenes tal y como son excepto un escalado para normalizar el valor de los píxeles por simplificar.

Las variables *entrenamiento_datagen* y *validacion_datagen* aplican estas transformaciones indicadas a las imágenes.

```
#Preprocesamiento de las imágenes para poder entrenar la red
entrenamiento_datagen = ImageDataGenerator(
    rescale = (1./255),
    shear_range = 0.3,
    zoom_range = 0.3,
    horizontal_flip = True)

validacion_datagen = ImageDataGenerator(
    rescale=1./255)
```

Código 4-4. Preprocesado de imágenes con *ImageDataGenerator* del modelo final.

A continuación, ya se generan las imágenes para entrenar a la red. Mediante el uso de la función *flow_from_directory* del generador creado en el paso anterior, se obtiene el set de imágenes de entrenamiento y validación indicando su directorio. Lo que hace es actuar como un iterador, pues recorre los elementos de un objeto, en este caso, de la carpeta entregada como argumento. Lee los archivos de las imágenes, decodifica el contenido de estas en cuadrículas de píxeles y ejecuta los cambios indicados [74].

Como ventajas de este modo de operación, no se guarda el conjunto completo de imágenes en memoria, puesto que puede dar problemas de almacenamiento al emplear una base de datos grande. Hace una partición del *dataset*, de manera que sólo se almacena en memoria la parte que se esté tratando y es la que se ofrece como entrada a la red neuronal. Para ello, se le pasa el tamaño de 'lote' que se desee como argumento (*batch_size*), también se establece el modo de clasificar cada imagen según las categorías (*categorical*) y se igualan las dimensiones para todas las imágenes (*altura* y *longitud*).

```
#Generación de las imágenes para entrenar a la red
entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    datos_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

validacion_generador = validacion_datagen.flow_from_directory(
    datos_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')
```

Código 4-5. Generación de imágenes de entrenamiento y validación.

En el siguiente paso es completar creación de la red neuronal convolucional y el diseño del modelo de capas de acuerdo con los conceptos explicados en el capítulo 3.4.

Lo primero ha sido llamar a la orden *sequential* para definir un modelo que consista en apilar las capas, para desde ahí comenzar a añadirlas con la función *add*.

Como se puede comprobar en el código 4-8, la jerarquía del modelo sigue los fundamentos explicados en la teoría. Primero, se tiene una capa convolucional con un total de 32 filtros, a cada imagen de entrada se le aplican los 32 filtros de tamaño 3x3, obteniendo 32 mapas de características a la salida. Se añade una función de activación ReLU, la más adecuada según las recomendaciones del apartado 3.2.6. Como es la primera capa de entrada, también se indican las dimensiones de las imágenes de entrada en el formato: (altura, anchura, dimensión), al haber normalizado la altura y la anchura de todas las imágenes, estos valores son fijos en todo el transcurso del diseño.

Al argumento *padding* se le ha dado el valor *same*, el cual indica que se debe completar el mapa de características de salida con tantas columnas y filas de ceros como haga falta para rellenarlo y garantizar las mismas dimensiones que las de entrada, ya que los píxeles agregados no afectan al realizar las operaciones de multiplicación características del filtro debido a que su valor es nulo. Así, se permite que cada verdadero píxel de la figura pueda encontrarse en medio del filtro y evita problemas al hacer el barrido con un parche deslizante, puesto que, si no, los píxeles centrales se procesan más veces que los exteriores y se puede obviar información relevante de los bordes de la imagen. A pesar de añadir más capas y profundidad, si todas mantienen al argumento *padding='same'* el tamaño de la salida queda fijo al indicado en *input_shape*. En la siguiente figura se muestra un ejemplo de *zero-padding* aplicado en un mapa de características.

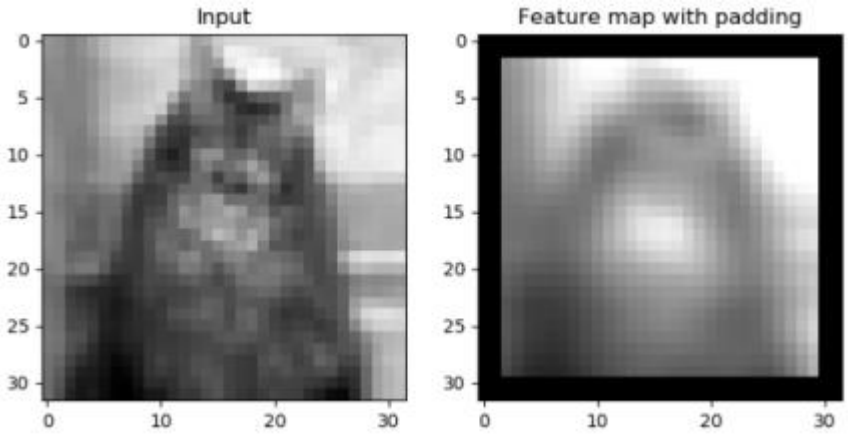

Figura 4-11. Mapa de características con relleno de ceros [75].

A continuación, se añade una capa de *pooling* para disminuir las dimensiones sin aumentar la profundidad, con el método empleado *MaxPooling* con un filtro de tamaño 2x2 [76].

Después, se inserta la segunda capa convolucional, con las mismas características que la anterior, aunque con distinto número de filtros. Lo mismo ocurre con la tercera capa.

Una vez realizado el paso por estas capas, se determina que el número de mapas de características es suficientemente elevado y la resolución de la imagen pequeña como para obtener los resultados esperados. Se ha alcanzado la profundidad necesaria.

Ahora, las imágenes se pasan a un vector unidimensional que almacena toda la información. Para aplanarlo, se usa la función *flatten*.

En este punto exacto, se da por concluido el diseño de la red neuronal convolucional y los datos obtenidos sirven como entrada de la red neuronal clásica que se encarga de clasificar las imágenes en las tres posibles categorías.

La primera capa densa de esta ANN, conectada con las salidas de la capa anterior, consta de 256 neuronas y de nuevo usa una activación ReLU. Se combina con un *dropout* de 0.8, es decir, a esta capa se le 'apagan' el 80% de las neuronas en cada paso de manera aleatoria para evitar *overfitting*. Agregadas a ella, se ponen tres capas densas más, que van disminuyendo a la mitad el número de neuronas que tienen y decrementando el *dropout*

progresivamente. Por último, se añade una capa final con tantas neuronas como clases existentes y ofrece resultados de clasificación según la mayor probabilidad en el rango [0,1], obtenida con la función softmax.

```
#Parametros de la red convolucional
filtrosConv1 = 32
filtrosConv2 = 64
filtrosConv3 = 128
pix_filtro1 = (3,3)
pix_filtro2 = (3,3)
pix_filtro3 = (3,3)
pooling_size = (2,2)
clases = 3
lr = 0.0005
```

Código 4-7. Parámetros internos de la red final.

```
#Creación de la red neuronal convolucional
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, pix_filtro1, padding ="same", input_shape=(longitud, altura, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Convolution2D(filtrosConv2, pix_filtro2, padding ="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Convolution2D(filtrosConv3, pix_filtro3, padding ="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))

cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.8))
cnn.add(Dense(128, activation='relu'))
cnn.add(Dropout(0.6))
cnn.add(Dense(64, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(32, activation='relu'))
cnn.add(Dropout(0.4))
cnn.add(Dense(clases, activation='softmax'))
```

Código 4-8. Estructura de la red final.

Ya se ha diseñado y estructurado el modelo de red neuronal convolucional y la posterior clasificación. Ahora, se va a tratar el método de aprendizaje con la función *compile* para la optimización del sistema. Se determina cual será la función de pérdidas, que se usa para evaluar la diferencia entre las salidas esperadas con los datos de entrenamiento y la realmente obtenidas, se usa la función *categorical_crossentropy*, porque la salida elige entre las distintas categorías que hay. También se escoge el optimizador *Adam* para minimizar la función de pérdidas y para ello hay que ajustar ciertos valores de forma dinámica en la red neuronal, algo que no se puede hacer de forma metódica, pero sí a través de optimizadores iterativos con una tasa de aprendizaje dada. Finalmente, la métrica es la precisión escogida, que estudia es el porcentaje de imágenes que han sido clasificadas tal y como se esperaba.

```
#Parámetros para la optimización
cnn.compile(loss='categorical_crossentropy',
            optimizer=optimizers.Adam(learning_rate=lr),
            metrics=['accuracy'])
```

Código 4-9. Optimización de la red final.

La parte final del proceso consiste en entrenar la red y validarla, pues la arquitectura está completamente finalizada. Cada época de la etapa de entrenamiento tiene un número de pasos para no entrar en un bucle infinito y una vez finalizada, se cargan las imágenes de validación y se establecen los pasos propios de validación. Como se puede ver en el código 4-3, el número de pasos se divide entre *batch_size* y se queda con la parte entera de esta fracción, por lo que, si se descarta la parte decimal, se pierden algunas imágenes de ese paso. No obstante,

como la selección de imágenes es aleatoria se acaban usando todas al correr todas las épocas y no es estrictamente necesario que el número de imágenes sea divisible entre el tamaño del lote.

```
#Entrenamiento de la red.
cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epochs,
    validation_data=validacion_generador,
    validation_steps=pasos_validacion)
```

Código 4-10. Entrenamiento de la red final.

Se usa la función *fit_generator* y se entregan como argumentos ambos conjuntos de imágenes (entrenamiento y validación), los pasos y el número de épocas.

Cuando se ejecute el fichero, se creará un modelo con una configuración determinada. Este modelo se guarda en un archivo, para no tener que volver a entrenar al sistema de nuevo en cada predicción y sólo tener que reusarlo, tanto la estructura como los pesos. Si no existe el directorio, es decir, si es la primera vez que se entrena, se crea este archivo.

```
#Se guarda el modelo y los pesos en un archivo.
target_dir = './modelo/'
if not os.path.exists(target_dir):
  os.mkdir(target_dir)
cnn.save('./modelo/modelo.h5')
cnn.save_weights('./modelo/pesos.h5')
```

Código 4-11. Guardado del modelo y los pesos de la red final.

5 EVALUACIÓN DE LOS RESULTADOS Y CAMBIOS DEL EXPERIMENTO

En este capítulo se van a estudiar los resultados obtenidos, no sólo en la red final seleccionada, que es la presentada en el capítulo anterior, si no las transformaciones que se han ido haciendo desde la primera red inicial que se diseñó.

En este memoria sólo se reflejan algunos de los cambios que ha sufrido la red, puesto que se han realizado multitud de cambios de parámetros y estructurales que se han decidido no añadir, ya que no aportaban nueva información respecto de otras situaciones ya explicadas y abarcaban conceptos redundantes.

5.1. Experimentos

En este apartado se analizan los distintos experimentos desarrollados, junto con una especificación de los motivos por los que algunos de ellos no han funcionado como se requiere. Se comenta qué se ha cambiado para mejorar los resultados de precision y pérdidas, se estudia aquello que debilita la red y se intenta reforzar, poco a poco, su estructura. En resumen, se tienen tres modelos distintos que resultan de cambiar parámetros internos de la red, un cambio que proviene de usar una base de datos diferente y por ultimo, se escribe sobre las redes neuronales preentrenadas y se emplea una de ellas, AlexNet, como ejemplo práctico.

5.1.1. Primer modelo de la CNN.

Como se puede verificar en el código que procede, el problema de la red del primer modelo es que es demasiado sencilla. En la parte convolucional, sólo se tienen dos capas, de 32 y 64 filtros de dimensiones (3,3) y (2,2) respectivamente, acompañados de otras capas de *MaxPooling.* La red añadida encargada de clasificar tiene una única capa densa de 256 neuronas y la capa final de *softmax*. El proceso total consta de 75 épocas, con un tamaño del *batch* de 32 y una normalización de las medidas de la imágenes de 64x64, un valor típico en este tipo de aplicaciones [77] [76]. Estas dimensiones fijas son adecuadas para la base de datos empleada, pues a pesar de que algunas imágenes superen esta altura y longitud, el cuadrado originado recorta desde el centro y el objeto objetivo se sitúa en el medio. Por ejemplo, los datos de entrenamiento con mascarilla ofrecen distinta diversidad en sus medidas, dentro de un rango que va desde 25x25 hasta 563x563; aunque de las 400 imágenes de este directorio, sólo son 67 (fracción correspondiente con un 16.75%) las que superan medidas de 100x100. Por lo general, se tienen imágenes pequeñas y fáciles de procesar, con propiedades como las expuestas a continuación en la figura 5-1.

El parámetro *batch_size* de 32 es bastante común, pues está relacionado con el punto de convergencia y el tiempo de alcance que requiere. Es buena práctica empezar con valores bajos, como es 32 o 64, e ir aumentando si se desea, así como emplear una potencia de dos para aprovechar mejor los recursos de procesamiento. Por otro lado, para tamaños de lote pequeños se recomiendan tasa de aprendizaje pequeñas como es 0.0005, pues son parámetros altamente correlacionados [78].

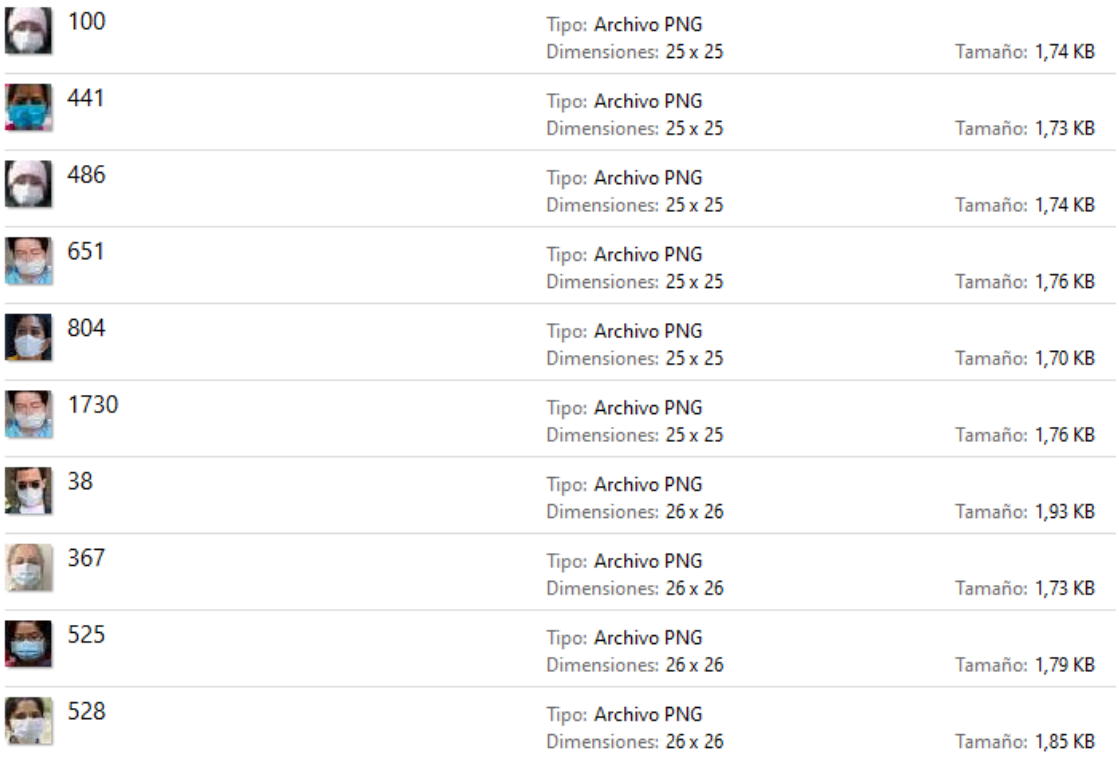

Figura 5-1. Tamaño de las imágenes de entrenamiento de la red.

En las próximas imágenes se insertan fragmentos del fichero de entrenamiento de la red, aunque no completo, sólo de las partes que cambian de un modelo a otro.

```
#Establecer hiperparámetros
epochs = 75
altura = 64
longitud = 64
batch_size = 32
pasos = (518/batch_size)
pasos_validacion = (38/batch_size)
```

Código 5-1. Datos del primer modelo.

```
#Preprocesamiento de las imágenes para poder entrenar la red
entrenamiento_datagen = ImageDataGenerator(
    rescale = (1./255),
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True)

validacion_datagen = ImageDataGenerator(
    rescale=1./255)
```

Código 5-2. Preprocesado de imágenes de entrenamiento y validación del primer modelo.

```
#Parametros de la red convolucional
filtrosConv1 = 32
filtrosConv2 = 64
pix_filtro1 = (3,3)
pix_filtro2 = (2,2)
pooling_size = (2,2)
clases = 3
lr = 0.0005
```

Código 5-3. Parámetros internos de la red del primer modelo.

```
#Creación de la red neuronal convolucional
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, pix_filtro1, padding ="same", input_shape=(longitud, altura, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Convolution2D(filtrosConv2, pix_filtro2, padding ="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))
```

Código 5-4. Estructura de la red del primer modelo.

Tras ejecutar el código, se poder observar que el resultado no es nada satisfactorio. En las capturas de pantalla adjuntas se ve que las pérdidas de validación van aumentando con el paso de las épocas, un efecto que siempre se debe eludir. Además, la pérdida y la precisión del entrenamiento son excesivamente ideales, llegando esta última a la perfección con valor de 1.000, algo improbable en un modelo real. Por tanto, todo indica que existe un caso claro de sobreajuste, a pesar de haber utilizado las técnicas recomendadas de *Data Augmentation* y *Dropout*.

```
16/16 [==============================] - 4s 198ms/step - loss: 0.9636 - accuracy: 0.5551 - val_loss: 0.8530 - val_accuracy: 0.6406
Epoch 2/74
16/16 [==============================] - 3s 176ms/step - loss: 0.5803 - accuracy: 0.7977 - val_loss: 0.4230 - val_accuracy: 0.8594
Epoch 3/74
16/16 [==============================] - 3s 183ms/step - loss: 0.4127 - accuracy: 0.8536 - val_loss: 0.4498 - val_accuracy: 0.7969
Epoch 4/74
16/16 [==============================] - 3s 182ms/step - loss: 0.2990 - accuracy: 0.9094 - val_loss: 0.5051 - val_accuracy: 0.8125
Epoch 5/74
16/16 [==============================] - 3s 186ms/step - loss: 0.2714 - accuracy: 0.9099 - val_loss: 0.7400 - val_accuracy: 0.7344
Epoch 6/74
16/16 [==============================] - 3s 188ms/step - loss: 0.2240 - accuracy: 0.9375 - val_loss: 0.7012 - val_accuracy: 0.7500
Epoch 7/74
16/16 [==============================] - 3s 189ms/step - loss: 0.2175 - accuracy: 0.9383 - val_loss: 1.0092 - val_accuracy: 0.6719
Epoch 8/74
16/16 [==============================] - 3s 192ms/step - loss: 0.1656 - accuracy: 0.9540 - val_loss: 0.7428 - val_accuracy: 0.7969
Epoch 9/74
16/16 [==============================] - 3s 173ms/step - loss: 0.1745 - accuracy: 0.9441 - val_loss: 0.8596 - val_accuracy: 0.6875
Epoch 10/74
16/16 [==============================] - 3s 172ms/step - loss: 0.1271 - accuracy: 0.9651 - val_loss: 0.5906 - val_accuracy: 0.8125
```

Código 5-5. 10 primeras épocas del primer modelo.

```
Epoch 64/74
16/16 [==============================] - 3s 181ms/step - loss: 0.0246 - accuracy: 0.9963 - val_loss: 1.4168 - val_accuracy: 0.7969
Epoch 65/74
16/16 [==============================] - 4s 222ms/step - loss: 0.0328 - accuracy: 0.9942 - val_loss: 1.5871 - val_accuracy: 0.7656
Epoch 66/74
16/16 [==============================] - 3s 162ms/step - loss: 0.0150 - accuracy: 0.9942 - val_loss: 1.3278 - val_accuracy: 0.7656
Epoch 67/74
16/16 [==============================] - 3s 157ms/step - loss: 0.0287 - accuracy: 0.9942 - val_loss: 1.3821 - val_accuracy: 0.7500
Epoch 68/74
16/16 [==============================] - 3s 191ms/step - loss: 0.0332 - accuracy: 0.9871 - val_loss: 1.1704 - val_accuracy: 0.8281
Epoch 69/74
16/16 [==============================] - 3s 203ms/step - loss: 0.0281 - accuracy: 0.9827 - val_loss: 2.2316 - val_accuracy: 0.7344
Epoch 70/74
16/16 [==============================] - 3s 179ms/step - loss: 0.0277 - accuracy: 0.9945 - val_loss: 1.1967 - val_accuracy: 0.7500
Epoch 71/74
16/16 [==============================] - 3s 170ms/step - loss: 0.0095 - accuracy: 0.9982 - val_loss: 1.7300 - val_accuracy: 0.7500
Epoch 72/74
16/16 [==============================] - 3s 180ms/step - loss: 0.0155 - accuracy: 0.9945 - val_loss: 2.6033 - val_accuracy: 0.6875
Epoch 73/74
16/16 [==============================] - 4s 232ms/step - loss: 0.0165 - accuracy: 0.9942 - val_loss: 2.0163 - val_accuracy: 0.7344
Epoch 74/74
16/16 [==============================] - 4s 228ms/step - loss: 0.0081 - accuracy: 1.0000 - val_loss: 2.0997 - val_accuracy: 0.7188
```

Código 5-6. 10 últimas épocas del primer modelo.

Las figuras 5-2 y 5-3 muestran la precisión de la clasificación y la función de pérdidas a medida que avanzan las épocas. Se verifica de manera gráfica que el entrenamiento es prácticamente perfecto mientras que la validación es muy inestable y se aleja de lo buscado. Se obtiene una precisión máxima de validación de 85.94%, un resultado aceptable, pero teniendo en cuenta las otras adversidades, la red es inútil.

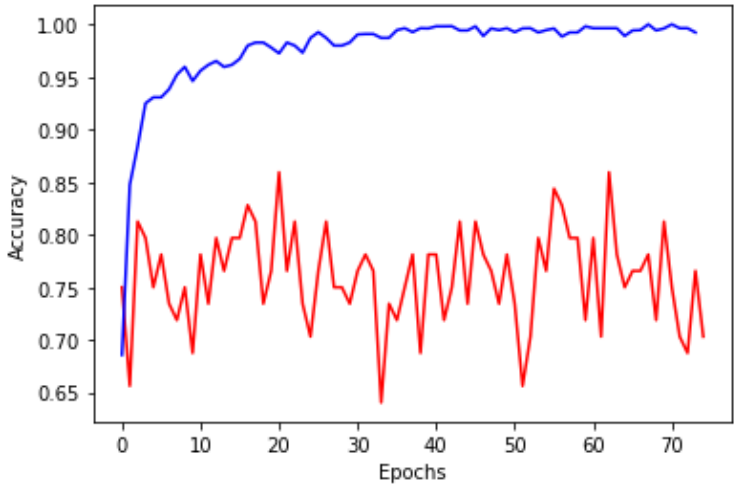

Figura 5-2. Precisión de entrenamiento y validación del primer modelo.

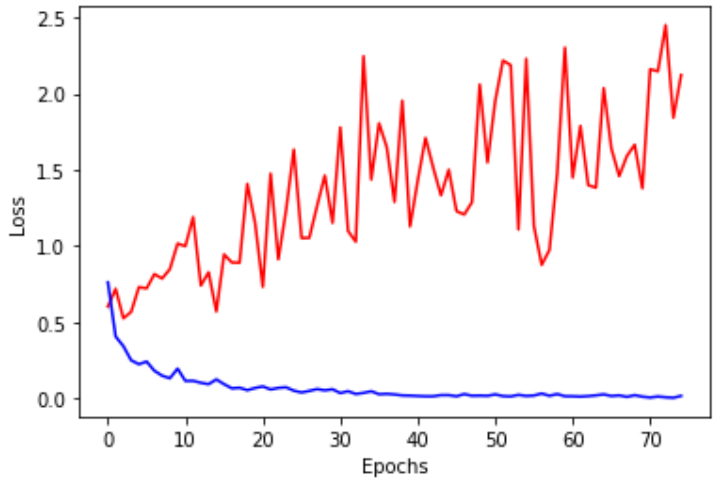

Figura 5-3. Pérdidas de entrenamiento y validación del primer modelo.

Es importante especificar que en todas las gráficas ofrecidas en este quinto capítulo la leyenda es la siguiente: los datos de color azul pertenecen a la fase de entrenamiento y los datos de color rojo pertenecen a la fase de validación.

Para solucionar estos conflictos se va a considerar un red más robusta, con más capas y más neuronas.

5.1.2. Segundo modelo de la CNN.

En este segundo modelo de la red, se intenta hacer un poco más complicada su arquitectura. Se añade otra capa densa junto con un abandon adicional de 0.5 para intentar mitigar el sobreajustre, quedando cada capa con 128 y 64 neuronas. Se ha dejado igual el número de filtros de la red convolucional, pero incrementando el tamaño de los conjunto de filtros a 3x3 y 4x4 respectivamente. Por lo demás, todo continúa como antes.

```
#Parametros de la red convolucional
filtrosConv1 = 32
filtrosConv2 = 64
pix_filtro1 = (4,4)
pix_filtro2 = (3,3)
pooling_size = (2,2)
clases = 3
lr = 0.0005
```

Código 5-7. Parámetros internos de la red del segundo modelo.

```
#Creación de la red neuronal convolucional
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, pix_filtro1, padding ="same", input_shape=(64,64,3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Convolution2D(filtrosConv2, pix_filtro2, padding ="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))

cnn.add(Flatten())
cnn.add(Dense(128, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(64, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))
```

Código 5-8. Estructura de la red del segundo modelo.

```
Epoch 1/75
16/16 [==============================] - 6s 331ms/step - loss: 1.0726 - accuracy: 0.5037 - val_loss: 0.8959 - val_accuracy: 0.7656
Epoch 2/75
16/16 [==============================] - 5s 298ms/step - loss: 0.7670 - accuracy: 0.6985 - val_loss: 0.5608 - val_accuracy: 0.7344
Epoch 3/75
16/16 [==============================] - 7s 414ms/step - loss: 0.5843 - accuracy: 0.7757 - val_loss: 0.4850 - val_accuracy: 0.8281
Epoch 4/75
16/16 [==============================] - 6s 359ms/step - loss: 0.4657 - accuracy: 0.8254 - val_loss: 0.4547 - val_accuracy: 0.8750
Epoch 5/75
16/16 [==============================] - 5s 336ms/step - loss: 0.4967 - accuracy: 0.8217 - val_loss: 0.4234 - val_accuracy: 0.8438
Epoch 6/75
16/16 [==============================] - 5s 293ms/step - loss: 0.4362 - accuracy: 0.8456 - val_loss: 0.4826 - val_accuracy: 0.7969
Epoch 7/75
16/16 [==============================] - 4s 276ms/step - loss: 0.3752 - accuracy: 0.8787 - val_loss: 0.4316 - val_accuracy: 0.8281
Epoch 8/75
16/16 [==============================] - 4s 269ms/step - loss: 0.3357 - accuracy: 0.8824 - val_loss: 0.3951 - val_accuracy: 0.8594
Epoch 9/75
16/16 [==============================] - 4s 257ms/step - loss: 0.3667 - accuracy: 0.8879 - val_loss: 0.3809 - val_accuracy: 0.8594
Epoch 10/75
16/16 [==============================] - 4s 273ms/step - loss: 0.3486 - accuracy: 0.8842 - val_loss: 0.5174 - val_accuracy: 0.7656
```

Código 5-9. 10 primeras épocas del segundo modelo.

```
Epoch 65/75
16/16 [==============================] - 3s 167ms/step - loss: 0.0775 - accuracy: 0.9761 - val_loss: 0.3221 - val_accuracy: 0.9375
Epoch 66/75
16/16 [==============================] - 3s 169ms/step - loss: 0.0320 - accuracy: 0.9926 - val_loss: 0.5238 - val_accuracy: 0.8438
Epoch 67/75
16/16 [==============================] - 3s 168ms/step - loss: 0.0214 - accuracy: 0.9982 - val_loss: 0.4108 - val_accuracy: 0.9062
Epoch 68/75
16/16 [==============================] - 3s 171ms/step - loss: 0.0249 - accuracy: 0.9890 - val_loss: 1.1579 - val_accuracy: 0.7812
Epoch 69/75
16/16 [==============================] - 3s 167ms/step - loss: 0.0547 - accuracy: 0.9926 - val_loss: 1.5653 - val_accuracy: 0.7656
Epoch 70/75
16/16 [==============================] - 3s 171ms/step - loss: 0.0430 - accuracy: 0.9835 - val_loss: 0.6250 - val_accuracy: 0.8594
Epoch 71/75
16/16 [==============================] - 3s 171ms/step - loss: 0.0547 - accuracy: 0.9761 - val_loss: 0.5367 - val_accuracy: 0.8750
Epoch 72/75
16/16 [==============================] - 3s 167ms/step - loss: 0.0435 - accuracy: 0.9853 - val_loss: 0.8110 - val_accuracy: 0.8125
Epoch 73/75
16/16 [==============================] - 3s 171ms/step - loss: 0.0416 - accuracy: 0.9908 - val_loss: 0.9077 - val_accuracy: 0.7969
Epoch 74/75
16/16 [==============================] - 3s 169ms/step - loss: 0.0461 - accuracy: 0.9871 - val_loss: 1.0697 - val_accuracy: 0.8281
Epoch 75/75
16/16 [==============================] - 3s 169ms/step - loss: 0.0397 - accuracy: 0.9871 - val_loss: 0.4786 - val_accuracy: 0.8594
```

Código 5-10. 10 últimas épocas del segundo modelo.

En las 10 primeras y últimas épocas que se acaban de compartir, se puede observar cómo los resultados mejoran, pero siguen sin ser excelentes. Sobre todo, las pérdidas de validación dejan mucho que desear, alcanzando picos de hasta 1.5653. Estos comportamientos se reflejan de igual manera en las gráficas.

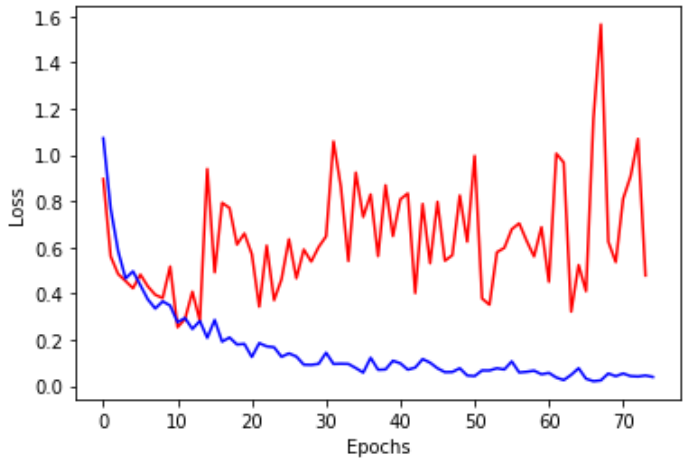

Figura 5-4. Precisión de entrenamiento y validación del segundo modelo.

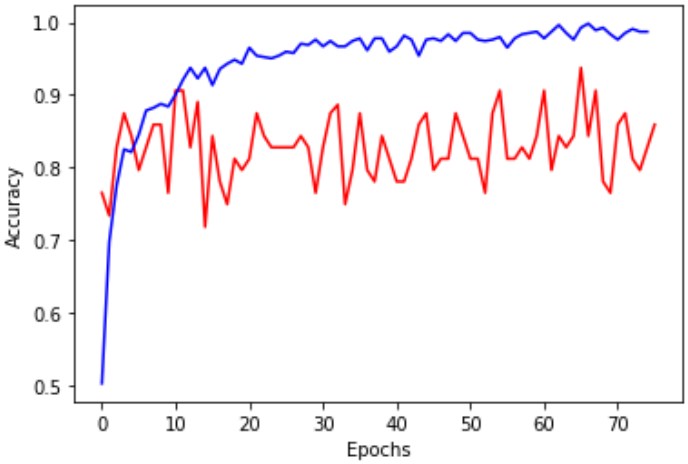

Figura 5-5. Pérdidas de entrenamiento y validación del segundo modelo.

Por lo tanto, en esta ejecución el sobreentrenamiento es menos dañino, pero igualmenete sigue existiendo. La curva de las pérdidas no crece de forma tan lineal como antes, pero no es suficiente mejora.

5.1.3. Tercer modelo de la CNN.

Tras investigar, se ha practicado un efecto más agresivo de Data Augmentation, pues se han incrementado las propiedades del generador de 20% al 30% de zoom e inclinación.

La red se ha hecho más grande y compleja. En la CNN se ha insertado una tercera capa más con 128 filtros de tamaño (3,3), el valor estándar esta vez constante para todos los filtros según fuentes de información consultadas [79]. La ANN clásica cuenta con otra capa más y un efecto de abandono mayor, quedando las parejas de neuronas de la capa y el posterior *dropout* tal que así: 256 neuronas y 0.8, 128 neuronas y 0.6, 64 neuronas y 0.5 y 32 neuronas y 0.4. Esta configuración es más adecuada para evitar caer en el *overfitting,* ya que, aunque 0.5 valor de de dropout sea un valor intermedio y parezca el óptimo, es muy probable que la red entrene mejor con un valor más cercano a 1 que a 0.5 mientras más proximidad haya con respecto a la primera capa de entrada [80].

```
#Parametros de la red convolucional
filtrosConv1 = 32
filtrosConv2 = 64
filtrosConv3 = 128
pix_filtro1 = (3,3)
pix_filtro2 = (3,3)
pix_filtro3 = (3,3)
pooling_size = (2,2)
clases = 3
lr = 0.0005
```

Código 5-11. Parámetros internos de la red del tercer modelo.

```
#Preprocesamiento de las imágenes para poder entrenar la red
entrenamiento_datagen = ImageDataGenerator(
    rescale = (1./255),
    shear_range = 0.3,
    zoom_range = 0.3,
    horizontal_flip = True)

validacion_datagen = ImageDataGenerator(
    rescale=1./255)
```

Código 5-12. Preprocesado de imágenes de entrenamiento y validación del tercer modelo.

```
#Creación de la red neuronal convolucional
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, pix_filtro1, padding ="same", input_shape=(64,64,3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Convolution2D(filtrosConv2, pix_filtro2, padding ="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))
cnn.add(Convolution2D(filtrosConv3, pix_filtro3, padding ="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=pooling_size))

cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.8))
cnn.add(Dense(128, activation='relu'))
cnn.add(Dropout(0.6))
cnn.add(Dense(64, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(32, activation='relu'))
cnn.add(Dropout(0.4))
cnn.add(Dense(clases, activation='softmax'))
```

Código 5-13. Estructura del tercer modelo.

Se exponen las 10 primeras y 10 últimas épocas. Se puede probar ahora que los resultados son similares a los esperados: no hay apenas sobreajuste. Esta vez las pérdidas de validación no aumentan a medida que avanzan las iteraciones y el entrenamiento se asemeja a un problema común y no tan idílico.

```
Epoch 1/75
16/16 [==============================] - 5s 254ms/step - loss: 1.1013 - accuracy: 0.4136 - val_loss: 1.0790 - val_accuracy: 0.3438
Epoch 2/75
16/16 [==============================] - 3s 200ms/step - loss: 1.0788 - accuracy: 0.4393 - val_loss: 1.0761 - val_accuracy: 0.3750
Epoch 3/75
16/16 [==============================] - 3s 197ms/step - loss: 1.0756 - accuracy: 0.4173 - val_loss: 1.0422 - val_accuracy: 0.6406
Epoch 4/75
16/16 [==============================] - 3s 190ms/step - loss: 1.0276 - accuracy: 0.4412 - val_loss: 1.0735 - val_accuracy: 0.4531
Epoch 5/75
16/16 [==============================] - 3s 197ms/step - loss: 1.0223 - accuracy: 0.4706 - val_loss: 1.0451 - val_accuracy: 0.5938
Epoch 6/75
16/16 [==============================] - 3s 190ms/step - loss: 0.9936 - accuracy: 0.5257 - val_loss: 0.9878 - val_accuracy: 0.7344
Epoch 7/75
16/16 [==============================] - 3s 195ms/step - loss: 0.9087 - accuracy: 0.5680 - val_loss: 0.9313 - val_accuracy: 0.7656
Epoch 8/75
16/16 [==============================] - 3s 201ms/step - loss: 0.8600 - accuracy: 0.6324 - val_loss: 0.9230 - val_accuracy: 0.7031
Epoch 9/75
16/16 [==============================] - 3s 192ms/step - loss: 0.8590 - accuracy: 0.6710 - val_loss: 0.7683 - val_accuracy: 0.7344
Epoch 10/75
16/16 [==============================] - 3s 193ms/step - loss: 0.8150 - accuracy: 0.7371 - val_loss: 0.8099 - val_accuracy: 0.7031
```

Código 5-14. 10 primeras épocas del tercer modelo.

```
Epoch 65/75
16/16 [==============================] - 4s 229ms/step - loss: 0.3300 - accuracy: 0.8768 - val_loss: 0.3247 - val_accuracy: 0.8438
Epoch 66/75
16/16 [==============================] - 3s 200ms/step - loss: 0.2794 - accuracy: 0.9044 - val_loss: 0.3161 - val_accuracy: 0.8125
Epoch 67/75
16/16 [==============================] - 4s 273ms/step - loss: 0.3433 - accuracy: 0.9044 - val_loss: 0.3608 - val_accuracy: 0.8281
Epoch 68/75
16/16 [==============================] - 4s 230ms/step - loss: 0.2590 - accuracy: 0.9173 - val_loss: 0.2813 - val_accuracy: 0.8750
Epoch 69/75
16/16 [==============================] - 5s 282ms/step - loss: 0.2366 - accuracy: 0.9246 - val_loss: 0.3380 - val_accuracy: 0.8438
Epoch 70/75
16/16 [==============================] - 4s 234ms/step - loss: 0.2576 - accuracy: 0.8952 - val_loss: 0.3190 - val_accuracy: 0.8281
Epoch 71/75
16/16 [==============================] - 4s 277ms/step - loss: 0.2774 - accuracy: 0.9118 - val_loss: 0.4450 - val_accuracy: 0.7656
Epoch 72/75
16/16 [==============================] - 4s 221ms/step - loss: 0.3454 - accuracy: 0.8787 - val_loss: 0.2927 - val_accuracy: 0.8281
Epoch 73/75
16/16 [==============================] - 5s 281ms/step - loss: 0.2367 - accuracy: 0.9265 - val_loss: 0.4134 - val_accuracy: 0.7500
Epoch 74/75
16/16 [==============================] - 5s 282ms/step - loss: 0.2012 - accuracy: 0.9430 - val_loss: 0.3982 - val_accuracy: 0.8125
Epoch 75/75
16/16 [==============================] - 4s 225ms/step - loss: 0.2029 - accuracy: 0.9338 - val_loss: 0.2582 - val_accuracy: 0.8750
```

Código 5-15. 10 últimas épocas del tercer modelo.

En las gráficas 5-6 y 5-7 se visualiza el comportamiento de forma más intuitiva. Es normal que la validación tenga unos resultados algo inferiores que el entrenamiento y como debe ser, la precisión tiende a aumentar y las pérdidas a disminuir en todo el transcurso del sistema. Se logra una precisión máxima en validación de 95.31%, valor bastante bueno de acuerdo con los informes tomados como referencia [81] [76].

La precisión máxima llega a ese excelente valor de 0.9531 debido a un pico, pero es cierto que su media en las últimas épocas es ligeramente inferior. Además, el valor mínimo de pérdidas es de 0.2231, pero ocurre lo mismo, en realidad es un poco más alto si no hubiera discontinuidades. Por ello, lo ideal sería suavizar los picos existentes y hacer una curva más estable, considerando un posible perfeccionamiento de los resultados a pesar de tener una red de alto nivel.

Se ha probado a mejorar la base de datos aplicando más alteraciones a través de Data Augmentation, pero los resultados eran muy similares. Después de hacer varios cambios y no obtener resultados notablemente mejores que el anterior, se va a probar con otra BBDD. No obstante, se va a hacer un estudio de los resultados obtenidos en estas circunstancias, pues este tercer modelo es el escogido finalmente como modelo final del proyecto.

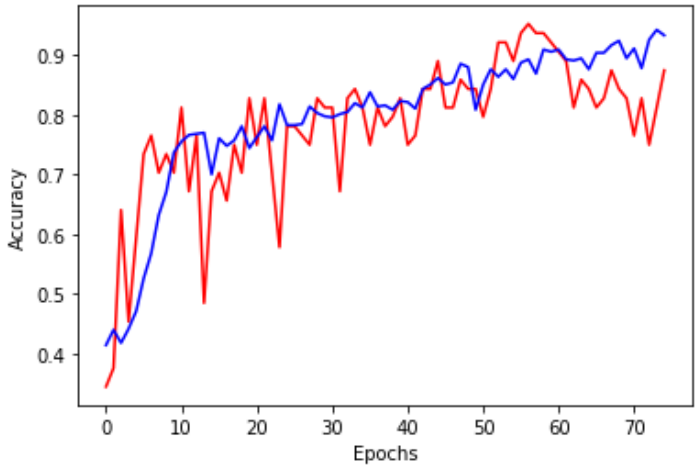

Figura 5-6. Precisión de entrenamiento y validación del tercer modelo.

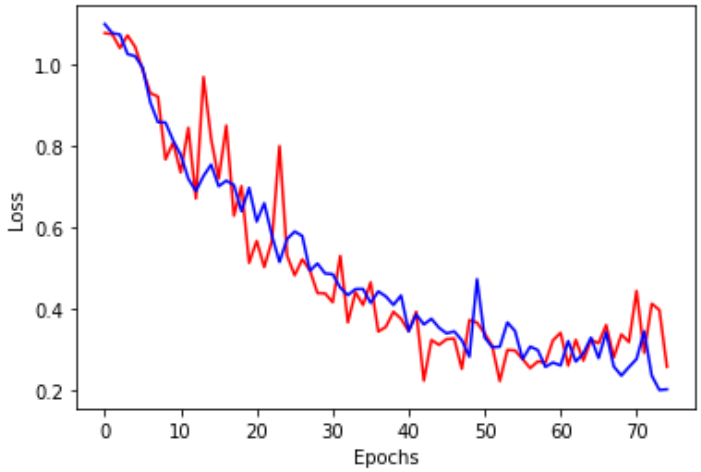

Figura 5-7. Precisión de entrenamiento y validación del primer modelo.

Predicción

En primer lugar, se procede a hacer una predicción real con datos desconocidos. Para ello se usa un fichero *predicción.py*, con un código como el ahora presentado. Se hace la siguiente importación desde las APIs.

```
import numpy as np
from tensorflow.python.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.keras.models import load_model
from sklearn.metrics import confusion_matrix,classification_report
import matplotlib.pyplot as plt
```

Código 5-16. Librerías importadas en el fichero de entrenamiento.

La altura y longitud de las imágenes deben coincidir con las dimensiones de las imágenes de entrenamiento para evitar errores innecesarios. Se importan los archivos guardados para poder reutilizar la red, de forma que sólo con indicar el camino se carga el modelo y los pesos a una variable denominada *cnn*.

```
longitud, altura = 64, 64
modelo = './modelo/modelo.h5'
pesos = './modelo/pesos.h5'
cnn = load_model(modelo)
cnn.load_weights(pesos)
```

Código 5-17. Importación de archivos del modelo.

La segunda parte del fichero consiste en predecir a través de la función *prediccion* a la que se le pasa la ruta de la imagen determinada. Se carga la imagen y después, se pasa a un vector unidimensional de los valores que la representa, añadiendo en el eje cero un dimensión adicional que permite procesar mejor la información. La orden *predict* se encarga de escoger la categoría a la que pertenece la clase, de forma que, si la imagen pertenece a una categoría concreta, en su posición del vector se pone un uno, y en el resto de las posiciones un cero; el vector tiene tantas posiciones como clases. Finalmente, esta función facilita el índice del valor más alto, es decir, del uno, y dependiendo del índice que sea se clasifica en una clase u otra. La clase 0 es sinónimo de llevar bien la mascarilla, la clase 1 de llevarla de forma incorrecta y la clase 2 de no llevarla.

```
#Se hace una función que recibe la ruta de la imagen y predice que clase es.
def prediccion(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  clase = np.argmax(result)
  if clase == 0:
    print("La imagen de entrada pertenece a la clase con mascarilla")
  elif clase == 1:
    print("La imagen de entrada pertenece a la clase mascarilla incorrecta")
  elif clase == 2:
    print("La imagen de entrada pertenece a la clase sin mascarilla")

  return
```

Código 5-18. Función de predicción.

A continuación, se muestran unos ejemplos de predicción que pueden dar lugar a confusión ya que no son tan evidentes como el resto. Como se observa en el código 5-19, la red predice de forma correcta todos estos ejemplos excepto la *imagencinco.jpg*, debido a que identifica la barba como si fuera una mascarilla. Con esto sólo se desea mostrar que en la mayoría de los casos la clasificación es correcta, aunque esta falta de perfección es la que puede provocar errores como este.

Figura 5-8. Ejemplos de imágenes confusas de clasificación.

```
In [111]: prediccion('./datos/prueba/imagenuno.jpg')
La imagen de entrada pertenece a la clase sin mascarilla

In [112]: prediccion('./datos/prueba/imagendos.jpg')
La imagen de entrada pertenece a la clase mascarilla incorrecta

In [113]: prediccion('./datos/prueba/imagentres.jpg')
La imagen de entrada pertenece a la clase con mascarilla

In [114]: prediccion('./datos/prueba/imagencuatro.jpg')
La imagen de entrada pertenece a la clase sin mascarilla

In [115]: prediccion('./datos/prueba/imagencinco.jpg')
La imagen de entrada pertenece a la clase con mascarilla

In [116]: prediccion('./datos/prueba/imagenseis.jpg')
La imagen de entrada pertenece a la clase con mascarilla
```

Código 5-19. Predicciones resultantes de los datos de la figura 5-8.

Matriz de confusión

La matriz de confusión es una métrica que mide los errores e imperfecciones del aprendizaje automático para evaluar el rendimiento de la clasificación. Además de medir la precisión, que es el número de predicciones correctas en base al número total de predicciones, sirve para medir otros resultados más recónditos. Se hallan así para cada campo de aplicación diferentes tipos de errores, algunos más tolerables que otros, por lo que es importante saber cómo tratar los resultados y prevenir interpretaciones incorrectas.

Las matrices de confusión tienen multitud de utilidades, entre otras: inspeccionar los errores de cada categoría concreta, ayudar a optimizar el ajuste de parámetros e interpretar la sensibilidad de los errores en cada dominio [82].

Para el estudio de los errores, se tiene en cuenta que la matriz se analiza por filas y por columnas y de ellas, se pueden deducir métricas complicadas derivadas de otras más sencillas.

La matriz de confusión en un escenario con dos clases se asocia a una tabla bidimensional, como la mostrada en la tabla 5-1. En ella, una dimensión es corresponde con clasificaciones actuales o instancias reales (columnas) y la otra con el número de predicciones de una categoría (filas) [83].

	Positivos reales (1)	**Negativos reales** (0)
Positivos predichos (1)	**Positivos verdaderos (VP)**	**Positivos falsos (FP)**
Negativos predichos (0)	**Negativos falsos (FN)**	**Negativos verdaderos (VN)**

Tabla 5-1. Matriz de confusión.

Para cada categoría, se codifican diferentes errores. La situación ideal se alcanza si no hay ningún falso negativo ni falso positivo, también visible como tener todos los datos en la diagonal principal y que el resto de los valores queden nulos. Por cada falso negativo de una clase, se incrementa un falso positivo en otra clase. Los términos para comprender el funcionamiento de esta herramienta son los expuestos:

- **VP** (Positivo verdadero). El dato es positivo y se predice en la categoría positiva correcta.
- **VN** (Negativo verdadero). El dato es negativo y se predice en la categoría negativa correcta.
- **FP** (Positivo falso). El dato es negativo y se predice un positivo o error estadístico tipo I.
- **FN** (Negativo falso). El dato es positivo y se predice un negativo o error estadístico tipo II

Algunas de las métricas más importantes que ofrece la matriz de confusión y sus respectivas fórmulas son las siguientes [84] [85]:

- La **exactitud** define lo próxima que queda una clasificación del resultado verdadero, es decir, el número de predicciones que se realizaron de forma correcta. Se define como el número de verdaderos positivos y negativos entre el total de realizaciones. No es una métrica fiable si los conjuntos de datos de cada clase tienen un reparto no igualado, ya que, si la mayoría de las personas no llevan mascarilla, es mucho más obvio acertar afirmando que no la llevan. En esta ocasión, es mejor fiarse del resto de métricas de calidad de la clasificación.

$$Exactitud = \frac{VP + VN}{VP + FP + VN + FN}$$

- La **precisión** es la métrica de calidad que representa la fracción de predicciones positivas detectadas respecto al número total de verdaderos positivos.

$$Precisión = \frac{VP}{VP + FP}$$

- La **sensibilidad** o tasa de verdaderos positivos mide la cantidad de positivos que son bien clasificados en la prueba, es decir, la capacidad de identificar un positivo y discriminarlo de un negativo.

$$Sensibilidad = \frac{VP}{VP + FN}$$

- La **especificidad** o tasa de verdaderos negativos es justo lo contrario que la sensibilidad, o sea, permite medir el porcentaje de los negativos identificados de forma adecuada.

$$Especificidad = \frac{VN}{VN + FP}$$

- El **F1 score** es una medida muy útil, ya que recoge tanto precisión como sensibilidad, ponderadas por igual, en único valor y se suele emplear cuando la distribución de las categorías no es simétrica. En el caso de tener clases desequilibradas, pero igual de importantes, se busca un valor alto de F1 para cada clase. Mientras que, en clases desequilibradas, pero de distinta importancia, se busca un valor alto de F1 en la clase más relevante [86].

$$F1 = 2 * \frac{Sensibilidad * Precisión}{Sensibilidad + Precisión}$$

Se van a evaluar estas métricas sobre la precisión de la red. Para ello, se usan las 30 imágenes de prueba de la figura 5-9, distribuidas de forma equitativa entre las tres categorías existentes: 10 con mascarilla, 10 sin mascarilla y 10 con mascarilla incorrecta, todas siendo imágenes nuevas para la red.

Figura 5-9. Imágenes de prueba.

Se adjuntan las líneas de código que permiten la llamada a la función de predicción de estas imágenes y el resultado obtenido.

```
In [217]: prediccion('./datos/prueba/prueba1.png')
     ...: prediccion('./datos/prueba/prueba2.png')
     ...: prediccion('./datos/prueba/prueba3.png')
     ...: prediccion('./datos/prueba/prueba4.png')
     ...: prediccion('./datos/prueba/prueba5.png')
     ...: prediccion('./datos/prueba/prueba6.png')
     ...: prediccion('./datos/prueba/prueba7.png')
     ...: prediccion('./datos/prueba/prueba8.png')
     ...: prediccion('./datos/prueba/prueba9.png')
     ...: prediccion('./datos/prueba/prueba10.png')
     ...: prediccion('./datos/prueba/prueba11.png')
     ...: prediccion('./datos/prueba/prueba12.png')
     ...: prediccion('./datos/prueba/prueba13.png')
     ...: prediccion('./datos/prueba/prueba14.png')
     ...: prediccion('./datos/prueba/prueba15.png')
     ...: prediccion('./datos/prueba/prueba16.png')
     ...: prediccion('./datos/prueba/prueba17.png')
     ...: prediccion('./datos/prueba/prueba18.png')
     ...: prediccion('./datos/prueba/prueba19.png')
     ...: prediccion('./datos/prueba/prueba20.jpg')
     ...: prediccion('./datos/prueba/prueba21.jpg')
     ...: prediccion('./datos/prueba/prueba22.jpg')
     ...: prediccion('./datos/prueba/prueba23.jpg')
     ...: prediccion('./datos/prueba/prueba24.jpg')
     ...: prediccion('./datos/prueba/prueba25.jpg')
     ...: prediccion('./datos/prueba/prueba26.jpg')
     ...: prediccion('./datos/prueba/prueba27.jpg')
     ...: prediccion('./datos/prueba/prueba28.jpg')
     ...: prediccion('./datos/prueba/prueba29.jpg')
     ...: prediccion('./datos/prueba/prueba30.jpg')
```

Código 5-20. Llamada a la función de predicción con las imágenes de prueba.

```
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase sin mascarilla
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
La imagen de entrada pertenece a la clase mascarilla incorrecta
```

Código 5-21. Predicciones de prueba.

Mediante el código 5-22, se calcula la matriz de confusión con la función importada *confusion_matrix*, su representación y el resto de las métricas con *classification_report*. Se solicitan los datos reales y predichos como argumentos, los cuales se han introducido de forma manual ya que se conocen ambos correspondientes a los códigos 5-20 y 5-21.

```
y_test = [2,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1]
y_pred = [2,2,2,2,2,2,2,2,2,2,2,0,1,0,0,0,1,2,0,0,1,1,1,1,1,1,1,1,1]
cm = confusion_matrix(y_test,y_pred)
print(cm)

figura, ax = plt.subplots(figsize=(10,5))
ax.matshow(cm)
plt.title('Matriz de confusión')
plt.ylabel('Clasificación actual')
plt.xlabel('Predicción')
for (i,j),z in np.ndenumerate(cm):
    ax.text(j,i,'{:0.1f}'.format(z),ha='center',va='center')

informe = classification_report(y_test, y_pred)
print(informe)
```

Código 5-22. Generación de la matriz de confusión y del informe de las métricas.

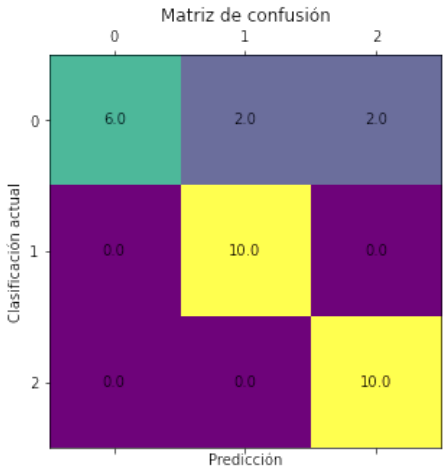

Figura 5-10. Matriz de confusión del tercer modelo.

```
              precision    recall  f1-score   support

           0       1.00      0.60      0.75        10
           1       0.83      1.00      0.91        10
           2       0.83      1.00      0.91        10
```

Código 5-23. Informe de las métricas del tercer modelo.

Examinando los resultados, hay cuatro posibles combinaciones de sensibilidad y precisión para estudiar por cada clase:

- Alta sensibilidad y alta precisión: la clasificación en esta categoría es excelente.
- Baja sensibilidad y alta precisión: esta categoría no suele ser bien detectada, pero cuando la identifica, tiene gran probabilidad de ser una predicción certera.
- Alta sensibilidad y baja precisión: es una clase que se detecta con facilidad, aunque suele incluir datos erróneos pertenecientes a otra clase.
- Baja sensibilidad y baja precisión: la clasificación en esta categoría no es adecuada.

Básicamente, la precisión indica cómo de fiables son los verdaderos positivos y la sensibilidad da una estimación de la cantidad de positivos que no se detectan. Dependiendo de la aplicación que se desea desarrollar, los resultados tendrán mayor o menor relevancia y se buscan unos u otros ajustes, por ejemplo, la precisión es un dato relevante en una situación en la que los falsos positivos tienen mayor importancia que los falsos negativos. En el caso particular del proyecto, se considera más importante detectar personas con mascarilla incorrecta (clase 1) o mascarilla sin colocar (clase 2) que personas con mascarilla bien puesta (clase 0). Además, es más importante que no se escapen los verdaderos positivos a que todos los positivos detectados sean correctos, entonces hay que prestarle mayor atención a la medida de sensibilidad, pasando la precisión a segundo lugar; según lo dicho, la clase 0 que es la de menor *recall* debe mejorar.

5.1.4. Cuarto modelo de la CNN. Cambio de base de datos.

Debido al empleo de una base de datos con una gran variedad de tipos de mascarilla (colores, forma, tamaño…) se prueba la misma red, pero con otra base de datos de Kaggle [87], de la que sólo se toman las imágenes con mascarilla (clase 0) puesto que son las que provocan mayor error en la red de acuerdo con los datos ofrecidos por la matriz de confusión. En particular, esta base de datos usa mascarillas artificiales dibujadas que imitan a una mascarilla quirúrgica, como ejemplo se pueden apreciar las fotografías de la figura 5-11. Con el nuevo conjunto de datos, se tienen en total 960 imágenes para entrenamiento y 242 para la validación.

Figura 5-11. Ejemplo de la nueva base de datos.

Se usa un modelo de red idéntico al del apartado 5.1.3, pero los resultados que se obtienen son muy diferentes. Primero, las épocas muestran una precisión tanto de entrenamiento como de validación prácticamente perfectas, hasta alcanza en varias ocasiones la unidad. Las pérdidas son despreciables, muchos valores quedan incluso por debajo de una décima. Entonces, con resultados tan buenos en entrenamiento y validación, ¿cuál es el problema?

```
Epoch 1/75
16/16 [==============================] - 16s 1s/step - loss: 1.0994 - accuracy: 0.3842 - val_loss: 1.0678 - val_accuracy: 0.4062
Epoch 2/75
16/16 [==============================] - 14s 890ms/step - loss: 1.0640 - accuracy: 0.3824 - val_loss: 1.0731 - val_accuracy: 0.3906
Epoch 3/75
16/16 [==============================] - 15s 943ms/step - loss: 1.0575 - accuracy: 0.4007 - val_loss: 1.0676 - val_accuracy: 0.4531
Epoch 4/75
16/16 [==============================] - 14s 864ms/step - loss: 1.0368 - accuracy: 0.4540 - val_loss: 1.0396 - val_accuracy: 0.5156
Epoch 5/75
16/16 [==============================] - 14s 848ms/step - loss: 0.9905 - accuracy: 0.4669 - val_loss: 1.0340 - val_accuracy: 0.6250
Epoch 6/75
16/16 [==============================] - 14s 907ms/step - loss: 0.9086 - accuracy: 0.5074 - val_loss: 0.8815 - val_accuracy: 0.6719
Epoch 7/75
16/16 [==============================] - 15s 936ms/step - loss: 0.8990 - accuracy: 0.5018 - val_loss: 0.8332 - val_accuracy: 0.6875
Epoch 8/75
16/16 [==============================] - 15s 929ms/step - loss: 0.8809 - accuracy: 0.5735 - val_loss: 0.9770 - val_accuracy: 0.6562
Epoch 9/75
16/16 [==============================] - 14s 838ms/step - loss: 0.8253 - accuracy: 0.6342 - val_loss: 0.7910 - val_accuracy: 0.7188
Epoch 10/75
16/16 [==============================] - 15s 934ms/step - loss: 0.7540 - accuracy: 0.7261 - val_loss: 0.7384 - val_accuracy: 0.7969
```

Código 5-24. 10 primeras épocas del cuarto modelo.

```
Epoch 65/75
16/16 [==============================] - 26s 2s/step - loss: 0.1072 - accuracy: 0.9706 - val_loss: 0.0427 - val_accuracy: 0.9844
Epoch 66/75
16/16 [==============================] - 28s 2s/step - loss: 0.0832 - accuracy: 0.9871 - val_loss: 0.0269 - val_accuracy: 1.0000
Epoch 67/75
16/16 [==============================] - 31s 2s/step - loss: 0.1346 - accuracy: 0.9743 - val_loss: 0.0390 - val_accuracy: 1.0000
Epoch 68/75
16/16 [==============================] - 27s 2s/step - loss: 0.0993 - accuracy: 0.9724 - val_loss: 0.0262 - val_accuracy: 1.0000
Epoch 69/75
16/16 [==============================] - 28s 2s/step - loss: 0.0970 - accuracy: 0.9743 - val_loss: 0.0217 - val_accuracy: 1.0000
Epoch 70/75
16/16 [==============================] - 27s 2s/step - loss: 0.1012 - accuracy: 0.9706 - val_loss: 0.1064 - val_accuracy: 0.9531
Epoch 71/75
16/16 [==============================] - 27s 2s/step - loss: 0.0717 - accuracy: 0.9835 - val_loss: 0.1517 - val_accuracy: 0.9844
Epoch 72/75
16/16 [==============================] - 28s 2s/step - loss: 0.0428 - accuracy: 0.9926 - val_loss: 0.1272 - val_accuracy: 0.9844
Epoch 73/75
16/16 [==============================] - 29s 2s/step - loss: 0.0699 - accuracy: 0.9816 - val_loss: 0.0193 - val_accuracy: 1.0000
Epoch 74/75
16/16 [==============================] - 30s 2s/step - loss: 0.0795 - accuracy: 0.9798 - val_loss: 0.0948 - val_accuracy: 0.9844
Epoch 75/75
16/16 [==============================] - 26s 2s/step - loss: 0.0630 - accuracy: 0.9835 - val_loss: 0.0177 - val_accuracy: 1.0000
```

Código 5-25. 10 últimas épocas del cuarto modelo.

El máximo de precisión en validación es 1.000 y la mínima pérdida de validación es 0.0177, es decir, se reiteran la excelente calidad de los resultados, que también se obvian de las gráficas.

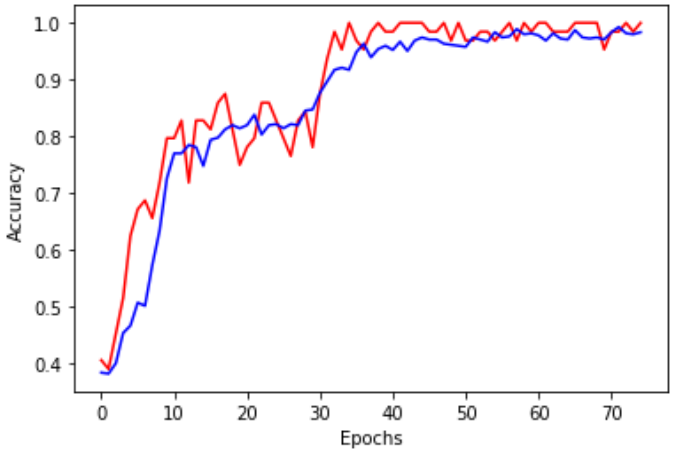

Figura 5-12. Precisión de entrenamiento y validación del cuarto modelo.

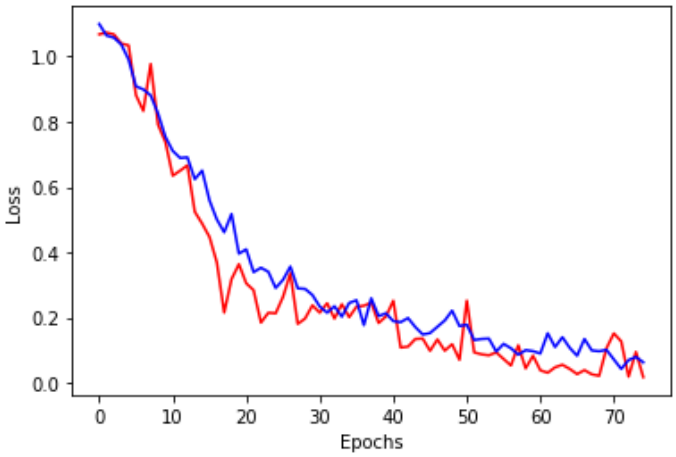

Figura 5-13. Pérdidas de entrenamiento y validación del cuarto modelo.

Lo único que queda pendiente es el cálculo de la predicción con los datos de prueba de la figura 5-9 anterior, para ver si son tan exactos como parece. Sin exponer de nuevo el código completo, se inserta el código 5-26, que señala las predicciones obtenidas. Acto seguido, se incluye la matriz de confusión y las métricas.

```
y_test = [2,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1]
y_pred = [2,2,2,2,2,2,2,2,2,2,2,0,1,0,0,0,0,1,0,2,1,1,1,1,1,1,1,2,1]
```

Código 5-26. Resultados de la predicción con los datos de prueba.

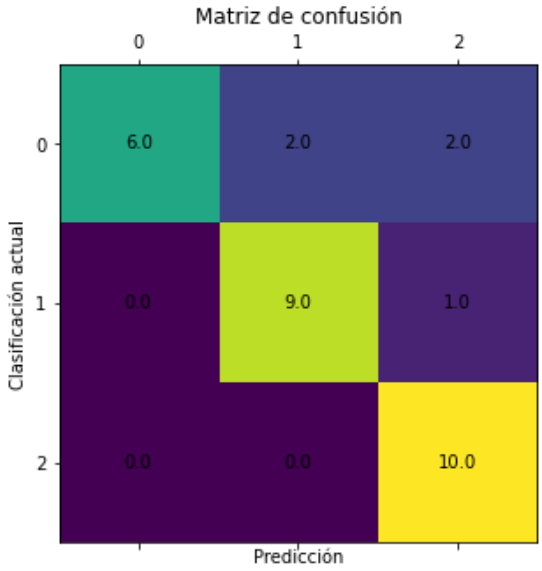

Figura 5-14. Matriz de confusión del cuarto modelo.

	precision	recall	f1-score	support
0	1.00	0.60	0.75	10
1	0.82	0.90	0.86	10
2	0.77	1.00	0.87	10

Código 5-27. Informe de las métricas del cuarto modelo.

Pero esta vez, los resultados son incluso peores. El problema es que la base de datos no ofrece la diversidad que se exige después en una clasificación real. Es decir, el entrenamiento se ha adaptado a este tipo de mascarillas ficticias y, por lo tanto, después no es capaz de diferenciar una mascarilla que difiera del único tipo con el que ha trabajado. De hecho, haciendo la misma predicción para las imágenes de la figura 5-8 anterior, se mantiene el mismo error que antes de la imagen cinco, pero, además, ahora no sabe clasificar las imágenes tres y seis, donde sí se lleva mascarilla.

```
In [260]: prediccion('./datos/prueba/imagenuno.jpg')
La imagen de entrada pertenece a la clase sin mascarilla

In [261]: prediccion('./datos/prueba/imagendos.jpg')
La imagen de entrada pertenece a la clase mascarilla incorrecta

In [262]: prediccion('./datos/prueba/imagentres.jpg')
La imagen de entrada pertenece a la clase mascarilla incorrecta

In [263]: prediccion('./datos/prueba/imagencuatro.jpg')
La imagen de entrada pertenece a la clase sin mascarilla

In [264]: prediccion('./datos/prueba/imagencinco.jpg')
La imagen de entrada pertenece a la clase mascarilla incorrecta

In [265]: prediccion('./datos/prueba/imagenseis.jpg')
La imagen de entrada pertenece a la clase sin mascarilla
```

Código 5-28. Predicciones resultantes de los datos del cuarto modelo.

Sin embargo, si las imágenes seleccionadas para probar son de personas mirando al frente, con una mascarilla quirúrgica blanca, sí que reconoce el patrón sin mayor problema. Como este no es el contexto que se desea, la red tiene menor calidad que antes y no interesa hacer este cambio en este proyecto en particular.

Figura 5-15. Imágenes de prueba con una mascarilla quirúrgica blanca.

```
In [266]: prediccion('./datos/prueba/pruebauno.jpg')
     ...: prediccion('./datos/prueba/pruebados.jpg')
     ...: prediccion('./datos/prueba/pruebatres.jpg')
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase con mascarilla
La imagen de entrada pertenece a la clase con mascarilla
```

Código 5-29. Predicciones de imágenes con mascarillas quirúrgicas blancas.

5.2. Redes pre-entrenadas. Ejemplo red *AlexNet*.

Cómo técnica alternativa, existen modelos ya preentrenados, los cuales han demostrado que actúan de forma adecuada, como Xception, diferentes versiones de VGG, ResNet o AlexNet entre otros; todos se aprecian en la figura 5-16 ordenados según su precisión (de izquierda a derecha). Se pueden usar modelos previamente entrenados para la extracción de características y a pesar de estar ya diseñados, se pueden incluso mejorar mediante un ajuste fino. Sin embargo, como en este trabajo no es necesaria una red excesivamente compleja y se tienen suficientes datos, se ha diseñado el modelo anterior desde cero. Aun así, se va a probar cómo funcionan para verificarlo y que no quede en una simple suposición.

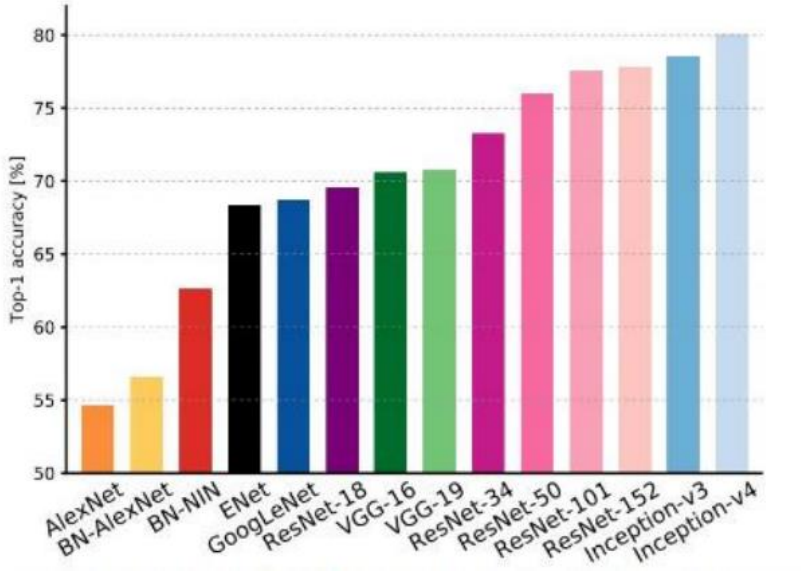

Figura 5-16. Redes preentrenadas de Keras según su precisión [37].

Se ha hecho una prueba con la red AlexNet, debido a que pesar de no ser la red más precisa, su implementación computacional es aceptable y resulta suficiente para probar que no es recomendable su empleo en esta ocasión. Otras CNNs de última generación son excesivamente complejas y no es posible ejecutarlas en este proyecto. La red AlexNet está conformada por ocho capas, cinco de ellas convolucionales, seguidas de métodos de agrupación máxima y funciones de activación ReLU, y el resto capas densas completamente conectadas. Es importante mencionar que sólo acepta imágenes de entrada de dimensiones 227x227, por lo que hay que normalizarlas obligatoriamente. No obstante, a pesar de que en el *dataset* empleado hay imágenes de dimensiones mucho menores, la precisión de la clasificación no se ve perjudicada debido al efecto del relleno con ceros, mientras que el tiempo de entrenamiento sí disminuye notablemente de forma proporcional a la cantidad añadida de ceros para alcanzar las dimensiones fijas [88]. Su estructura es la representada en la figura 5-17 [89] [90].

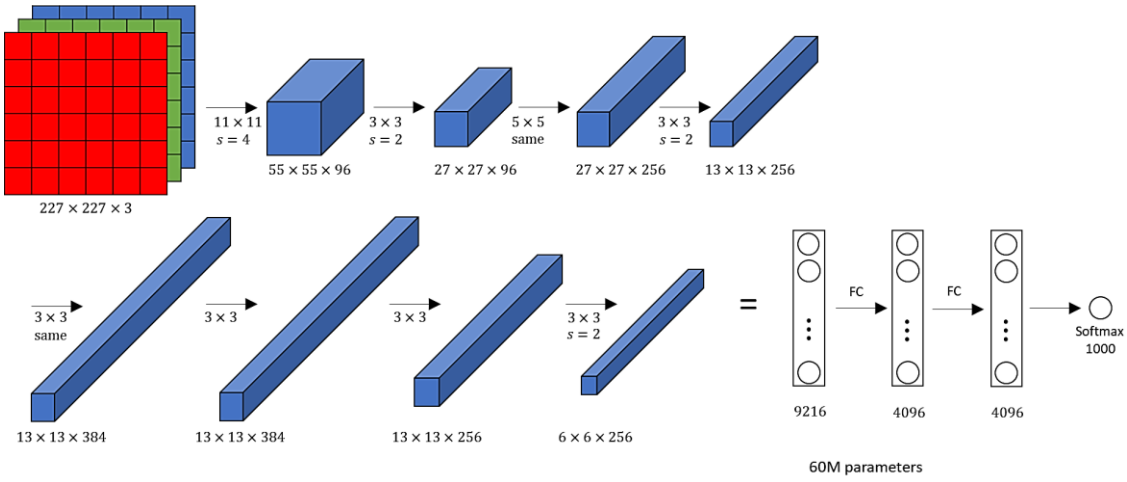

Figura 5-17. Estructura de la red AlexNet.

```
#Creación de la red neuronal convolucional
cnn = tensorflow.keras.models.Sequential([
    tensorflow.keras.layers.Conv2D(96, (11,11), strides=(4,4), activation='relu', input_shape=(227,227,3)),
    tensorflow.keras.layers.BatchNormalization(),
    tensorflow.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tensorflow.keras.layers.Conv2D(256, (5,5), strides=(1,1), activation='relu', padding="same"),
    tensorflow.keras.layers.BatchNormalization(),
    tensorflow.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tensorflow.keras.layers.Conv2D(384, (3,3), strides=(1,1), activation='relu', padding="same"),
    tensorflow.keras.layers.BatchNormalization(),
    tensorflow.keras.layers.Conv2D(384, (3,3), strides=(1,1), activation='relu', padding="same"),
    tensorflow.keras.layers.BatchNormalization(),
    tensorflow.keras.layers.Conv2D(256, (3,3), strides=(1,1), activation='relu', padding="same"),
    tensorflow.keras.layers.BatchNormalization(),
    tensorflow.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tensorflow.keras.layers.Flatten(),
    tensorflow.keras.layers.Dense(4096, activation='relu'),
    tensorflow.keras.layers.Dropout(0.5),
    tensorflow.keras.layers.Dense(4096, activation='relu'),
    tensorflow.keras.layers.Dropout(0.5),
    tensorflow.keras.layers.Dense(clases, activation='softmax')
    ])
```

Código 5-30. Estructura de la red AlexNet.

Las capas de *batch_normalization*, posteriores a las capas convolucionales y previas a las capas de *pooling*, normalizan la salida de las capas convolucionales con técnicas de escalado y desplazamiento adicionales. Gracias a ellas, una red poco profunda ofrece un mayor rendimiento y precisión que otras idénticas que carecen de normalización por cada *batch*. La normalización de lotes minimiza el cambio interno de variables, por lo que también el entrenamiento es más rápido. Se debe tener en cuenta que la técnica normalización por lotes funciona de manera distinta durante el entrenamiento y la prueba. En el entrenamiento, la media y la varianza se calculan independientemente para cada lote y, durante el test, se utilizan valores estadísticos promedios: la media móvil y la varianza de todas las imágenes vistas por la red. Por este motivo, en algunos experimentos la normalización por lotes no es sólo recomendable, si no altamente demandada para un correcto ajuste [91] [92].

El argumento de *stride* o zancada indica el número de pasos de píxeles que realiza el filtro cuando se mueve de una activación de imagen hacia otra, que sistemáticamente se hace de izquierda a derecha y de arriba abajo hasta completar un recorrido por toda la imagen. Es habitual utilizar una zancada unitaria, el valor por defecto, aunque se puede aumentar para imágenes de dimensiones mayores o configurar diferentes valores de la zancada para cada conjunto de píxeles. Se indica mediante una tupla de altura y anchura respectivamente, aunque lo normal es que ambos valores sean simétricos y se indique ese único valor [93]. En la figura 5-18 se compara el paso con *stride* unitario y con *stride* de valor dos y como consecuencia, en el segundo caso las dimensiones de salida del mapa de características se reducen a la mitad [94].

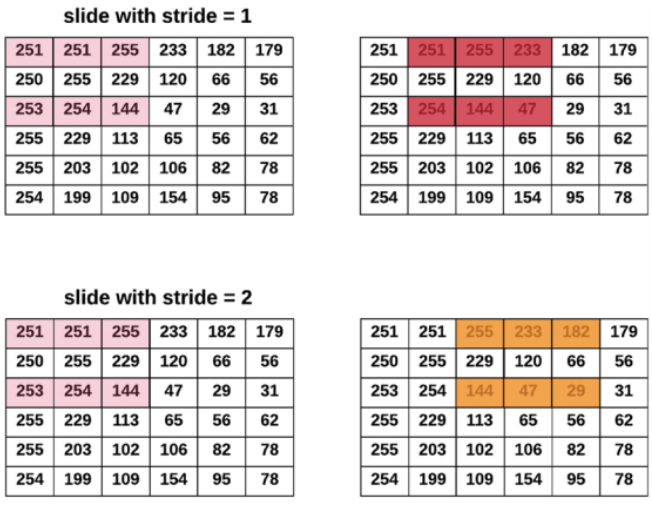

Figura 5-18. Comparación de *stride*=1 y *stride* =2 [95].

Tras codificar la estructura (véase código 5-30), las primeras y últimas épocas son las siguientes:

```
Epoch 1/50
16/16 [==============================] - 42s 3s/step - loss: 6.3401 - accuracy: 0.6746 - val_loss: 25.2207 - val_accuracy: 0.1719
Epoch 2/50
16/16 [==============================] - 36s 2s/step - loss: 3.2852 - accuracy: 0.8107 - val_loss: 8.0871 - val_accuracy: 0.4844
Epoch 3/50
16/16 [==============================] - 38s 2s/step - loss: 1.9415 - accuracy: 0.8676 - val_loss: 10.6846 - val_accuracy: 0.6094
Epoch 4/50
16/16 [==============================] - 43s 3s/step - loss: 1.6239 - accuracy: 0.8750 - val_loss: 10.6616 - val_accuracy: 0.4844
Epoch 5/50
16/16 [==============================] - 38s 2s/step - loss: 1.0673 - accuracy: 0.8934 - val_loss: 1.7949 - val_accuracy: 0.7188
Epoch 6/50
16/16 [==============================] - 40s 2s/step - loss: 1.6973 - accuracy: 0.8566 - val_loss: 4.4014 - val_accuracy: 0.5469
Epoch 7/50
16/16 [==============================] - 42s 3s/step - loss: 2.3139 - accuracy: 0.8860 - val_loss: 5.8998 - val_accuracy: 0.6094
Epoch 8/50
16/16 [==============================] - 42s 3s/step - loss: 0.8141 - accuracy: 0.8952 - val_loss: 4.6325 - val_accuracy: 0.5000
Epoch 9/50
16/16 [==============================] - 46s 3s/step - loss: 1.1611 - accuracy: 0.8971 - val_loss: 2.7920 - val_accuracy: 0.6875
Epoch 10/50
16/16 [==============================] - 45s 3s/step - loss: 1.6788 - accuracy: 0.8860 - val_loss: 3.4521 - val_accuracy: 0.6719
```

Código 5-31. 10 primeras épocas del modelo AlexNet.

```
Epoch 40/50
16/16 [==============================] - 43s 3s/step - loss: 0.1667 - accuracy: 0.9522 - val_loss: 0.0950 - val_accuracy: 0.9531
Epoch 41/50
16/16 [==============================] - 42s 3s/step - loss: 0.2634 - accuracy: 0.9596 - val_loss: 1.4195 - val_accuracy: 0.7500
Epoch 42/50
16/16 [==============================] - 42s 3s/step - loss: 0.3299 - accuracy: 0.9412 - val_loss: 1.3007 - val_accuracy: 0.8281
Epoch 43/50
16/16 [==============================] - 44s 3s/step - loss: 0.3599 - accuracy: 0.9485 - val_loss: 1.5865 - val_accuracy: 0.7188
Epoch 44/50
16/16 [==============================] - 44s 3s/step - loss: 0.3983 - accuracy: 0.9301 - val_loss: 0.5161 - val_accuracy: 0.8594
Epoch 45/50
16/16 [==============================] - 40s 2s/step - loss: 0.4075 - accuracy: 0.9301 - val_loss: 2.2855 - val_accuracy: 0.7656
Epoch 46/50
16/16 [==============================] - 40s 2s/step - loss: 0.2587 - accuracy: 0.9540 - val_loss: 1.3100 - val_accuracy: 0.8438
Epoch 47/50
16/16 [==============================] - 40s 2s/step - loss: 0.2826 - accuracy: 0.9504 - val_loss: 2.1141 - val_accuracy: 0.6719
Epoch 48/50
16/16 [==============================] - 40s 2s/step - loss: 0.1917 - accuracy: 0.9540 - val_loss: 1.1144 - val_accuracy: 0.7656
Epoch 49/50
16/16 [==============================] - 40s 2s/step - loss: 0.2394 - accuracy: 0.9669 - val_loss: 0.4193 - val_accuracy: 0.8594
Epoch 50/50
16/16 [==============================] - 40s 2s/step - loss: 0.1015 - accuracy: 0.9651 - val_loss: 1.0116 - val_accuracy: 0.7500
```

Código 5-32. 10 últimas épocas del modelo AlexNet.

Visualizando el resultado de los códigos 5-31 y 5-32, se verifica que las desventajas frente a la red empleada son múltiples: el tiempo consumido es mayor, la precisión de validación es ligeramente menor, las pérdidas de validación no alcanzan durante todo el entrenamiento los valores esperados y, además, existe un claro sobreajuste brindado por la complejidad de la red. En definitiva, este modelo de nuevo se aleja de lo esperado y no conviene su uso para los recursos existentes.

6 CONCLUSIONES

Durante el transcurso de este proyecto, que ha quedado reflejado en esta memoria, se han trabajado distintas fases del proyecto de implementación de algoritmos basados en aprendizaje profundo para la clasificación de imágenes basadas en las mascarillas faciales para prevenir el contagio de COVID-19.

Primero, se ha realizado una explicación teórica exhaustiva de los distintos conceptos que engloban las disciplinas de inteligencia artificial, aprendizaje máquina, aprendizaje profundo y redes neuronales. Después, se ha expuesto el entorno de trabajo y las herramientas requeridas, así como la propia implementación de la red neuronal convolucional de clasificación. Para desarrollar el trabajo, se han recopilado algunas de las transformaciones internas que ha sufrido la red para llegar a un buen ajuste de parámetros que de lugar a los resultados finales: las dimensiones de normalización, las capas, la técnica de abandono o la base de datos. Estos resultados se muestran mediante diversas predicciones y métricas derivadas de la matriz de confusión, obtenida al ofrecerle a la red nuevos datos de entrada distintos de los usados para entrenar.

En la tabla 6-1 se recoge de forma esquematizada los cambios hechos a la CNN diseñada y las consecuencias que han traído, para tener una comparación directa y visual de unos modelos con otros, acompañada de anotaciones que afectan en gran medida a la calidad del sistema.

	Primer Modelo	**Segundo Modelo**	**Tercer Modelo**	**Cuarto modelo**	**Modelo AlexNet**
Data Augmentation	Inclinación, zoom = **0.2**	Inclinación, zoom = **0.2**	Inclinación, zoom = **0.3**	Inclinación, zoom = **0.3**	Inclinación, zoom = **0.3**
Dimensiones	**64x64**	**64x64**	**64x64**	**64x64**	**227x227**
Capas convolucionales	Dos de **32** y **64** filtros de tamaño (3,3) y (2,2)	Dos de **32** y **64** filtros de tamaño (4,4) y (3,3)	Tres de **23**, **64** y **128** filtros de tamaño (3,3)	Tres de **32**, **64** y **128** filtros de tamaño (3,3)	Cinco de **96**, **256**, **384**, **384** y **256** filtros de tamaño (11,11), (5,5), (3,3), (3,3) y (3,3).
Capas densas	Una de **256** neuronas + *softmax*	Dos de **128** y **64** neuronas + *sofmax*	Cuatro de **256**, **128**, **64** y **32** neuronas + *softmax*	Cuatro de **256**, **128**, **64** y **32** neuronas + *softmax*	Dos de **4096** neuronas + *softmax*
Dropout	Una vez de valor **0.5**	Dos veces de valores **0.5**	Cuatro veces de valores **0.8**, **0.6**, **0.5** y **0.4**.	Cuatro veces de valores **0.8, 0.6, 0.5** y **0.4.**	Dos veces de valores **0.5**
Precisión máx. validación	**85.94%**	**93.57%**	**95.31%**	**100.00%**	**95.31%**
Pérdidas mín. validación	**52.54%**	**25.49%**	**22.31%**	**1.77%**	**9.5%**
Overfitting	Sí	Sí	No	No	Sí

Tabla 6-1. Distintos modelos de CNN del proyecto.

En conlcusión, se han obtenido una serie de ideas clave sobre el diseño y la implementación de las redes neuronales convolucionales. Primero, hay que aclarar que una red demasiado sencilla tiene una alta probabilidad de no adaptarse de forma adecuada a los datos y no predecir bien (primer modelo), pero esto no significa que el otro extremo sea el correcto: si la red es demasiado compleja, puede causar sobre ajuste, además de incrementar el coste computacional (AlexNet). En resumen, lo más adecuado es hacer pruebas y mejoras progresivas, estudiar los resultados, revisar la base de datos e ir solventando todos los conflictos internos del modelo, como se ha hecho mediante las modificaciones del segundo y del tercer modelo, tras investigar y leer informes que han sido el soporte de todas estas alteraciones.

La base de datos tiene un papel vital y puede provocar confusiones a la hora de interpretar la calidad del sistema. Al hacer el cambio en la base de datos (cuarto modelo) se obtiene la mejor precisión y pérdidas de validación, pero realmente es inútil de cara a las diversas predicciones que se desean. Un buen *dataset* debe ser variado y albergar todas las categorías de la clasificación en su justa medida, sin cargarlo con datos redundantes o innecesarios. Como recomendación, para funciones sencillas es mejor usar imágenes que pesen poco, como referencia es bueno no sobrepasar 500 kB; todo aquello que ayude a tener un modelo liviano y poco pesado, siempre que no afecte a la precisión, es positivo.

6.1. Líneas futuras de investigación.

En este último apartado, se van a nombrar e introducir ciertos mecanismos y técnicas que pueden servir de mejora de este proyecto en un futuro, a pesar de haber quedado excluidos del alcance de este o no haber alcanzado un desarrollo profundo.

Una limitación de este estudio es que sólo se trabaja con una red preentrenada, AlexNet, debido a que es posible su implementación con los recursos existentes en este proyecto por su baja carga computacional. Como líneas futuras de investigación se recomienda, entrenar con otras redes ya diseñadas más complejas que se escapan del alcance de este trabajo, ya que alguna se puede adecuar a los resultados que se buscan (véase la figura 5-16).

Otra tarea futura para futuros estudios es un análisis más detallado de la matriz de confusión y sus métricas derivadas. Desde un punto de vista experto, se le puede sacar un enorme rendimiento a esta herramienta, de la cual sólo se ha visto una introducción, tanto teórica como práctica, pero tiene mucho más que ofrecer y exprimir de ella.

Finalmente, lo más característico de una red neuronal es su arquitectura y el ajuste de parámetros. Se ha mencionado la existencia de novedosos parámetros adaptativos que van evolucionando junto con la red para sacarle el máximo potencial. Algunos de ellos son la tasa de aprendizaje, el *stride* o el *batch* adaptativo, y sería muy interesante trabajar con ellos en un futuro.

REFERENCIAS

[1] Pérez, AMR et al. (2020) *Características clínico-epidemiológicas de la COVID 19.* Revista Habanera de Ciencias Médicas ;19(2):1-15.

[2] Caicoya, M. (2020). *El papel de las mascarillas en el control de la epidemia COVID-19.* J Healthc Qual Res. ;35(4):203-205.

[3] Arellano, C et al. (2020). *Eficacia de las mascarillas para uso odontológico en la prevención del Covid-19*. Una revisión de literatura.

[4] Deng, L (2018). *Artificial Intelligence in the Rising Wave of Deep Learning: The Historical Path and Future Outlook [Perspectives]*. IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 180-177.

[5] El Naqa I. y Murphy MJ. (2015). *¿Qué es el aprendizaje automático?* Machine Learning in Radiation Oncology. Springer, Cham.

[6] Khanzode, Ku Chhaya A y Ravindra D. Sarode. (2020). *Advantages and Disadvantages of Artificial Intelligence and Machine Learning: A Literature Review*. International Journal of Library & Information Science (IJLIS).

[7] López, R et al. (2021). *Inteligencia Artificial y Machine Learning en trastornos del movimiento*. MANUAL SEN 2021.

[8] Dalal, K.R. (2020). *Analysing the Role of Supervised and Unsupervised Machine Learning in IoT*. International Conference on Electronics and Sustainable Communication Systems (ICESC).

[9] Tian, Y. and Compere M. (2019). *A Case Study on Visual-Inertial Odometry using Supervised, Semi-Supervised and Unsupervised Learning Methods*. 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), pp. 203-2034.

[10] Oquab, M. et al. (2015). *Is object localization for free? - weakly-supervised learning with convolutional neural networks.* Proceedings of the IEEE conference on computer vision and pattern recognition.

[11] Ren, H. et al. (2020). *Stereo Disparity Estimation via Joint Supervised, Unsupervised, and Weakly Supervised Learning,* 2020 IEEE International Conference on Image Processing (ICIP), pp. 2760-2764-

[12] Reese, H. (2017). *Understanding the differences between AI, machine learning, and deep learning*. URL: https://www. techrepublic. com/article/understandingthedifferencesbetweenaimachine learninganddeeplearning

[13] Lauzon, FQ. (2012) *An introduction to deep learning*. 2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA). IEEE.

[14] Ghazimirsaeed, S.M et al. (2020). *Accelerating GPU-based Machine Learning in Python using MPI Library: A Case Study with MVAPICH2-GDR*. IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S).

[15] Ketkar, N. (2017) *Introduction to Deep Learning. In: Deep Learning with Python*. Apress, Berkeley, CA.

[16] Illingworth, W.T. (1989). *Beginner's guide to neural networks.* IEEE Aerospace and Electronic Systems Magazine, vol. 4, no. 9, pp. 44-49.

[17] Aldabas-Rubira, E. (2002). *Introducción al reconocimiento de patrones mediante redes neuronales*. IX Jornades de Conferències d'Enginyeria Electrònica.

[18] Tandeitnik, P. and Guterman, H (1996). *Modelling of biological neurons by artificial neural networks*, Proceedings of 19th Convention of Electrical and Electronics Engineers in Israel, pp. 239-242.

[19] Steck, J.E. and Dalton J.S. (1992). *Modeling the neurodynamics of a biological neuron using a feedforward*

artificial neural network. IJCNN International Joint Conference on Neural Networks.

[20] Tablada, C.J y Torres G.A. (2009). *Redes neuronales artificiales*. Revista de educación matemática 24.3.

[21] Santillán, D. et al. (2014). *Predicción de lecturas de aforos de filtraciones de presas bóveda mediante redes neuronales artificiales.* Tecnología y ciencias del agua.

[22] Gallant, S.I. (1990). *Perceptron-based learning algorithms*. IEEE Transactions on neural networks. 179-191.

[23] Gong, D. et al. (2020). *Learning Deep Gradient Descent Optimization for Image Deconvolution*. IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 12, pp. 5468-5482.

[24] Moreno, A. (1994). *Aprendizaje automático*. Universidad Politécnica de Cataluña, Edicions UPC.

[25] Zeiler, M.D. (2012). *An adaptive learning rate method*. Adadelta.

[26] Hirasawa, K. et al. (1996). *Forward propagation universal learning network*. Proceedings of International Conference on Neural Networks (ICNN'96). Vol. 1. IEEE.

[27] Han, X. et al. (2016). *An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks*. IEEE 34th International Conference on Computer Design (ICCD).

[28] Liu, R. et al. (2019). *Learning Converged Propagations With Deep Prior Ensemble for Image Enhancement*. IEEE Transactions on Image Processing, vol. 28, no. 3, pp. 1528-1543.

[29] Fooladi, N. et al. (2014). *Permeability Prediction of an Iranian Reservoir Using Hybrid Neural Genetic Algorithm.*

[30] Torres, L. G. (1994). *Redes neuronales y aproximación de funciones.* Bol. Matemáticas, 1, 35-58.

[31] Sharma, A.V. (2017). *Understanding Activation Functions in Neural Meteros*. Medium.

[32] Gao, Y.W.L. y Lombardi, F. (2020). *Design and Implementation of an Approximate Softmax Layer for Deep Neural Networks*. 2020 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE.

[33] Kouretas, I. y Vassilis, P. (2019). *Simplified hardware implementation of the softmax activation function*. 8th international conference on modern circuits and systems technologies (MOCAST). IEEE.

[34] Nwankpa, C. et al. (2018). *Activation functions: Comparison of trends in practice and research for deep learning*. arXiv preprint.

[35] Setiono, R. y Liu, H. (1997). *Neural-network feature selector*. IEEE Transactions on Neural Networks, vol. 8, no. 3, pp. 654-662.

[36] Sandoval Serrano, L.J. (2018). *Algoritmos de aprendizaje automático para análisis y predicción datos*. Revista Tecnológica; no. 11.

[37] Artola Moreno, Á. (2019). *Clasificación de imágenes usando redes neuronales convolucionales en Python*. (Trabajo Fin de Grado). Universidad de Sevilla, Sevilla.

[38] Bullinaria, J.A. (2015). *Bias and variance, under-fitting and over-fitting*. Neural Computation: Lecture 9.

[39] Koehrsen, W. (2018). *Overfitting vs. underfitting: A complete example*. Towards Data Science.

[40] Smith, L.N. (2018). *A disciplined approach to neural network hyper-parameters: Part 1--learning rate, batch size, momentum, and weight decay.* arXiv preprint.

[41] Gupta, A. (2018). *Introduction to deep learning: part 1.* Chemical Engineering Progress 114.6, 22-29.

[42] Albawi S. et al. (2017). *Understanding of a convolutional neural network*. 2017 International Conference on Engineering and Technology (ICET), pp. 1-6.

[43] Aloysius, N. y Geetha, M. (2017). *A review on deep convolutional neural networks*. 2017 International Conference on Communication and Signal Processing (ICCSP). IEEE.

[44] Pizarro, J. T. (2019). *Detección y clasificación de diferentes formas eritrocitarias anómalas mediante redes neuronales profundas* (Trabajo Fina de Grado). UPC, Escola d'Enginyeria de Barcelona.

[45] Mairal, J. et al. (2014). *Convolutional kernel networks*. Advances in neural information processing systems 27.

[46] Cifuentes, A. et al. (2019). *Desarrollo de una red neuronal convolucional para reconocer patrones en imágenes.* Investigación y desarrollo en TIC 10.2.

[47] Nagi, J. et al. (2011). *Max-pooling convolutional neural networks for vision-based hand gesture recognition*" 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA).

[48] Masci, J. et al. (2012). *Steel defect classification with Max-Pooling Convolutional Neural Networks.* The 2012 International Joint Conference on Neural Networks (IJCNN).

[49] Wu, H. y Xiaodong G. (2015). *Max-pooling dropout for regularization of convolutional neural networks.* International Conference on Neural Information Processing. Springer.

[50] Yani, M.T. (2019*). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail.* Journal of Physics*:* Conference Series.

[51] Dileep, D. y Bora P.K. (2020). *Dense Layer Dropout Based CNN Architecture for Automatic Modulation Classification.* 2020 National Conference on Communications (NCC).

[52] Srivastava, N. (2013). *Improving neural networks with dropout*. University of Toronto 182.566: 7.

[53] Sanjar, K. et al. (2020). *Weight Dropout for Preventing Neural Networks from Overfitting*. 8th International Conference on Orange Technology (ICOT).

[54] Naskar, A. et al. (2019). *Convolutional Neural Networks*. 10.13140/RG.2.2.23743.05282.

[55] Rolon-Mérette, D. et al. (2016). *Introduction to Anaconda and Python: Installation and setup.*

[56] Lee, W.M. (2019). *Python machine learning.* John Wiley & Sons.

[57] PopularitY of Programming Language: https://pypl.github.io/PYPL.html [último acceso: septiembre 2021].

[58] Librería OS: https://docs.python.org/es/3/tutorial/stdlib.html [último acceso: septiembre 2021].

[59] Librería SYS: https://docs.python.org/es/3.10/library/sys.html [último acceso: septiembre 2021].

[60] Librería NUMPY: https://numpy.org/ [último acceso: septiembre 2021].

[61] Singhla, R et al. (2021). *Image Classification Using Tensor Flow.* 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS).

[62] Scikit-Learn: https://scikit-learn.org/stable/ [último acceso: septiembre 2021].

[63] Kaggle: https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset [último acceso: agosto 2021].

[64] Yadav, S. y Shukla, S. (2016). *Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification.* 2016 IEEE 6th International Conference on Advanced Computing (IACC).

[65] Probst, P. et al. (2019). *Tunability: importance of hyperparameters of machine learning algorithms*. The Journal of Machine Learning Research 20.1.

[66] Claesen, M. y De Moor, B. (2015). *Hyperparameter search in machine learning*. arXiv preprint arXiv:1502.02127.

[67] Kim, Y. y Chung, M. (2019). *An Approach to Hyperparameter Optimization for the Objective Function in Machine Learning. Electronics*. https://doi.org/10.3390/electronics8111267

[68] Brownlee, J. (2018). *What is the Difference Between a Batch and an Epoch in a Neural Network?* Machine Learning Mastery 20.

[69] Devarakonda, A et al. (2017). *Adabatch: Adaptive batch sizes for training deep neural networks*. arXiv preprint arXiv:1712.02029.

[70] Shorten, C. and Taghi M.K. (2019). *A survey on image data augmentation for deep learning*. Journal of Big Data 6.1.

[71] Perez, L. y Wang, J. (2017). *The effectiveness of data augmentation in image classification using deep learning*. arXiv preprint arXiv:1712.04621.

[72] Bowden, G.J. et al. (2002). *Optimal division of data for neural network models in water resources applications*. Water Resources Research 38.2.

[73] Manry T.M. et al. (2013). *Minimizing validation error with respect to network size and number of training epochs*. The 2013 international joint conference on neural networks (IJCNN). IEEE.

[74] Lai, K.T., (2020). *Convolutional Neural Network*."

[75] Cheng, H.J.H. (2020). *Empirical Study on the Effect of Zero-Padding in Text Classification with CNN.* University of California, Los Angeles,

[76] Naufal, M.F. et al. (2021). *Comparative Analysis of Image Classification Algorithms for Face Mask Detection.* Journal of Information Systems Engineering and Business Intelligence 7.1.

[77] Terliksiz, A.S. y D. Turgay, A. (2019). *Use of deep neural networks for crop yield prediction: A case study of soybean yield in lauderdale county, Alabama, USA*." 2019 8th International Conference on Agro-Geoinformatics (Agro-Geoinformatics). IEEE.

[78] Ahmed, W.S. (2020). *The impact of filter size and number of filters on classification accuracy in cnn*. 2020 International conference on computer science and software engineering (CSASE). IEEE.

[79] Ahmed, W.S. (2020). *The impact of filter size and number of filters on classification accuracy in cnn*. 2020 International conference on computer science and software engineering (CSASE). IEEE.

[80] Srivastava, Ni. et al. (2014). *Dropout: a simple way to prevent neural networks from overfitting*. The journal of machine learning research 15.1.

[81] Jeny, J.R.V. (2021). *Deep Learning Framework for Face Mask Detection,* 5th International Conference on Trends in Electronics and Informatics (ICOEI).

[82] Beauxis-Aussalet, E. y Hardman, L. (2014). *Visualization of confusion matrix for non-expert users*. IEEE Conference on Visual Analytics Science and Technology (VAST)-Poster Proceedings.

[83] Luque, A. et al. (2019). *The impact of class imbalance in classification performance metrics based on the binary confusion matrix.* Pattern Recognition 91.

[84] Tatbul, N. et al. (2018). *Precision and recall for time series."* arXiv preprint arXiv:1803.03639.

[85] Beauxis-Aussalet, E. y Hardman, L. (2014). *Simplifying the visualization of confusion matrix.* 26th Benelux Conference on Artificial Intelligence (BNAIC).

[86] Yacouby, R. y Axman, D. (2020), *Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models.* Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems.

[87] Kaggle: https://www.kaggle.com/prasoonkottarathil/face-mask-lite-dataset [último acceso: septiembre 2021].

[88] Agarwal, A. et al. (2021). *Lung Cancer Detection and Classification Based on Alexnet CNN*. 2021 6th International Conference on Communication and Electronics Systems (ICCES).

[89] Minhas, R.A., et al. (2019). *Shot classification of field sports videos using AlexNet Convolutional Neural Network.* Applied Sciences 9.3.

[90] Gong, W. et al. (2019). *Palmprint Recognition Based on Convolutional Neural Network-Alexnet.* 2019 Federated Conference on Computer Science and Information Systems (FedCSIS).

[91] Thakkar, V. et al. (2018). *Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data.* 2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT).

[92] Hasani, M. y Khotanlou, H. (2019). *An Empirical Study on Position of the Batch Normalization Layer in Convolutional Neural Networks*. 2019 5th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS).

[93] Kong, C. y Lucey, S. (2017). Take it in your stride: ¿Do we need striding in CNNs? arXiv preprint arXiv:1712.02502.

[94] Keras API Layers Reference: https://keras.io/api/layers/convolution_layers/convolution2d/ [último acceso: septiembre 2021].

[95] Bisong, E. (2019). Convolutional Neural Networks (CNN). Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA