Reconocimiento de imágenes médicas mediante aprendizaje automático

Daniel Valverde Menasalvas

UNIVERSIDAD COMPLUTENSE MADRID

Trabajo de Fin de Grado
Grado en Matemáticas

Dirigido por
Ana Carpio

Medical image recognition with supervised learning

Madrid, 2020-2021

Reconocimiento de imágenes médicas mediante aprendizaje automático

Memoria que se presenta para el Trabajo de Fin de Grado de

Daniel Valverde Menasalvas

Dirigido por

Ana Carpio

Departamento de Análisis Matemático y Matemática Aplicada
Facultad de Matemáticas
Universidad Complutense de Madrid

Madrid, 2021

Resumen

Este proyecto trata de aplicar técnicas de deep learning para solucionar un problema de categorización con múltiples etiquetas en imágenes médicas. En concreto, se utiliza un corpus de radiografías de pulmón con cuatro etiquetas: Normal, COVID-19, Opacidad de Pulmón y Neumonía Viral.

El trabajo consta de un estudio inicial de las técnicas de reconocimiento de imagen usando deep learning, tratando los tipos de redes neuronales más comunes y los conceptos claves en el entrenamiento de estas. También se tratan conceptos más avanzados de las redes convolucionales que permitirán mejorar algunos puntos el rendimiento del modelo. Además, se realiza una experimentación sencilla con redes neuronales que identifican letras a modo de introducción a la tecnología.

Finalmente, se proponen una serie de modelos para solventar el problema, se entrenan y se evalúan los resultados. Se ha llegado a conseguir una exactitud de *92,91 %* en el conjunto de validación, lo cual es un resultado bastante satisfactorio, teniendo en cuenta el problema planteado.

Palabras clave: Deep Learning, Redes Neuronales, Computer Vision, Redes Convolucionales, CNN, Reconocimiento de Imagen Médica, COVID-19.

Abstract

The purpose of this project is to apply deep learning techniques to solve a multi label classification problem with medical images. For that, a corpus of lung radiographies with four labels (Normal, COVID-19, Lung Opacity and Viral Pneumonia) will be used.

The initial part of the project will consist in an study of the image recognition deep learning techniques. Most common types of neural networks will be explained as well as the fundamental concepts of their training. More advanced topics will also be treated such as separable convolutional layers or augmentation and batch normalization techniques. After that, a letter recognition problem will be solved using convolutional neural networks to get familiarized with the technology.

Finally, several models will be proposed and trained for the medical image problems, following with an evaluation of the results. An accuracy of *92.91 %* was acquired in the validation set, which is definitely a good result taking into account the proposed problem.

Key words: Deep Learning, Neural Networks, Computer Vision, Convolutional Neural Networks, CNN, Medical Image Recognition, COVID-19.

Índice general

Página

1. Introducción y objetivos **1**
1.1. Objetivos 1

2. Deep learning y redes neuronales **3**
2.1. Deep learning 3
2.1.1. Inteligencia artificial y machine learning 3
2.1.2. Deep learning y redes neuronales 4
2.1.3. Las cuatro ramas del machine learning 7
2.1.4. ¿Qué ha conseguido y qué podemos esperar del deep learning? . . 8
2.2. Conceptos fundamentales de Machine Learning 9
2.2.1. Preprocesamiento de los datos 9
2.2.2. Sobreajuste y desajuste 10
2.2.3. Evaluando modelos de Machine Learning 11
2.2.4. Métricas comunes en deep learning 13
2.2.5. La transformación más básica: la capa densa 14
2.2.6. Función de activación, función de pérdida y optimizador 15
2.2.7. Experimentación con redes neuronales 15
2.3. Redes recurrentes 16
2.3.1. Redes de memoria a largo-corto plazo (LSTM) y redes con unidades cerradas recurrentes(GRU) 17
2.4. Redes convolucionales(CNN) 18
2.4.1. La operación de convolución 18
2.4.2. La operación de agrupación máxima 20
2.4.3. Aumento de datos 21
2.4.4. Normalización por lotes 22
2.4.5. Convolución separable por profundidad 23
2.5. Redes generativas antagónicas (GANs) 23

3. Reconocimiento de letras/dígitos **26**
3.1. Problema y datos de entrenamiento 26
3.2. Arquitectura propuesta 27
3.3. Resultados 28

4. Reconocimiento de imágenes **31**
4.1. Problema y datos de entrenamiento 31
4.2. Arquitecturas propuestas y resultados 32

4.2.1. Modelos convolucionales clásicos 33
4.2.2. Modelos convolucionales separables 36
4.2.3. Modelos convolucionales separables con mayor clasificador 40

5. Conclusiones **45**

6. Bibliografía y enlaces de referencia **47**

Índice de figuras

2.1. Deep learning enmarcado en el campo de la inteligencia artificial. 3
2.2. Cambio de enfoque entre la inteligencia artificial simbólica y el machine learning. 4
2.3. Representaciones por capas en el deep learning. 5
2.4. Algoritmo de propagación hacia atrás. 6
2.5. Campos históricamente complicados para el machine learning donde el deep learning ha conseguido avances muy considerables 8
2.6. Transformación de una imagen en un tensor. 10
2.7. Explicación gráfica de la técnica del abandono en deep learning. 11
2.8. Técnica de la validación con K particiones. 12
2.9. Matriz de confusión del ejemplo de detección de gatos en imágenes. . . . 13
2.10. Funciones de activación. 15
2.11. Funciones de activación para la última capa y funciones de pérdida según el tipo de problema. 16
2.12. Funcionamiento de una RNN. 17
2.13. Funcionamiento de una red LSTM. 17
2.14. Patrones locales reconocidos por una CNN. 18
2.15. Descomposición de una imagen en patrones locales. 19
2.16. Ejemplo del efecto de un filtro sobre una imagen y el mapa de características resultante. 19
2.17. Efecto de una convolución sobre un mapa de características de entrada. . 20
2.18. Funcionamiento de la capa de agrupación máxima. 21
2.19. Imagen de un gato a la que se le ha aplicado la técnica del aumento de datos. 22
2.20. Estructura de una convolución separable por profundidad. 23
2.21. Funcionamiento de una red generativa antagónica. 24
2.22. Imágenes sintéticas generadas por una red generativa antagónica. 25

3.1. Distribución del conjunto de imágenes de letras. 26
3.2. Muestra de las imágenes de dígitos. 27
3.3. Resumen del modelo 1 28
3.4. Resumen del modelo 2 28
3.5. Resumen del modelo 3 28
3.6. Resumen del modelo 4 28

4.1. Algunas imágenes del conjunto de datos tras realizar las transformaciones 32
4.2. Arquitectura del modelo 2Conv_1Dense. 34
4.3. Arquitectura del modelo 3Conv_1Dense 34
4.4. Arquitectura del modelo 4Conv_1Dense 34

4.5. Exactitud en entrenamiento (rojo) y validación (azul) del modelo 2Conv_1Dense 35
4.6. Matriz de confusión del modelo 2Conv_1Dense 35
4.7. Exactitud en entrenamiento (naranja) y validación (azul) del modelo 3Conv_1Dense 35
4.8. Matriz de confusión del modelo 3Conv_1Dense 35
4.9. Exactitud en entrenamiento (gris) y validación (naranja) del modelo 4Conv_1Dense 36
4.10. Matriz de confusión del modelo 4Conv_1Dense 36
4.11. Arquitectura del modelo 2ConvSep_1Dense. 38
4.12. Arquitectura del modelo 3ConvSep_1Dense. 38
4.13. Arquitectura del modelo 4ConvSep_1Dense. 38
4.14. Exactitud en entrenamiento (rosa) y validación (verde) del modelo 2ConvSep_1Dense 39
4.15. Matriz de confusión del modelo 2ConvSep_1Dense 39
4.16. Exactitud en entrenamiento (azul) y validación (rosa) del modelo 3ConvSep_1Dense 39
4.17. Matriz de confusión del modelo 3ConvSep_1Dense 39
4.18. Exactitud en entrenamiento (naranja) y validación (azul) del modelo 4ConvSep_1Dense 40
4.19. Matriz de confusión del modelo 4ConvSep_1Dense 40
4.20. Arquitectura del modelo 3ConvSep_2Dense. 41
4.21. Arquitectura del modelo 3ConvSep_3Dense. 41
4.22. Arquitectura del modelo 4ConvSep_2Dense. 41
4.23. Exactitud en entrenamiento (rosa) y validación (verde) del modelo 3ConvSep_2Dense 42
4.24. Matriz de confusión del modelo 3ConvSep_2Dense 42
4.25. Exactitud en entrenamiento (azul) y validación (rosa) del modelo 3ConvSep_3Dense 43
4.26. Matriz de confusión del modelo 3ConvSep_3Dense 43
4.27. Exactitud en entrenamiento (naranja) y validación (azul) del modelo 4ConvSep_2Dense 43
4.28. Matriz de confusión del modelo 4ConvSep_2Dense 43
4.29. Exactitud en entrenamiento (gris) y validación (naranja) del modelo 4ConvSep_3Dense 44
4.30. Matriz de confusión del modelo 4ConvSep_3Dense 44

Índice de tablas

3.1. Resultados de los experimentos planteados con los cuatro modelos iniciales. 29
3.2. Resultados de aumentar el tamaño de las capas en los modelos 1 y 3. . . 29
3.3. Resultados tras añadir abandono al modelo 1. 29

4.1. Distribución de las imágenes de radiografías de pulmón. 31
4.2. Valor numérico que usaremos para balancear cada clase en el conjunto de datos. 31
4.3. Exactitud en validación de los modelos convolucionales. 34
4.4. Exactitud en validación de los modelos convolucionales y los convolucionales separables. 37
4.5. Exactitud de los modelos con tres y cuatro bloques convolucionales variando el número de bloques densos. 42

Capítulo 1

Introducción y objetivos

Las técnicas de inteligencia artificial han cobrado gran popularidad en los últimos años debido al amplio abanico de problemas que permiten resolver. En concreto, el machine learning, y más concretamente el deep learning, están consiguiendo excelentes resultados en problemas que hace unas décadas se pensaban fuera del alcance de un ordenador. Algunos ejemplos son el procesamiento de lenguaje natural, la conducción autónoma o el reconocimiento de imágenes.

Inicialmente, esta tecnología estaba al alcance de muy pocos debido a su alta complejidad tanto teórica como técnica. Sin embargo, el surgimiento de herramientas como Tensorflow o PyTorch ha hecho que cualquier persona con conocimientos básicos de programación pueda diseñar y entrenar sus propios modelos, ya que nos proporcionan interfaces razonablemente sencillas donde solo tenemos que elegir los parámetros de nuestros modelos para poder entrenarlos.

Además, el auge de las tecnologías de computación en la nube como Amazon Web Services o Google Colab ha facilitado aún más el proceso. Estas, nos liberan de las restricciones que pueda tener el hardware utilizado permitiendo alquilar hardware donde entrenar nuestros modelos a precios muy asequibles. De esta manera, abaratamos costes y no tenemos que tener un ordenador dedicado completamente al proceso de entrenamiento.

En este trabajo se proponen dos problemas de clasificación de imágenes con múltiples etiquetas. El primero será identificación de letras y el segundo tratará de la clasificación de radiografías de pulmón identificando diferentes patologías. Para ello, se utilizarán modelos de redes neuronales convolucionales, experimentando con diferentes arquitecturas y utilizando técnicas de optimización para conseguir los mejores resultados posibles.

1.1. Objetivos

Los objetivos del trabajo son los siguientes:

- Investigar sobre el deep learning en el contexto del machine learning adquiriendo conocimientos sobre los problemas que ha conseguido solucionar y su potencial para el futuro.
- Adquirir conocimiento sobre los conceptos técnicos presentes en modelos de deep learning.

- Investigar sobre los tipos de modelos de deep learning más populares: redes convolucionales, redes recurrentes y redes antagónicas.
- Resolver un problema de reconocimiento de letras en imágenes como iniciación a la tecnología.
- Resolver un problema de clasificación de radiografías de pulmón según patologías como muestra del potencial de la tecnología.

Capítulo 2

Deep learning y redes neuronales

2.1. Deep learning

Antes de comenzar a hablar de deep learning y redes neuronales, es pertinente definir los conceptos de inteligencia artificial y machine learning dentro de la ciencia de la computación.

2.1.1. Inteligencia artificial y machine learning

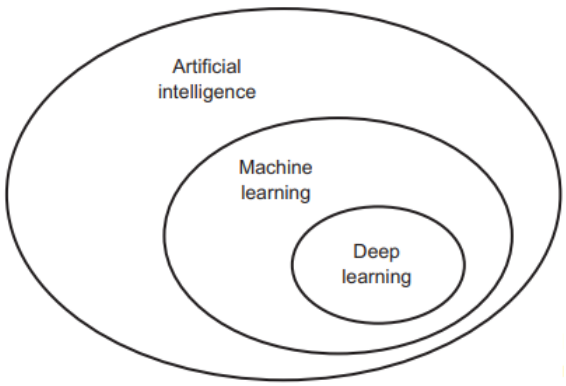

Figura 2.1: Deep learning enmarcado en el campo de la inteligencia artificial.

La **inteligencia artificial(AI)** se puede definir como *el esfuerzo para automatizar tareas intelectuales normalmente realizadas por humanos*. Esta definición es tremendamente genérica y engloba diversos campos como vemos en la figura 2.1. Por ejemplo, los primeros programas para jugar al ajedrez solo contenían reglas definidas manualmente por programadores, por lo que no se consideraban machine learning. Esto se conoce como enfoque simbólico y fue el paradigma predominante en la AI desde la década de los 50 hasta finales de la década de los 80. Este enfoque era válido para problemas lógicos bien definidos como jugar al ajedrez. Sin embargo, resultó ser poco adecuado a la hora de resolver problemas con definiciones más confusas como la clasificación de imágenes, interpretar audio o traducir textos. Por ello, surgió un nuevo enfoque para sustituir la inteligencia artificial simbólica, el machine learning.

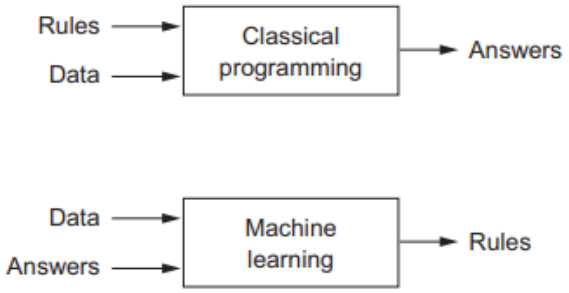

Figura 2.2: Cambio de enfoque entre la inteligencia artificial simbólica y el machine learning.

El **machine learning(ML)** surge de preguntarnos si un ordenador puede ir más allá de lo que nosotros le programamos y aprender a realizar una tarea por si solo. Esta pregunta abre la puerta a un paradigma de programación completamente nuevo. Un sistema de machine learning es entrenado en vez de programado. Se le presenta un conjunto de ejemplos relevantes para resolver una tarea y encuentra una estructura estadística que finalmente le permite automatizar dicha tarea. En la inteligencia artificial simbólica proporcionábamos normas y datos a un ordenador esperando respuestas; ahora proporcionamos datos y respuestas a un ordenador esperando nuevas normas que puedan ser utilizadas sobre datos nuevos para generar respuesta totalmente originales (ver figura 2.2).

Aunque el machine learning empezó a florecer en la década de los 90, se ha convertido rápidamente en el subcampo más exitoso de la AI, debido a la disponibilidad de hardware más veloz y conjuntos de datos de mayor tamaño.

La inteligencia artificial está muy relacionada con la estadística matemática pero tiene algunas diferencias significativas. A diferencia de la estadística, el machine learning trabaja con conjuntos de datos grandes y complejos (como un conjunto de millones de imágenes cada una compuesta por decenas de miles de píxeles), para los cuales los análisis estadísticos clásicos como el Bayesiano serían poco prácticos. Por consiguiente, el machine learning, y especialmente el deep learning, contiene poca teoría matemática. Es una disciplina práctica donde las ideas suelen probarse empíricamente más que de forma teórica.

2.1.2. Deep learning y redes neuronales

Para explicar que es el **deep learning(DL)**[1][2], primero hablaremos de learning (aprendizaje) y luego de deep (profundo). Para utilizar un algoritmo de machine learning necesitamos tres cosas:

- Datos de entrada: En una tarea de clasificación de imágenes serían imágenes.
- Ejemplos de datos de salida: En una tarea de clasificación de imágenes serían etiquetas, por ejemplo 'perro', 'gato'...
- Una métrica para determinar el desempeño del algoritmo (*función de pérdida o función objetivo*): Esta métrica debe medir la distancia entre la salida actual y la salida esperada y se usa como feedback para ajustar la forma en la que el algoritmo trabaja. Este ajuste es lo que llamamos learning (aprendizaje).

A la hora de transformar la entrada en la salida, será clave la representación que el algoritmo haga de los datos de entrada. Por ejemplo, una imagen se puede representar en formato RGB (rojo, verde, azul por sus siglas en inglés) o en formato HSV (matiz, saturación, valor por siglas en inglés). El formato RGB será mucho más adecuado para la tarea de «seleccionar todos los píxeles rojos en una imagen» mientras que el formato HSV será más adecuado para la tarea de «reducir la saturación de la imagen».

Todos los algoritmos de machine learning consisten en encontrar de forma automática esas representaciones más útiles para cada tarea. Cuando trabajemos con vectores de números (cosa que sucede en prácticamente todas las tareas en deep learning), estas operaciones pueden ser cambios de coordenadas, proyecciones lineales, traslaciones, operaciones no lineales, etc. Normalmente, los algoritmos de ML no son originales a la hora de encontrar esas transformaciones, solo prueban dentro de un conjunto predeterminado al que denominamos *espacio de hipótesis*.

Por lo tanto, a nivel técnico, el machine learning consiste en buscar representaciones útiles de unos datos de entrada, dentro de un espacio de posibilidades, usando como guía una señal de feedback. Este enfoque tan simple nos permite resolver un número de tareas complejas, desde transcribir un discurso a conducir un coche de forma automática. Ahora que ya entendemos a que nos referimos con learning, vamos a ver que hace al deep learning especial.

El deep learning es un subcampo específico del machine learning que pone énfasis en aprender sucesivas capas (*layers*) con representaciones de los datos cada vez más esclarecedoras. Otro nombre apropiado para el campo podría haber sido «aprendizaje mediante representaciones por capas». Los modelos de deep learning pueden contener decenas o incluso centenas de capas sucesivas, y todas aprenden de forma automática al exponerlas a los datos de entrenamiento.

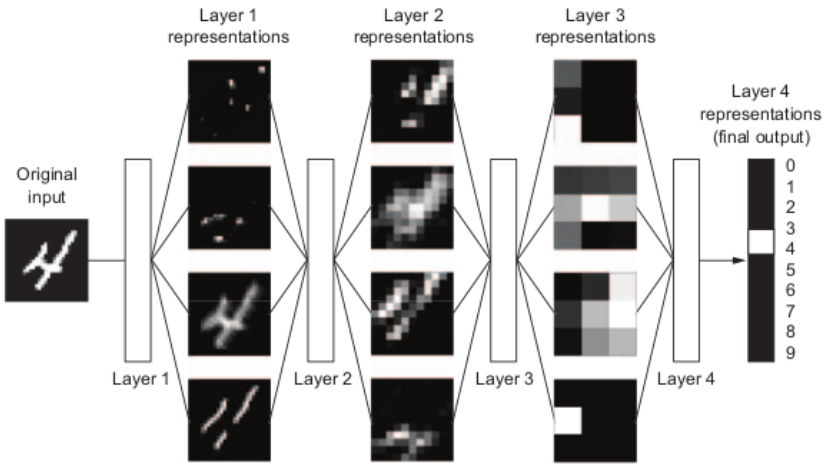

Figura 2.3: Representaciones por capas en el deep learning.

En deep learning, estas representaciones por capas aprenden casi siempre usando modelos llamados **redes neuronales** (ver figura 2.3). El nombre de red neuronal es una referencia a la neurobiología pero, aunque algunos de los conceptos centrales del deep learning fueran inspirados por nuestro conocimiento del cerebro, no hay ninguna evidencia de que el cerebro implemente nada parecido a los mecanismos usados en los modelos actuales de

DL. En lo que a nosotros nos concierne, el deep learning es un marco matemático para aprender representaciones de datos.

Podemos pensar en una red neuronal como una operación de destilación de información con varias etapas. Para controlar la salida de la red neuronal, necesitamos medir como de lejos está de la salida esperada, ese es el trabajo de la función de pérdida. Esta función coge las predicciones hechas por la red y las predicciones que esperas que la red haga y calcula una distancia que representa el desempeño de la red en esta tarea concreta. Esta medida de pérdida se usa como feedback para ajustar los parámetros de las capas de la red (pesos), en una dirección que disminuirá la función de pérdida. Este trabajo lo realiza el *optimizador*, implementando un algoritmo llamado *propagación hacia atrás*, el algoritmo central del deep learning (ver figura 2.4).

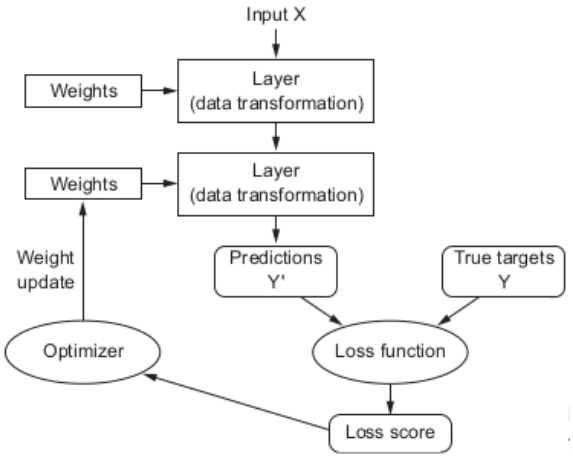

Figura 2.4: Algoritmo de propagación hacia atrás.

Sin duda alguna, la parte más problemática de la ejecución de la red neuronal es el ajuste de los pesos. Es natural preguntarse cuales son las bases del optimizador a la hora de ajustar los pesos en cada iteración, ya que esto no es una tarea nada trivial. Por suerte, podemos aprovechar que todas las transformaciones realizadas en las capas de la red neuronal son diferenciables y, por lo tanto, podemos calcular su gradiente. Basta entonces con mover los coeficientes en dirección contraria al gradiente para disminuir la medida de pérdida. La magnitud del ajuste de los parámetros de la red se denomina *tasa de aprendizaje* (learning rate). Normalmente, no esperamos a hacer una pasada por todos datos de entrada para hacer el ajuste, si no que partimos los datos de entrada en grupos de tamaño fijo denominados *lotes* (batches en ingles) y hacemos el ajuste tras probar la red neuronal en cada uno de ellos. Cada iteración por todo el conjunto de datos de entrada (y por lo tanto por todos los lotes) se la denomina *época*.

Por suerte, los aspectos técnicos de este proceso ya están implementados en librerías como TensorFlow[3] (desarrollada por Google) o Pytorch[4] (desarrollada por Facebook), lo que permite a mucha más gente entrar al mundo del deep learning, ya que minimiza al máximo los conceptos técnicos que se han de manejar.

Inicialmente, se les asignan valores aleatorios a los pesos de las capas de la red. Luego, tras cada iteración, los pesos se van ajustando disminuyendo la medida de pérdida. Esto es llamado el *bucle de entrenamiento* el cual, repetido unas cuantas veces (normalmente

decenas de iteraciones sobre miles de ejemplos), consigue unos pesos que minimizan la función de pérdida. Una red con una perdida mínima, es una red con los pesos de las capas tan cerca como es posible del objetivo, es decir, una red entrenada.

Por lo tanto, técnicamente el deep learning es un método para aprender representaciones de datos formado por varias etapas. Es una idea simple, pero mecanismos simples escalados de forma correcta pueden conseguir resultados extraordinarios.

2.1.3. Las cuatro ramas del machine learning

Hasta ahora, hemos explicado como funcionan las tareas de machine learning en aprendizaje supervisado. Sin embargo, esto es solo la punta del iceberg, el machine learning es un campo muy extenso que contiene gran variedad de subcategorías. Los algoritmos de machine learning normalmente se clasifican usando alguna de las siguientes categorías generales:

- Aprendizaje supervisado: Este es, con diferencia, el caso más común. Consiste en aprender como relacionar datos de entrada con etiquetas conocidas (también llamadas *anotaciones*), dado un conjunto de ejemplos normalmente anotados por humanos. La gran mayoría de aplicaciones conocidas de deep learning se encuentra dentro de esta categoría. Algunos ejemplos son: reconocimiento de imágenes, transcripción de discursos, traducción de textos, detección de objetos...
- Aprendizaje no supervisado: Consiste en encontrar transformaciones de los datos sin la ayuda de etiquetas, con el objetivo de comprender, visualizar, eliminar ruido o encontrar relaciones presentes en un conjunto de datos. El aprendizaje no supervisado es tremendamente común en el análisis de datos y es a menudo un paso necesario antes de intentar resolver un problema con aprendizaje supervisado.
- Aprendizaje autosupervisado: Este es un caso concreto del aprendizaje supervisado, pero es lo suficientemente diferente para merecer un apartado. Se trata de un aprendizaje supervisado en el que las anotaciones no son hechas por humanos. Este tipo de algoritmos siguen usando anotaciones, pero estas son generadas a partir de los datos de entrada, normalmente usando un algoritmo eurístico. Ejemplos de este tipo de problemas pueden ser adivinar la siguiente palabra de un texto o el siguiente fotograma de un vídeo.
- Aprendizaje reforzado: Esta rama del ML ganó su fama cuando el algoritmo DeepMind[5] de Google consiguió aprender a jugar a videojuegos de Atari y al Go al máximo nivel. En el aprendizaje reforzado, un *agente* recibe información sobre su entorno y aprende a elegir acciones que maximizarán un premio. Por ejemplo, una red neuronal que «mira» la pantalla de un videojuego y devuelve acciones del juego para maximizar la puntuación obtenida puede ser entrenada con aprendizaje reforzado. Por ahora, este área no ha conseguido grandes éxitos más allá de los videojuegos, pero se espera que consiga grandes logros en tareas del mundo real como coches autónomos, robótica, gestión de recursos, educación etc.

2.1.4. ¿Qué ha conseguido y qué podemos esperar del deep learning?

Aunque el deep learning es un campo del machine learning bastante antiguo, no empezó a ganar popularidad hasta la década de 2010. Desde entonces, ha conseguido revolucionar este área de investigación, con resultados especialmente buenos en tareas como ver y escuchar; tareas intuitivas para los humanos, pero que habían supuesto grandes retos para las máquinas. Estos son algunos de los progresos que ha conseguido el deep learning hasta la fecha (ver figura 2.5), todos en ámbitos históricamente complicados para el machine learning:

- Clasificación de imágenes a niveles cercanos al humano[6].
- Análisis de textos a niveles cercanos al humano[7].
- Transcripción de textos escritos a niveles cercanos al humano.
- Asistentes digitales como el de Google, Amazon (Alexa) o Apple (Siri).
- Conducción autónoma a niveles cercanos al humano[8].
- Jugar al Go (juego de mesa más complejo que el ajedrez) mejor que cualquier humano[5].
- Mejora de resultados de búsqueda en la red.
- Mejora de personalización de anuncios, usado por empresas como Google o Bing.
- Gestión de carteras y predicción de movimientos en los mercados a niveles cercanos al humano[9].

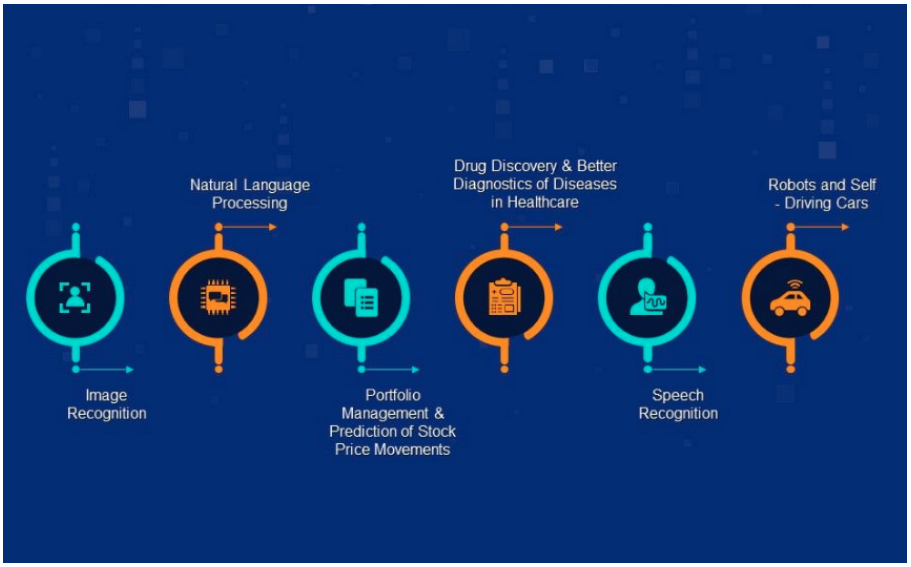

Figura 2.5: Campos históricamente complicados para el machine learning donde el deep learning ha conseguido avances muy considerables

Aún se están explorando las capacidades que el deep learning puede tener. Se está empezando a aplicar en campos fuera de la percepción de las máquinas o del procesamiento del lenguaje natural, como el razonamiento formal. Esto podría llevarnos a una nueva era

donde el deep learning asista a los humanos en el desarrollo de la ciencia, del software y quién sabe que más.

Sin embargo, no sería la primera vez que la industria se deja llevar por las hinchadas expectativas a corto plazo y, al ver que estas no se cumplen, dar lugar a una sequía de financiación que paraliza el desarrollo de la tecnología durante unos años (como ya pasó a principio de la década de los 2000s). No nos debemos dejar llevar por las expectativas a corto plazo, si no que debemos centrarnos en la visión a largo plazo. Puede que lleve un tiempo hasta que la inteligencia artificial llegue a su máximo desarrollo, pero esos avances están por venir y transformarán nuestra vida de una manera que nunca antes hubiéramos imaginado.

2.2. Conceptos fundamentales de Machine Learning

Antes de terminar la parte teórica del trabajo, es pertinente dedicar una sección a explicar metodologías que serán aplicadas en la parte práctica y también a aclarar cuales son las decisiones que se han de tomar a la hora de diseñar un modelo de deep learning con redes neuronales. El mundo del machine learning es esencialmente experimental, por lo que para decidir cual es el mejor diseño de red para un problema concreto se han de realizar diferentes experimentos hasta dar con la combinación de parámetros que obtenga mejores resultados. Sin embargo, estos experimentos se han de realizar con una base teórica detrás que nos guíe a la hora de decidir que modificaciones de la red pueden tener sentido. Esta sección tratará de desarrollar esa base teórica.

2.2.1. Preprocesamiento de los datos

Una pregunta esencial que debemos hacernos es como debemos preparar nuestros datos antes de introducirlos en la red neuronal como entrada. El formato de los datos de entrada es esencial y puede llegar a tener un efecto muy grande en la calidad de las predicciones de la red.

El primer paso del preprocesado de datos es la *vectorización*. A los vectores en machine learning se les denomina *tensores*. Todos los datos de entrada y los valores esperados de las predicciones que introducimos a la red deben ser tensores de números reales (float), aunque en casos concretos también se usan tensores de números enteros. Las imágenes se codifican como tensores de tres dimensiones, las dos primeras (altura y anchura) hacen referencia al píxel en el que nos encontramos y la última a los canales de color de la codificación de la imagen . De esta manera, para una codificación RGB tendríamos tres canales (ver figura 2.6).

A la vectorización le sigue la *normalización*. En general, no es buena idea introducir datos que tomen valores relativamente grandes (números de varios dígitos) o datos que sean heterogéneos (una característica con valores en $[0, 1]$ y otra con valores en $[100, 200]$). Para que la red aprenda de forma correcta, los datos de entrada han de tomar valores pequeños (normalmente en el intervalo $[0, 1]$ o $[-1, 1]$) y homogéneos (todos las características en un rango de valores similar). Por lo tanto, aunque no siempre es necesario, se suele hacer la siguiente normalización para cada característica de forma independiente:

- Normalizar la media a cero: $x = x - x.mean()$

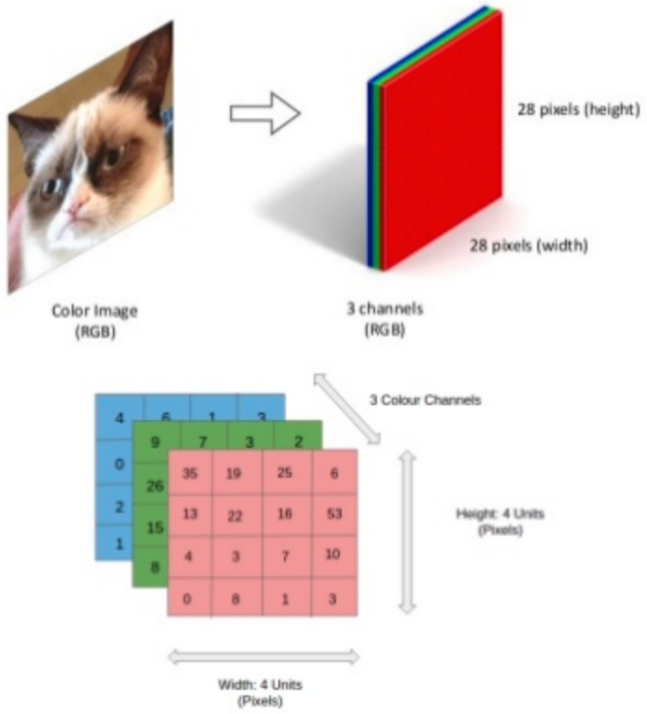

Figura 2.6: Transformación de una imagen en un tensor.

- Normalizar la desviación estándar a 1: $x = x/x.std()$

2.2.2. Sobreajuste y desajuste

Durante el entrenamiento, los modelos de deep learning llegan a su máximo rendimiento cuando pasan unas cuantas épocas y luego empieza a decaer. Esto es debido a que el modelo aprende características demasiado concretas de los datos de entrenamiento, lo que hace que el rendimiento en nuevos datos empiece a empeorar, a este fenómeno lo denominamos *sobreajuste*[10] (overfitting en inglés). Aprender a como gestionar el sobreajuste es esencial cuando se trabaja con machine learning.

El problema fundamental del machine learning es encontrar el equilibrio entre optimización y generalización. Por optimización entendemos el proceso de ajustar el modelo para que funcione lo mejor posible en el conjunto de entrenamiento, mientras que generalización es como se desenvuelve el modelo cuando se le presentan datos que no había visto antes. Por supuesto, el objetivo es maximizar la generalización. Pero como no podemos controlar la generalización, debemos ajustar el modelo basándonos en los datos de entrenamiento.

Para evitar que el modelo aprenda patrones incorrectos o irrelevantes, la mejor solución es conseguir más datos. Un modelo entrenado con más datos generalizará mejor, como es natural. Cuando esta opción no es posible, tenemos tres herramientas para combatir el sobreajuste:

- **Reducir el tamaño de la red**: Reducimos los parámetros que aprende la red reduciendo el número de capas o su tamaño. No hay ninguna fórmula mágica que nos guíe a la hora de saber cuantas capas deberíamos utilizar o que tamaño debe-

rían de tener. Deberemos evaluar diferentes combinaciones en nuestro conjunto de validación (luego explicaremos qué es eso) para poder dar con la más adecuada. La manera habitual de proceder es comenzar con modelos pequeños e ir aumentando el número de capas y su tamaño hasta que el valor de la función de pérdida en el conjunto de valoración empiece a decaer.

- **Regularizar los pesos de la red**: Otra forma de simplificar los modelos es forzar a que los pesos tomen valores pequeños, haciendo que su distribución sea más regular. A esto se le llama *regularización de pesos*. En la práctica se añade un coste a la función de pérdida asociado a los pesos con valores grandes en la red. El coste puede ser proporcional al valor absoluto de los pesos (regularización L1) o proporcional al cuadrado del valor de los pesos (regularización L2).
- **Usar abandono**: El *abandono* (dropout en inglés) es una técnica que consiste en cambiar algunos datos aleatorios de salida de una capa de la red por 0s durante el entrenamiento (ver figura 2.7). La *tasa de abandono* es la fracción de datos que se transforman en 0, normalmente se sitúa entre 0,2 y 0,5. Cuando se prueba la red, los valores no son suprimidos por 0s, pero todos los valores de salida se multiplican por la tasa de abandono para compensar que hay más unidades activas que en el entrenamiento. La idea detrás de esta técnica es que al introducir ruido en la salida de una capa se pueden romper patrones inservibles que la red hubiera aprendido en caso de no estar el abandono presente.

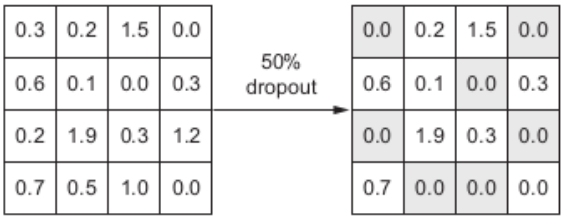

Figura 2.7: Explicación gráfica de la técnica del abandono en deep learning.

2.2.3. Evaluando modelos de Machine Learning

El objetivo de los modelos de deep learning es aprender patrones de unos datos de entrenamiento y poder *generalizar* esos patrones a nuevos datos que el modelo no haya visto previamente. Al entrenar con unos datos concretos, corremos el riesgo de que el modelo aprenda esos datos «de memoria» y no sea capaz de generalizar a nuevos datos (sobreajuste). Por ello, para medir el rendimiento de nuestro modelo entrenado, deberemos usar datos que el modelo no haya visto previamente. En concreto, los ejemplos que tengamos para entrenar la red se dividirán en tres conjuntos:

- Entrenamiento (training): El conjunto que se usará para entrenar la red. La red hará predicciones sobre este conjunto, se compararán con los valores reales usando la función de pérdida y se utilizará el optimizador para ajustar los pesos de la red y mejorar sus predicciones. Este proceso se repetirá durante varias iteraciones (llamadas épocas).

- Validación (validation): Este conjunto se usará para observar la capacidad de generalización del modelo. Si el valor de la función de pérdida mejora para el conjunto de entrenamiento pero no para el de validación, la red estará haciendo *sobreajuste*.
- Prueba (test): Este conjunto se usará para evaluar el rendimiento de la red una vez que se haya decidido su diseño y se haya terminado de entrenar.

Al ver esta división surge de forma natural la siguiente pregunta: ¿cuál es la función del conjunto de prueba? La realidad es que el conjunto de validación, aunque no se use directamente para entrenar la red, si tiene influencia a la hora de decidir el número de capas de la red, la profundidad de estas, la velocidad de aprendizaje... (todas estas decisiones se denominan *hiperparámetros* para distinguir de los *parámetros* que son los pesos de la red). Al hacer cambios en los hiperparámetros guiados por el rendimiento de la red en el conjunto de validación, podríamos conseguir un rendimiento artificialmente bueno de la red en el conjunto de validación. Para evitar esto, se hace una nueva valoración de la generalidad de la red con un conjunto de datos totalmente nuevo (conjunto de prueba) al final del proceso.

Es necesario que los datos sean variados en todos los conjuntos, por ejemplo, en una tarea de clasificación con varias etiquetas querremos que haya ejemplos de todas las etiquetas en todos los conjuntos. Para ello muchas veces es deseable reordenar los datos de forma aleatoria antes de hacer la división de los conjuntos. Algunas divisiones de conjuntos comunes son las siguientes:

- 80 % entrenamiento, 10 % validación, 10 % test
- 70 % entrenamiento, 15 % validación, 15 % test
- 60 % entrenamiento, 20 % validación, 20 % test

Cuando el conjunto de datos es pequeño, puede ser difícil obtener variedad de datos en todos los conjuntos haciendo una partición simple. Para solventar esto, existen técnicas avanzadas de partición como la validación con K particiones. Esta consiste en partir los datos de entrada en n conjuntos del mismo tamaño (3 o 5 son valores bastante comunes) y proceder a entrenar la red n veces usando en cada iteración uno de los conjuntos como conjunto de validación y los demás como conjunto de entrenamiento (ver figura 2.8).

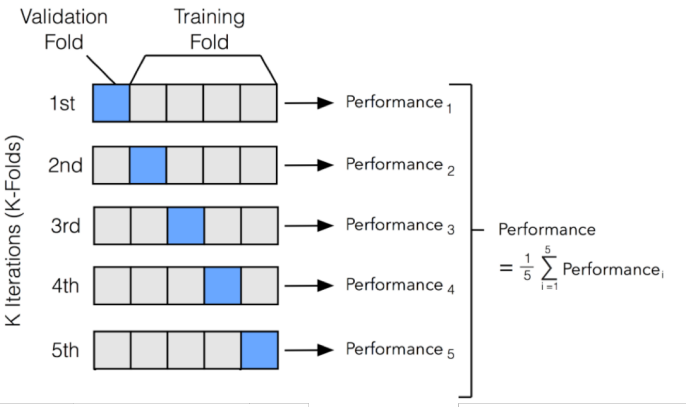

Figura 2.8: Técnica de la validación con K particiones.

2.2.4. Métricas comunes en deep learning

A la hora de crear un modelo de deep learning es esencial elegir una métrica adecuada que mida el rendimiento de este. Esta métrica será la que refleje como de óptimo es el rendimiento de cada modelo y la que nos servirá para comparar modelos y para guiarnos a la hora de diseñar modelos nuevos. Para el caso de modelos de clasificación, es común utilizar las métricas de exactitud, precisión, exhaustividad, valor F1 y la matriz de confusión.

Dado un problema de clasificación binario, la *matriz de confusión* es una forma de representar los aciertos y los fallos de una predicción. Para ello, utiliza los valores de verdadero positivo (TP), verdadero negativo (TN), falso positivo (FP) y falso negativo (FN). Pongamos un ejemplo en el que queremos distinguir entre fotos en las que aparece o no un gato. Tenemos 100 fotos, de las cuales solo 20 muestran un gato. En la figura 2.9 se muestra una matriz de confusión de una posible predicción. Esta misma idea idea se puede extender a problemas donde haya más de dos etiquetas.

		predicción	
		0	**1**
realidad	**0**	70	10
	1	15	5

		predicción	
		0	**1**
realidad	**0**	TN	FP
	1	FN	TP

Figura 2.9: Matriz de confusión del ejemplo de detección de gatos en imágenes.

La *exactitud* es una métrica que mide el porcentaje de casos en los que el modelo ha acertado. La fórmula empleada para ello es:

$$Exactitud = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Esta métrica es la más usada generalmente, sin embargo, este modelo es problemático cuando las clases no están balanceadas. Pongamos un modelo que, dado un texto, busque detectar palabras dentro de este relacionadas con cáncer. Imaginemos que solo una de cada 10 palabras está relacionada con cáncer en el texto. Si eso fuera así, un modelo que predijera que ninguna palabra está relacionada con cáncer tendría una exactitud del 90 % cuando en realidad sería un modelo totalmente inútil. Para solventar este problema, se usan las métricas precisión, exhaustividad y valor F1.

La *precisión* es una métrica que mide el porcentaje de aciertos al hacer una predicción positiva. En el ejemplo de los gatos sería el porcentaje de fotos con gatos entre las fotos que se predice que tienen un gato. La fórmula es la siguiente:

$$Precisión = \frac{TP}{TP + FP} \tag{2.2}$$

La *exhaustividad* nos informa sobre la cantidad de predicciones positivas que acertamos. En el ejemplo de los gatos sería el porcentaje de fotos con gatos acertadas entre la cantidad total de fotos con gatos. La fórmula es la siguiente:

$$Exhaustividad = \frac{TP}{TP + FN} \tag{2.3}$$

Finalmente, el valor F1 se usa para combinar las medidas de precisión y exhaustividad en un solo valor. Esto es práctico a la hora de comparar diferentes modelos atendiendo solo a una métrica. La fórmula es la media armónica entre la precisión y la exhaustividad:

$$F1 = 2 \cdot \frac{precisión \cdot exhaustividad}{precisión + exhaustividad} \tag{2.4}$$

Para problemas de clasificación con varias etiquetas, estas métricas han de ser calculadas para cada una de las etiquetas lo que dificulta hacer comparaciones de los modelos. A diferencia de las demás, la métrica de exactitud tiene una generalización natural para problemas de varias etiquetas. Denotando el conjunto de las diferentes etiquetas como I y N el número total de muestras:

$$Exactitud = \frac{\sum_{i \in I} TP_i}{N} \tag{2.5}$$

Para obtener una única métrica en un problema de clasificación con varias clases no balanceadas, podemos multiplicar los TP_i por un factor α_i que trate de dar una relevancia similar a todas las clases en la exactitud.

Para comodidad de los usuarios, hay diversas librerías como Sklearn[11] que ya implementan todas estás métricas.

2.2.5. La transformación más básica: la capa densa

Las capas densas[12] (del inglés *dense layers*) son el tipo de capa más simple en deep learning y están presentes en la solución de prácticamente cualquier problema. Estas se basan en una transformación lineal del tensor de entrada seguida de una transformación no lineal (denominada *función de activación*). En notación de Python, la transformación quedaría así:

```
y = f(W*X + b)
```

En la fórmula anterior, X es el tensor de entrada, y el tensor de salida, f la función de activación y W y b los pesos de la capa. Al tensor W se le denomina *núcleo de la transformación* y al tensor b se le denomina *desviación*. El núcleo siempre será un tensor 2D con tamaño de la primera dimensión igual al de la última dimensión de la entrada y con tamaño de la segunda dimensión igual al tamaño de la salida de la capa densa (esto será un hiperparámetro de la capa densa). Si la entrada (X) tiene más de dos dimensiones, se hará el producto escalar de la última dimensión de la entrada con la primera dimensión del núcleo. Por ejemplo, si la entrada es un tensor 3D con forma $(batchSize, d0, d1)$, entonces se creará un núcleo 2D con forma $(d1, outputSize)$. El núcleo operará a lo largo de la última dimensión de la entrada sobre cada sub-tensor con forma $(1, 1, d1)$ (habrá $batchSize * d0$ tales tensores). En este caso, la salida tendrá la forma $(batchSize, d0, outputSize)$.

Notemos que, sin la función de activación, la capa densa sería simplemente una transformación lineal del tensor de entrada. El hecho de que la transformación sea no lineal es esencial en deep learning. Si no utilizásemos una función de activación, el espacio de hipótesis de la capa serían las posibles transformaciones lineales. Este espacio de hipótesis es demasiado restrictivo y no se beneficiaría de múltiples capas de representaciones,

ya que apilar varias capas lineales resulta en otra capa lineal, por lo que no estaríamos expandiendo el espacio de hipótesis.

Para poder acceder a espacios de soluciones mucho más ricos, necesitamos una función no lineal, o función de activación. La función de activación más popular en deep learning es *relu*, pero hay otras muchas candidatas[13] con nombres similares como *leacky relu*, *elu* etc (ver figura 2.10).

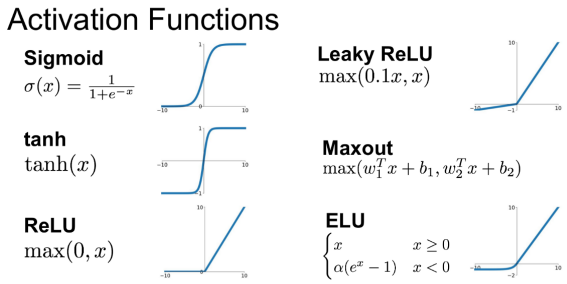

Figura 2.10: Funciones de activación.

Las capas densas pueden modelar cualquier función matemática. Sin embargo, siguen limitadas en el sentido de que no pueden detectar repeticiones en el tiempo (textos o voz) o en el espacio (imágenes) y para una misma entrada siempre devuelven una misma salida. Para sortear estas limitaciones, recurriremos a las redes recurrentes y a las redes convolucionales, las cuales trataremos en las siguientes secciones.

2.2.6. Función de activación, función de pérdida y optimizador

Una vez que hemos preparado nuestros datos de entrenamiento y hemos decidido la estructura que tendrá nuestro modelo de deep learning, aún quedan tres decisiones esenciales que tomar antes de poner nuestro a modelo a entrenar:

- **Función de activación de la última capa**: La última capa del modelo será la que nos proporcione la salida, por lo que la función de activación de la última capa debe ser acorde al tipo de salida que queremos obtener. Esto dependerá del problema que estemos tratando, por ejemplo, para un problema de clasificación binaria deberíamos usar *sigmoid* mientras que para un problema de clasificación de etiqueta única con varias clases deberíamos usar *softmax* (ver figura 2.11).
- **Función de pérdida**: Esta, al igual que la función de activación, también depende del problema al que nos estemos enfrentando. Para un problema de clasificación binaria usaríamos *entropía cruzada binaria* (ver figura 2.11).
- **Configuración de optimización**: Hemos de decidir el optimizador que utilizaremos y su tasa de aprendizaje. Normalmente, es seguro utilizar el optimizador *Adam*[14] con la tasa de aprendizaje por defecto (0,001).

2.2.7. Experimentación con redes neuronales

Finalmente, es pertinente explicar como va a ser el flujo típico de experimentación a la hora de construir un modelo de deep learning. Para ello tendremos que ir entrenando la

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Figura 2.11: Funciones de activación para la última capa y funciones de pérdida según el tipo de problema.

red e ir haciendo modificaciones buscando mejorar la métrica elegida, que en nuestro caso será la *exactitud*.

El primer paso será determinar cuales van a ser nuestro datos de entrada y qué clase de problema queremos resolver. Luego deberemos hipotetizar que los datos que ofreceremos al modelo son suficientes para predecir la salida que esperamos de este. A menudo se utilizan modelos de machine learning con los que se espera predecir el futuro de algún fenómeno utilizando datos del comportamiento pasado. Es común que estos datos no sean suficientes para predecir el comportamiento futuro, lo que hace que el modelo se comporte de forma deficiente.

A continuación, debemos elegir una métrica. Para ello tendremos en cuenta el problema en cuestión ya que esta métrica será la que guíe el modelo a la hora de aprender. Además deberemos elegir nuestro protocolo de evaluación, pudiendo mantener un conjunto de validación fijo (si tenemos muchos datos) o hacer una validación en K particiones. Finalmente, deberemos normalizar los datos y elegir nuestra función de activación, función de pérdida y configuración de optimización.

Llegado a este punto, sólo nos queda determinar la arquitectura del modelo y sus hiperparámetros. Para ello, la forma habitual de proceder es comenzar por un modelo pequeño e ir aumentando progresivamente las capas, su tamaño y el número de épocas hasta que el rendimiento se quede estancado o empiece a disminuir. Una vez determinado como de grande debe ser nuestro modelo, podemos probar modificaciones más sofisticadas como puede ser añadir abandono o regularización L1 o L2.

2.3. Redes recurrentes

La inteligencia biológica procesa información de forma incremental mientras mantiene un modelo interno de lo que se está procesando, creado con la información pasada y actualizándose constantemente conforme recibe la nueva información. Una **red neuronal recurrente (RNN)** adopta el mismo principio, aunque de una forma muy simplificada. Procesa secuencias iterando elementos de la secuencia y manteniendo un estado con información de los elementos que han sido procesados. Una RNN es un tipo de red neuronal que funciona con un bucle interno. El estado se reinicia entre dos secuencias diferentes (por ejemplo, dos frases distintas en una tarea de reconocimiento de lenguaje natural), por lo que una secuencia se considera una unidad de datos de entrada. Lo que hace especial a este tipo de redes neuronales es que los datos no se procesan en un solo paso, si no que la red itera sobre cada uno de los elementos de la secuencia (ver figura 2.12).

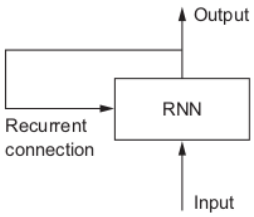

Figura 2.12: Funcionamiento de una RNN.

Por todo esto, las RNN son apropiadas para tareas en las que se trabajen con datos secuenciales, como el procesamiento de texto o sonido. Más adelante se expondrán tipos de redes neuronales más apropiadas para la clasificación de imágenes.

2.3.1. Redes de memoria a largo-corto plazo (LSTM) y redes con unidades cerradas recurrentes(GRU)

La LSTM[15] es uno de los tipos de RNN más conocidos. Las LSTM son una variantes de las redes RNN simples, expuestas anteriormente. Su diferencia reside en que guardan las salidas de cada elemento de la secuencia para que estas puedan ser utilizadas después, evitando así que las señales antiguas se desvanezcan gradualmente durante el proceso (ver figura 2.13).

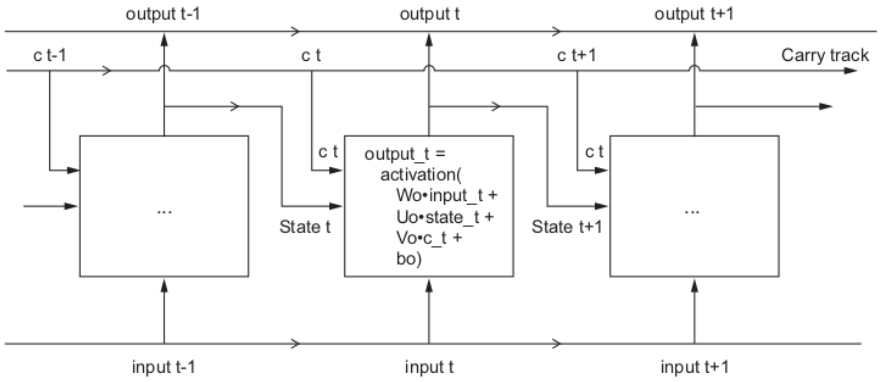

Figura 2.13: Funcionamiento de una red LSTM.

Otra variante muy común de las RNN son las GRU. Las GRU funcionan usando los mismos principios que las LSTM, pero de una forma algo simplificada y son, por lo tanto, más baratas computacionalmente (aunque no tienen tanta potencia de representación como las LSTM). Este intercambio entre coste computacional y potencia de representación es algo muy común dentro del machine learning.

Algunas técnicas avanzadas que se aplican con redes LSTM y GRU son: abandono recurrente, apilar varias capas recurrentes y utilizar capas recurrentes bidireccionales.

2.4. Redes convolucionales(CNN)

Las redes convolucionales son la primera opción a la hora de realizar tareas relacionadas con imágenes. La característica de estas es que aprenden patrones locales como vemos en la figura 2.14.

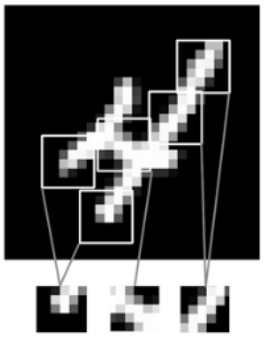

Figura 2.14: Patrones locales reconocidos por una CNN.

Esta característica fundamental de las CNN tiene las siguientes propiedades remarcables:

- **Los patrones aprendidos son invariantes por traslación**: Pueden reconocer patrones independientemente de en que parte de la imagen se sitúen. Esto hace a las CNN eficientes a la hora de procesar imágenes, ya que el mundo visual es fundamentalmente invariante por traslaciones. Por esto, necesitan menos ejemplos de entrenamiento para aprender representaciones que tengan potencial para ser generalizadas.
- **Pueden aprender patrones de jerarquías visuales**: Una primera CNN aprenderá pequeños patrones locales como vértices, una segunda CNN aprenderá patrones más grandes formados por los patrones de la primera, y así sucesivamente (ver figura 2.15). Esto permite a las CNN aprender patrones cada vez más complejos y conceptos visuales abstractos, ya que el mundo visual esta formado fundamentalmente por jerarquías visuales.

2.4.1. La operación de convolución

Las operaciones realizadas en las capas convolucionales se denominan *convoluciones*[16]. Las convoluciones operan sobre tensores de 3 dimensiones llamados *mapas de características*, con dos ejes espaciales (altura y anchura) y un eje de profundidad (también denominado eje de canales). Para una imagen en formato RGB, la dimensión del eje de profundidad es 3, ya que la imagen tiene tres canales de color (rojo, verde y azul). Para una imagen en blanco y negro, la dimensión del eje de colores es 1 (los niveles de gris).

La operación de convolución extrae fragmentos de su mapa de características y aplica la misma transformación a todos esos fragmentos, produciendo un *mapa de características de salida*. Esta salida es también un tensor de 3 dimensiones, con anchura y altura. El tamaño de la dimensión del eje de profundidad puede ser arbitraria ya que ya no

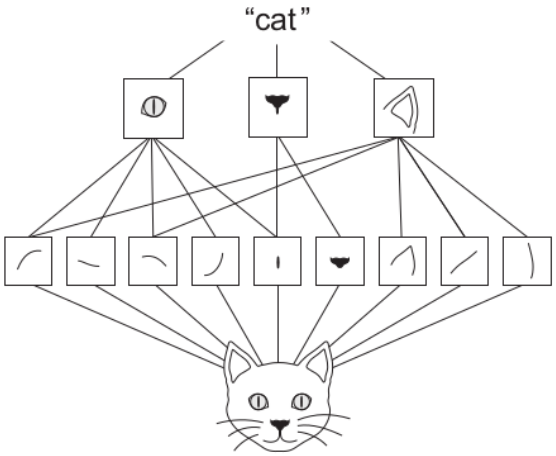

Figura 2.15: Descomposición de una imagen en patrones locales.

representa los canales de color de la imagen, si no que ahora representa *filtros*. Estos filtros codifican características específicas de los datos de entrada, por ejemplo un filtro podría codificar la «presencia de una cara en la entrada». Una salida de dimensiones $(26, 26, 32)$ computaría 32 filtros sobre la entrada. Cada filtro resultaría en una matriz de 26×26 llamada *mapa de respuesta* del filtro sobre la entrada, ésta codifica la respuesta del filtro en diferentes localizaciones de la entrada (ver figura 2.16). Por eso, utilizamos el término mapa de características, cada matriz de la salida es un mapa espacial de la respuesta de la característica (o filtro) sobre la entrada.

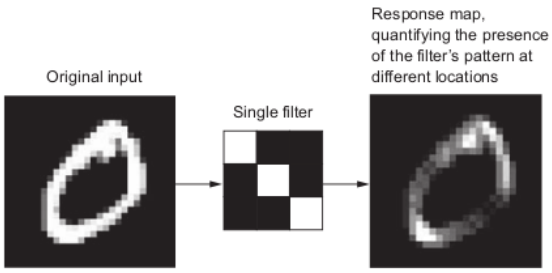

Figura 2.16: Ejemplo del efecto de un filtro sobre una imagen y el mapa de características resultante.

Las CNN son definidas por dos parámetros, que deberemos pasar a la función a la hora de implementar una CNN:

- El tamaño de los patrones extraídos de la entrada: Valores típicos de este parámetro son 3×3 o 5×5.
- La profundidad del mapa de características de salida: Valores típicos de este parámetro son 32 o 64.

Una convolución funciona desplazando esas ventanas de 3×3 o 5×5 sobre el mapa de entrada tridimensional, parando en cada posible localización, y extrayendo un tensor tridimensional (una pequeña ventana del mapa inicial). Cada tensor 3D es transformado (por producto de tensores con una misma matriz llamada *núcleo de la convolución*) en un tensor de una dimensión del tamaño de la profundidad del mapa de salida. Todos

estos tensores $1D$ se juntan para formar el mapa de características de salida (ver figura 2.17).

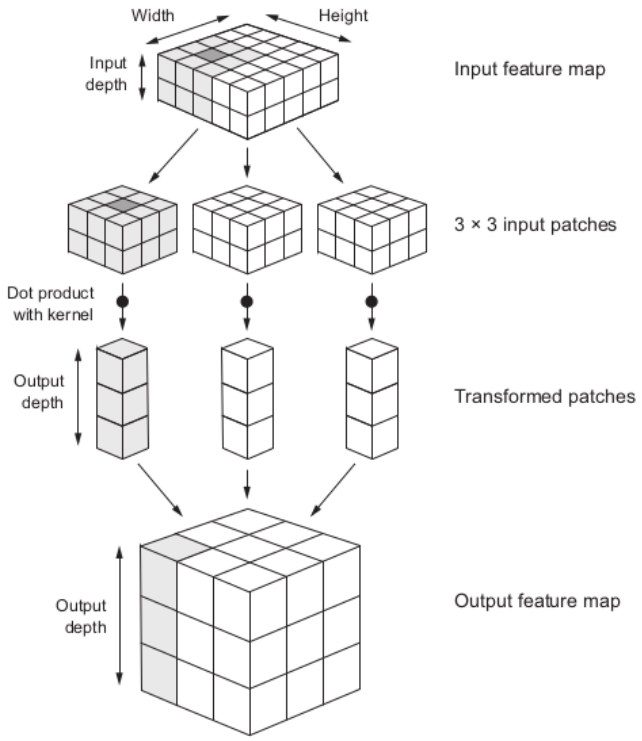

Figura 2.17: Efecto de una convolución sobre un mapa de características de entrada.

Aunque el mapa de salida no tiene porque tener el mismo tamaño que el mapa de entrada (si usamos patrones 3×3 la altura y la anchura se reducen 2 unidades, si usamos 5×5 se reducen 4 etc.), la distribución espacial se mantiene, es decir, la esquina superior derecha del mapa de salida se corresponde con la esquina superior derecha del mapa de entrada. Por ejemplo, con patrones 3×3, el tensor $salida[i, j, :]$ se corresponde con el tensor $entrada[i-1 : i+1, j-1 : j-1, :]$ (siempre que $0 < i < altura - 1$ y $0 < j < anchura - 1$). A veces se aumenta el tamaño del mapa de entrada introduciendo relleno en los bordes antes de realizar la convolución para que el tamaño del mapa de salida sea igual al del mapa de entrada.

2.4.2. La operación de agrupación máxima

La función de la capa de agrupación máxima(del inglés max-pooling) es disminuir la cantidad de datos del mapa de características después de aplicar una convolución. La agrupación máxima consiste en coger una ventana del mapa de características y devolver el valor máximo de cada canal, es decir, un vector de $1D$ con la misma profundidad que la entrada de la capa de agrupación máxima (ver figura 2.18). La idea es similar a las convoluciones salvo que, en vez de transformar las ventanas vía una transformación lineal aprendida, las transforma con una función vectorial de máximos, y esta es siempre la misma. Normalmente la agrupación máxima se suele realizar con ventanas 2×2 y no dejando que las ventanas se solapen (lo que técnicamente se denomina paso 2). Mientras que las convoluciones se realizan con ventanas 3×3 completamente solapadas (paso 1).

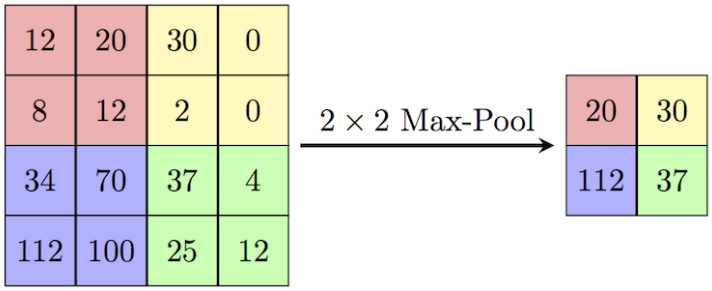

Figura 2.18: Funcionamiento de la capa de agrupación máxima.

¿Por qué reducir la cantidad de datos que tenemos de esta manera? Las capas de agrupación máxima solucionan dos problemas fundamentales:

- **Los patrones analizados por las redes serían demasiado pequeños**: Si superpusiéramos tres capas de convolución con ventanas 3×3 sin ninguna de agrupación máxima, la tercera capa contendría información de patrones 7×7. Ventanas de 7×7 píxeles son demasiado pequeñas para reconocer prácticamente cualquier elemento del mundo real.
- **El mapa final contendría demasiada información**: Imaginemos que el mapa final tiene dimensiones $22 \cdot 22 \cdot 64 = 30976$. Imaginemos que pasáramos la información por una capa densa con 512 como dimensión de salida para hacer las predicciones. Esta capa tendría casi 16 millones de parámetros. Esto es sin duda demasiado para un modelo pequeño y daría lugar a malas predicciones debido a que el modelo aprendería de memoria los patrones del entrenamiento y perdería capacidad para generalizar (overfitting).

La agrupación máxima no es la única forma de disminuir la cantidad de datos. También podríamos evitar el solapamiento (aumentar el paso) en las capas de convolución, consiguiendo una reducción de los datos similar. Otra opción sería obtener la media de cada canal (average-pooling) en vez del máximo. En la práctica, la agrupación máxima consigue los mejores resultados. Esto es debido a que los mapas de características suelen codificar la presencia de características concretas dentro de una parcela de la imagen (de ahí su nombre). Es más informativo mirar la presencia máxima de una característica que su presencia media, ya que la presencia media puede difuminar la información. Por lo tanto, el procedimiento más razonable es conseguir mapas densos de características (usando convoluciones solapadas) y luego mirar la presencia máxima de características en pequeñas parcelas (agrupación máxima). Esto es más efectivo que obtener mapas con información más distribuida (haciendo convoluciones no solapadas) u obtener la presencia media de las características (agrupación media).

2.4.3. Aumento de datos

El sobreajuste es provocado por la escasez de muestras en el conjunto de entrenamiento. Cuando tratamos con imágenes, es posible aumentar de forma artificial la cantidad total de muestras, a esta técnica la llamamos *aumento de datos*[17] (data augmentation en inglés). El aumento de datos consiste en aumentar el conjunto de datos de entrenamiento aplicando al conjunto inicial una serie de transformaciones aleatorias que dan como resultado imágenes que seguirían siendo válidas como imágenes de entrenamiento. Estas

transformaciones se hacen durante el entrenamiento de la red, de forma que la red nunca es expuesta a dos imágenes exactamente iguales. Algunas de las transformaciones más comunes son rotaciones, traslaciones, ampliar la imagen y aplicar simetrías, normalmente en el eje horizontal (ver figura 2.19).

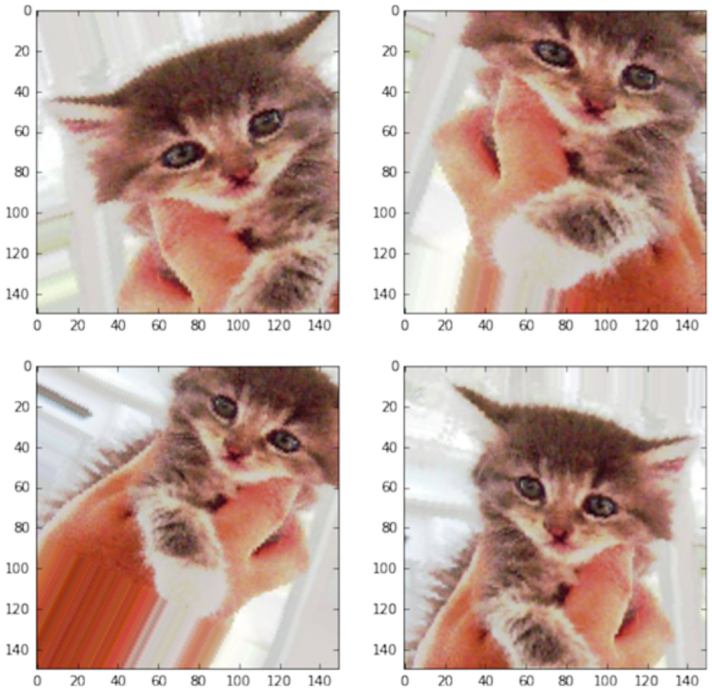

Figura 2.19: Imagen de un gato a la que se le ha aplicado la técnica del aumento de datos.

2.4.4. Normalización por lotes

La normalización es una amplia categoría de métodos que buscan hacer que un modelo de machine learning vea diferentes muestras más parecidas entre sí. De esta forma el modelo aprende y generaliza mejor. La forma más común de normalización es, como ya hemos visto anteriormente, centrar los datos en 0 restando la media de todos y buscar una desviación estándar unitaria dividiendo por la desviación estándar. Esto toma por supuesto que los datos tienen una distribución normal.

Hasta ahora habíamos normalizado los datos antes de introducirlos al modelo. Sin embargo, deberíamos preocuparnos por normalizar los datos después de cada transformación realizada por la red neuronal. Aunque los datos que entran a una capa densa o convolucional tengan media 0, esto no significa que los datos que salgan también la tendrán.

La *normalización por lotes*[18] (batch normalization en inglés) es una capa que normaliza de manera adaptativa los datas incluso cuando la media y la varianza cambian durante el entrenamiento. Para ello, guarda una media móvil exponencial de la media y la varianza de cada lote de datos de entrenamiento. La capa de normalización por lotes se suele aplicar después de capas densas o convolucionales.

El principal efecto de la normalización por lotes es que ayuda con la propagación del gradiente, permitiendo así modelos con más capas. Por ejemplo, la normalización por

lotes se usa en varios de los modelos convolucionales avanzados incluidos en Tensorflow como ResNet50, Inception V3 y Xception.

2.4.5. Convolución separable por profundidad

Curiosamente, hay una alternativa a la clásica convolución que consigue modelos más ligeros (con menos parámetros entrenables), más rápido (realiza menos operaciones en punto flotante) y mejora las métricas de nuestros experimentos en algunos puntos porcentuales. Esto es lo que hace las convoluciones separables por profundidad (*SeparableConv2D* en Tensorflow).

Esta capa hace una convolución en cada canal de la entrada de forma independiente y luego mezcla los canales resultantes con una convolución 1×1 (ver figura 2.20). Esto es equivalente a separar el aprendizaje de características espaciales del de características a nivel de canal. Esto tiene sentido si asumimos que informaciones próximas en la imagen están altamente relacionadas, pero que la información de diferentes canales es bastante independiente.

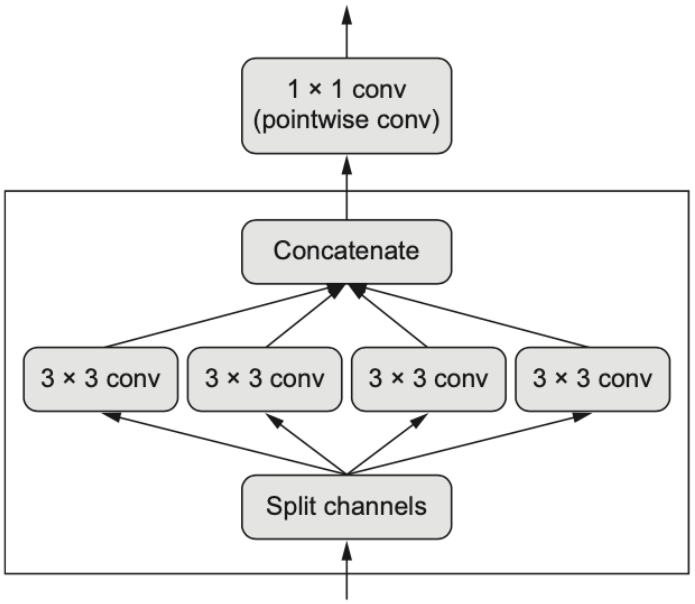

Figura 2.20: Estructura de una convolución separable por profundidad.

Esta forma de realizar la convolución requiere de menos parámetros y realiza menos operaciones, por lo que da lugar a modelos más pequeños y más rápidos. Además, como realiza la representación de los datos de forma más eficiente, tiende a realizar mejores representaciones usando menos datos, lo que nos lleva a modelos con mejor desempeño. Este tipo de capa es fundamental en modelos ampliamente famosos por su gran rendimiento con limitado tamaño como Xception[19].

2.5. Redes generativas antagónicas (GANs)

El potencial de la inteligencia artificial para emular el pensamiento humano va más allá de tareas pasivas o reactivas como reconocer objetos o conducir un coche. También se

extiende a tareas creativas. Las *redes generativas antagónicas (GANs)*[20] permiten la generación de imágenes sintéticas razonablemente realistas. Esto se consigue forzando a las imágenes generadas a ser estadísticamente indistinguibles de las reales.

Una metáfora útil para entender el funcionamiento de las GANs es el del falsificador de cuadros y el experto en arte. El falsificador es inicialmente malo imitando cuadros de Picasso. Junta las imitaciones con cuadros de verdad y se los enseña al experto. El experto evalúa la autenticidad de cada cuadro y da un feedback al falsificador sobre que es lo que hace a un Picasso parecer un Picasso y el falsificador vuelve a realizar nuevas imitaciones. Con el tiempo, el falsificador se vuelve realmente bueno falsificando cuadros y el experto se vuelve realmente bueno detectando fraudes. Al final, ambos acaban consiguiendo unos Picassos falsos de mucha calidad.

Así es como funciona una GAN, una red que falsifica y otra que distingue las falsificaciones de los originales, ambas entrenadas para batir a la otra:

- **Red generadora**: Coge como entrada un vector aleatorio y lo codifica como una imagen sintética.
- **Red discriminadora (o adversario)**: Coge como entrada una imagen (real o sintética) y predice si la imagen es del conjunto de entrenamiento o de la red generadora.

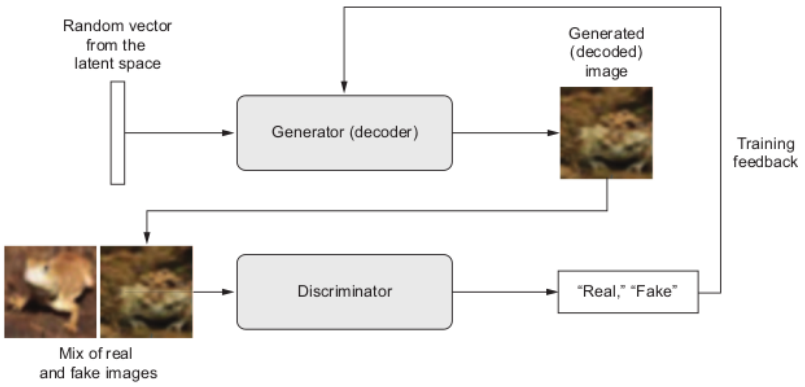

Figura 2.21: Funcionamiento de una red generativa antagónica.

La red generadora se entrena para engañar a la discriminadora y la discriminadora se entrena para poder distinguir las imitaciones cada vez mejor (ver figura 2.21). De esta manera obtenemos imágenes cada vez más realistas y un umbral de exigencia cada vez mayor. Al acabar el entrenamiento, la red generadora es capaz de transformar cualquier vector de su espacio de entrada en una imagen que engañe a la red discriminadora.

Una GAN es un sistema que no trata de optimizar un mínimo como las otras redes neuronales que hemos visto. Normalmente, el descenso de gradiente consiste en optimizar una función en un entorno estático. Sin embargo, con las GANs cada paso que damos cambia un poco el entorno. Es un sistema dinámico donde el proceso de optimización no trata de encontrar un mínimo, si no un equilibrio entre dos fuerzas. Por esto, las GANs son redes difíciles de entrenar; conseguir una GAN que funcione requiere de ajustar la arquitectura del modelo y sus parámetros de forma muy cuidadosa. A pesar de todo, se

pueden llegar a conseguir imágenes de alta calidad usando redes de este estilo (ver figura 2.22).

Figura 2.22: Imágenes sintéticas generadas por una red generativa antagónica.

Capítulo 3

Reconocimiento de letras/dígitos

3.1. Problema y datos de entrenamiento

Como primer acercamiento al procesamiento de imágenes con deep learning generaremos un modelo que, dada una imagen en la que se puede ver una letra, nos diga qué letra es. Para ello, contamos con un conjunto de datos formado por 10009 imágenes que representan las 26 letras del alfabeto inglés (ver figura 3.1). La distribución de las imágenes es la siguiente:

```
Total number of samples: 10009
Number of tags:  26
Letter distribution:
R: 400                U: 400                I: 208                N: 399                G: 395
Z: 399                T: 399                S: 400                A: 400                F: 401
O: 400                H: 400                M: 402                J: 211                C: 398
D: 400                V: 400                Q: 400                X: 400                E: 399
B: 400                K: 400                L: 400                Y: 400                P: 398
W: 400
```

Figura 3.1: Distribución del conjunto de imágenes de letras.

Las imágenes tienen formato 20×20 píxeles y están codificadas en RGB (3 canales de color). Por lo que las guardaremos como tensores de tres dimensiones con tamaño $(20, 20, 3)$. Los canales de RGB guardan valores entre el 0 y el 255. Antes de proporcionarlas como input a la red neuronal, dividiremos estos valores entre 255 para obtener valores en el rango $[0, 1]$ ya que la red neuronal se comporta mejor con valores pequeños como entrada. Las imágenes tienen el aspecto mostrado en la figura 3.2.

Debido a que los datos de entrenamiento no son tan numerosos, se ha decido utilizar la validación en 5 particiones, por lo que no utilizaremos conjunto de test. Para cada modelo propuesto se entrenará 5 veces utilizando en cada iteración un quinto de los datos como conjunto de validación y los restantes como conjunto de entrenamiento. La métrica que nos guiará a la hora de modificar el modelo será la exactitud, calcularemos la media aritmética de las cinco exactitudes obtenidas para obtener una idea general de como se comporta el modelo propuesto.

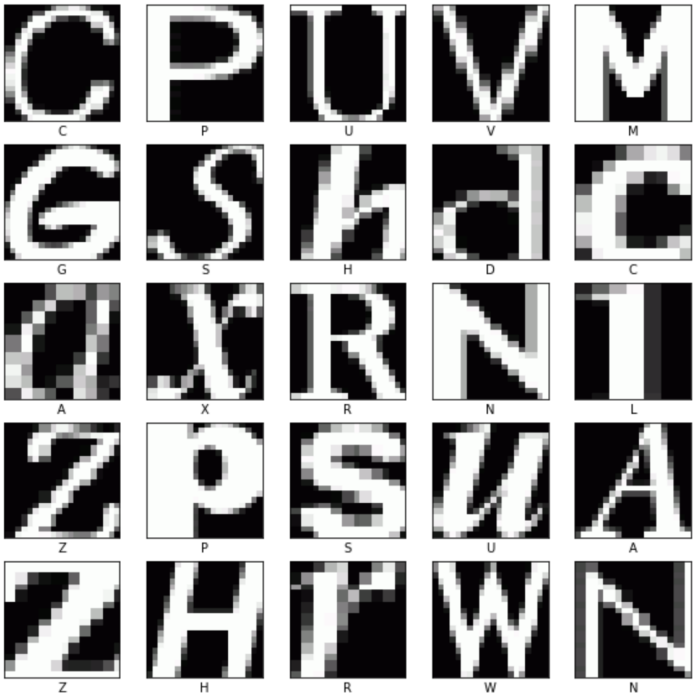

Figura 3.2: Muestra de las imágenes de dígitos.

3.2. Arquitectura propuesta

Al ser un problema de clasificación de imágenes, es claro que la técnica a utilizar ha de ser las redes neuronales convolucionales[21][22]. En este sentido, propondremos cuatro modelos. Inicialmente, entrenaremos estos modelos con pesos pequeños en cada una de sus capas y haremos sucesivos experimentos aumentando los pesos de las capas en los modelos más prometedores. Finalmente, añadiremos abandono cuando los modelos que mejor se comporten empiecen a mostrar sobreajuste. Los cuatro modelos propuestos son los siguientes:

- **Modelo 1**: Nuestro primer modelo tratará de ser lo más simple posible. Estará formado por una capa convolucional de tamaño 32 con ventanas 3×3 totalmente solapadas, seguida de una capa de agrupación máxima 2×2 sin solapamiento. Tras esto usaremos una capa de aplanado que reestructurará todos los datos en un vector de una dimensión. A continuación, utilizaremos una capa densa de tamaño 32 y relu como función de activación. Finalmente, otra capa densa con tamaño el número de etiquetas (26) y con función de activación softmax, la cual nos proporcionará nuestra salida (ver figura 3.3).
- **Modelo 2**: El modelo 2 será igual que el modelo 1 pero duplicando el bloque que forman la capa convolucional y la de agrupación máxima (ver figura 3.4).
- **Modelo 3**: El modelo 3 será igual al modelo 1 pero duplicando la capa densa con relu como función de activación (ver figura 3.5).
- **Modelo 4**: El modelo 4 surgirá de añadir al modelo 1 ambas modificaciones, es decir, constará de dos bloques convolucionales y dos capas densas con función de activación relu (ver figura 3.6).

Layer (type)	Output Shape	Param #
rescaling_90 (Rescaling)	(None, 20, 20, 3)	0
conv2d_128 (Conv2D)	(None, 18, 18, 32)	896
max_pooling2d_126 (MaxPoolin	(None, 9, 9, 32)	0
flatten_91 (Flatten)	(None, 2592)	0
dense_209 (Dense)	(None, 32)	82976
dense_210 (Dense)	(None, 26)	858

Total params: 84,730
Trainable params: 84,730
Non-trainable params: 0

Figura 3.3: Resumen del modelo 1

Layer (type)	Output Shape	Param #
rescaling_91 (Rescaling)	(None, 20, 20, 3)	0
conv2d_129 (Conv2D)	(None, 18, 18, 32)	896
max_pooling2d_127 (MaxPoolin	(None, 9, 9, 32)	0
conv2d_130 (Conv2D)	(None, 7, 7, 32)	9248
max_pooling2d_128 (MaxPoolin	(None, 3, 3, 32)	0
flatten_92 (Flatten)	(None, 288)	0
dense_211 (Dense)	(None, 32)	9248
dense_212 (Dense)	(None, 26)	858

Total params: 20,250
Trainable params: 20,250
Non-trainable params: 0

Figura 3.4: Resumen del modelo 2

Layer (type)	Output Shape	Param #
rescaling_93 (Rescaling)	(None, 20, 20, 3)	0
conv2d_133 (Conv2D)	(None, 18, 18, 32)	896
max_pooling2d_131 (MaxPoolin	(None, 9, 9, 32)	0
flatten_94 (Flatten)	(None, 2592)	0
dense_216 (Dense)	(None, 32)	82976
dense_217 (Dense)	(None, 32)	1056
dense_218 (Dense)	(None, 26)	858

Total params: 85,786
Trainable params: 85,786
Non-trainable params: 0

Figura 3.5: Resumen del modelo 3

Layer (type)	Output Shape	Param #
rescaling_92 (Rescaling)	(None, 20, 20, 3)	0
conv2d_131 (Conv2D)	(None, 18, 18, 32)	896
max_pooling2d_129 (MaxPoolin	(None, 9, 9, 32)	0
conv2d_132 (Conv2D)	(None, 7, 7, 32)	9248
max_pooling2d_130 (MaxPoolin	(None, 3, 3, 32)	0
flatten_93 (Flatten)	(None, 288)	0
dense_213 (Dense)	(None, 32)	9248
dense_214 (Dense)	(None, 32)	1056
dense_215 (Dense)	(None, 26)	858

Total params: 21,306
Trainable params: 21,306
Non-trainable params: 0

Figura 3.6: Resumen del modelo 4

-***Nota***: Todas los modelos tienen al inicio una capa de escalado que no ha sido mencionada. Esta capa se encarga de dividir los valores de las imágenes entre 255 (normalizar) como se ha comentado antes. La razón de que lo añadamos como parte del modelo en vez de realizarlo de forma previa es por pura comodidad a la hora de escribir el código.

Los hiperparámetros que no han sido mencionados todavía los dejaremos fijos a lo largo de los diferentes experimentos. En concreto, utilizaremos Adam como optimizador, entrenaremos por 10 épocas y dejaremos la tasa de aprendizaje y el tamaño de lote a los predefinidos por Tensorflow (0.001 y 32 respectivamente).

Al ser sencillos tanto el problema como los modelos propuestos, basta la CPU de un ordenador medio de sobremesa para entrenar los modelos. En particular, los modelos se entrenaron en un *MacBook Pro 2019* con un procesador *Intel Core i7* de 2,6 GHz y 6 núcleos. Con este hardware, se pudieron entrenar todos los modelos propuestos en menos de dos horas.

3.3. Resultados

Tras experimentar con los modelos propuestos los resultados fueron los mostrados en la tabla 3.1. Como podemos observar, los modelos 1 y 3 obtuvieron resultados significativamente mejores que los modelos 2 y 4. Comparando el modelo 1 y el modelo 2, podemos deducir que una segunda capa convolucional lejos de mejorar el funcionamiento de la red, lo empeora. Esta hipótesis se confirma en los experimentos de los modelos 3 y 4 ya que estos también se diferencian exclusivamente en la segunda capa convolucional y el

3 se comporta significativamente mejor que el 4. Comparando los modelos 1 y 3, también podemos observar que una segunda capa densa con activación relu no empeora el rendimiento de la red, pero tampoco lo mejora significativamente ya que tenemos una diferencia de menos del 0,1 % en la exactitud.

	Modelo 1	**Modelo 2**	**Modelo 3**	**Modelo 4**
Exactitud en validación	96,31	91,79	**96,46**	90,90

Tabla 3.1: Resultados de los experimentos planteados con los cuatro modelos iniciales.

Basándonos en la primera ronda de experimentos, se decidió realizar otra segunda ronda descartando los modelos 2 y 4 debido a su menor éxito. En esta segunda ronda se irá duplicando el tamaño de las capas convolucionales y de las densas con activación relu. De esta forma, experimentamos con los modelos 1 y 3 con capas de tamaños 64 y 128. Los resultados fueron los mostrados en la tabla 3.2.

Tamaño de capas	**Modelo 1**	**Modelo 3**
32	96,31	96,46
64	97,15	96,50
128	**97,43**	96,55
256	97,23	97,16

Tabla 3.2: Resultados de aumentar el tamaño de las capas en los modelos 1 y 3.

Los resultados de esta serie de experimentos son claros. Lo primero que nos llama la atención es que los resultados del modelo 1 son ligeramente mejores que los del modelo 3. Al ser el modelo 3 más exigente computacionalmente, esto deja al modelo 1 como claro ganador. Curiosamente, la arquitectura más sencilla es la que mejor se comporta con el problema planteado. Esto seguramente se deba a que el problema planteado es también bastante sencillo ya que utilizamos imágenes en blanco y negro con todas las letras del mismo tamaño y en la misma posición.

La segunda apreciación que podemos hacer es que, en efecto, un mayor tamaño de las capas de la red resulta en un mejor funcionamiento de ésta. Sin embargo, esta mejora llega a un límite en el modelo 1 cuando utilizamos capas de tamaño 128. El siguiente aumento del tamaño ya no resulta en un aumento de la efectividad del modelo, si no en un ligero empeoramiento de este. Esto es debido a que el modelo empieza a hacer sobreajuste en ese punto. El límite del sobreajuste parece estar algo más alto en el modelo 3.

Abandono	**Modelo 1 (tam 256)**
0	0.97232
20 %	0.97202
50 %	**0.97682**

Tabla 3.3: Resultados tras añadir abandono al modelo 1.

Para paliar los efectos del sobreajuste en el modelo 1, realizamos una última ronda de experimentos añadiendo abandono al modelo 1 con 256 de tamaño. Los efectos fueron los mostrados en la tabla 3.3. En esta tanda de experimentos conseguimos el mejor resultado hasta el momento, una exactitud del $97,68\,\%$. Como podemos ver, un abandono

de 20 % no es suficiente para cambiar el desempeño del modelo, pero un abandono de 50 % si que implica una modesta pero significativa mejora. No obstante, la diferencia de rendimiento entre el modelo de tamaño 256 con abandono 50 % y el modelo de tamaño 128 es de tan solo 20 %. La decisión de si compensa o no este incremento del tamaño del modelo a cambio de esta muy ligera mejora del rendimiento se deberá tomar atendiendo principalmente a la capacidad de computación disponible.

Como conclusiones generales, podemos decir que el modelo 1 es sin duda la mejor opción entre las propuestas, debido tanto a su sencillez como a su rendimiento. El modelo obtiene resultados más que válidos ($96,31\,\%$) con capas de tamaño 32 y sin abandono. Sin embargo, es posible mejorar su rendimiento alrededor de un $1,3\,\%$ si recurrimos a aumentar el tamaño de las capas hasta 128 e incluso 256 si utilizamos un abandono de 50 %.

Capítulo 4

Reconocimiento de imágenes

4.1. Problema y datos de entrenamiento

Como aplicación práctica final de las técnicas de deep learning expuestas, realizaremos un modelo para la clasificación de imágenes médicas. El corpus[23] utilizado constará de 21173 imágenes de radiografías de pulmón etiquetadas respecto a cuatro categorías: Covid, Neumonía Vírica, Opacidad de Pulmón y Normal. La distribución de las imágenes es no balanceada como se muestra en la tabla 4.1.

Etiqueta	***COVID***	***Neumonía vírica***	***Opacidad pulmón***	***Normal***
Nº imágenes	3616	1345	6012	10200
Porcentaje del total	17,80 %	6,35 %	28,39 %	48,17 %

Tabla 4.1: Distribución de las imágenes de radiografías de pulmón.

El hecho de que el conjunto de datos tenga una distribución de clases no balanceadas puede dar lugar a modelos sesgados que no funcionen de forma correcta. Por ejemplo, un modelo que etiquetase todas las imágenes como normal tendría una exactitud de casi 50 % debido a que la clase normal esta sobrerepresentada en nuestro conjunto de datos. Para compensar esto, utilizaremos la funcionalidad de *class_ weight* que proporciona Tensorflow [24] por la cual se da más importancia a ciertas clases (las menos representadas) a la hora de calcular la función de pérdida. Los pesos que utilizaremos para cada categoría son los mostrados en la tabla 4.2 y para calcularlos utilizaremos la siguiente función:

$$pesoClase = \frac{Número\,total\,de\,muestras}{Número\,de\,clases \cdot Número\,de\,muestras\,de\,esta\,clase}$$

Etiqueta	***COVID***	***Neumonía vírica***	***Opacidad de pulmón***	***Normal***
Peso	1,46	3,89	0,88	0,52

Tabla 4.2: Valor numérico que usaremos para balancear cada clase en el conjunto de datos.

Por otro lado, modificaremos las imágenes usando la técnica del aumento de datos para que el modelo aprenda a generalizar mejor durante el entrenamiento. Las transformaciones que se realizarán a las imágenes serán estiramientos, rotación, espejo horizontal

y ampliación, todas en rangos de deformación bastante limitados (ver figura 4.1). Para ello, utilizaremos la función de Tensorflow llamada *ImageDataGenerator* con la siguiente configuración:

- **Estiramiento**: 15 %.
- **Rotación**: 12 grados en ambos sentidos.
- **Espejo horizontal** : 50 % de las imágenes.
- **Ampliación**: 10 %.

La separación de los datos en conjuntos de entrenamiento se realizará con la siguiente distribución: entrenamiento 80 % y validación 20 %. Debido a que no se harán gran cantidad de ajustes a los hiperparámetros basándonos en resultados ya obtenidos, los resultados entre el conjunto de validación y test serían poco notables, por lo que se ha decidido omitir el conjunto de test en nuestra separación.

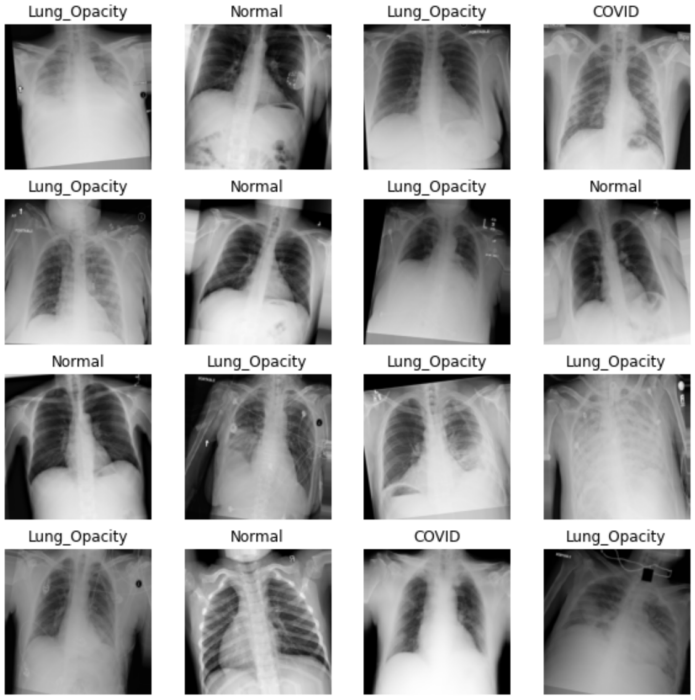

Figura 4.1: Algunas imágenes del conjunto de datos tras realizar las transformaciones

4.2. Arquitecturas propuestas y resultados

Como es natural, las redes apropiadas para tratar este problema son las redes convolucionales [25][26]. Sin embargo, hay muchas incógnitas a resolver para este problema en concreto (número de capas, tamaño de estas, capas convolucionales normales o separables, cuanto abandono aplicamos...). La forma más común de solventar estas incógnitas en el terreno del deep learning es comenzar con una red muy simple e ir añadiendo capas y tamaño de las capas a esta hasta que lleguemos al sobreajuste, así que esta será la forma en la que procederemos.

Debido al tamaño del conjunto de datos y a la complejidad de los modelos propuestos, se decidió por una opción de computación en linea para entrenarlos. Se utilizo la suscripción Colab Pro[27] que permite entrenar los modelos haciendo uso de una GPU NVIDIA T4[28] de forma remota. Con este hardware, cada modelo tardó en entrenarse alrededor de 8 horas.

4.2.1. Modelos convolucionales clásicos

La primer arquitectura propuesta será una estructura típica convolucional[29], la cual estará esencialmente formada por la reiteración de bloques convolucionales. Los bloques convolucionales pasarán su salida mediante una capa de allanamiento a un clasificador. El clasificador estará formado por una capa densa con activación relu, una capa de normalización por lotes, una de abandono al 50 % y una capa densa con softmax como función de activación. Siguiendo esta arquitectura, propondremos tres modelos:

- **2Conv_1Dense**: Modelo con dos bloques convolucionales (ver figura 4.2).
- **3Conv_1Dense**: Modelo con tres bloques convolucionales (ver figura 4.3).
- **4Conv_1Dense**: Modelo con cuatro bloques convolucionales (ver figura 4.4).

Respecto al tamaño de las capas, las primera capa convolucional tendrá tamaño 64 y cada capa convolucional posterior irá duplicando su tamaño hasta llegar a la cuarta con tamaño 512 (ver figura 4.4). La única capa densa del clasificador tendrá tamaño 512 y la capa densa final tendrá tamaño 4, pues es la que nos debe indicar en cual de las cuatro categorías cae cada imagen.

Utilizaremos la métrica de *exactitud* y también calcularemos la matriz de confusión final para cada modelo. Utilizaremos Adam como optimizador con una tasa de aprendizaje inicial del 0,001. La función de pérdida será la *entropía cruzada categórica* (del inglés Sparse Categorical Crossentropy).

También utilizaremos el callback *ReduceLROnPlateau*, una herramienta de Tensorflow que nos permite modificar la tasa de aprendizaje durante el entrenamiento. Nosotros multiplicaremos la tasa de aprendizaje por 0,1 cada vez que la exactitud en valoración lleve sin mejorar 7 épocas. Entrenamos por un total de 50 épocas con un tamaño de lote de 32 y los resultados fueron los mostrados en la tabla 4.3.

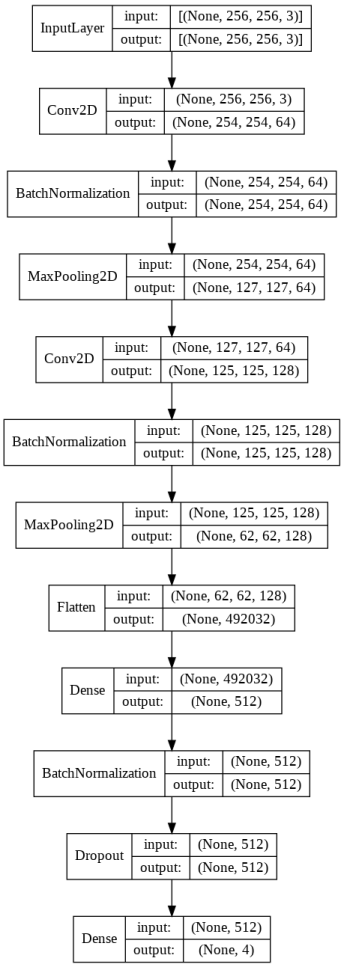

Figura 4.2: Arquitectura del modelo 2Conv_1Dense.

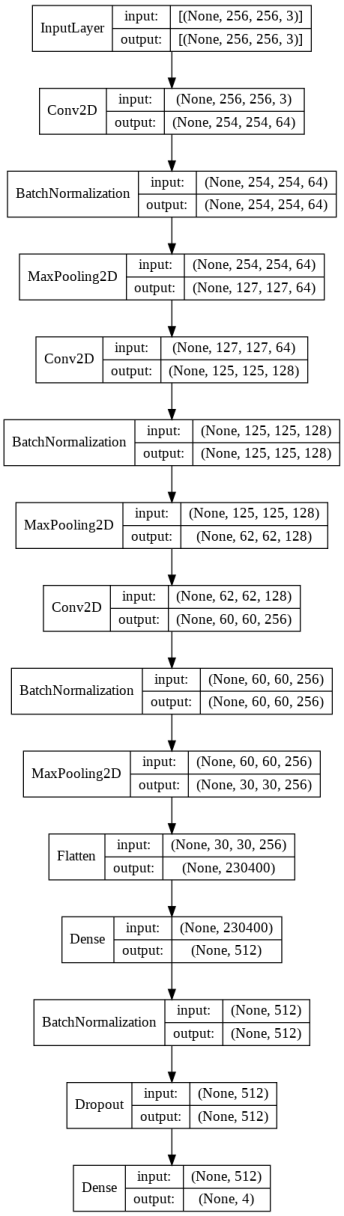

Figura 4.3: Arquitectura del modelo 3Conv_1Dense

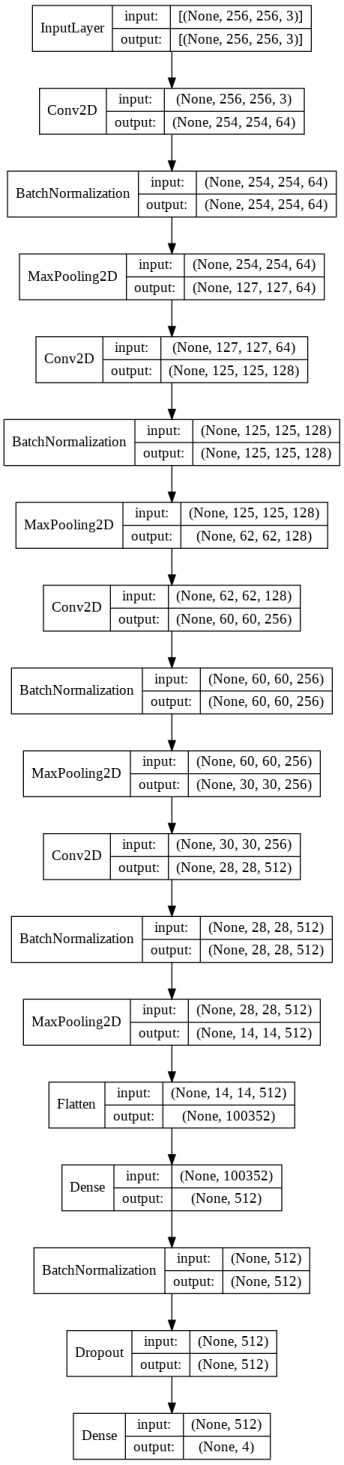

Figura 4.4: Arquitectura del modelo 4Conv_1Dense

Modelo	**Exactitud en validación(%)**
2Conv_1Dense	86,53
3Conv_1Dense	87,36
4Conv_1Dense	**90,34**

Tabla 4.3: Exactitud en validación de los modelos convolucionales.

Atendiendo a los resultados de la tabla 4.3 parece que el modelo se beneficia de tener cuantos más bloques convolucionales y no llega al sobreajuste con el cuarto bloque. Para hacer un análisis más a fondo del entrenamiento de los modelos mostraremos la gráfica de la exactitud en entrenamiento y validación durante el entrenamiento y la matriz de confusión resultante de cada modelo.

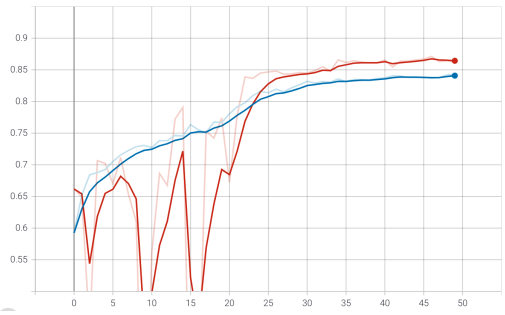

Figura 4.5: Exactitud en entrenamiento (rojo) y validación (azul) del modelo 2Conv_1Dense

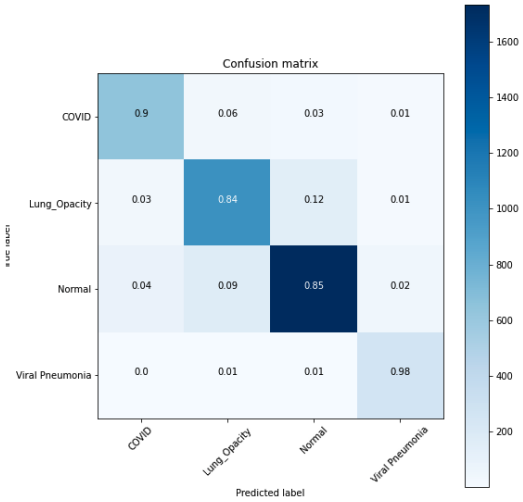

Figura 4.6: Matriz de confusión del modelo 2Conv_1Dense

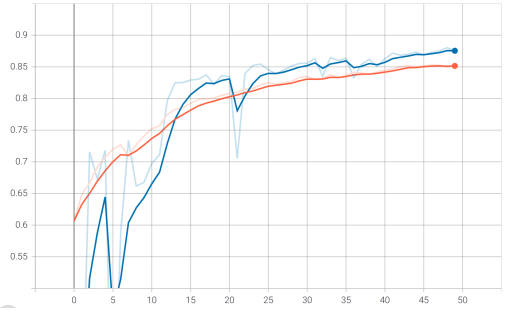

Figura 4.7: Exactitud en entrenamiento (naranja) y validación (azul) del modelo 3Conv_1Dense

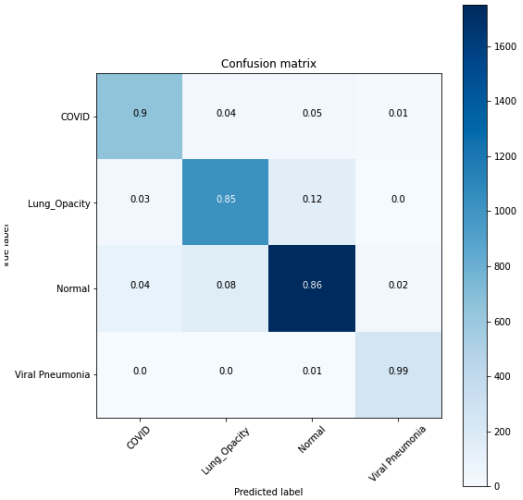

Figura 4.8: Matriz de confusión del modelo 3Conv_1Dense

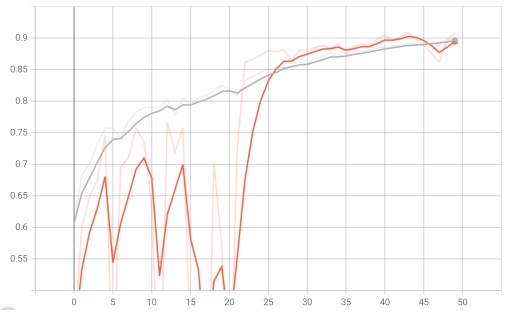

Figura 4.9: Exactitud en entrenamiento (gris) y validación (naranja) del modelo 4Conv_1Dense

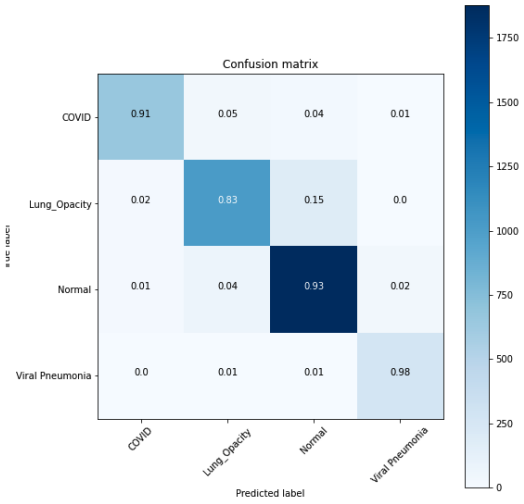

Figura 4.10: Matriz de confusión del modelo 4Conv_1Dense

Al observar las gráficas de exactitud (ver figuras 4.5, 4.7 y 4.9) observamos que las tres se comportan de forma similar. Inicialmente tienen un rendimiento oscilante en validación hasta la época 25 aproximadamente, a partir de ahí se comportan de forma asintótica con apenas variación. Curiosamente, en todas las gráficas observamos que la exactitud en validación termina siendo ligeramente mayor que la de entrenamiento (unos dos puntos porcentuales por encima). Esto podría ser debido a que el conjunto de validación no es una muestra lo suficientemente amplia de los datos, lo cual no parece probable ya que son alrededor de 4000 fotos para representar 4 clases. Otra explicación posible sería que estuviésemos usando un abandono excesivo, aunque una sola capa de abandono al 50 % a priori no parece demasiado. Utilizaremos pues las matrices de confusión para asegurarnos que nuestros modelos están rindiendo como se espera.

Como podemos ver en la figuras 4.6, 4.8 y 4.10 las matrices de confusión son también similares entre los tres modelos y reflejan un comportamiento correcto. Los modelos tienen rendimientos de, al menos, 84 % para todas las etiquetas. Estos rinden ligeramente mejor en la etiquetas con menos representación (COVID y neumonía viral) que en las de normal y opacidad pulmonar, las cuales tienen una ligera tendencia a confundir entre sí. La principal diferencia entre el model Conv3_1Dense y el modelo Conv4_1Dense es que el segundo actúa ligeramente peor con la opacidad de pulmón a cambio de una mejora en la precisión en la etiqueta normal.

4.2.2. Modelos convolucionales separables

Para continuar con nuestra experimentación, propondremos una segunda arquitectura utilizando capas convolucionales separables[19] en vez de convolucionales clásicas. Como buscamos comparar los resultados con los de la sección 4.2.1, trataremos de realizar los mínimos cambios a los modelos, para que las diferencias se expliquen exclusivamente por el tipo de capa convolucional utilizada.

En concreto, utilizaremos una arquitectura copiada de la de la sección 4.2.1 pero cambiando cada capa convolucional por dos capas convolucionales separables del mismo tamaño que la convolucional anterior. El motivo de utilizar dos capas en vez de una es que las

capas convolucionales separables son mucho más ligeras que las convolucionales clásicas, hasta tal punto que dos capas convolucionales separables siguen añadiendo menos parámetros al modelo que una convolucional clásica.

Siguiendo esta arquitectura, proponemos los siguientes modelos:

- **2ConvSep_1Dense**: Modelo con dos bloques convolucionales separables (ver figura 4.11).
- **3ConvSep_1Dense**: Modelo con tres bloques convolucionales separables (ver figura 4.12).
- **4ConvSep_1Dense**: Modelo con cuatro bloques convolucionales separables (ver figura 4.13).

Pasemos a comentar la tabla 4.4 con los resultados de la experimentación. En ella vemos que los modelos convolucionales separables superan siempre a los convolucionales clásicos. Sin embargo, en este caso si que llegamos al sobreajuste, pues vemos que el modelo 3ConvSep_1Dense rinde ligeramente mejor que el modelo 4ConvSep_1Dense, lo que indica que este último bloque convolucional separable añadido hace que el modelo pierda capacidad de generalización. Como hemos hecho en la sección 4.2.1, pasaremos a estudiar las gráficas de exactitud y las matrices de convolución para poder hacer un análisis más profundo de los modelos.

Modelo	**Exactitud en val(%)**	**Modelo**	**Exactitud en val(%)**
2Conv_1Dense	86,53	*2ConvSep_1Dense*	87,69
3Conv_1Dense	87,36	*3ConvSep_1Dense*	**91,59**
4Conv_1Dense	90,34	*4ConvSep_1Dense*	90,97

Tabla 4.4: Exactitud en validación de los modelos convolucionales y los convolucionales separables.

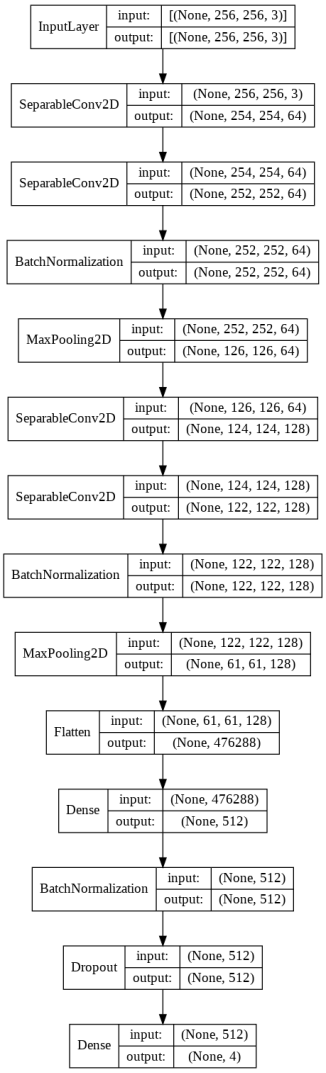

Figura 4.11: Arquitectura del modelo 2Conv-Sep_1Dense.

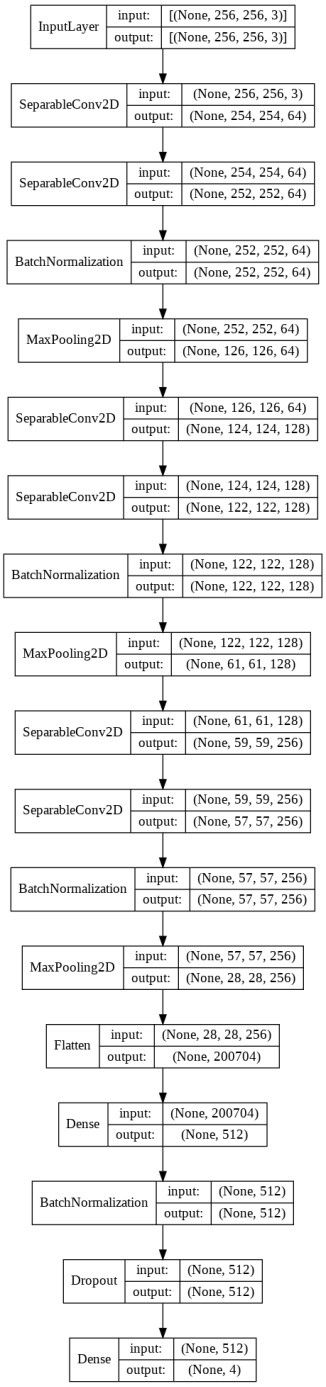

Figura 4.12: Arquitectura del modelo 3Conv-Sep_1Dense.

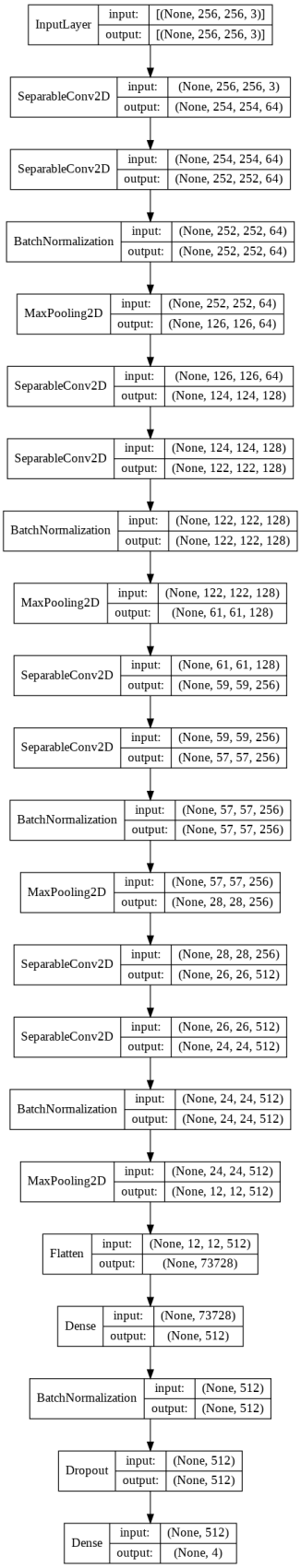

Figura 4.13: Arquitectura del modelo 4Conv-Sep_1Dense.

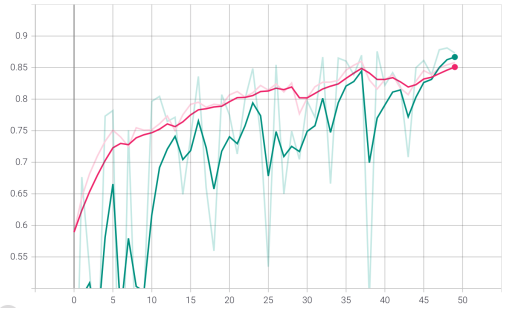

Figura 4.14: Exactitud en entrenamiento (rosa) y validación (verde) del modelo 2ConvSep_1Dense

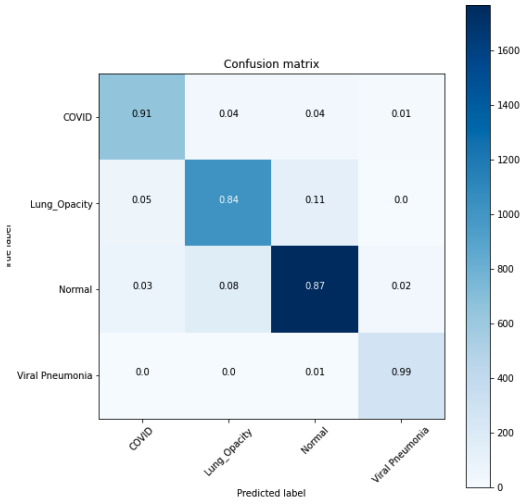

Figura 4.15: Matriz de confusión del modelo 2ConvSep_1Dense

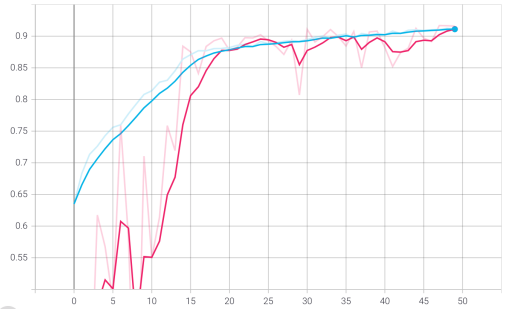

Figura 4.16: Exactitud en entrenamiento (azul) y validación (rosa) del modelo 3ConvSep_1Dense

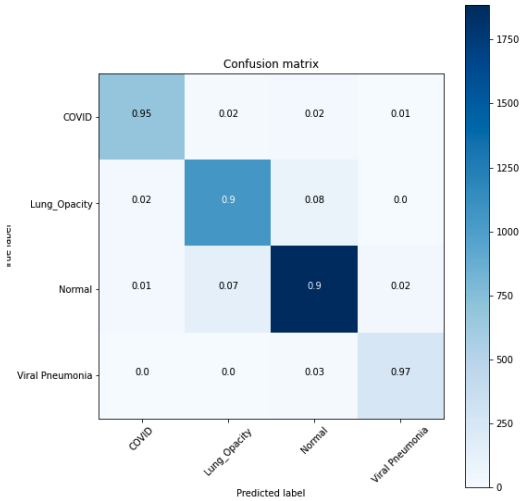

Figura 4.17: Matriz de confusión del modelo 3ConvSep_1Dense

Al observar las figuras 4.14, 4.16 y 4.18 lo primero que llama la atención es que la exactitud en validación oscila más que las gráficas de las sección 4.2.1. Esto probablemente se deba a que las capas convolucionales separables, al tener menos parámetros, son más sensibles a los cambios que las convolucionales clásicas. También vemos que la exactitud en validación ya no esta por encima de la de validación, lo que podría indicar que las capas convolucionales separables admiten más abandono que las convolucionales clásicas. Respecto al comportamiento, todas las gráficas parecen comportarse de forma asintótica respecto a un valor límite, como se esperaba.

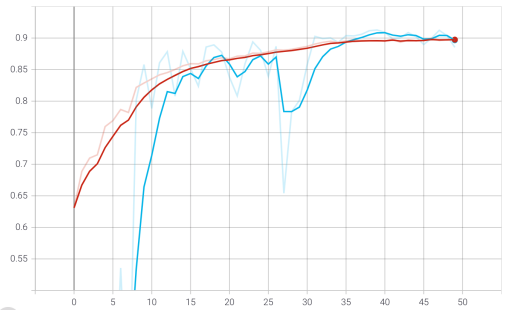

Figura 4.18: Exactitud en entrenamiento (naranja) y validación (azul) del modelo 4ConvSep_1Dense

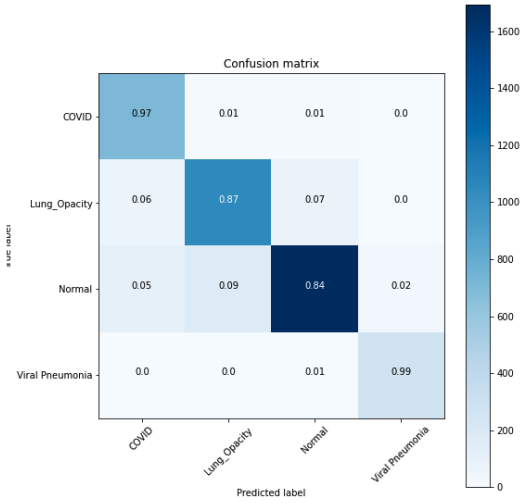

Figura 4.19: Matriz de confusión del modelo 4ConvSep_1Dense

Respecto a las matrices de confusión de las figuras 4.15, 4.17 y 4.19, también muestran los resultados esperados. Se mantiene la tendencia de los modelos de etiquetar ligeramente mejor las etiquetas de COVID y neumonía viral, mientras que sigue confundiendo en ocasiones la etiqueta normal con la de opacidad de pulmón.

4.2.3. Modelos convolucionales separables con mayor clasificador

Hasta el momento, solo hemos modificado la parte representativa de los modelos, es decir, la que incluye las capas convolucionales. En esta sección cogeremos los modelos más exitosos hasta el momento y ampliaremos su clasificador viendo si así podemos mejorar los resultados de exactitud algún punto porcentual más.

En concreto, cogeremos los modelos 3ConvSep_1Dense y 4ConvSep_1Dense y aumentaremos el número de bloques densos que contienen. Como bloque denso entenderemos una capa densa seguida de una de normalización por lotes y una de abandono al 50 %. El tamaño de las capas densas lo mantendremos fijo en 512. Probaremos, para cada modelo, usar dos y tres capas densas, lo que da lugar a los siguientes modelos:

- **3ConvSep_2Dense**: Modelo con tres bloques convolucionales separables y dos bloques densos (ver figura 4.20).
- **3ConvSep_3Dense**: Modelo con tres bloques convolucionales separables y tres bloques densos (ver figura 4.21).
- **4ConvSep_2Dense**: Modelo con cuatro bloques convolucionales separables y dos densos (ver figura 4.22).
- **4ConvSep_3Dense**: Modelo con cuatro bloques convolucionales separables y tres densos (no lo mostramos debido a su gran tamaño).

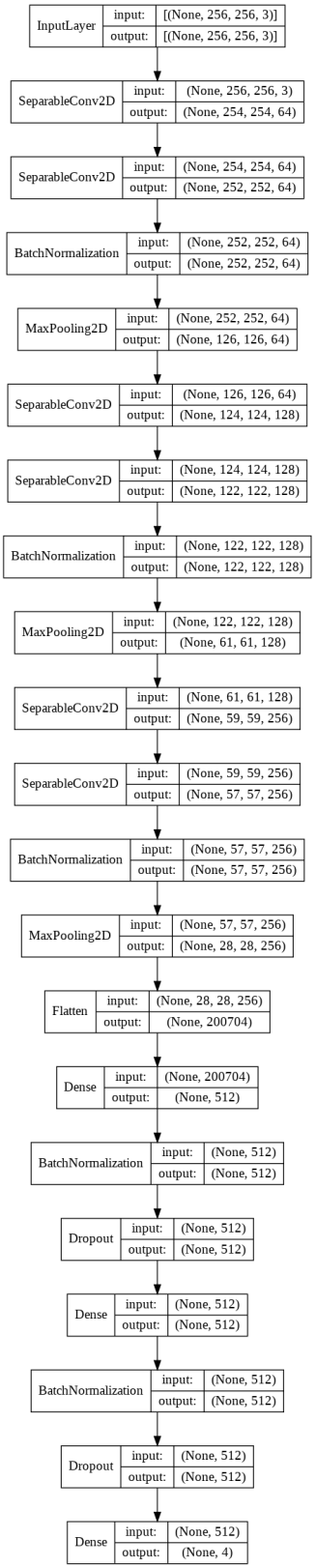

Figura 4.20: Arquitectura del modelo 3Conv-Sep_2Dense.

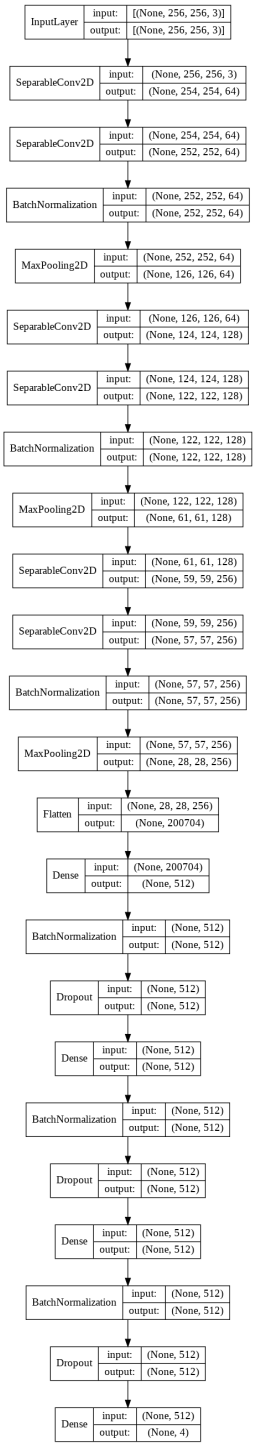

Figura 4.21: Arquitectura del modelo 3Conv-Sep_3Dense.

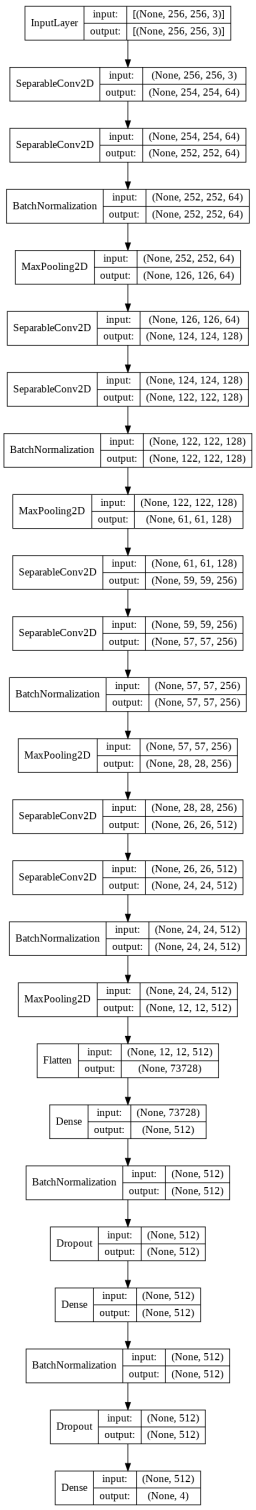

Figura 4.22: Arquitectura del modelo 4Conv-Sep_2Dense.

Modelo	Exactitud val(%)	Modelo	Exactitud val(%)
3ConvSep_1Dense	91,59	*4ConvSep_1Dense*	90,97
3ConvSep_2Dense	91,89	*4ConvSep_2Dense*	**92,91**
3ConvSep_3Dense	91,68	*4ConvSep_3Dense*	92,27

Tabla 4.5: Exactitud de los modelos con tres y cuatro bloques convolucionales variando el número de bloques densos.

Como vemos en los resultados de exactitud de la tabla 4.5, los modelos sí se benefician del segundo bloque denso, pero al añadir el tercero empiezan a sobreajustar. La mejora es especialmente notable entre el modelo 4ConvSep_1Dense y el modelo 4ConvSep_2Dense, pues la exactitud aumenta en casi dos puntos porcentuales. Pasemos a estudiar las gráficas de la exactitud y las matrices de confusión para obtener una visión más profunda de estas mejoras en rendimiento.

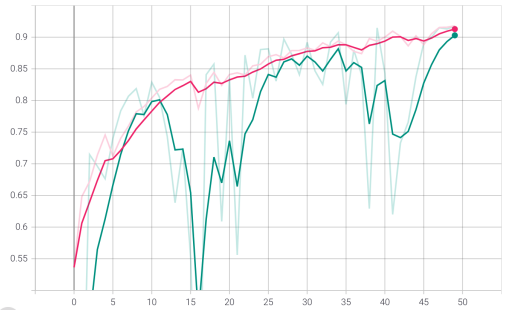

Figura 4.23: Exactitud en entrenamiento (rosa) y validación (verde) del modelo 3ConvSep_2Dense

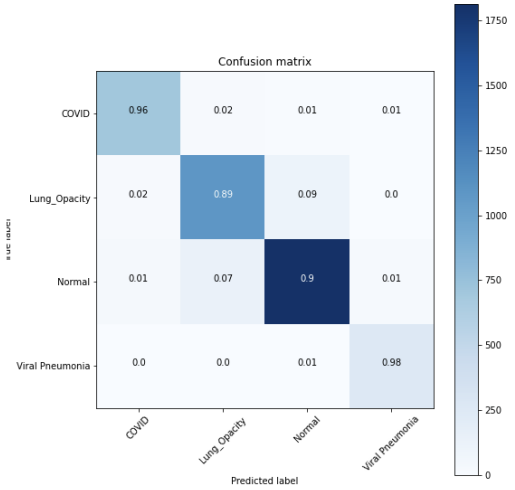

Figura 4.24: Matriz de confusión del modelo 3ConvSep_2Dense

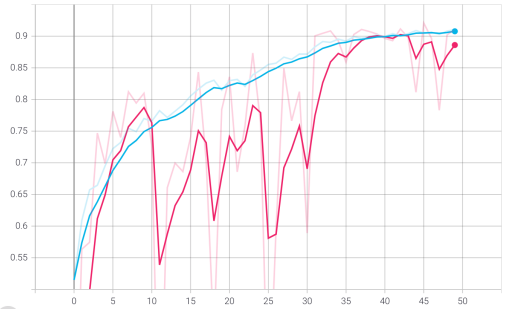

Figura 4.25: Exactitud en entrenamiento (azul) y validación (rosa) del modelo 3ConvSep_3Dense

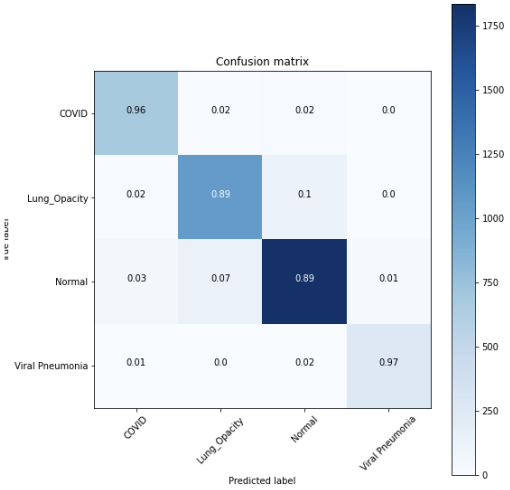

Figura 4.26: Matriz de confusión del modelo 3ConvSep_3Dense

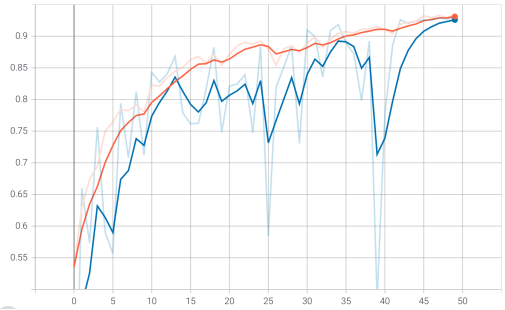

Figura 4.27: Exactitud en entrenamiento (naranja) y validación (azul) del modelo 4ConvSep_2Dense

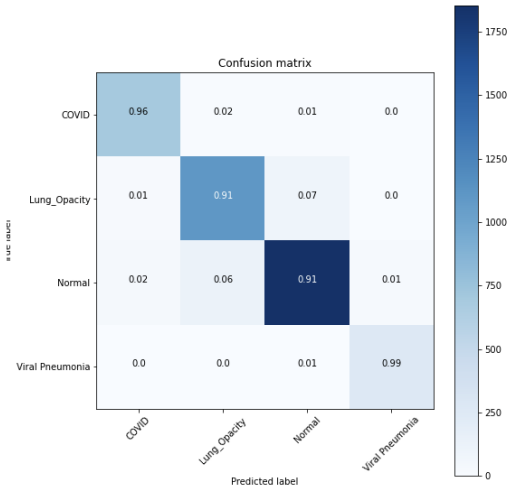

Figura 4.28: Matriz de confusión del modelo 4ConvSep_2Dense

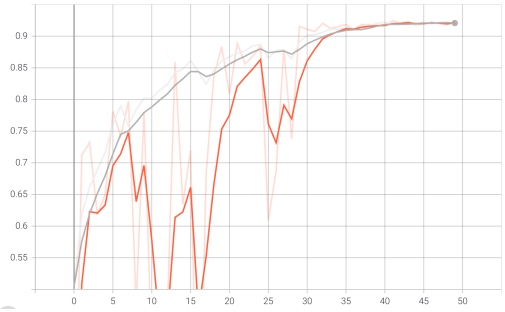

Figura 4.29: Exactitud en entrenamiento (gris) y validación (naranja) del modelo 4ConvSep_3Dense

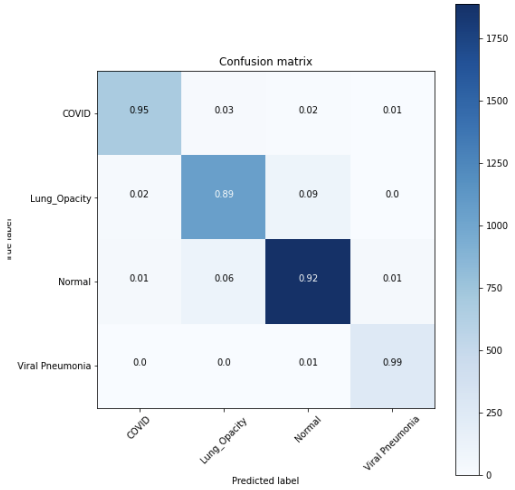

Figura 4.30: Matriz de confusión del modelo 4ConvSep_3Dense

Al observar las gráficas de exactitud de las figuras 4.23, 4.25, 4.27 y 4.29 lo más llamativo es que las oscilaciones comentadas en la sección 4.2.2 son aún más acusadas en estos casos. A estos modelos les cuesta más converger hacia una exactitud límite. Esto teniendo en cuenta que disminuimos la tasa de aprendizaje durante el entrenamiento, es decir, que el entrenamiento de los modelos acabó con una tasa de aprendizaje de $0,00001$. Por otra parte, en estos modelos la exactitud en validación no supera la de entrenamiento, lo cual es tranquilizador.

Respecto a las matrices de confusión de las figuras 4.24, 4.26, 4.28 y 4.30 no vemos grandes cambios. Se mantienen las tendencias mencionadas en las secciones 4.2.1 y 4.2.2, pero con un aumento en la precisión en las etiquetas Normal y Opacidad de Pulmón.

Capítulo 5

Conclusiones

El machine learning es una de las tecnologías que mayor impacto ha tenido en nuestras vidas durante los últimos años. En concreto, el deep learning nos abre la puerta a un inmenso mundo de posibilidades donde los ordenadores resuelven problemas que hace tan solo unas décadas se pensaban demasiado complejos para estos.

Los modelos de redes neuronales permiten enseñar a las máquinas a desempeñar tareas que los propios humanos tardan meses o años en dominar, como pueden ser la conducción autónoma, el reconocimiento de patologías en una imagen médica, la interpretación de un texto o jugar a juegos complejos como el ajedrez. Sin embargo, a pesar de todos los logros que ya ha obtenido este campo, la tecnología sigue en constante evolución y es muy probable que siga aumentando el número de tareas que facilita y soluciona con el paso del tiempo.

Para aplicar el potencial de la tecnología, se ha centrado la investigación en los modelos de deep learning para el procesamiento y reconocimiento de imágenes, también llamados redes convolucionales. Tras una introducción teórica de los conceptos claves de las redes convolucionales en la actualidad, se ha propuesto un problema sencillo de reconocimiento de letras el cual se ha conseguido resolver con bastante éxito (97,68 % de exactitud en valoración) haciendo uso de estructuras convolucionales sencillas.

Para finalizar, se ha propuesto solucionar un problema de actualidad como es el reconocimiento de COVID-19 en una radiografía de pulmón. Para ello, se obtuvo un corpus de 21173 imágenes etiquetadas en cuatro categorías y se propusieron una serie de modelos que iban aumentando progresivamente de complejidad. Comparando los diferentes modelos, se ha llegado a la conclusión de que el modelo se beneficiaba de mayor profundidad hasta un límite de sobreajuste que se situaba en la elección de cuatro capas convolucionales y dos densas. Además, el uso de capas convolucionales separables ha demostrado ser superior al de las capas convolucionales clásicas para el problema planteado. Los resultados fueron de nuevo muy satisfactorios llegando a obtener un 92,91 % de exactitud en el conjunto de valoración.

Se espera que este trabajo pueda servir de guía a cualquier persona que quiera adentrarse en el mundo del deep learning y más en concreto en el de los modelos convolucionales. Además, aunque la elección de modelos en deep learning está fuertemente condicionada por el problema concreto que se plantea, la resolución del problema de clasificación de

radiografías de pulmón podrá servir como referencia a la hora de plantear cualquier modelo para la clasificación de imágenes con múltiples etiquetas.

Bibliografía

[1] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2018.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] "Why tensorflow." https://www.tensorflow.org/. Última consulta: 2021-05-02.

[4] "Pytorch." https://pytorch.org/. Última consulta: 2021-05-02.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] P. Lakhani and B. Sundaram, "Deep learning at chest radiography: Automated classification of pulmonary tuberculosis by using convolutional neural networks," *Radiology*, vol. 284, no. 2, pp. 574–582, 2017.

[7] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: A comprehensive review," *ACM Comput. Surv.*, vol. 54, Apr. 2021.

[8] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[9] J. B. Heaton, N. G. Polson, and J. H. Witte, "Deep learning for finance: deep portfolios," *Applied Stochastic Models in Business and Industry*, vol. 33, no. 1, pp. 3–12, 2017.

[10] L. Rice, E. Wong, and Z. Kolter, "Overfitting in adversarially robust deep learning," in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 8093–8104, PMLR, 13–18 Jul 2020.

[11] Sklearn, "Metrics and scoring: quantifying the quality of predictions." https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics. Última consulta: 2021-05-02.

[12] "Tensorflow, dense layer." https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense. Última consulta: 2021-05-02.

[13] S. S. S Sharma and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, pp. 310–316, 2020.

[14] S. Bock and M. Weiß, "A proof of local convergence for the adam optimizer," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019.

[15] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[16] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[17] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[18] J. Bjorck, C. P. Gomes, and B. Selman, "Understanding batch normalization," *CoRR*, vol. abs/1806.02375, 2018.

[19] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, p. 139–144, Oct. 2020.

[21] Y. Wang, F. Li, H. Sun, W. Li, C. Zhong, X. Wu, H. Wang, and P. Wang, "Improvement of MNIST image recognition based on CNN," *IOP Conference Series: Earth and Environmental Science*, vol. 428, p. 012097, jan 2020.

[22] C. Kusuma, A. Fadhilah, and A. Afiahayati, "Convolutional neural networks for handwritten javanese character recognition," *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, vol. 12, p. 83, 01 2018.

[23] "Covid-19 radiography database." `https://www.kaggle.com/tawsifurrahman/covid19-radiography-database`. Última consulta: 2021-05-02.

[24] Tensorflow, "Classification on imbalanced data." `https://www.tensorflow.org/tutorials/structured_data/imbalanced_data`. Última consulta: 2021-05-02.

[25] H. Farhat, G. E. Sakr, and R. Kilany, "Deep learning applications in pulmonary medical imaging: recent updates and insights on covid-19," *Machine Vision and Applications*, vol. 31, no. 6, p. 53, 2020.

[26] Y. Dong, Y. Pan, J. Zhang, and W. Xu, "Learning to read chest x-ray images from 16000+ examples using cnn," in *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pp. 51–57, 2017.

[27] "Colab pro." `https://colab.research.google.com/signup`. Última consulta: 2021-06-28.

[28] "Nvidia t4 tensor core gpu for ai inference." https://www.nvidia.com/en-us/data-center/tesla-t4/. Última consulta: 2021-06-28.

[29] A. Choi, O. Stephen, M. Sain, U. J. Maduh, and D.-U. Jeong, "An efficient deep learning approach to pneumonia classification in healthcare," *Journal of Healthcare Engineering*, vol. 2019, p. 4180949, 2019.

Daniel Valverde Menasalvas
2021
Ult. actualización 29 de junio de 2021
LaTeX lic. LPPL & powered by TEFLON CC-ZERO