Trabajo de final de grado

Grado en Ingeniería en Tecnologías Industriales

Diseño de un modelo Deep Learning de clasificación de imágenes para la detección de la neumonía

MEMORIA

Autor: Ana Blasi Sanchiz
Director: Jordi Fonollosa Magrinya
Ponente: Samir Kanaan Izquierdo
Convocatoria: Junio 2021

Escuela Técnica Superior de
Ingeniería Industrial de Barcelona

RESUMEN

Unas décadas atrás, nació Deep Learning, una técnica que revolucionó el mundo de la Inteligencia Artificial tal y como se conocía. Esta imita el funcionamiento neuronal de un cerebro humano para clasificar conjuntos de datos y encontrar patrones entre ellos.

Gracias a este sistema computacional, en este trabajo se ha podido diseñar un modelo clasificatorio de imágenes que detecte la neumonía. Para ello, se ha hecho uso de la metodología Transfer Learning. Esta se basa en utilizar modelos entrenados previamente para crear el nuevo modelo. Además se ha utilizado las redes neuronales convolucionales como extractoras de patrones.

Se ha decidido identificar esta patología porque la neumonía es una infección respiratoria que, si se complica, puede llegar a ser letal, sobre todo en niños menores de cinco años. Del mismo modo, se ha creado este modelo con el objetivo de que se pueda utilizar en estudios clínicos para así poder encontrar nuevos tratamientos o controlar la enfermedad.

Para crear el modelo, se ha utilizado la librería Fast.ai. debido a que tiene componentes de alto nivel que permiten proporcionar resultados de forma más sencilla y rápida. El conjunto de datos utilizados para crear el modelo se ha extraído de la página web *Kaggle*.

Como se quería obtener el mejor rendimiento posible, se han realizado cuatro modelos distintos a partir de cuatro modelos pre entrenados; ResNet 18, ResNet 34, ResNet 50 y AlexNet. Al final, se han acabado obteniendo resultados muy buenos a partir de todos los modelos, incluso en tres de ellos se ha alcanzado una precisión mayor del 99%. Aun así, ha sido a partir de ResNet 50 que se ha obtenido el mejor rendimiento y con el que se ha creado el programa que consigue detectar la neumonía.

ABSTRACT

A few decades ago, Deep Learning appeared revolutionizing the world of Artificial Intelligence as it was known. This technique mimics the neural functioning of a human brain to classify data sets and to find patterns among them.

Thanks to this computational system, it has been possible to design through this project a classifying model of images that detects pneumonia. Transfer Learning has been the methodology used to achieve so. This method is based on using previously trained models to create the new model. Furthermore, convolutional neural networks have been used as pattern extractors.

It was decided to identify this pathology because pneumonia is a respiratory infection that, if it gets bad, can be lethal, especially in children under five years old. Likewise, this model has been created with the aim of being used in clinical studies in order to find new treatments and to control the disease.

To create the model, the Fast.ai. library has been used because it has high-level components that can quickly and easily provide results. The dataset used to create the model was extracted from the Kaggle website.

As it was wanted to obtain the best possible performance, four different models have been made from four pre-trained models; ResNet 18, ResNet 34, ResNet 50 and AlexNet. In the end, very good results have been obtained from all the models, even in three of them an accuracy of more than 99% has been achieved. However, the best performance was obtained from ResNet 50. It was with this model that the program that detects pneumonia was created.

GLOSARIO

AI: Inteligencia artificial.

CNN: Redes neuronales convolucionales.

Dataset: Conjunto de datos.

Deep learning: Aprendizaje profundo.

Fine-tuning: Técnica que permite realizar cambios al modelo para obtener mejores resultados.

Kernel: Matrices que se utilizan en la convolución.

Machine learning: Aprendizaje automático.

Pooling: Técnica para comprimir datos disminuyendo parámetros

ReLu: Función de activación que pone los valores negativos a cero.

Softmax: Función de activación que clasifica datos.

Transfer learning: Aprendizaje transferido

Weights: Peso adaptativo de las redes neuronales.

ÍNDICE

ÍNDICE DE TABLAS, ECUACIONES Y CÓDIGOS

ÍNDICE DE FIGURAS

1. PREFACIO

1.1. Origen del trabajo

El trabajo de fin de grado presentado a continuación lleva el título de "Diseño de un modelo Deep Learning de clasificación de imágenes para la detección de la neumonía". El origen de este proyecto surge tras la idea de querer realizar un algoritmo capaz de clasificar imágenes a partir de los conocimientos de deep learning. El modelo podrá identificar la patología de la neumonía. Para realizarlo, se utilizará la librería Fast.ai.

1.2. Motivación

La motivación de este proyecto surgió tras realizar unas prácticas profesionales en la empresa Napptilus Tech cinco meses antes de empezar este proyecto. Esta experiencia laboral me acercó al mundo del deep learning y *data science*, por lo que tuve la oportunidad de trabajar con bases de datos y así poder desarrollar los primeros pasos para este proyecto.

1.3. Requerimientos previos

Para la ejecución de este trabajo ha sido necesario contener una serie de requerimientos previos. Se debía poseer una base de programación en Python y ciertos conocimientos sobre deep learning y sobre redes neuronales. En las prácticas profesionales se exigió la visualización del curso de Fast.ai sobre deep learning y clasificación de imágenes (37). También, se procedió al estudio del libro "Deep Learning with Python" de François Chollet (1). Además, durante el realizamiento de este trabajo, el autor ha cursado la asignatura *Laboratory of machine learning* con Agostino Di Ciaccio, como profesor, en la Universidad de Roma, Sapienza.

2. INTRODUCCIÓN

En el mundo de la inteligencia artificial nació hace unos años machine learning, la ciencia donde el ordenador aprende y mejora automáticamente a partir de experiencias. El interés ha ido aumentando con la aparición de deep learning. Este es un tipo de aprendizaje que trabaja a partir de redes neuronales artificiales. Las dos prácticas se emplean en el día a día sin que el usuario se dé cuenta. Se encuentran en vehículos autónomos, en apps de reconocimiento de voz automático y en clasificadores de imágenes entre muchos otros. Deep learning, empleando dichas redes, ha conseguido mejorar la clasificación, el reconocimiento, la detección y la descripción.

A su vez, la medicina también ha avanzado exponencialmente en todos sus ámbitos en las últimas décadas. En una era donde la tecnología y la computación definen el cambio, nace la necesidad de encontrar nuevos recursos y nuevas técnicas para mejorar el diagnóstico y la planificación de tratamientos y estudios. Normalmente es el médico el que llega a un juicio final sobre el diagnóstico del enfermo, a partir de radiografías o análisis. Aun así, surge la necesidad de obtener resultados con una mayor rapidez. Como por ejemplo, a raíz de la creación de unos estudios clínicos donde se exige examinar un gran volumen de datos. De ahí nace el objetivo de este trabajo.

El trabajo se dividirá en dos partes, una parte más teórica y otra más práctica.

En la parte teórica, primero se hará una breve introducción de la inteligencia y se adentrará en el mundo de machine learning y deep learning. Se explicarán qué son las redes neuronales artificiales y en concreto las redes neuronales convolucionales. (capítulo 3).

La segunda parte se basa en la creación de un modelo que clasifique imágenes utilizando los conocimientos teóricos de la parte anterior. Este modelo conseguirá identificar si en una radiografía se observa la infección de neumonía o no. Para ello, se explicará qué metodología se ha seguido para crear el modelo. En este, se hará un repaso sobre las bases biológicas de la neumonía y se explicará cómo se ha utilizado la metodología transfer learning para crear el modelo. A continuación, se analizarán los resultados obtenidos llegando a una serie de

conclusiones y se realizará el programa que clasifique diferentes imágenes para detectar la neumonía (capítulo 5).

Por último, se explicará qué planificación se ha seguido para realizar el trabajo (capítulo 6), y se expondrán cuáles han sido los impactos medioambientales y económicos (capítulos 7 y 8).

2.1 Objetivos

El objetivo final del trabajo es utilizar conocimientos de deep learning para crear un modelo capaz de clasificar imágenes y que identifique cuando un paciente padece neumonía o no. Este modelo deberá tener una buena actuación, que los resultados den un *accuracy* cerca del 100%.

Para poder llevar a cabo este objetivo final, se ha tenido que plantear una serie de objetivos más concretos.

El primero es aprender y revisar conocimientos de machine learning y deep learning. En concreto sobre las redes neuronales convolucionales.

También, estudiar el método de transfer learning y sus técnicas; las redes neuronales convolucionales como extractoras de patrones y fine_tuning.

El siguiente objetivo es ser capaz de realizar un algoritmo utilizando metodologías explicadas. Como se quiere conseguir resultados muy precisos, se realizará cuatro modelos diferentes a partir de cuatro arquitecturas pre entrenadas.

Comparar los resultados obtenidos a partir de los diferentes modelos construidos.

Por último, realizar un programa clasificador a partir del modelo que ha obtenido mayor precisión.

3. MACHINE LEARNING Y DEEP LEARNING

3.1. Inteligencia artificial

En los últimos años las principales potencias del mundo han dirigido gran parte de sus recursos al desarrollo de la inteligencia artificial. Tal y como comentó Vladímir Putin, Presidente de la Federación Rusa; "La inteligencia artificial es el futuro, no solo para Rusia, sino para toda la humanidad. Viene con oportunidades colosales, pero también con amenazas difíciles de predecir. Quien se convierta en el líder en esta esfera se convertirá en el gobernante del mundo". Dicha tecnología está creciendo a pasos agigantados sobre todo en algunos países, en concreto Estados Unidos de América y China.

La inteligencia artificial (AI) es una amplia rama de la informática que se centra en construir máquinas ingeniosas capaces de realizar tareas que requieren inteligencia humana. A partir de los años cuarenta, cuando se inició el desarrollo del ordenador digital, se pudo probar que estos podían realizar tareas muy complejas tras programarlos adecuadamente. Como por ejemplo descubrir pruebas de teoremas matemáticos o jugar al ajedrez (2). Aun así, todavía no hay ordenadores capaces de actuar como un ser humano. El objetivo final de la inteligencia artificial, por lo tanto, es crear programas donde los ordenadores puedan aprender y solucionar problemas con la misma flexibilidad que un humano.

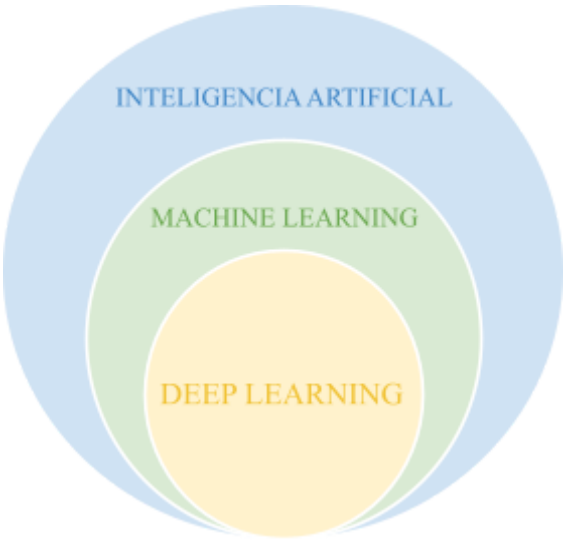


Figura 1: Inteligencia artificial, machine learning y deep learning. (Fuente: (3))

Dentro del mundo de la inteligencia artificial se encuentra machine learning, y dentro de esta, deep learning. Gracias a estos dos subconjuntos los sistemas AI están empezando a cumplir los propósitos y han conseguido cambiar paradójicamente muchos sectores de la industria tecnológica.

3.2. Machine learning

Machine learning, en español llamado aprendizaje automático, es la ciencia que logra que un ordenador actúe sin que esté explícitamente programado. En los últimos años, machine learning ha participado en motores de búsqueda de Internet, filtros de correo electrónico para clasificar el spam, sitios web para hacer recomendaciones personalizadas, software bancario para detectar transacciones inusuales y muchas aplicaciones en nuestros teléfonos, como el reconocimiento de voz. Ha propiciado una búsqueda más efectiva de páginas web además de mejorar el entendimiento sobre el genoma humano. Esta práctica ha crecido tanto que la mayoría de las personas lo utilizan una docena de veces al día sin darse cuenta (6).

Básicamente, machine learning emplea algoritmos para analizar datos, aprender de ellos y, finalmente, poder realizar una predicción o sugerencia sobre algo. El proceso de aprendizaje empieza con observaciones, datos, experiencias y/o instrucciones que se analizan para encontrar patrones y, así, poder tomar mejores decisiones sobre el futuro basadas en los ejemplos que se ofrecen. La finalidad es permitir que los ordenadores aprendan automáticamente sin la ayuda del ser humano ni ninguna asistencia y que se ajuste con las acciones pertinentes (8).

Machine learning también se puede definir como un proceso en el que recopilando un *dataset* formado por unos datos entrenados y teniendo una respuesta clara, a partir de un algoritmo, se puede obtener un modelo estadístico basado en ese conjunto de datos (7).

Figura 2: Metodología genérica de machine learning. (Fuente:(7))

3.3. Deep learning

Deep learning, en español conocido como aprendizaje profundo, es un subcampo de aprendizaje automático que se centra en los algoritmos basados en la estructura y función del cerebro llamado redes neuronales artificiales. En esta práctica se emplean configuraciones lógicas que se parecen en gran medida a la estructura de un sistema nervioso central. A partir de unos objetivos específicos, se tienen unas capas de unidades de proceso, neuronas artificiales, que se dedican a detectar determinadas características (9).

También, a raíz de deep learning nació un tipo de aprendizaje representativo hacia los datos donde se prioriza el aprendizaje por capas (l*ayers*) sucesivas de representaciones cada vez más significativas. La palabra profundo (*deep*) tiene una importancia semántica que se refiere a esta imagen de capas sucesivas, más capas más profundo es el aprendizaje y, por lo tanto, el modelo (1). Y estas son las llamadas redes neuronales.

Por último hace falta resaltar que, a diferencia de otros algoritmos de aprendizaje, deep learning mejora sus resultados cuanto mayor es la red neuronal y más datos tiene.

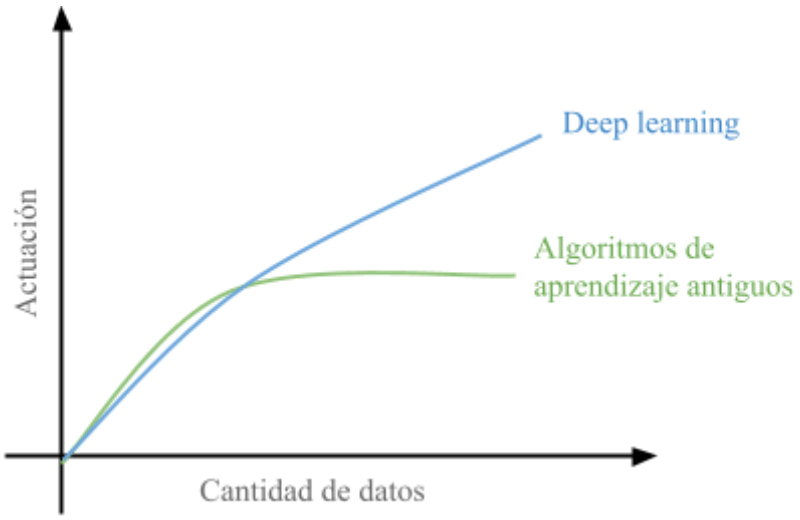


Figura 3: La evolución de la actuación de un modelo en los algoritmos de aprendizaje antiguos y en los de deep learning. (Fuente:(10))

3.4. Redes neuronales

3.4.1. Descripción general de las redes neuronales biológicas

El cerebro humano funciona internamente a partir de neuronas y de redes neuronales conocidas como redes neuronales biológicas. Se ha hecho una estimación donde, según Wikipedia, este contiene aproximadamente 100.000 millones de neuronas unidas entre sí por redes [(11) Diapositiva: 2].

A un nivel muy elevado, las neuronas interactúan y se comunican entre ellas mediante el nexo de unión entre los terminales de axones que están conectados a las dendritas a través de una brecha.

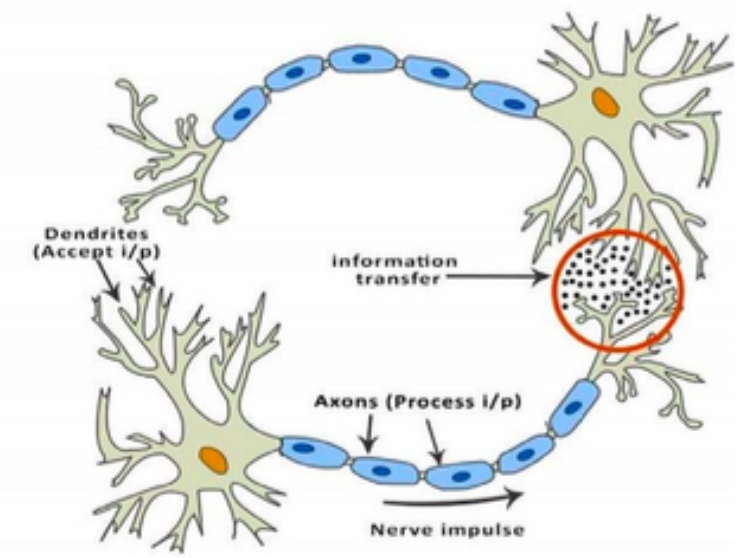


Figura 4: Dibujo de un par de neuronas. Se puede observar cómo los impulsos son transferidos por los axones hacia las dendritas, que es donde se transfiere la información. (Fuente: [(11) Diapositiva: 2]).

En un sentido más matemático, cuando una neurona le envía un mensaje a otra a través de esta red solo si la suma de las señales de entrada ponderadas de una o más neuronas es lo suficientemente grande se transmitirá el mensaje. Se habla de activación cuando se supera el umbral y el mensaje se transmite a la siguiente neurona [(11) Diapositiva: 6].

3.4.2. Redes neuronales artificiales en deep learning

El término red neuronal en deep learning, como se ha dado a entender previamente, es una referencia neurobiológica. Aun así, los modelos de deep learning no son como los del cerebro, simplemente los arquitectos de esta práctica se inspiraron, de forma generalizada, en el entendimiento de este. No hay evidencia alguna de que el órgano implemente nada parecido a los mecanismos de aprendizaje empleados en los modelos modernos. Simplemente, tratan de simular el comportamiento de dichas neuronas biológicas (1).

Como estas últimas, las redes neuronales artificiales también tienen ramificaciones que entran y salen de los nodos o núcleos. Se transmite la información por dichas ramificaciones y, por lo contrario, se procesa en los nodos. El funcionamiento básico de las neuronas se basa en modelar y procesar relaciones no lineales entre señales de entrada y salida en paralelo.

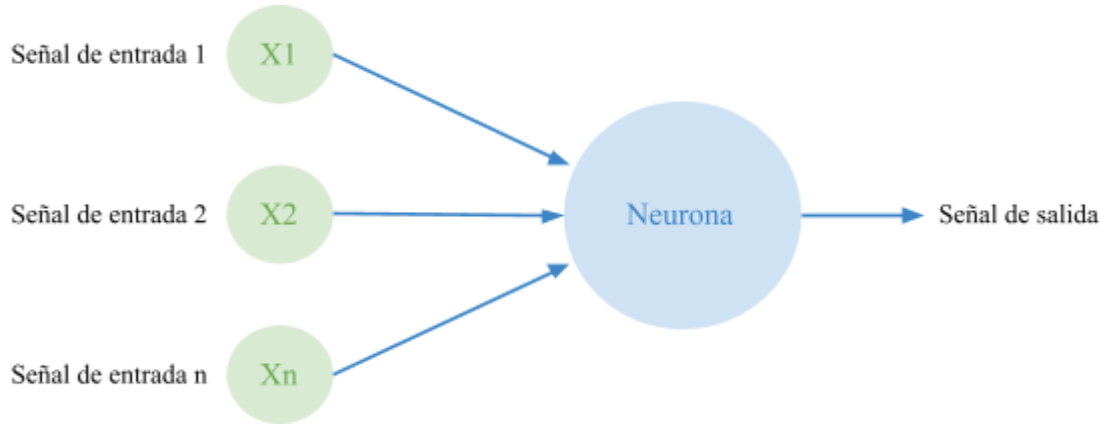


Figura 5: Estructura del funcionamiento de una neurona artificial. A la neurona le entran n señales de entrada y a esta le sale una señal de salida. (Fuente: 13)

Se deberían utilizar las redes neuronales porque ayudan a encontrar relaciones complejas entre un gran número de datos. Estas utilizan lo aprendido para realizar predicciones sobre el

comportamiento de múltiples estructuras. Estas redes pueden predecir relaciones lineales y no lineales en un *dataset* (1). Por otro lado también son capaces de realizar distintas actividades como clasificar imágenes o decidir en qué empresa invertir.

Asimismo, las redes neuronales artificiales se caracterizan por [(11) Diapositiva: 12]:

- Su arquitectura
- Valores de salida
- El peso (*weight*) adaptativo
- Cómo procesan la información
- Las funciones de activación
- Los algoritmos de aprendizaje
- La función de coste
- La regularización

3.4.3. Arquitectura de las redes neuronales

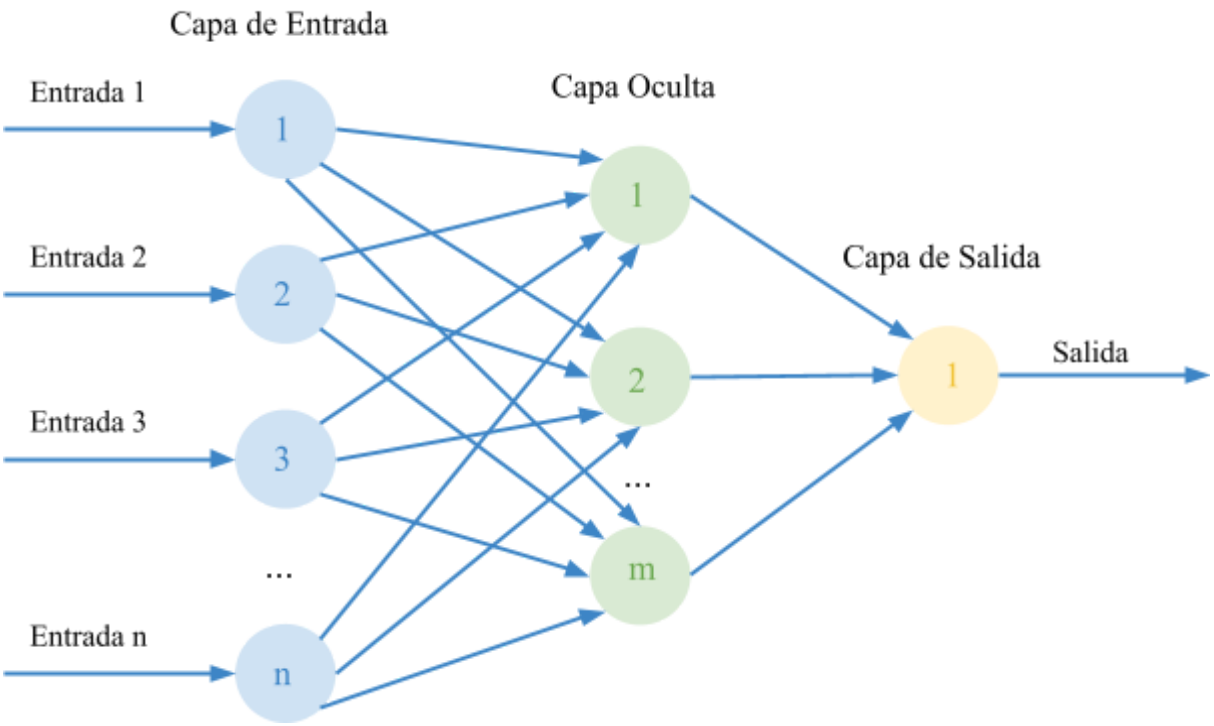


Figura 6: Arquitectura de una red neuronal, con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida . (Fuente: (18))

En un modelo simple la primera capa se llama capa de entrada, le sigue la capa oculta, y por último se encuentra la capa de salida. Cada capa puede contener una o más neuronas. Los modelos pueden llegar a una complejidad tan elevada que pueden requerir un gran número de capas ocultas con muchas neuronas en cada una de ellas. Esto se llama red neuronal profunda. Cabe resaltar que la complejidad y abstracción del modelo va ligado al *overfitting* (sobreajuste del modelo). En cuanto a las redes neuronales profundas, en las capas ocultas las señales de salida de algunas neuronas pueden ser las señales de entrada de otras, haciendo así que todo esté conectado [(11) Diapositiva: 6].

Aun así, en estos casos sí que hay una clara distinción entre las diferentes capas. La capa de entrada es aquella cuyas neuronas reciben la información del exterior y donde no se realiza ningún cálculo. Las capas ocultas, en cambio, no tienen ninguna conexión con el mundo exterior y, justamente, es donde se realizan todos los cálculos que se van transfiriendo de nodo a nodo. Por último, la capa de salida provee la información y el cálculo final desde la red neuronal al exterior.

3.4.4. Los valores de salida de las redes neuronales

En cuanto a los valores de salida, estos tienen la posibilidad de adoptar tres tipos de casos (13). Primero, pueden ser continuos, como el peso de una persona. Al igual que el peso, el valor de salida, en este caso, es estático y fijo, es decir, no puede cambiar. En segundo lugar, puede que sean binarios, como por ejemplo que una persona esté dormida o que esté despierta. Solo hay dos posibilidades, y estas son incompatibles. Por último, también puede ser una categoría. Un ejemplo de esta sería qué empresa de gas factura más en España. Se ha de tener en cuenta que, en este contexto, cuando se tienen dos categorías, también se puede encontrar un caso binario.

3.4.5. El peso adaptativo de las redes neuronales

El peso adaptativo, más conocido como *weights,* es de gran importancia para poder resolver el procesado de manera adecuada. Cada ramificación tiene asignado previamente un valor o peso y estos vienen representados de la siguiente forma:

$$w_1, w_2, \ldots, w_n \qquad n = número\ de\ ramificaciones\ totales\ por\ capa.$$

Ecuación 1: Weights.

Estos pesos pueden representarse como si fueran una fuerza que se transmite por cada ramificación. Es fundamental ajustar los pesos de tal forma que la red neuronal actúe de la forma que se busca, es más, los pesos mal colocados disminuyen el valor de la salida.

Cuando una red neuronal es entrenada, se inicializa con un conjunto de pesos. Después, estos pesos se optimizan durante el entrenamiento, y, pesos más óptimos se producen. Los pesos esencialmente reflejan cómo de importante es la entrada (19).

3.4.6. Procesado de información de una neurona artificial

Como se ha comentado anteriormente, las señales de entrada se transmiten, mediante ramificaciones, a los nodos donde se procesa esta, y a continuación, el resultado es enviado a una señal de salida. También es necesario recordar que hay una capa de entrada, algunas capas ocultas y la capa de salida.

La información se procesa en los núcleos de las neuronas cuando las señales de entrada interactúan con el peso ajustado inicialmente (13). Estas variables independientes que representan dichas señales de entrada son las siguientes:

$$x_1, x_2, \ldots, x_n \qquad n = número\ de\ señales\ de\ entrada\ totales\ por\ capa.$$

Ecuación 2: Señales de entrada.

Cada variable contiene una información concreta, es decir, un solo registro en la base de datos. Asimismo, en los núcleos, el peso y la señal de entrada se pueden procesar de distintas maneras. La forma más básica es hacer una combinación lineal entre el peso y su señal de entrada, es decir, sumar los productos de la señal y el peso de las ramificaciones.

$$x_1 \cdot w_1 + x_2 \cdot w_2 + \ldots + x_n \cdot w_n$$

Ecuación 3.

A continuación, se realiza a dicha suma la llamada función de activación, para así poder procesar la información de las redes neuronales dando como resultado la salida:

$$y = \phi(x_1 \cdot w_1 + x_2 \cdot w_2 + \ldots + x_n \cdot w_n)$$

Ecuación 4: Función de activación.

3.4.7. Las funciones de activación de las redes neuronales

Las funciones de activación, como se ha mencionado anteriormente, se encuentran en las neuronas y se encargan de procesar la información que reciben y transferirla por las señales de salida si esta lo requiere. Por eso reciben, también, el nombre de funciones de transmisión.

Hay una función de activación en cada nodo de la red neuronal y estas no tienen por qué ser las mismas. Dichas funciones se utilizan con el fin de resolver problemas más complejos donde haya un gran número de capas ocultas con muchos nodos conectados entre sí. También son las que añaden la no linealidad a la red neuronal, permitiendo así trabajar con una gran cantidad de datos (16).

Hay dos grandes tipos de función de activación: lineal y no lineal.

3.4.7.1. Función lineal

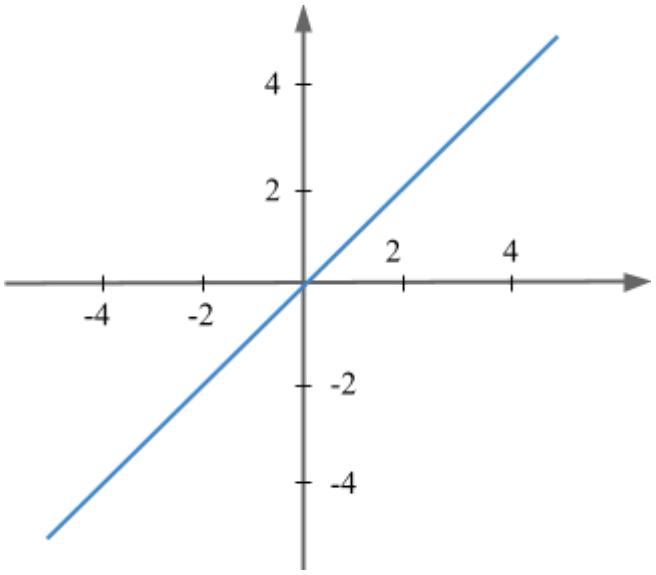


Figura 7: Gráfica de la función lineal continua.

$$f(x) = x$$

Ecuación 5: Función lineal.

Esta función de activación, también conocida como función de identidad, es una regresión lineal. Se utiliza principalmente cuando no se quiera transmitir ninguna información o se busca que la función genere un valor único. Si todos los nodos tuvieran una función de identidad no habría capas ocultas. En otras palabras, consigue que la señal de entrada sea igual a la de salida. No se puede emplear en problemas complejos, como la clasificación de imágenes, pero sí se puede utilizar cuando se quiere predecir un número concreto, como por ejemplo, el valor de la producción de un producto.

3.4.7.2. Funciones no lineales

Estas funciones son las más utilizadas en el mundo de las redes neuronales artificiales y se encuentran las siguientes:

→ Función escalón (*threshold*)

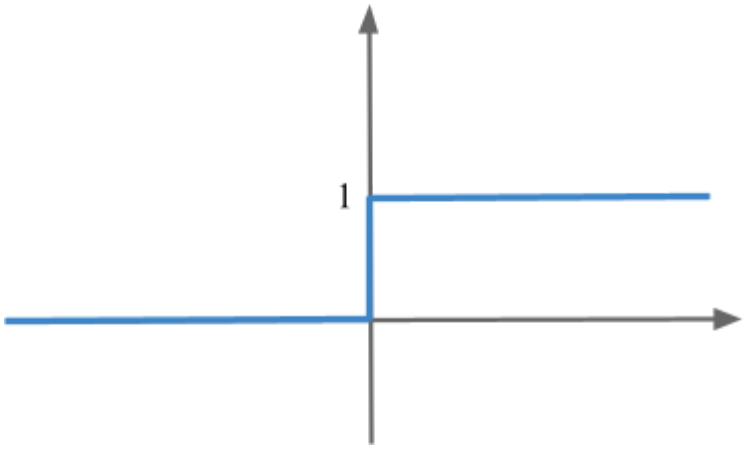


Figura 8: Gráfica de la función escalón.

$$f(x) = 0 \quad si \quad x \in (-\infty, 0)$$

$$f(x) = 1 \quad si \quad x \in (0, \infty)$$

Ecuación 6: Función escalón.

Esta función se utiliza principalmente cuando se tienen salidas binarias. Esta es la más estática, solo puede adoptar dos valores, cero cuando el valor de *x* es menor que cero y uno cuando *x* es mayor o igual que cero. El valor de la función se prolongará hacia la siguiente neurona con la señal de salida. Cuando la función es cero, nada cambiará, no se ajustará el peso de la neurona y por lo tanto dejará de responder a la señal de entrada. Este último fenómeno se llama desaparición del gradiente.

→ Función sigmoide

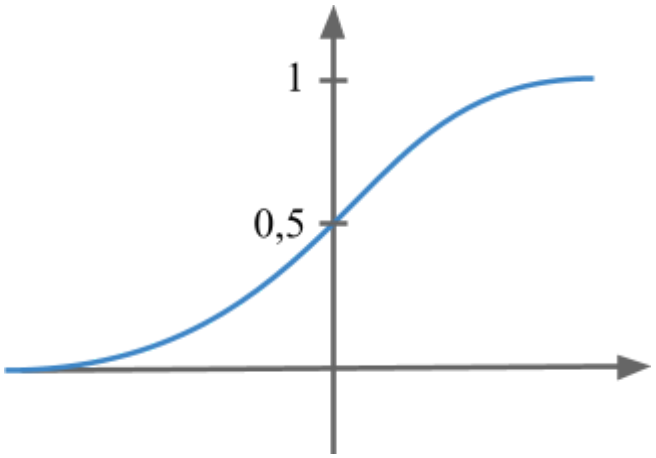


Figura 9: Gráfica de la función sigmoide.

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

Ecuación 7: Función sigmoide.

La sigmoide se puede llamar, además, función logística. También en esta es necesario distinguir entre los valores positivos y negativos aunque no sea tan fija como la anterior función. Es una función continua y tiene un rango que oscila del cero al uno. Cuando la *x* tienda al menos infinito, es decir, el valor sea muy negativo, la función tenderá a cero. Cuando tienda al más infinito esta tenderá a uno. De lo contrario el valor de la función será el de la ecuación. La sigmoide se utiliza mayoritariamente en los nodos de la capa de salida y sobre todo si estos son categóricos. También tiene el problema de la desaparición del gradiente.

→ Función tangente hiperbólica

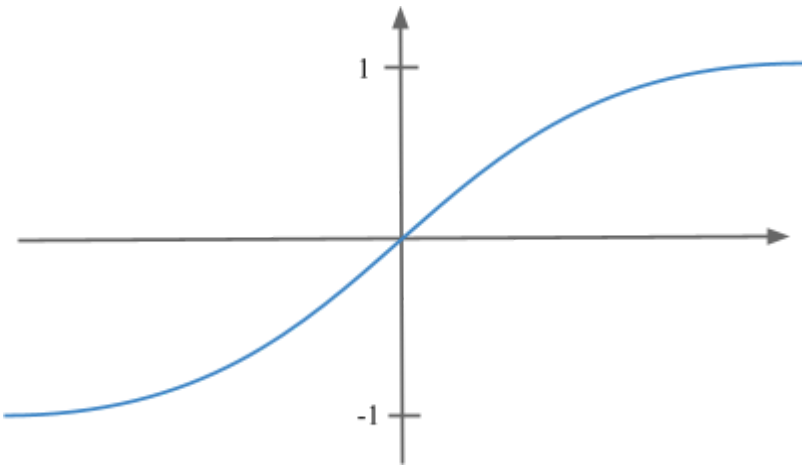


Figura 10: Gráfica de la función tangente hiperbólica.

$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

Ecuación 8: Función tangente hiperbólica.

La función hiperbólica también es continua y tiene un rango que ronda del -1 al 1. Se produce un escalamiento de la función logística para obtener esta función. El inconveniente de esta, al igual que en las otras, se produce cuando una desaparición del gradiente y el valor es cero. Aun así, el gradiente es más poderoso que en la función sigmoide.

→ Función ReLu

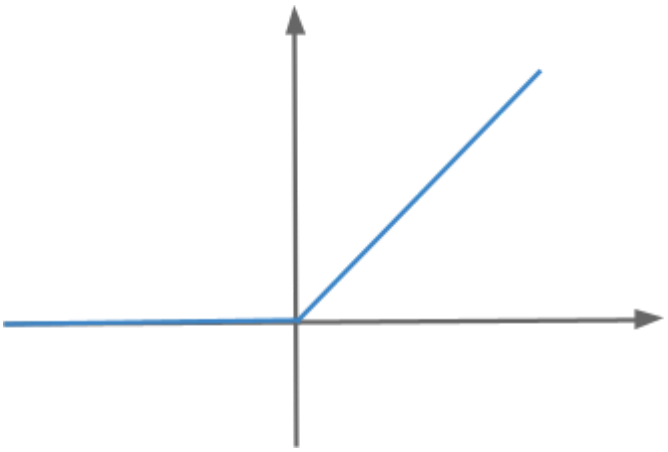


Figura 11: Gráfica de la función ReLu.

$$f(x) = 0 \quad si \quad x \in (-\infty, 0)$$

$$f(x) = x \quad si \quad x \in (0, \infty)$$

Ecuación 9: Función ReLu.

La función ReLu es la que más se utiliza en comparación a las otras debido a su rapidez. Es una función no continua con un rango de cero a más infinito. Si proporciona a la función un valor de entrada muy negativo, el resultado será cero. En cambio, si le da un valor positivo, el resultado seguirá siendo el mismo debido a que la pendiente de la función es cero. A pesar de que parece que también pueda suceder la desaparición del gradiente, la función ReLu tiene una variante llamada Leaky ReLu que evita la existencia de neuronas muertas. Lo hace creando una pendiente positiva muy pequeña en la región donde la función es negativa.

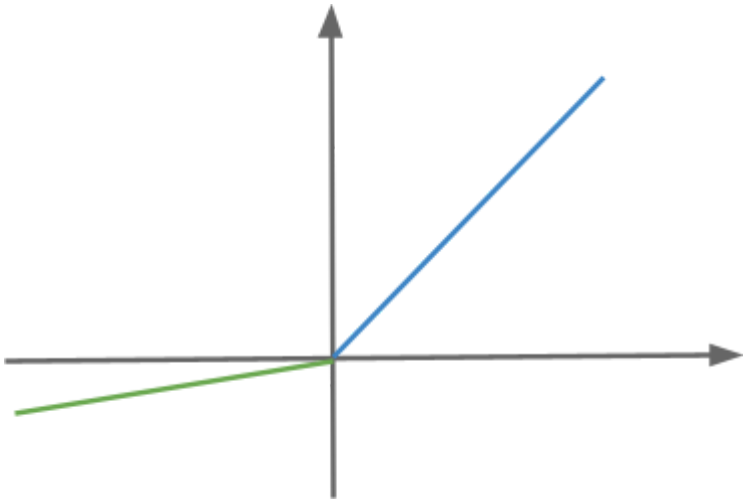


Figura 12: Gráfica de la función ReLu Leaky.

$$f(x) = 0,01x \quad si \quad x \in (-\infty, 0)$$

$$f(x) = x \quad si \quad x \in (0, \infty)$$

Ecuación 10: Función ReLu Leaky.

→ Función Softmax

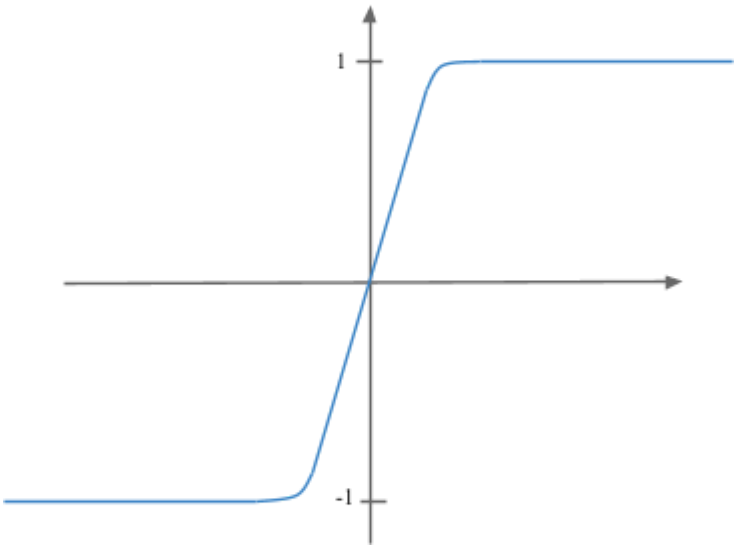


Figura 13: Gráfica de la función Softmax.

La función Softmax es una función continua con un rango que oscila del -1 al 1. Esta función se usa principalmente para clasificar datos. Por ejemplo, si queremos clasificar una imagen según el tipo de flor, la aplicación de la red softmax nos permitirá tener una probabilidad de 0.3% o 30 % de ser una rosa, 0,2 o 20% de ser una margarita y 0,5 o 50% de ser un lirio, por lo que el resultado será el de mayor probabilidad. Cabe señalar que la suma de estas probabilidades será igual a 1. También, esta función se une principalmente cuando la señal de salida tiene múltiples categorías y cuando se quiera asignar probabilidades a cada categoría que pertenezca a múltiples categorías.

3.4.8. Algoritmos de aprendizaje de las redes neuronales

En deep learning, los algoritmos de aprendizaje son un reflejo de cómo el cerebro humano analiza información y aprende de ella. Cuando un modelo se entrena, los algoritmos se centran en analizar los elementos desconocidos de las señales de entrada para extraer características, relacionar conceptos y descubrir patrones de datos útiles. El modelo de deep

learning utiliza diferentes algoritmos. Dependiendo del problema que se quiera solucionar, habrá ciertos algoritmos más adecuados. Elegir el algoritmo correcto ayuda a tener una comprensión profunda de todos los algoritmos principales que existen (20).

A continuación se expone una lista de los diez algoritmos de deep learning más utilizados: fuente (21):

1. Redes neuronales convolucionales (CNN)
2. Redes de memoria a corto plazo (LSTM)
3. Redes neuronales recurrentes (RNN)
4. Redes Generativas Adversariales (GANs)
5. Redes de funciones de base radial (RBFN)
6. Perceptrones multicapa (MLP)
7. Mapas autoorganizativos (SOM)
8. Redes de creencia profunda (DBN)
9. Máquinas de Boltzmann restringidas (RBM)
10. Auto Codificador

Este trabajo solo se centrará en las redes neuronales convolucionales, ya que considera que esta arquitectura es más poderosa que las otras. También, porque estas redes consiguen una muy alta precisión en los problemas de clasificación de imágenes, que es justamente lo que se hará en la parte más práctica del trabajo. Además, es capaz de detectar, sin supervisión humana, los parámetros más importantes.

3.4.9. Las funciones de coste de las redes neuronales

Cuando se entrena un modelo de redes neuronales están preestablecidos los valores de entrada y los de salida. Antes de comenzar a entrenarlo es necesario fijar los pesos de manera adecuada para que la red neuronal aprenda. También es de gran importancia que los valores de salida o estimados sean lo más parecido posible a los valores reales conocidos (13). En

resumen, las funciones de costes buscan optimizar los parámetros de la red neuronal determinando el error entre el valor estimado y el real. El objetivo es minimizar este error, esta función de costes. La manera de hacerlo es iterar el algoritmo y, tras esto, ir ajustando los pesos hasta que se encuentren unos parámetros que den un error lo más bajo posible. Este error también se llama pérdida (*loss*).

3.4.10. La regularización de las redes neuronales

El objetivo de la regularización es evitar el sobreajuste. Usualmente se utiliza el método Dropout. Este método de regularización se basa en eliminar temporalmente unidades de la red neuronal, es decir, las neuronas, además de todas sus conexiones, con el fin de reducir dicha red (22).

3.5. Redes neuronales convolucionales

Como se ha comentado previamente, las redes neuronales convolucionales son un tipo de algoritmo que se usa en deep learning y se suelen llamar CNN. Está diseñado principalmente para el procesamiento y la clasificación de imágenes y para la detección de objetos. Esencialmente, las CNN son un sistema donde se apila un gran número de neuronas juntas. Inicialmente, se puede pensar que la solución de un problema de clasificación de imágenes se podría basar en crear un mapa de redes neuronales por cada píxel que hay en la fotografía. Aun así, sería realmente caro desde el punto de vista computacional y, por lo tanto, no se hace. La solución se encuentra en las redes convolucionales porque ayuda a optimizar los cálculos sin perder la esencia de los datos. La convolución es básicamente un montón de matrices multiplicando y sumando estos resultados [(23) Diapositiva:15]. Las redes neuronales convolucionales, gracias a que tiene muchas capas ocultas, va detectando patrones capa a capa y va mejorando hasta que en las últimas ya es capaz de identificar la imagen.

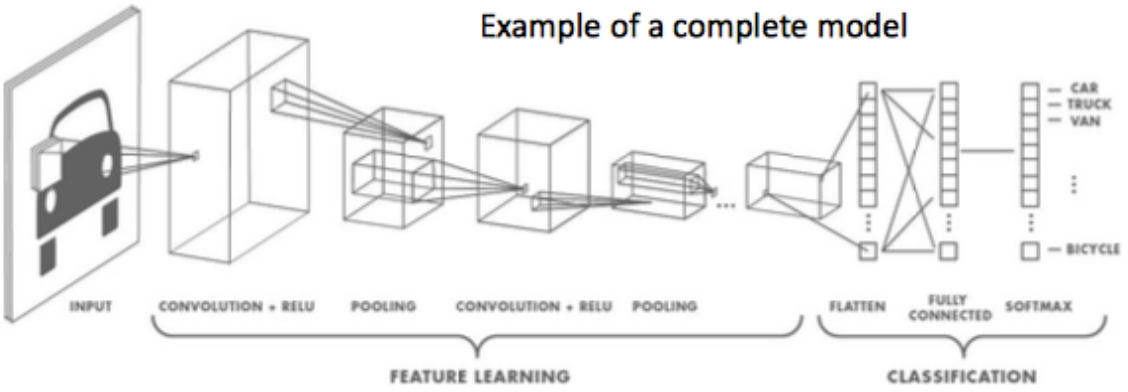


Figura 14: Ejemplo de un modelo completo. Se observa como en el modelo le entra una imagen de un vehículo. El modelo aprende a partir de esta entrada para poder, así, crear el modelo clasificatorio. (Fuente: [(23), Diapositiva: 15])

3.5.1. Patrones en la red convolucional

Como se ha aclarado al inicio, no es posible crear una red totalmente conectada para identificar una imagen. Aun así, algunos patrones son mucho más pequeños y útiles que la imagen completa. Como por ejemplo, si se piensa en una imagen de un avión, solo bastaría en fijarse en el ala para saber que es un avión. Esto es, precisamente, lo que hace la red neuronal convolucional representando una región pequeña con menos parámetros. Cuando se tiene un gran *dataset*, el modelo encuentra los mismos patrones en diferentes lugares y estos se comprimen con los mismos parámetros.

La CNN está formada principalmente por capas convolucionales. Cada capa convencional tiene una serie de filtros que realizan operaciones convencionales. Gracias a que las características de las CNN permiten la invariabilidad de las transformaciones afines a las imágenes que se alimentan por la red, es posible identificar los patrones desplazados, inclinados o deformados en las diferentes imágenes.

3.5.2. Filtros de la red convolucional

Se puede pensar en ellos como imágenes de menor tamaño que se deslizan sobre la imagen de entrada. El filtro puede estar en distintas regiones de la imagen y se multiplica en cada una de ellas. Es decir, se produce la convolución sobre las imágenes. Estas se representan mediante una matriz de forma (altura, anchura, canales), donde hay tres canales (rojo, verde, azul) para las imágenes en color, y un canal para las imágenes en escala de grises. Para que un filtro pueda realizar una convolución sobre la imagen, debe tener el mismo número de canales que la entrada. La salida es la suma de la multiplicación elemental de los elementos del filtro y la imagen (se puede considerar como producto de puntos) [(23) Diapositiva: 19].

- Se puede utilizar cualquier número de filtros y estos están compuestos por un conjunto de *kernels*, posteriormente se explicará lo que son.
- La dimensión de profundidad de salida será igual al número de filtros que se debe utilizar.

3.5.3. Convoluciones

Se ha hablado mucho sobre las señales de entrada de una red neuronal, pero aún no se ha especificado qué son estas señales exactamente. En un ejercicio real, como la clasificación de una imagen, las señales de entrada serán los píxeles de fotografía (25). Cuando se tiene una imagen en blanco y negro con 54x54 píxeles se usarían 2916 neuronas. Si esta última fuera en color, gracias a que se necesitan tres canales, se usarían 54x54x3, es decir, 8748 neuronas de entrada.

La convolución es una herramienta matemática que, en las redes neuronales, es capaz de crear relaciones entre píxeles utilizando pequeñas matrices de datos de entrada. Esta pequeña matriz es precisamente *kernel* y es operada matemáticamente con otro grupo de píxeles cercanos. Todas las imágenes se pueden representar en una gran matriz de valores de píxeles.

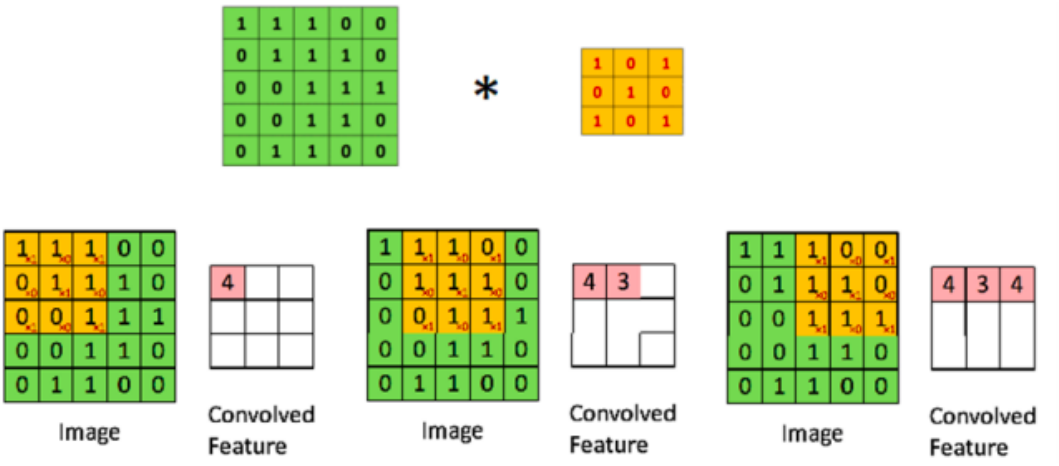


Figura 15: Convolución. (Fuente:(24))

Como se observa en la imagen anterior, se realiza la convolución con la matriz de píxeles de una imagen (matriz verde) con el filtro (matriz amarilla), y se va obteniendo un mapa de características (matriz rosa). Con diferentes filtros se pueden obtener diferentes mapas, y por lo tanto, se puede buscar distintas características dentro de una imagen, como por ejemplo formas.

A la hora de programar con CNN, los filtros se crean por su cuenta cuando se entrena el modelo. Cuantos más filtros, más características se pueden encontrar. Los mapas dependen del número de filtros, del número de píxeles por los que se desliza en el *kernel* sobre la matriz de entrada y de los rellenos de ceros. Este último también se llama zero-padding y se basa en añadir ceros alrededor de la matriz de entrada con tal de poder controlar el tamaño del mapa de características (25).

Principalmente hay tres tipos de capas en las CNN:

- Capas de convolución: En esta capa se realiza la convolución aplicando los *kernels* y obteniendo los mapas de características. El número de parámetros aprendibles en estas capas es igual al número de parámetros de cada filtro multiplicado por el número de filtros. Es decir, (altura del filtro) * (ancho del filtro) * (número de filtros). Normalmente se utiliza ReLU. También, es en esta capa es donde se realiza el llamado

pooling y se utiliza para reducir la muestra de la imagen. Se obtienen menos parámetros en las capas siguientes y, por lo tanto, la red de convolución puede ser un poco más profunda. Se ha comprobado que el pooling máximo funciona mejor tomando el máximo de esa región de la imagen. Las capas de pooling mantienen la dimensión de profundidad, es decir, el pooling se realiza de forma independiente en todos los canales de la entrada. En esta capa no hay parámetros aprendibles. Para que los mapas de características puedan identificar formas más complejas es necesario que haya más de una capa de convolución. En el siguiente ejemplo se puede observar la estructura si se realizara dos capas.

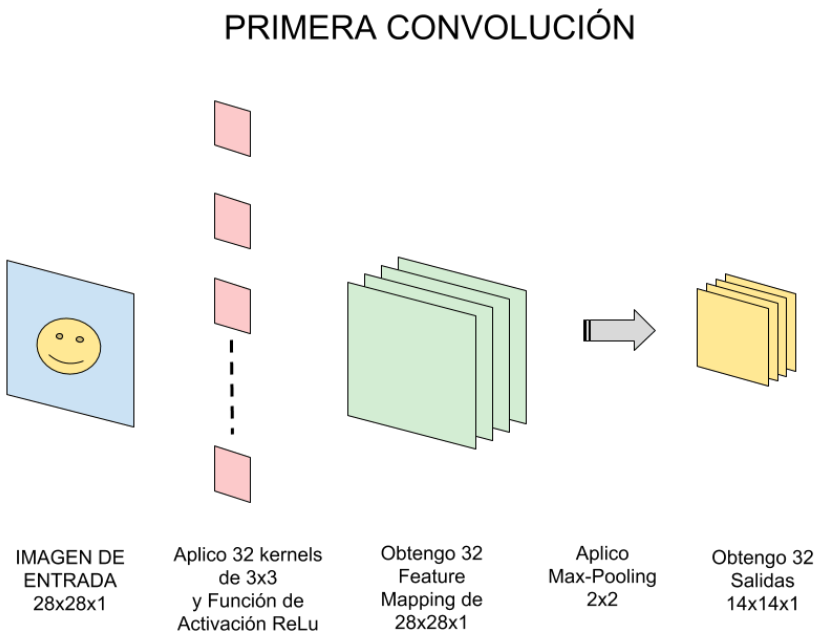


Figura 16: Ejemplo de los pasos que se siguen en la primera convolución. (Fuente:(25))

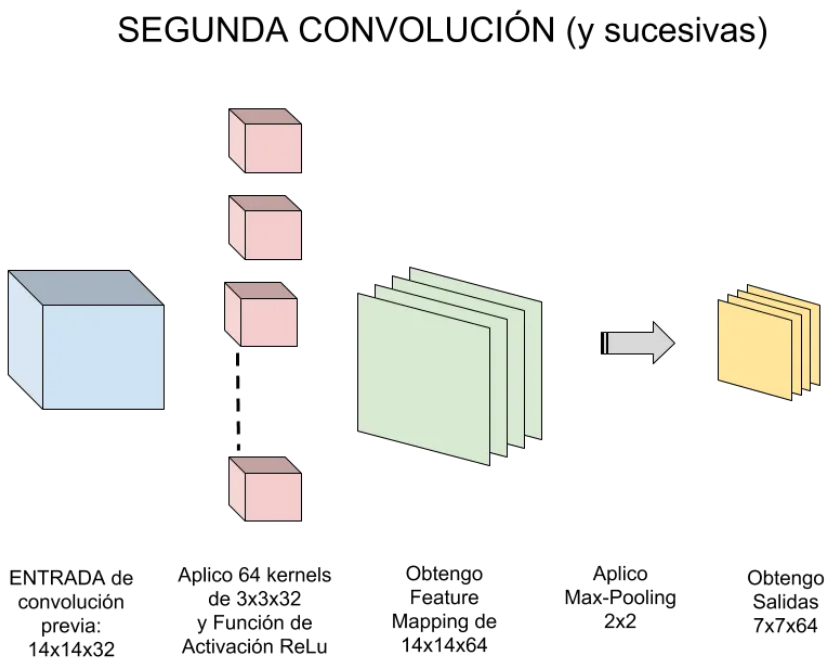


Figura 17: Ejemplo de los pasos que se siguen en la segunda convolución. (Fuente:(25))

- Capas totalmente conectadas: Suelen estar al final de la red de convección. Son capas simples de avance, y tienen tantos parámetros aprendibles como los que tendría una red normal de avance.
- Capa de salida: Se realiza la función de Softmax a la anterior capa para obtener la capa de salida. Es en esta última capa que se obtienen los resultados del problema. En un ejercicio de clasificación de imágenes se obtendrían las predicciones.

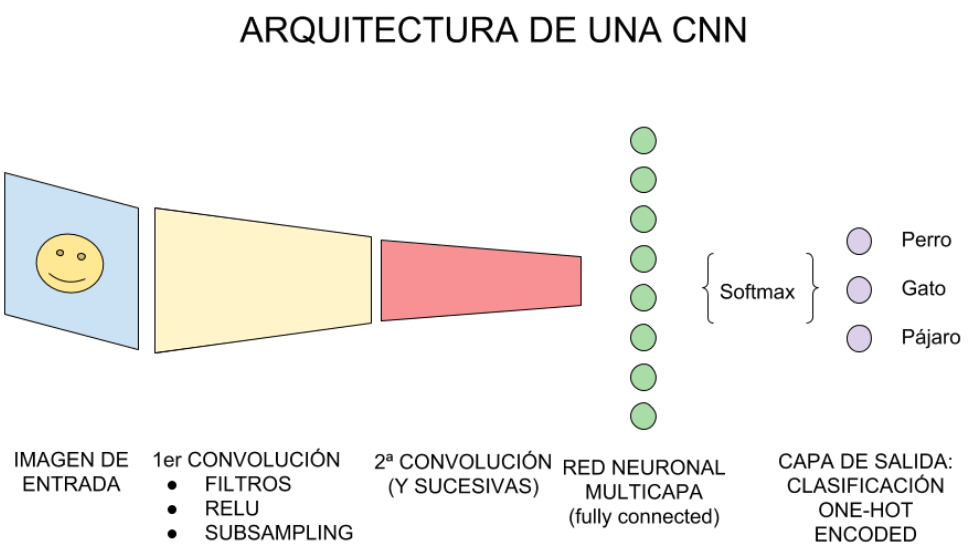


Figura 18: Arquitectura de una red neuronal convolucional. (Fuente:(25))

3.5.4. Pooling

Como se ha comentado anteriormente, el pooling ayuda a comprimir datos disminuyendo los parámetros, pero manteniendo la profundidad, esto se denomina subsampling. El pooling también reduce el tiempo de entrenamiento. La idea principal de pooling es crear pequeños "resúmenes" de cada mapa de características, manteniendo así lo realmente relevante (25).

En las redes neuronales convolucionales se suele utilizar Max-pooling. Como se puede ver en la siguiente figura, esta práctica se basa en crear matrices reducidas cogiendo el número más alto de cada submatriz.

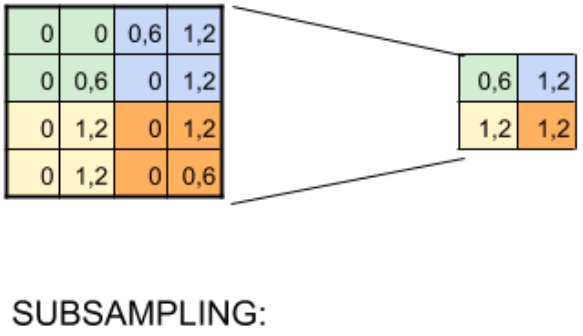


Figura 19: Matriz resultante tras realizar Subsampling. Se aplica Max-Pooling de 2x2 y se reduce la salida a la mitad. (Fuente:(25))

Las CNN tienen un rendimiento excepcional cuando clasifican imágenes muy parecidas al conjunto de datos. Cuando alguna imagen del *dataset* está un poco inclinada o está orientada de manera distinta, la red convolucional no funciona de manera correcta. Esto se soluciona con pooling porque consigue crear una invariabilidad posicional. Con esta metodología, una CNN es tolerable a los pequeños cambios de punto de vista y se adapta fácilmente para que no se confunda con el posicionamiento espacial dentro de la imagen.

3.6. Transfer learning

Se ha comentado en un gran número de ocasiones de que las redes neuronales son capaces de clasificar imágenes, aun así, no se ha hablado de qué metodología se ha de seguir para entrenar estas. Para crear este modelo se ha decidido utilizar la técnica de transfer learning. Esta técnica utiliza modelos entrenados previamente para solucionar problemas. Básicamente, se trata de utilizar el conocimiento aprendido anteriormente en otro ejercicio, transfiriendo los pesos. Es a partir de un modelo con muchos datos etiquetados disponibles que se crea el modelo inicial. Por lo tanto, se utiliza ese conocimiento para resolver tareas que no tengan muchos datos (40).

> *"Transfer learning es la mejora del aprendizaje de una nueva tarea mediante la transferencia de conocimientos sobre una tarea que ya se ha aprendido."* - Capítulo 11: Transfer Learning, Handbook of Research on Machine Learning Applications, 2009. (42)

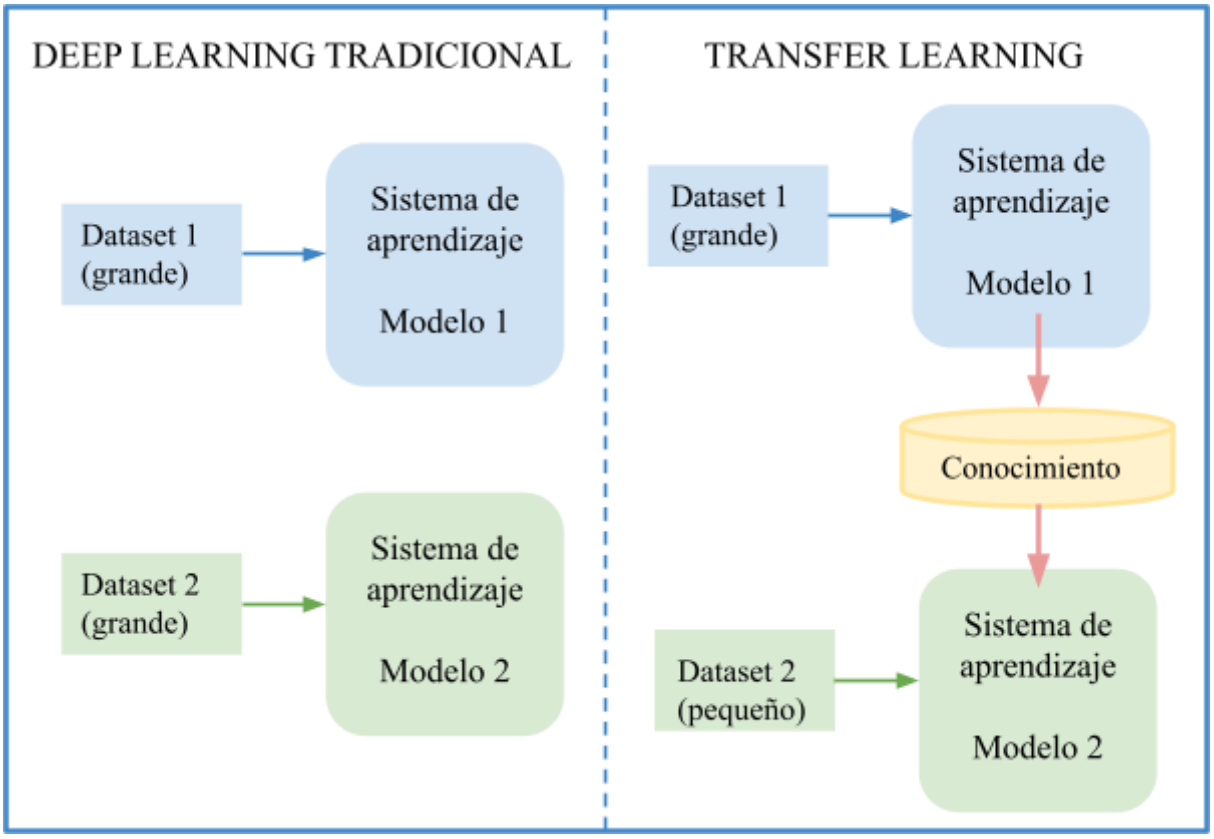


Figura 20: Deep learning tradicional vs transfer learning. (Fuente: (39))

Las redes neuronales tratan de encontrar patrones y características en los datos para poder hacer predicciones, y transfer learning facilita ese objetivo. También, cuando se realiza este método, se utilizan las redes neuronales convolucionales. Así pues, si se utiliza transfer learning es necesario seguir estos pasos (40)(43):

1) Selecciona un modelo de origen: El primer paso es escoger un modelo entrenado previamente. La biblioteca de código abierto Keras tiene, por ejemplo, nueve modelos pre entrenados que pueden ser utilizados para transfer learning.
2) Reutilizar modelo: Ahora es el momento de utilizar el modelo escogido y emplearlo para crear el segundo modelo a partir de las CNN, donde se extraen características y patrones.
3) Realizar Fine-tuning: Este paso no es obligatorio. Si se requiere, se puede adaptar o refinar el modelo en los datos de entrada y de salida según las especificaciones.

3.6.1. Modelos pre entrenados

Siguiendo con lo ya mencionado, los modelos pre entrenados son aquellos que han sido entrenados previamente utilizando una gran cantidad de datos. Estos son capaces de identificar los patrones y las características del nuevo dataset. En este trabajo solamente se utilizarán los modelos ResNet y Alexnet, a pesar de que existen muchos más. Estos dos modelos se entrenaron en el desafío de ImageNet.

3.6.1.1. ResNet

Las redes reducidas o ResNet es un modelo que disminuye la profundidad de las redes eliminando el problema de desaparición del gradiente.

Como se ha aclarado anteriormente, las redes neuronales, cuanto más profundas, más fácil es encontrar características y patrones y, por lo tanto, mejor es el desempeño de los resultados. Aun así, estas acaban siendo operaciones muy complejas a llevar a cabo debido al gran número de capas que contienen las redes. A fin de solucionar este problema, los ingenieros informáticos se plantearon si era realmente cierto lo que decía la teoría, que cuanto más profundo y cuantas más capas tenía esa red neuronal más precisión se obtenía. Al contrario de lo que pensaban, se descubrió que la precisión del entrenamiento iba disminuyendo después de iterar el problema unas cuantas veces. Vieron que la realidad se alejaba de la teoría (44).

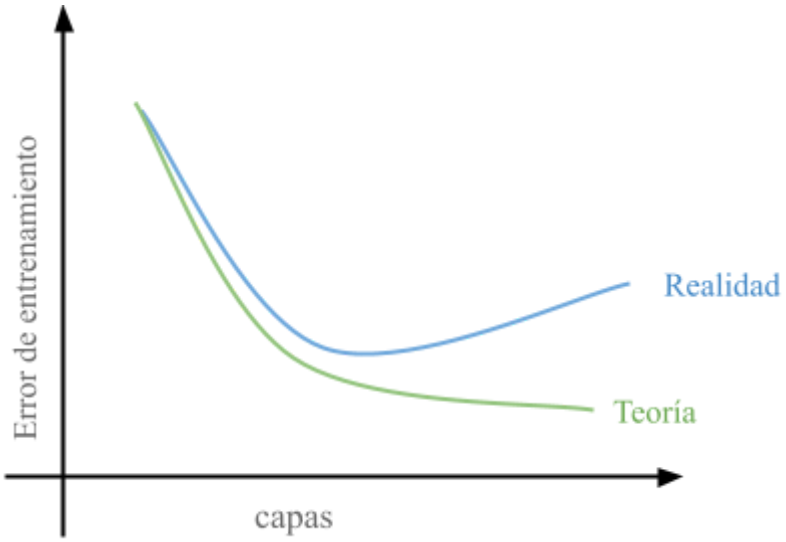


Figura 21: Teoría vs Realidad sobre la variación en el error de entrenamiento con un aumento en las capas. (Fuente: (44))

Encontraron que el problema estaba en la desaparición del gradiente. Que los gradientes desaparecieran provocaron que se obstaculizara la convergencia, lo que hizo que se obtuvieran malos resultados. Aun así, observaron que aunque se solucionará el problema, seguían obteniendo más precisión cuantas menos capas tenía la red (46).

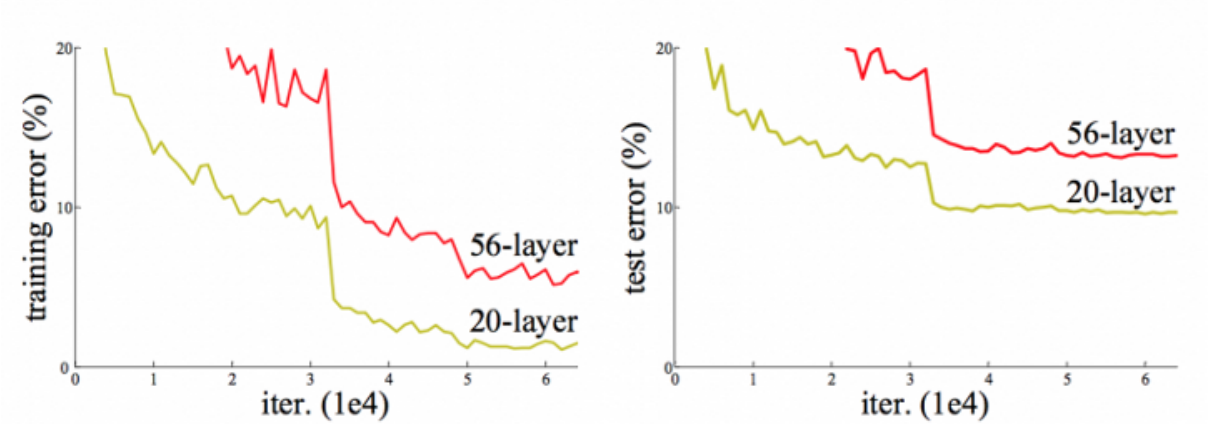


Figura 22: Gráficas del error de entrenamiento y del error de prueba con redes de 20 capas y 56 capas. La red más profunda tiene un mayor error de entrenamiento y, por tanto, de prueba. (Fuente: (46))

Al final, fue un equipo de investigación de Microsoft el que creó los ResNets. Fueron estas redes residuales las que ayudaron a acabar con el problema de degradación. Se demostró que estas eran más fáciles de optimizar obteniendo una alta precisión aun cuando las redes neuronales fueran muy profundas. ResNet, es capaz de obtener estos resultados omitiendo datos de una capa a otra. Este método se denomina conexiones de acceso directo o conexiones de omisión. Gracias a las redes residuales, los datos fluyen entre capas sin empeorar el aprendizaje de un modelo profundo. Esto se debe a que si existe alguna capa que perjudica el rendimiento del modelo, este omitirá dicha capa (44).

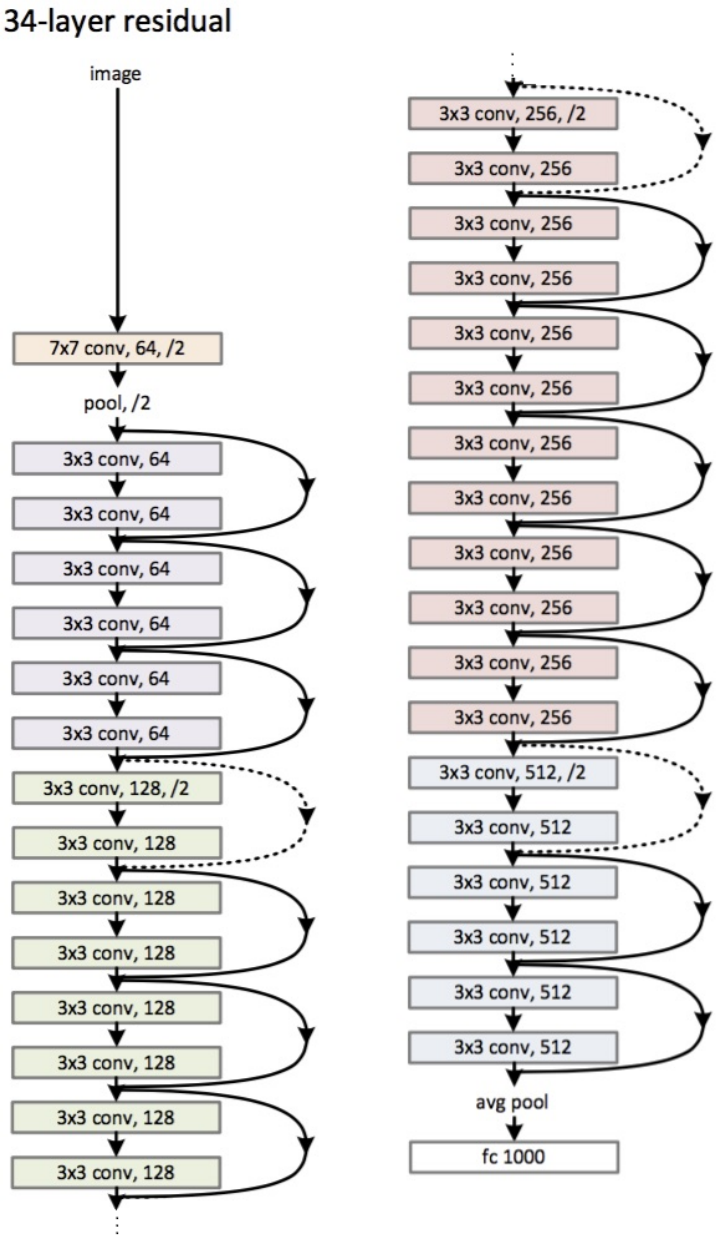


Figura 23: Ejemplos de arquitecturas de red para ResNet 34. (Fuente: (46))

En conclusión, ResNet ha conseguido mejorar considerablemente los resultados y la precisión de las redes neuronales más profundas. En la siguiente figura se puede observar cómo, efectivamente, las redes neuronales mejoran el rendimiento del modelo.

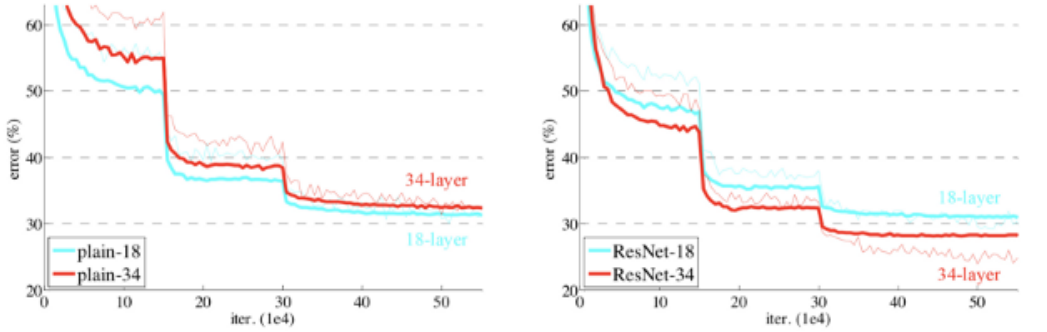


Figura 24: Entrenamiento simple vs entrenamiento con ResNet. Las curvas delgadas denotan el error de entrenamiento y las curvas en negrita denotan el error de validación de los cultivos centrales. En este gráfico, las redes residuales no tienen ningún parámetro adicional en comparación con sus homólogas simples. (Fuente: (46))

3.6.1.2 AlexNet

El modelo pre entrenado AlexNet es una arquitectura que revolucionó el mundo de deep learning cuando se presentó en el concurso de ImageNet. Está formado por ocho capas; cinco capas convolucionales y tres capas totalmente conectadas. A continuación se explicarán las razones por la cual este modelo es tan importante (47):

1) Utiliza la función no lineal ReLU: Una de las ventajas de utilizar ReLU es que el tiempo de entrenamiento es menor.
2) Utiliza LRN (Normalización de respuesta local): Tras las funciones de activación y agrupación se usa LNR. Este mecanismo resalta las neuronas con respuesta grande y suprime las neuronas con respuesta pequeña. En resumen, mejora la precisión del modelo.
3) Utiliza múltiples GPU y CUDA: AlexNet utiliza múltiples GPUs poniendo la mitad de las neuronas del modelo en una GPU y la otra mitad en otra. Permite entrenar modelos de grandes dimensiones además de disminuir el tiempo de entrenamiento.
4) Utiliza Max-Pooling: El modelo realiza una agrupación de Max-Pooling. Gracias a eso se obtienen más patrones y más características.

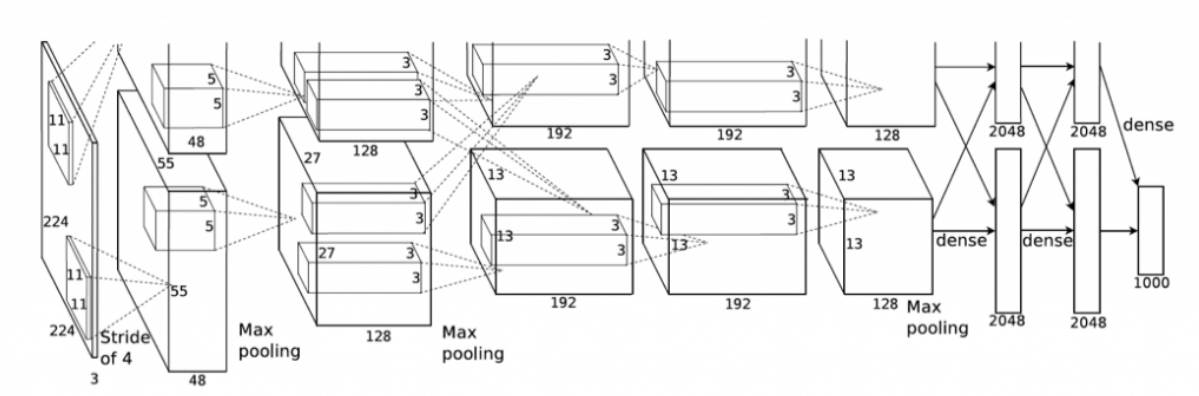


Figura 25: Ilustración de la arquitectura de AlexNet. (Fuente: (47))

3.6.2. CNN cómo extractor de patrones

Como ya se ha mencionado, las redes neuronales convolucionales actúan como extractores fijos de patrones y características. La mayoría de las CNN pre entrenadas salen de ImageNet. La extracción de patrones se basa en utilizar una base convolucional de una red entrenada previamente, pasando por ella los nuevos datos, y luego, entrenando un nuevo clasificador con el resultado (1).

A partir de las CNN, se congelan las primeras capas convolucionales de la red neuronal del modelo pre entrenado usando solamente la última capa. Esta es la que hace una predicción del nuevo modelo y actúa como clasificador. Tras extraer esta, se agrega una nueva capa con parámetros aleatorios que se encargará de la nueva clasificación con los nuevos datos. Por último, cuando es el momento de entrenar el modelo, se congela todo exceptuando esta última capa, que es la que entrena y aprende.

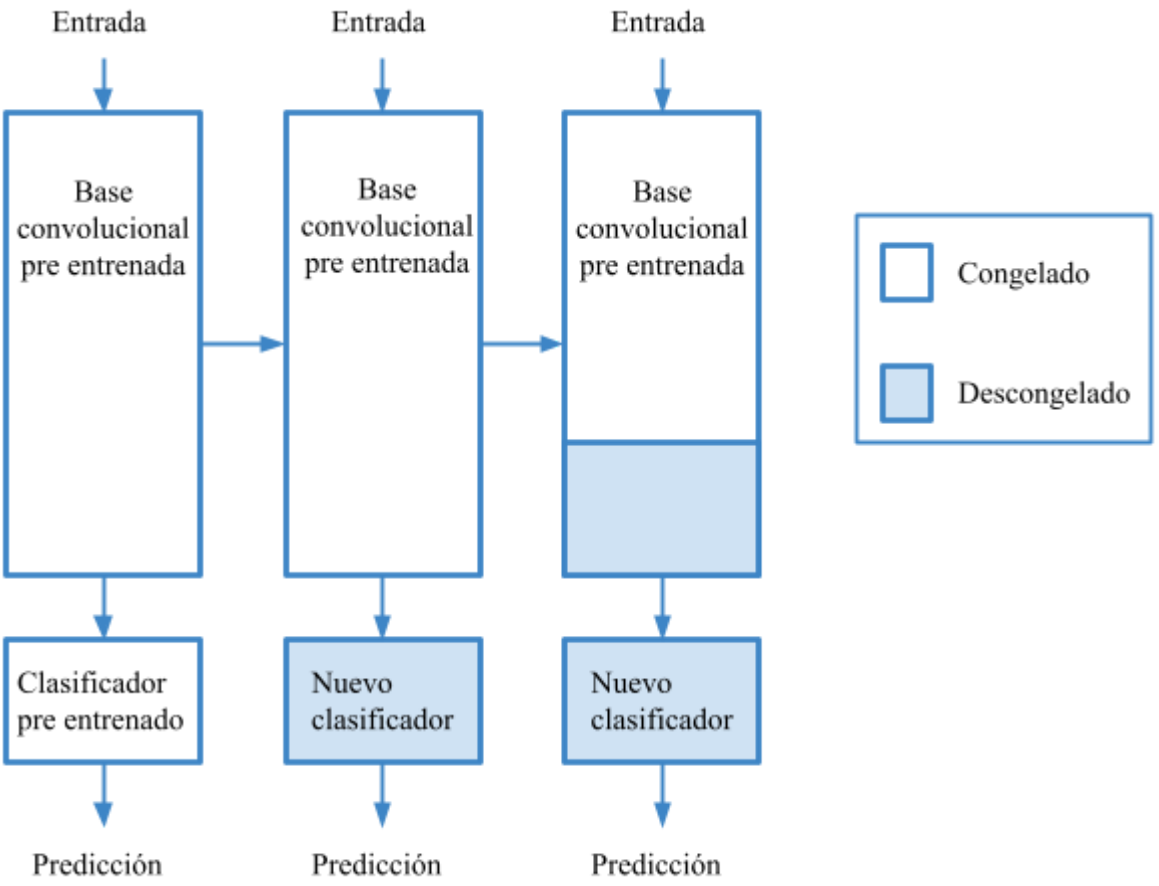


Figura 26: Diagrama de transfer learning a partir de un modelo CNN pre entrenado. (Fuente: (50))

3.6.3. Fine-tuning

Fine-tuning se define como la técnica que permite realizar pequeños cambios al modelo para obtener el resultado deseado. Con fine-tuning, se utilizan los weights de un modelo pre entrenado en otro modelo. Este proceso también consigue disminuir el tiempo de entrenamiento y volverlo más preciso.

El libro *Deep Learning con Python* escrito por François Chollet describe Fine Tuning de la siguiente manera; "Fine-tuning consiste en descongelar algunas de las capas superiores de un modelo base congelado utilizado para la extracción de características, y entrenar

conjuntamente la parte recién añadida del modelo (en este caso, el clasificador totalmente conectado) y estas capas superiores."(1)

Para poder entender esta definición, es necesario explicar que normalmente, después de entrenar el modelo es necesario congelarlo, es decir, bloquearlo. Gracias a eso se evita que los pesos de las diferentes capas cambien, también disminuye el tiempo de entrenamiento.

Aun así, si se quiere seguir entrenando el modelo realizando fine-tuning, se descongelaran algunas de las primeras capas del modelo pre entrenado. Después, cuando se entrene el nuevo modelo se usarán los pesos de dichas capas para obtener resultados más precisos.

4. METODOLOGÍA

En esta segunda parte del trabajo se pondrá en práctica algunos de los conocimientos desarrollados anteriormente. El objetivo es crear un clasificador de imágenes que identifique con una alta precisión si un paciente tiene neumonía. Para ello se utilizarán conceptos de deep learning para realizar el código donde se entrenará el modelo de dicho clasificador.

En los siguientes subapartados se repasará qué es la neumonía y su incidencia. Se explicará cuál ha sido la motivación de crear un clasificador identifique esta enfermedad y no otra. Por último, se enseñará, paso a paso, cómo se ha utilizado transfer learning para crear el programa. En concreto, se ha utilizado las redes neuronales convolucionales y la técnica de fine-tuning.

4.1. Neumonía

4.1.1. Bases biológicas

Como define Mayo Clínica, la neumonía es una infección que inflama los sacos aéreos de uno o ambos pulmones. Los sacos aéreos se pueden llenar de líquido o pus (material purulento), lo que provoca tos con flema o pus, fiebre, escalofríos y dificultad para respirar. Diversos microorganismos, como bacterias, virus y hongos, pueden provocar neumonía (29).

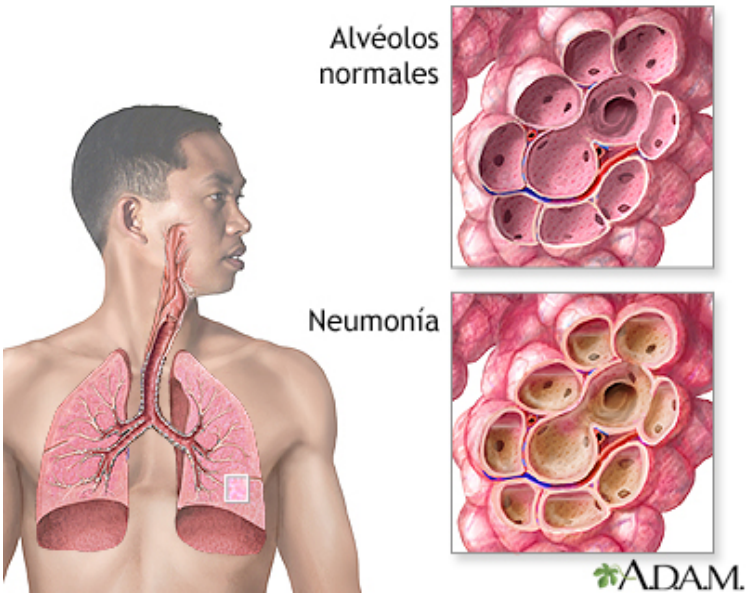


Figura 27: Loa alvéolos de los pulmones normales y con neumonía. (Fuente:(28))

Aunque todo el mundo puede padecerla, suele ser más grave en bebés, niños pequeños, personas mayores de 65 años y en personas con problemas de salud o sistemas inmunitarios debilitados. El gran problema de la neumonía es que cada caso es distinto. Aunque esta se pueda prevenir, puede llegar a ser mortal. Por eso, es tan importante detectar la enfermedad a tiempo (29).

Sus síntomas pueden incluir dolor en el pecho al respirar o toser, tos, fatiga, fiebre, escalofríos, dolor de cabeza y dificultad para respirar (28).

El tratamiento de la neumonía depende de la gravedad y del tipo de esta. La mayoría de personas se recuperan en sus casas descansando e ingiriendo mucho líquido. Cuando la infección se complica, es necesario hospitalizar al paciente donde se le puede administrar líquidos y antibióticos de manera intravenosa, y donde puede recibir oxigenoterapia y tratamientos respiratorios.

El único método de prevención para los adultos es la vacuna de la gripe. En el caso de los niños pequeños se puede prevenir aumentando las medidas de protección, como una nutrición adecuada, y reduciendo los factores de riesgo como la contaminación atmosférica (que hace que los pulmones sean más vulnerables a las infecciones) y utilizando buenas prácticas de higiene (55).

Una de las formas de diagnóstico para detectar esta patología es a partir de una radiografía torácica, donde se podrá apreciar la infección pulmonar. En una radiografía, la neumonía se puede identificar si se observa una zona de color blanco. Estos son los alvéolos llenos de líquido. La zona oscura son los alvéolos "sanos", aquellos que están llenos de aire. Al observar este color blanco, se puede confirmar el diagnóstico de la infección.

4.1.2 Incidencia de la neumonía

La neumonía es una infección que puede provocar una enfermedad muy grave con un gran impacto económico. Según distintos estudios, de cada 1.000 habitantes al año hay entre 2 y 10 casos de neumonía. (32) En España, el INE publicó que en 2019 murieron 9.384 personas por esta infección. El 51,55% fueron hombres y el 48,45% restantes, mujeres. La neumonía representó ese año el 2,24% de las muertes totales (31).

Es una enfermedad que, si es necesario, se debe internar al paciente en hospitales. La doctora Isabel Jimeno, responsable del Grupo de Vacunas de la Sociedad Española de Médicos Generales y de Familia (SEMG), señaló; "Cada día son hospitalizadas 274 personas por neumonía en España. Sin embargo, existe una percepción muy baja del riesgo de padecerla y de las consecuencias que puede tener a corto y a largo plazo. El objetivo es informar a la población de su importancia, así como promover intervenciones para proteger, prevenir y tratar esta enfermedad". Aun así, estos datos solo muestran las personas hospitalizadas. La doctora Rosario Menéndez, jefa del Servicio de Neumología del *Hospital Universitari i Politècnic La Fe* dijo que solamente el 41% de los adultos que padecen neumonía son hospitalizados (33).

En el caso de los niños pequeños la neumonía se convierte en algo mucho más letal. Unicef publicó un artículo que decía que la neumonía cobra la vida de más de 800.000 niños menores de cinco años cada año, incluidos más de 153.000 recién nacidos, que son especialmente los más vulnerables a la infección. Esto significa que un niño muere de neumonía cada 39 segundos y que casi todas estas muertes son evitables (55).

4.1.3 Motivación

Como se ha podido observar en los apartados anteriores la neumonía es un problema muy grave en el mundo. Y, aunque muchas personas mueren al año por esta enfermedad, la neumonía no es una enfermedad de declaración obligatoria. En cierta forma, dicha enfermedad suele subestimarse, y por eso es tan importante concienciar a la población de su gravedad. Además, justamente porque se infravalora, existen muy pocos datos y estudios poblacionales sobre la incidencia de la neumonía.

Con este clasificador, se busca crear un programa que identifique de manera más rápida y concisa si el sujeto a estudio padece neumonía o no. Del mismo modo, se intenta fomentar la creación de dichos estudios para así poder encontrar nuevos tratamientos o controlar la enfermedad. Sería beneficioso si esos estudios estuvieran dirigidos a niños menores de cinco años, debido a que esa es una de las franjas de edades que más puede afectar la neumonía.

Además, se ha decidido hacer un clasificador de imágenes para poder poner en práctica lo aprendido en la parte más teórica del trabajo.

4.2. Dataset

Para poder crear un buen clasificador, es muy importante el *dataset* o conjunto de datos. Cuanto más equilibrado sea este conjunto de datos, mayor precisión se obtendrá. Por lo tanto, se ha decidido utilizar un *dataset* de la página web Kaggle creado por Paul Mooney. Este *dataset* se ha descargado 123.009 veces. Se puede encontrar en el siguiente link: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia (34)

El *dataset* está formado por imágenes de radiografías del tórax (anterior-posterior) que les sacaron a pacientes pediátricos de entre uno a cinco años en el Centro Médico de Mujeres y Niños de Guangzhou. Todas las radiografías se realizaron como parte de una atención clínica rutinaria de los pacientes.

Para el análisis de las imágenes, todas las radiografías fueron sometidas a un control de calidad exhaustivo, eliminaron todas las exploraciones de baja calidad y las que eran ilegibles. Además, los diagnósticos de las imágenes fueron calificados por dos médicos expertos antes de ser autorizados para el entrenamiento de sistemas IA. Para evitar cualquier error de clasificación, las imágenes ya clasificadas se volvieron a revisar por un tercer experto.

Hay, en total, 5.863 imágenes de rayos x que están guardadas en formato JPEG. La carpeta entera pesa 1,15 Gbytes. Cada imagen tiene unas dimensiones distintas, por lo tanto, distintos píxeles. Este *dataset* está formado por imágenes de radiografías torácicas con y sin neumonía.

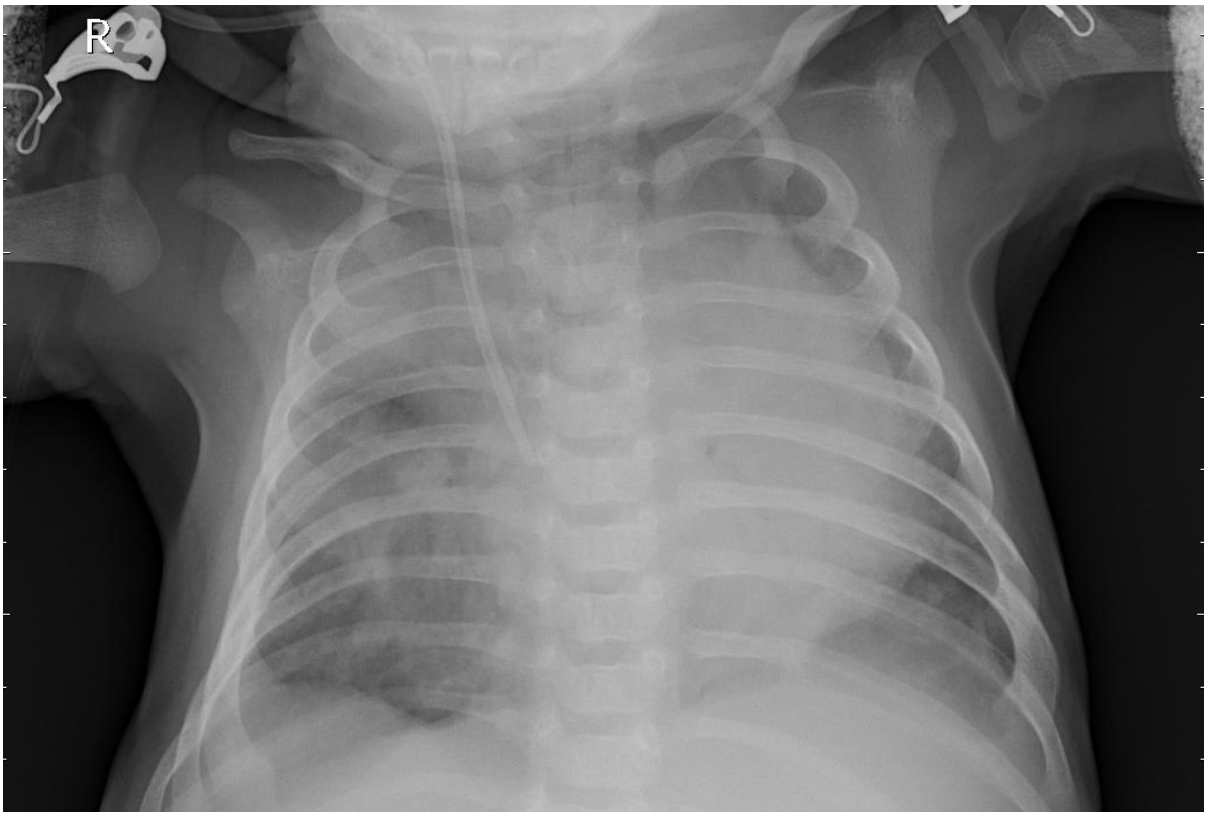


Figura 28: Imagen del dataset de una radiografía torácica donde se presenta neumonía. (Fuente:(34))

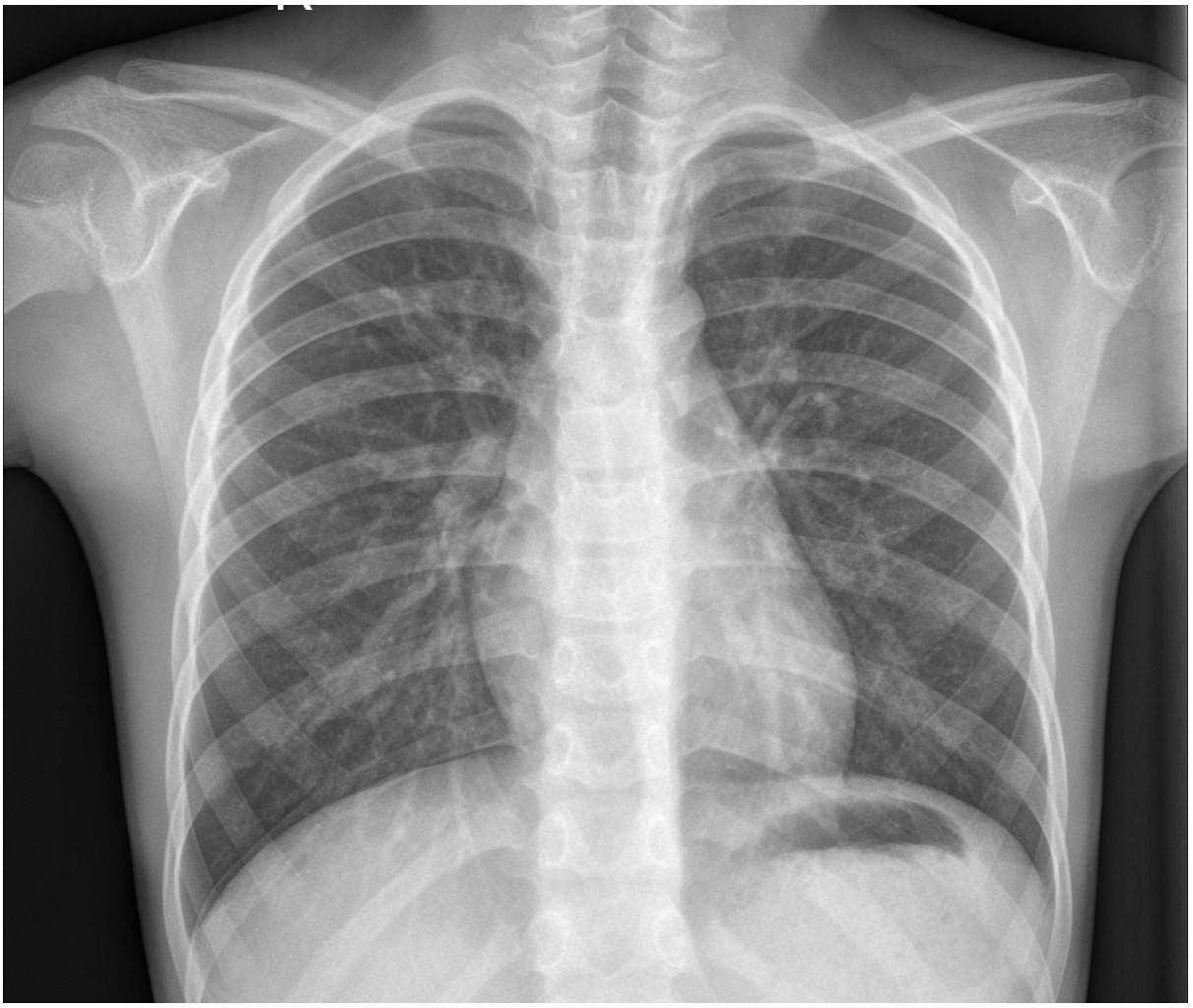


Figura 29: Imagen del dataset de una radiografía torácica normal. (Fuente:(34))

Se observa en las anteriores imágenes la comparación que hay entre unos pulmones infectados por neumonía y unos sanos. En los infectados, se aprecia esa zona blanca que se hablaba anteriormente.

La carpeta a descargar se llama *Chest_xray.* En esta hay tres carpetas; *test*, *train* y *val.* Y, en cada una de estas hay otras dos carpetas más; *normal* y *pneumonia*. Es en estas últimas donde están las imágenes. Véase que los nombres de las carpetas están en Inglés.

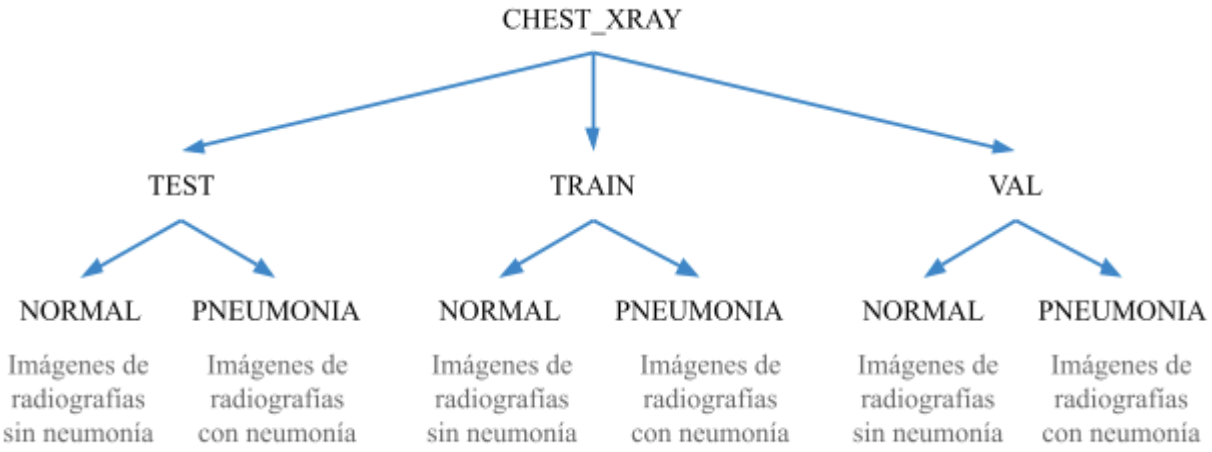


Figura 30: Estructura del dataset de Kaggle.

Para entrenar el modelo, se cambiará un poco la estructura de este dataset. Para empezar, se utilizarán solamente las imágenes de las carpetas *Train* y *Val,* y estas se unirán creando una nueva carpeta llamada *Imágenes modelo*. Dentro de esta habrá dos carpetas; Normal y Neumonía. También se cambiará el nombre de las imágenes enumerándolas. La carpeta *Test* se dejará para probar el modelo. Por lo tanto, el dataset tiene la siguiente estructura.

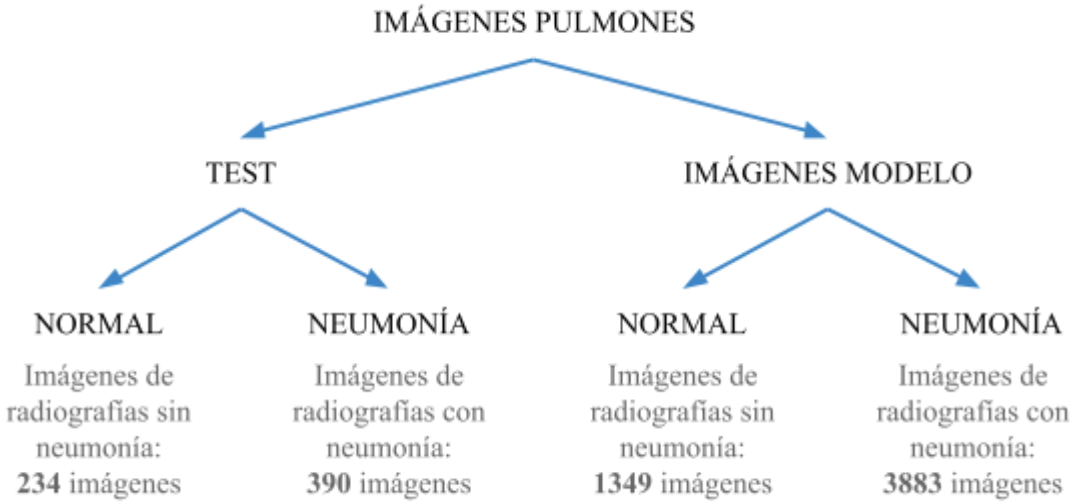


Figura 31: Estructura del nuevo dataset.

4.3. Python

El funcionamiento de un ordenador está basado en combinaciones de números que se procesan y se convierten en acciones. Estos son muy difíciles de entender. Es por eso que existen los lenguajes de programación. Estos permiten que el programador pueda comunicarse con el ordenador, darle una serie de instrucciones e indicarle que quiere hacer. En cierta medida estos lenguajes son traducciones más simples del lenguaje del ordenador. Existen varios lenguajes como C, C++, Java, PHP, Python, C#, ASP, entre otros. Dependiendo de lo que se quiera desarrollar, algunos lenguajes son más útiles que otros.

FINALIDAD	LENGUAJES		
Lenguajes para programar páginas web	JavaScript	PHP	HTML
Lenguajes para programar videojuegos	C++	C#	Ruby
Lenguajes enfocados a la ingeniería	Python	Matlab	Objective-C
Lenguajes para crear aplicaciones Apple	Swift	Objective-C	
Lenguajes para crear aplicaciones Android	Java	Kotlin	HTML+CSS

Tabla 1: Lenguajes informáticos y sus principales aplicaciones. (Fuente: (35))

Para realizar el clasificador se ha decidido utilizar el lenguaje de programación Python (56). Es un lenguaje limpio y sencillo de aprender. Además, ya se tenían conocimientos previos de este lenguaje. También, se ha escogido Python porque es uno de los mejores lenguajes si se busca desarrollar proyectos de inteligencia artificial, machine learning y deep learning (38). Una de las ventajas de este programa es que contiene una amplia variedad de librerías y entornos de trabajo, como por ejemplo PyTorch y Fast.ai.

Figura 32: Logo de Python. (Fuente:(36))

Para escribir el código del clasificador se utilizará la librería de deep learning; Fast.ai. El objetivo de dicha librería es agilizar lo máximo posible el entrenamiento de las redes neuronales profundas. A su vez, Fast.ai se basa en buscar las mejores prácticas de deep learning intentando que el entrenamiento se vuelva rápido y preciso. Fue fundada por Jeremy Howard y Rachel Thomas en 2016 (37).

Esta librería está escrita en Python y construida sobre PyTorch, uno de los principales entornos de trabajo de deep learning (57). También, Fast.ai se caracteriza por, además de ser

una librería, ofrecer cursos de aprendizaje y crear una comunidad de programadores aprendiendo sobre deep learning.

Por último, para poder realizar los códigos, se ha utilizado Google Colab, un servicio cloud basado en los Notebooks de Jupyter, que permite escribir en Python (58). Desde este se puede acceder a todos los archivos de Google Drive (60), lo que es una ventaja. También, ofrece un uso gratuito de las GPUs.

Los programas de los modelos se guardarán en Github, una plataforma que permite el alojamiento de códigos, para el control de las diferentes versiones y la colaboración entre distintos programadores (59).

4.4. Transfer learning

Tras anunciar la metodología que se usará para realizar el algoritmo clasificador, es el momento de crear dicho código según las especificaciones. Como se observará, no se ha obtenido un buen rendimiento en el primer intento. En este apartado se estudiará cómo se ha implementado la metodología transfer learning para crear los modelos.

También, como se quiere obtener los resultados con mejor desempeño posible, se entrenará cuatro modelos a partir de cuatro modelos entrenados previamente. Estos serán los siguientes:

- ResNet 18
- ResNet 34
- ResNet 50
- AlexNet

Antes de seguir, se dirá que la estructura del código se ha creado independientemente del tipo de modelo pre entrenado que se ha utilizado. Por lo tanto, el análisis de la metodología se hará, solamente, del modelo obtenido con ResNet 18.

Para realizar el algoritmo, primero, se estableció la estructura que debía seguir este. De forma conceptual, el código debería contener tres partes esenciales. La primera fue la preparación del dataset. La segunda fue utilizar las CNN como extractor de características. La última era realizar fine-tuning. Fue de gran importancia que, en el código, se escribiera, de forma correcta, cada una de estas partes.

Al final, los códigos empleados para crear los modelos se pueden encontrar en el siguiente link:

https://github.com/anablasi/Detector-de-neumonia (61)

4.4.1. Preparación del dataset

Tras ordenar el dataset como se ha explicado anteriormente, se decide que la manera más óptima para preparar los datos que van a ser entrenados es utilizar la clase *dataloaders*. Para ello, antes se tendrá que utilizar otra clase. La que encaja más con las especificaciones ha sido *DataBlock*. A esta clase se le han de definir una serie de detalles.

El primero es definir qué es la entrada. Lógicamente se ha determinado que al modelo le entran imágenes de tipo categórico, es decir, que se dividen en categorías. También se utilizará la función *get_image_files* para que el modelo pueda obtener las imágenes. Esta clase separa el dataset en datos de entrenamiento y datos de validación. Por lo tanto, es necesario especificar el porcentaje de datos totales que se validará. En este caso se ha decidido que un 20% del conjunto de datos totales sean de validación. La siguiente especificación es definir las etiquetas, en este caso serán; normal y neumonía. Para eso se utiliza la función *parent_label.* Esta función utiliza los nombres de las carpetas que pertenecen las imágenes para definir las etiquetas. Por último, se cambia el tamaño de las imágenes a 224x224 píxeles porque siempre da muy buenos resultados. También se busca que todas las imágenes tengan la misma dimensión para que se puedan cotejar en tensores que se pasarán a la GPU.

```
chest = DataBlock(
   blocks=(ImageBlock, CategoryBlock),
   get_items= get_image_files,
   splitter=RandomSplitter(valid_pct=0.2,
seed=42),
   get_y= parent_label,
   item_tfms=Resize(224)
   )
```

Código 1: Creación de los conjuntos de entrenamiento y validación.

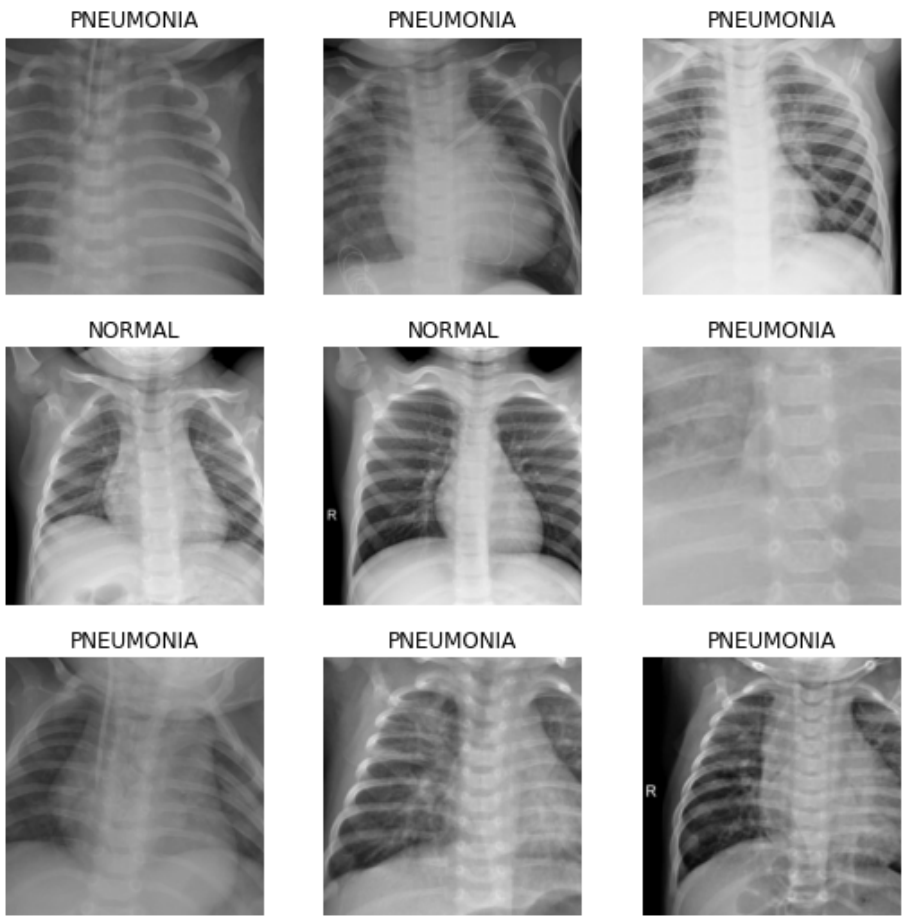
A esta nueva variable *chest*, se le pasó la clase *dataloaders* para preparar las imágenes que se van a entrenar. Es en esta clase donde se especifica dónde están estos datos, cuál es la ruta de las imágenes. También, es necesario especificar el *batch size*, el tamaño del lote. Este lote es un conjunto de imágenes que se usa en una iteración (es decir, una actualización del gradiente) del entrenamiento de modelos (54). Se ha decidido utilizar un *batch size* de 20. Por último, se ha decidido especificar el *wd,* que es el decaimiento del peso que se utilizará al entrenar el modelo.

```
dls = chest.dataloaders(IMAGES_FOR_MODEL, bs=20,
wd=1e-1)
```

Código 2: Preparación del dataset que se va a entrenar.

En la siguiente celda se puede observar una parte del lote:

```
dls.show_batch()
```



Código 3: Un conjunto de imágenes del batch.

4.4.2. CNN cómo extractor de patrones

A continuación, se utilizará las redes neuronales convolucionales con el objetivo detectar los patrones. Para ello se decidió utilizar la función de Fast.ai; *cnn_learner*. A esta función se le

ha de especificar los datos, el modelo y las métricas. Las métricas son simplemente las funciones que adoptan los tensores de entrada y destino, también, devuelve la métrica de interés para el entrenamiento (37). Se ha definido *metrics* cómo *accuracy*, es decir, la precisión.

```
learn = cnn_learner(dls, resnet18,
metrics=accuracy)
```

Código 4: Creación del nuevo modelo.

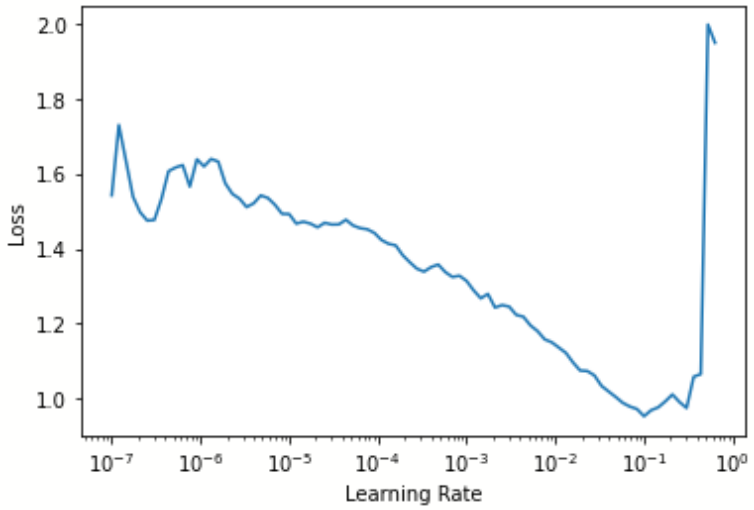
4.4.3. Fine-tuning

A continuación, se analizó la mejor manera para realizar fine-tuning. Se analizaron las diferentes funciones que tenía Fast.ai para realizar este método y, al final, se decidió utilizar una función que se llama *fine_tune,* esta se creó principalmente para Transfer learning.

Tras estudiar las diferentes especificaciones que podía tener esta función se consideró que, a esta función, se le debía definir el número de *epochs* y el *learning rate.* Este último se usa para entrenar un modelo a través del descenso de gradientes. Primero, se busca cúal es este learning rate y luego se crea la función. En la fase de experimentación se ha decidido hacer 10 *epochs.*

```
learn.lr_find()
```

```
   SuggestedLRs(lr_min=0.010000000149, lr_steep=1.5848931980e-06)
```




```
learn.fine_tune(10, base_lr=1e-1)
```

Código 5: Entrenamiento fine-tuning.

Tras ejecutar la función *fine-tune* con las especificaciones que se han definido anteriormente, obtenemos los siguientes resultados:

epoch	train_loss	valid_loss	accuracy
0	0,46130	0,29563	0,93308
1	0,30599	151,48889	0,84704
2	0,51731	3,06660	0,94933
3	0,34499	2,95476	0,76577
4	0,22527	0,13085	0,97132
5	0,20603	0,11737	0,96367
6	0,12065	41,13545	0,95411
7	0,05874	2,14526	0,96558
8	0,04635	0,04952	0,98088
9	0,02711	0,47679	0,98279

Tabla 2: Resultados del modelo ResNet 18 en la fase 1 de fine-tuning.

A simple vista, se observa una serie de anomalías. El *train_loss* y el *valid_loss* suben y bajan a raíz que van pasando los epochs. Se observa, también, que en los *epochs* 1 y 6, el *valid_loss* es demasiado grande. Además, el *accuracy* no es nada preciso y varía bastante.

Se llega a la conclusión que el rendimiento obtenido no ha sido correcto. El *valid_loss* no debería ser tan grande, este tendría que ser ligeramente diferente al *train_loss* (1).

Tras estudiar las posibles causas de estas anomalías se descubre que no se ha realizado fine-tuning de manera correcta. Se ha descartado el problema de overfitting porque, aunque el *valid_loss,* cuando sucede overfitting, también es muy alto, este solo debería crecer. En cambio, en este caso, sube y baja. Además, cuando se analiza la metodología que sigue Fast.ai para realizar transfer learning, se descubre cual ha sido el problema.

El error yacía en cómo se ha utilizado la función *fine_tune.* En la fase anterior se buscaba el *learning_rate* del modelo para definirlo en dicha función. Se descubre, que cuando se realiza transfer learning, este no debe ser predefinido, se debería utilizar un *learning_rate* discriminativo. Al especificarlo, cuando se entrenaba el modelo, no se aprovechaba al máximo la red ya entrenada.

Por lo tanto, se decide realizar el método de *fine_tune* solamente de la siguiente manera:

```
learn.fine_tune(10
)
```

Código 6: Entrenamiento fine-tuning.

Cuando se vuelve a ejecutar el código tras realizar los cambios propuestos se observa, ya se ha solucionado el problema anterior, no se presentan las anomalías anteriores. Las cifras que refleja la tabla entran dentro del rango correcto y normal. Se aprecia que el *valid_loss* y *train_loss* son reducidos y tienen una diferencia pequeña. Además al fijarse en la columna de *accuracy* se observa un rendimiento muy preciso.

epoch	train_loss	calid_loss	accuracy
0	0,19163	0,10201	0,96750
1	0,08376	0,09567	0,97419
2	0,08544	0,15823	0,95794
3	0,08565	0,09334	0,96941
4	0,08618	0,06223	0,97706
5	0,04403	0,06024	0,97992
6	0,05950	0,05734	0,97992
7	0,03750	0,03077	0,99140
8	0,03290	0,02867	0,98757
9	0,03091	0,02360	0,99140

Tabla 3: Resultados del modelo ResNet 18 en la fase 2 de fine-tuning.

Como conclusión del proceso de análisis de cómo se realiza *fine-tune* para transfer learning, se dirá que no se debe especificar el *learning_rate*. Los resultados obtenidos cuando se utiliza la función correctamente, son muy precisos, y, por ello, se ha decidido entrenar el resto de modelos a partir de esta estructura.

5. RESULTADOS

Una vez se ha definido la metodología que se usará para crear el modelo y se ha realizado el algoritmo, se hará un análisis de los resultados obtenidos. Se recuerda que se han obtenido cuatro modelos distintos a partir de cuatro modelos pre entrenados. Al final, se estudiará cuál ha obtenido el mejor rendimiento.

5.1. ResNet

Primero, se entrenará el modelo con las redes residuales (capítulo 3.6.1.1). Se ha decidido utilizar tres tipos de redes; ResNet 18, ResNet 34 y ResNet 50. Estas redes residuales se diferencian en su tamaño, el número de capas. Las razones por las cuales se han escogido estas tres y no otras de mayor tamaño son porque, primero, el ordenador no soportaría tal espacio, y, segundo, se cree que con estos ResNet se obtendrán resultados lo suficiente precisos.

En los siguientes subapartados, se puede observar los resultados del entrenamiento de los tres tipos de ResNet utilizados. Se observará la evolución del *train_loss* y *valid_loss* a partir de los *epochs*, también la evolución del *accuracy*. Además, se utilizarán las matrices de confusión para analizar los modelos. El programa utiliza las imágenes de validación para observar la confusión obtenida. Estas matrices te muestran cuántas veces se ha predecido mal una imagen según el tipo de error.

5.1.1. ResNet 18

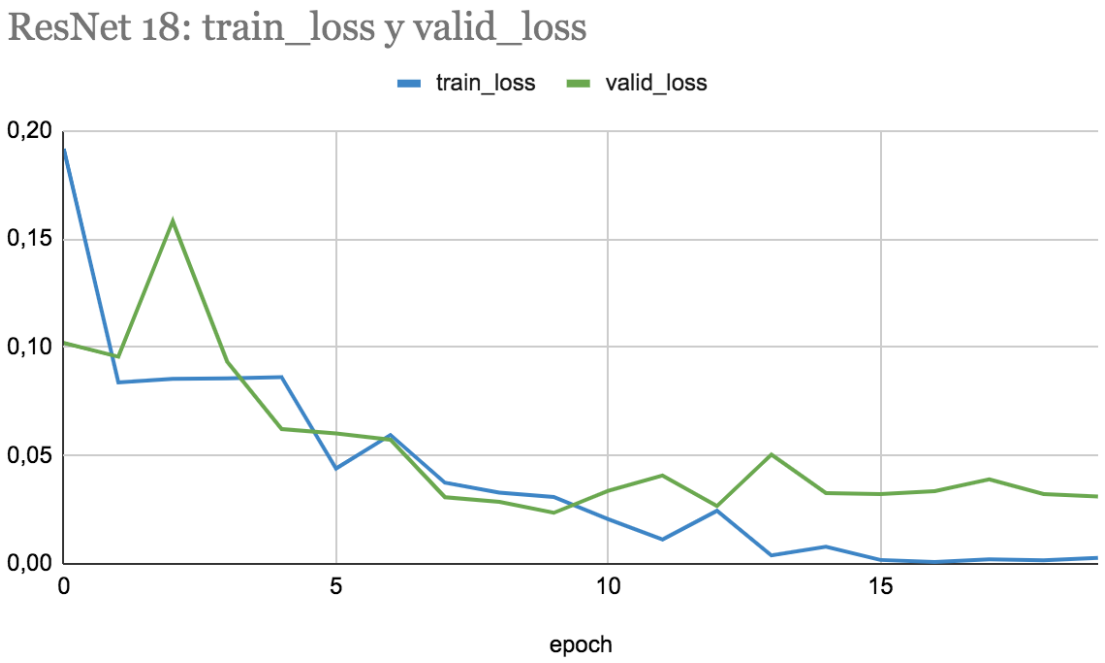


Figura 33: Evolución de train_loss y valid_loss frente los epochs en el modelo ResNet 18.

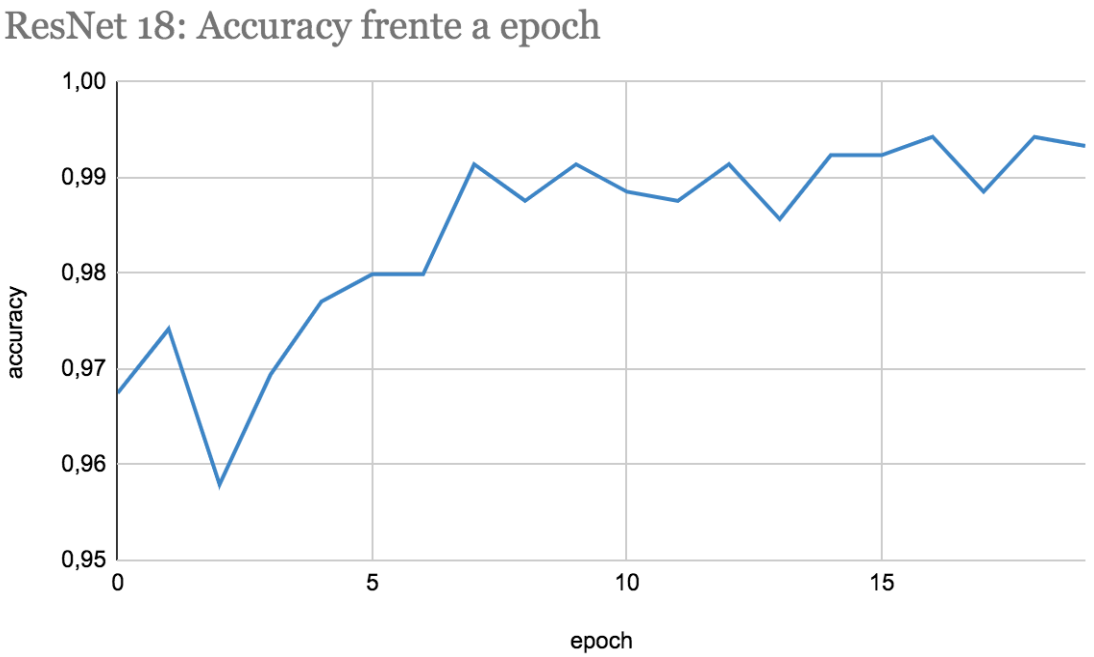


Figura 34: Evolución de accuracy frente los epochs en el modelo ResNet 18.

Se observa como el *train_loss* y el *valid_loss* van disminuyendo como deberían. Además, se aprecia que a partir del *epoch* 9 el *valid_loss* ya es mayor que el *train_loss*. También se observa una evolución del *accuracy* muy favorable. El modelo ha conseguido alcanzar una precisión del 0,9931.

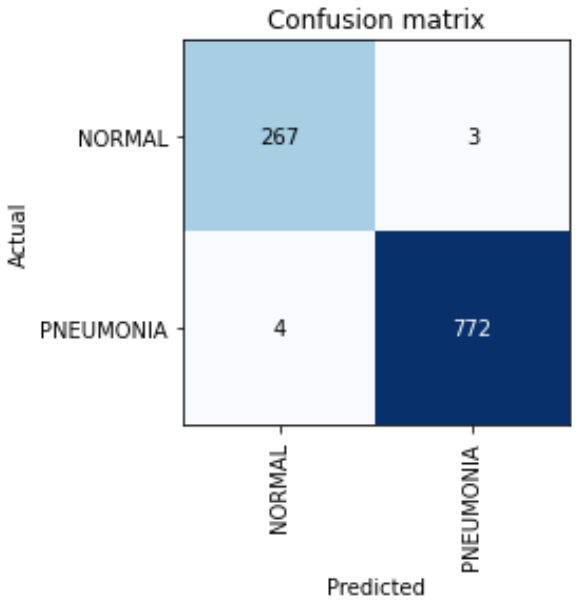


Figura 35: Matriz de confusión del modelo ResNet 18.

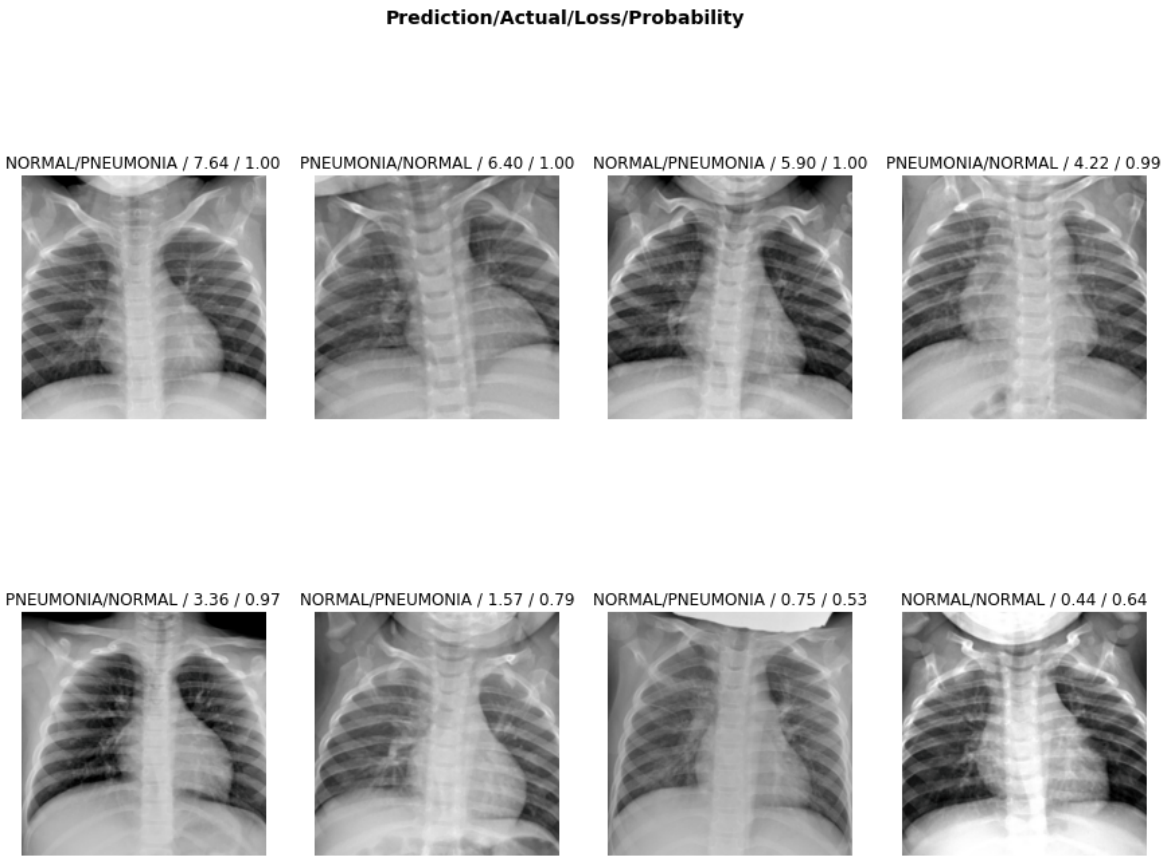


Figura 36: Las 8 imágenes más confundidas del modelo ResNet 18.

Tras estudiar los resultados obtenidos a partir de la matriz de confusión se observa que de 1046 imágenes de validación, solamente se han confundido 7. Este modelo ha dado resultados muy buenos. En 4 de ellas se ha previsto que eran normales, cuando, en realidad, eran neumonía. Con las 3 imágenes restantes ha sucedido lo contrario. Se ha previsto que eran neumonía cuando en realidad eran normales. Justamente, se observan estas 7 imágenes en la figura 32. La última imagen se ha predecido adecuadamente.

5.1.2. ResNet 34

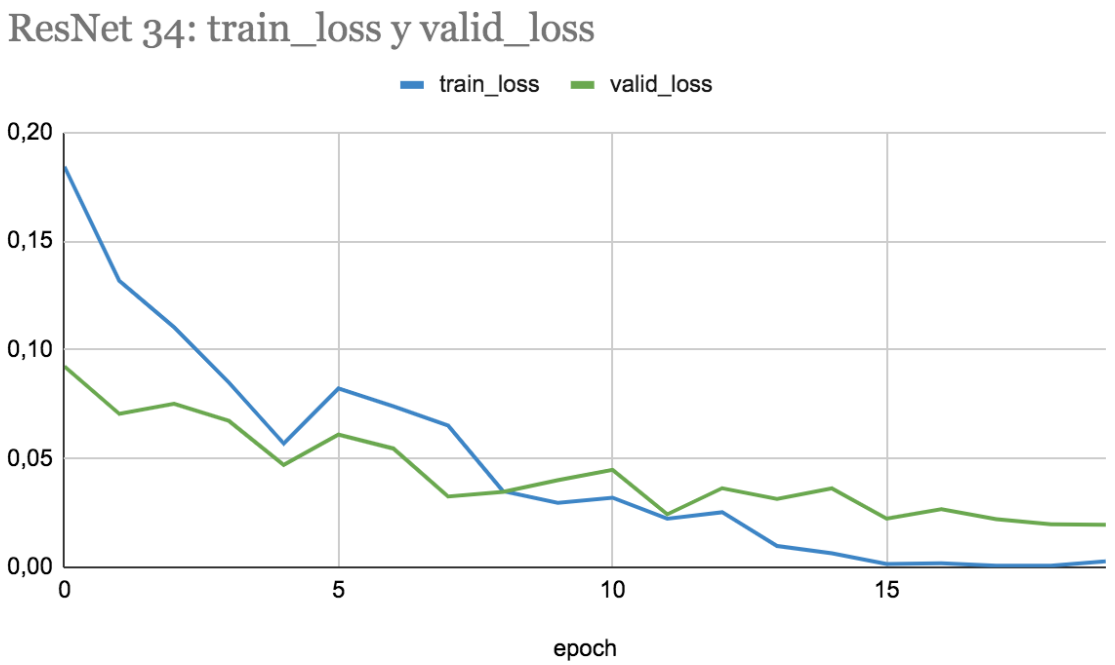


Figura 37: Evolución de train_loss y valid_loss frente los epochs en el modelo ResNet 34.

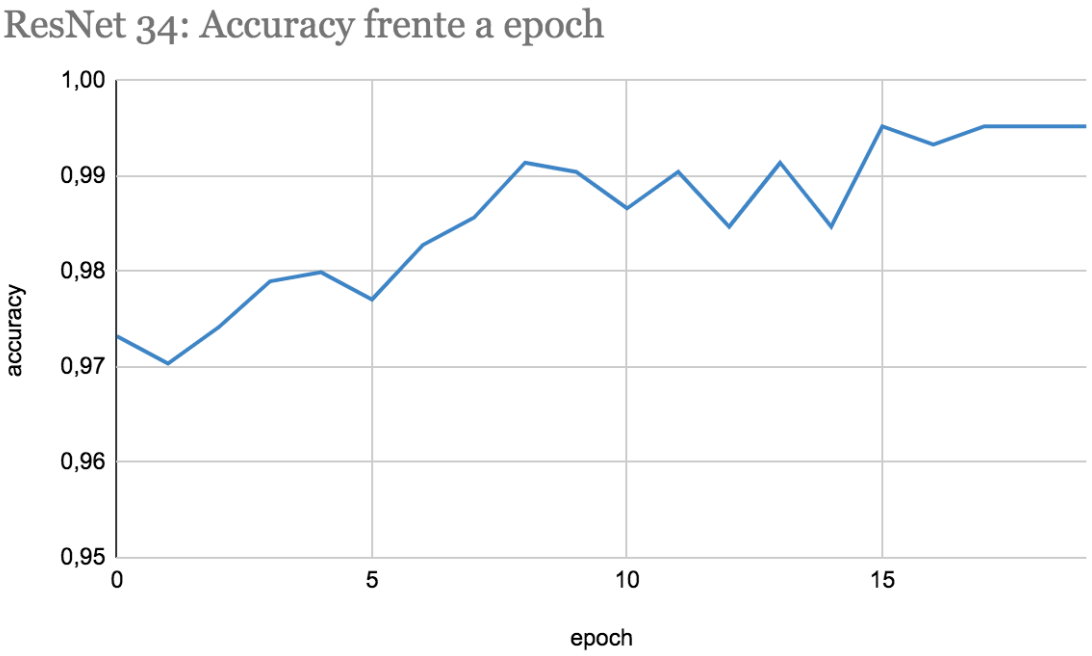


Figura 38: Evolución de accuracy frente los epochs en el modelo ResNet 34.

Se aprecia en la figura 38 como el *train_loss* y el *valid_loss* disminuyen frente a los epochs. También con este modelo se ha conseguido que en el epoch 8 el v*alid_loss* sea mayor que el *train_loss*. Además, si se analiza la figura 39, se aprecia que con el modelo ResNet 34 también se han conseguido muy buenos resultados. La precisión ha llegado a 0,99522.

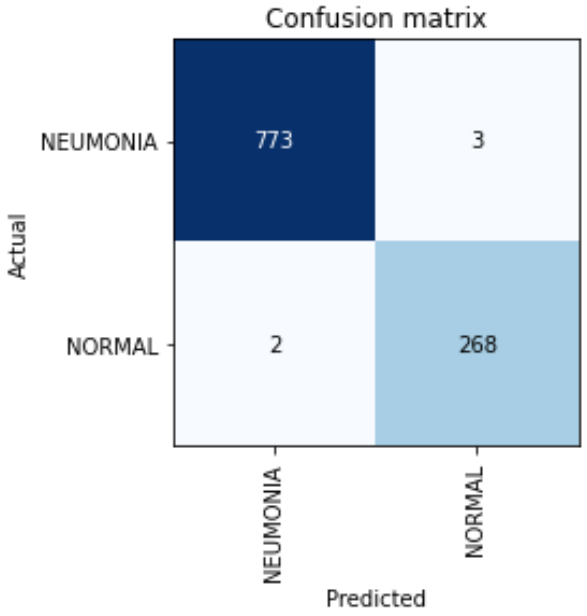


Figura 39: Matriz de confusión del modelo ResNet 34.

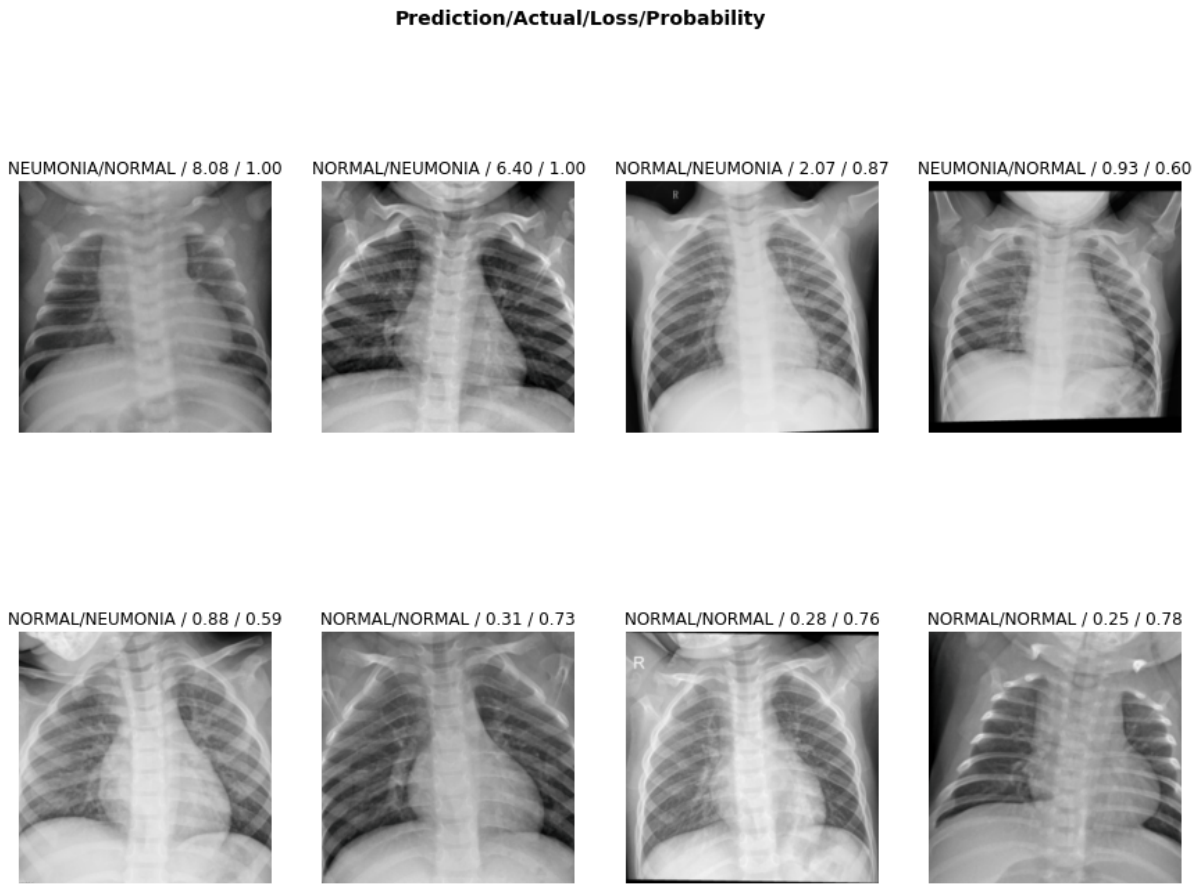


Figura 40: Las 8 imágenes más confundidas del modelo ResNet 34.

En la matriz de confusión se observa que solamente hay 5 imágenes que se han confundido. Como era esperado, este modelo ha dado mejores resultados que ResNet 18, se puede demostrar en el hecho que ResNet 34 ha confundido solamente 5 imágenes y el otro 7. En 2 de ellas se ha proveído que eran neumonía, cuando en realidad, eran normales. A las otras imágenes les ha pasado lo contrario.

5.1.3. ResNet 50

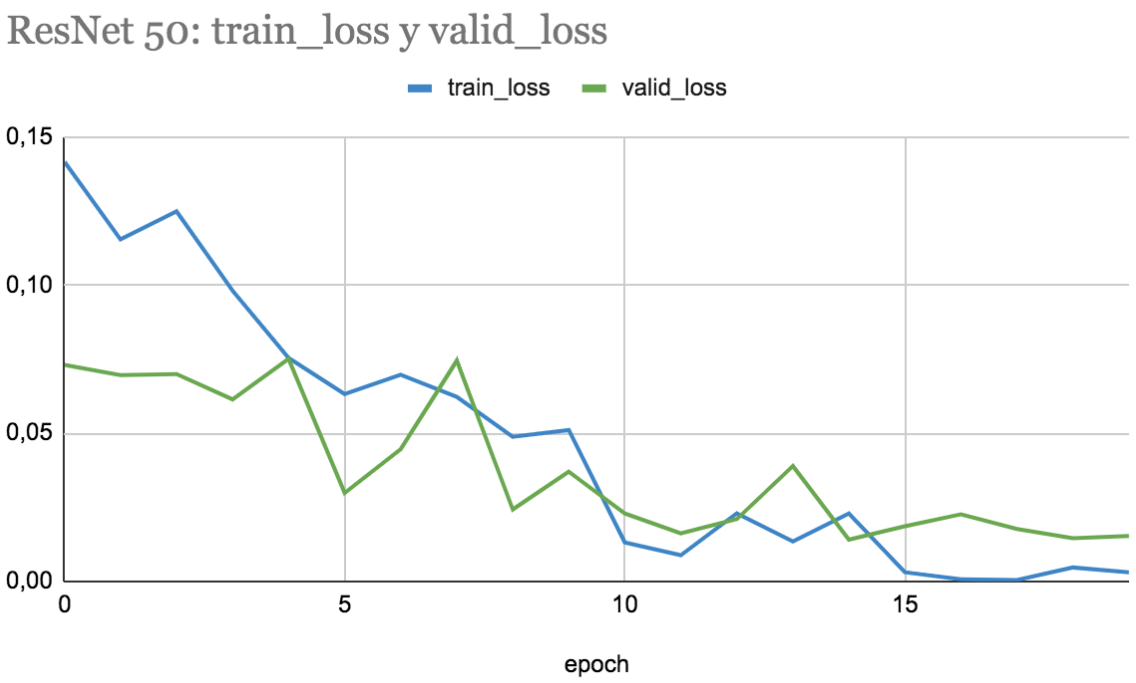


Figura 41: Evolución de train_loss y valid_loss frente los epochs en el modelo ResNet 50.

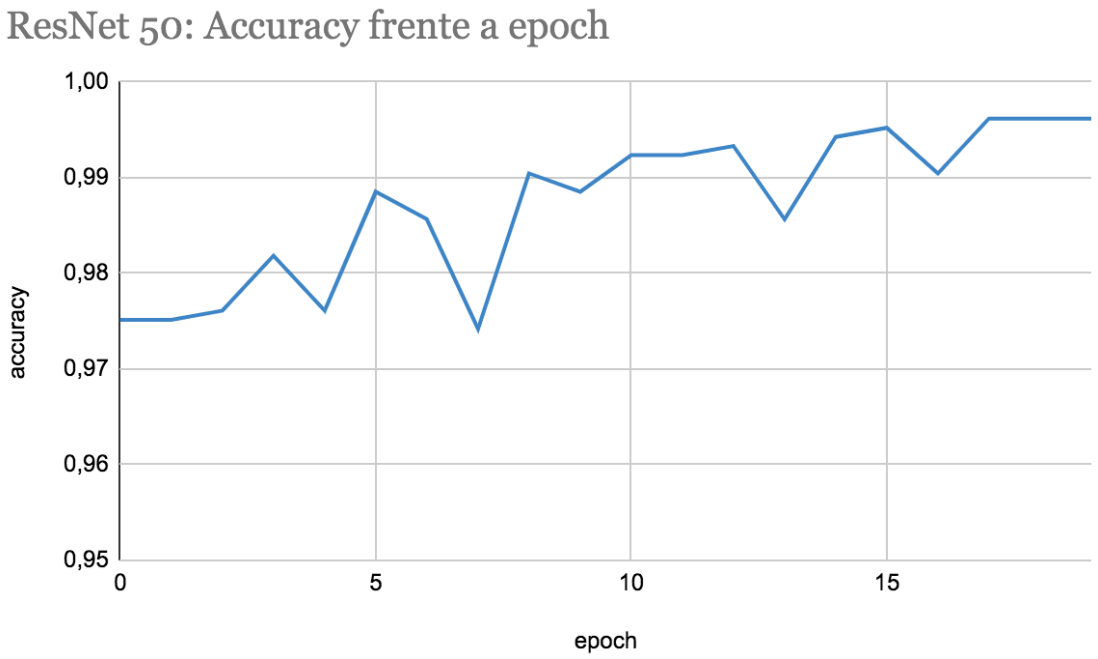


Figura 42: Evolución de accuracy frente los epochs en el modelo ResNet 50.

Por último, dentro de las redes reducidas, se ha utilizado ResNet 50. Este ha sido el modelo con el que se ha obtenido el mejor rendimiento. Se ha alcanzado un accuracy del 0.99618. Las dos tipos de pérdidas también han disminuido. Con este modelo se ha conseguido que el *valid_loss* sea mayor que el *train_loss* en el epoch 14.

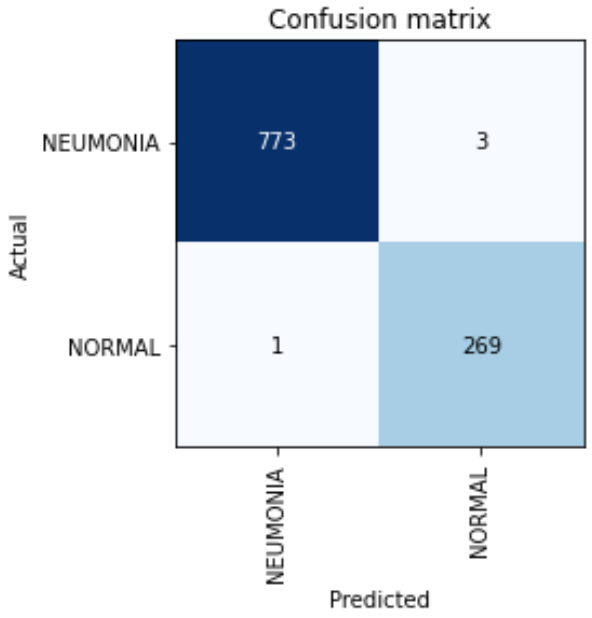


Figura 43: Matriz de confusión del modelo ResNet 50.

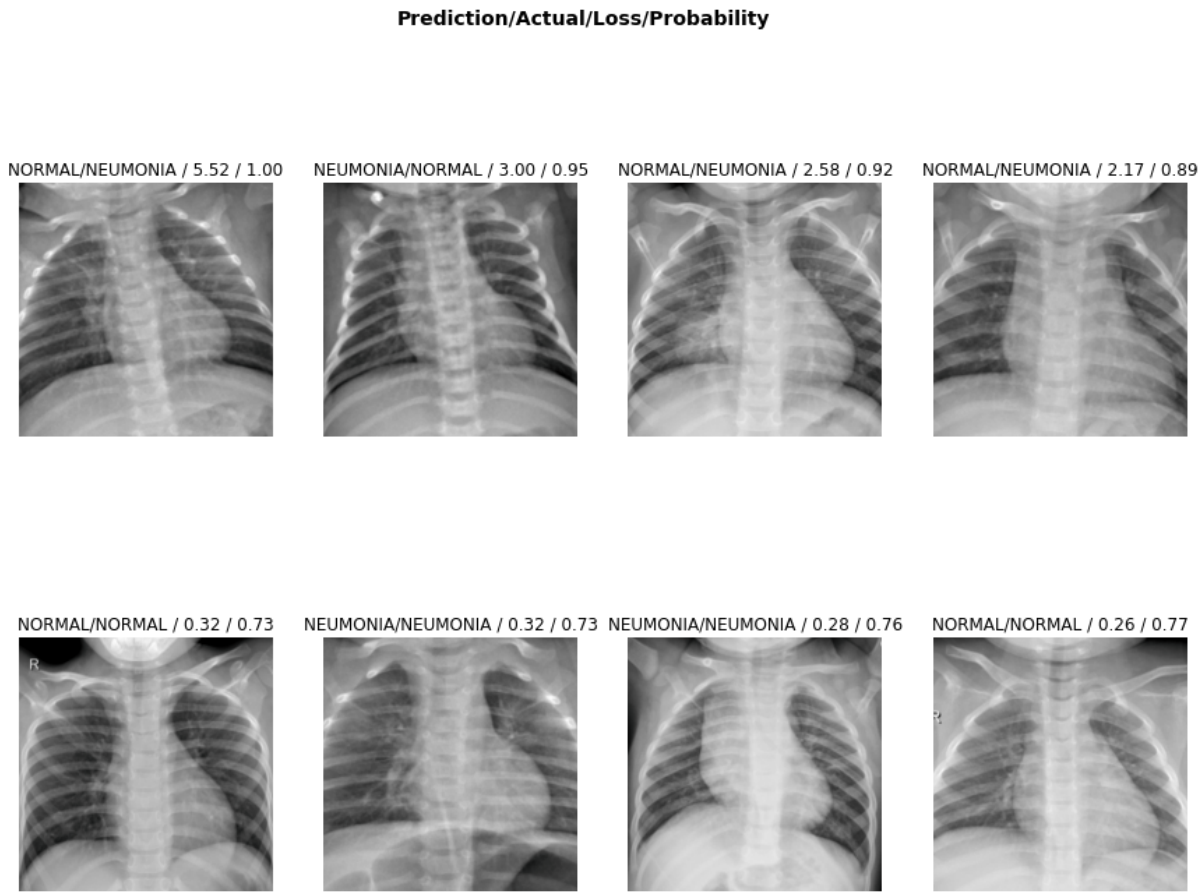


Figura 44: Las 8 imágenes más confundidas del modelo ResNet 50.

Solamente se han obtenido 4 confusiones de 1046 imágenes. En la figura 45 se puede observar, justamente, estas 4 imágenes. Resnet 50 ha seguido un muy buen entrenamiento y se refleja en los resultados obtenidos.

5.2. AlexNet

A continuación, se examinarán los resultados obtenidos a partir del modelo AlexNet (capítulo 3.6.1.2). También se ha entrenado con 20 *epochs*.

Figura 45: Evolución de train_loss y valid_loss frente los epochs en el modelo AlexNet.

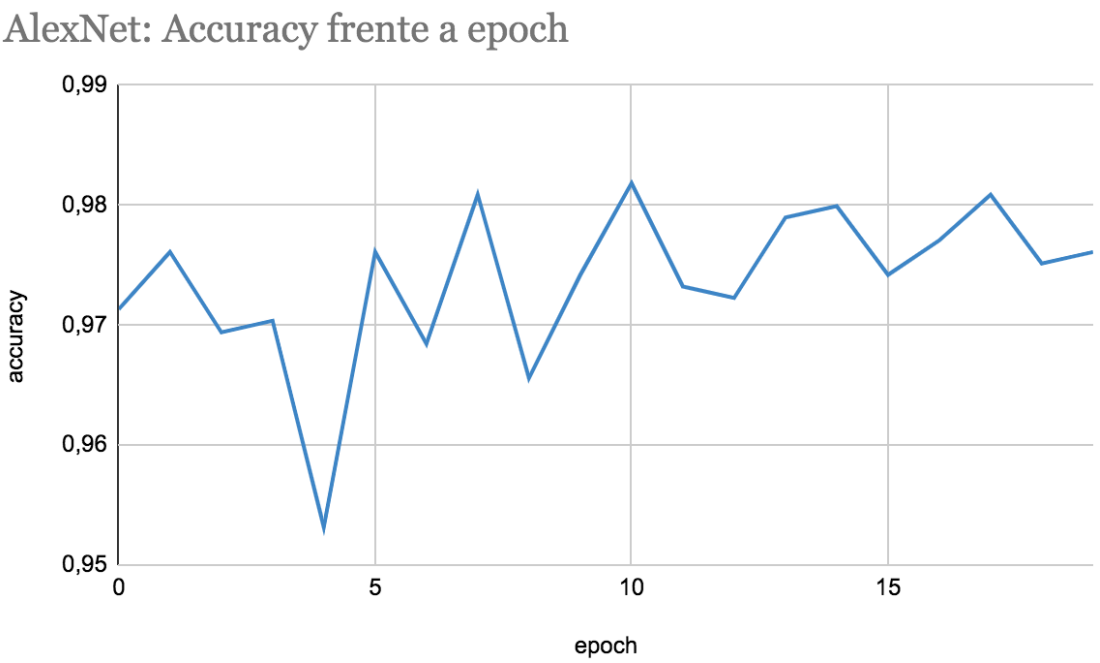


Figura 46: Evolución de accuracy frente los epochs en el modelo AlexNet.

A partir de AlexNet se ha obtenido el modelo que ha tenido el peor rendimiento entre todos los modelos entrenados. Como se observa en las figuras 46 y 47, también ha sido el que ha realizado más fallos. El *train_loss*, excepto en algunos *epochs*, va disminuyendo como se espera, en cambio, el *valid_loss* no. Aun así, se observa que en el *epoch* 4, el *valid_loss* aumenta hasta ser mayor que el *train_loss*.

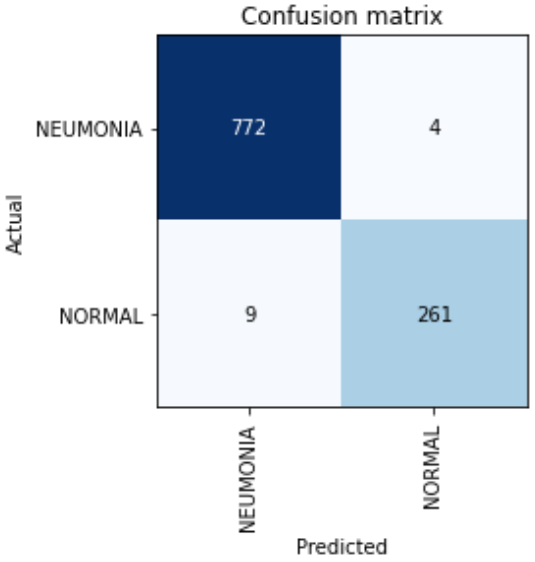


Figura 47: Matriz de confusión del modelo AlexNet.

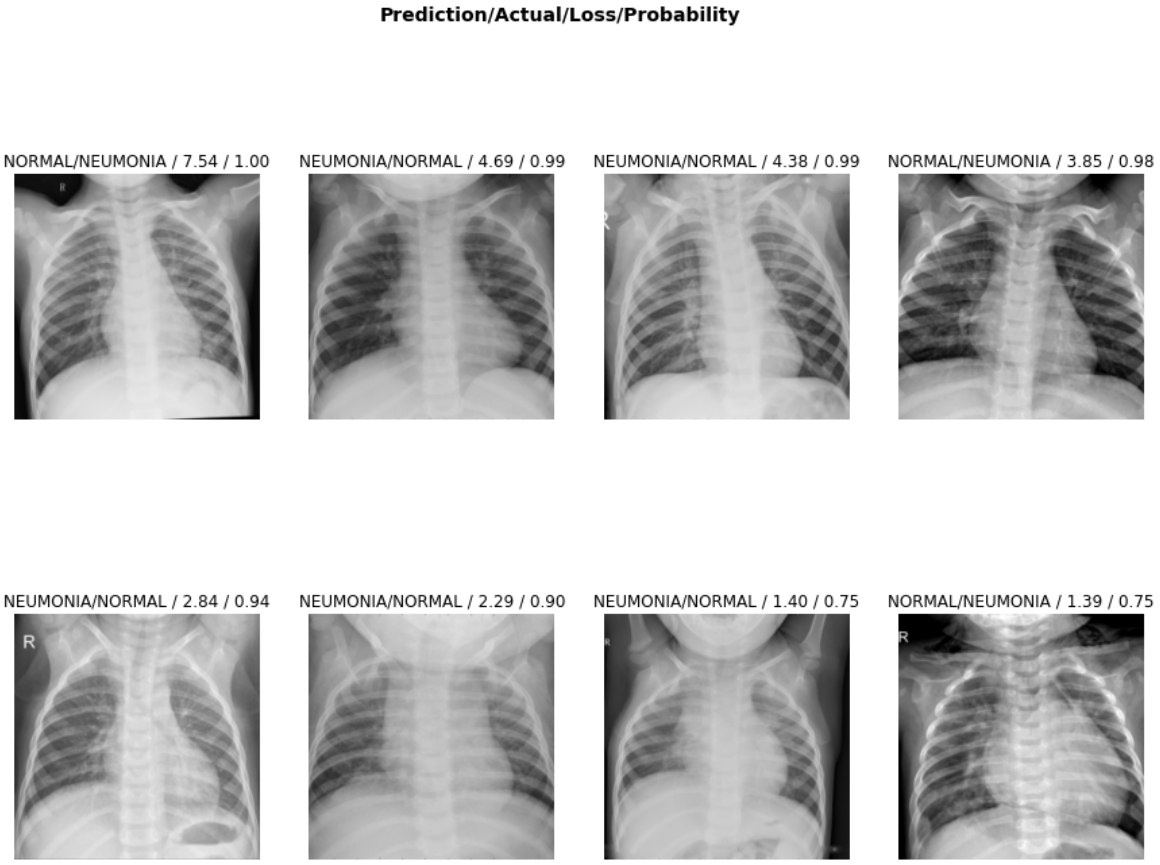


Figura 48: Las 8 imágenes más confundidas del modelo AlexNet.

En la matriz de confusión se observa como 9 imágenes se han predecido erróneamente, el modelo creía que eran neumonía cuando actualmente eran normales. También, el modelo creía que 4 imágenes eran normales cuando en realidad eran neumonía.

5.3 Resultados finales

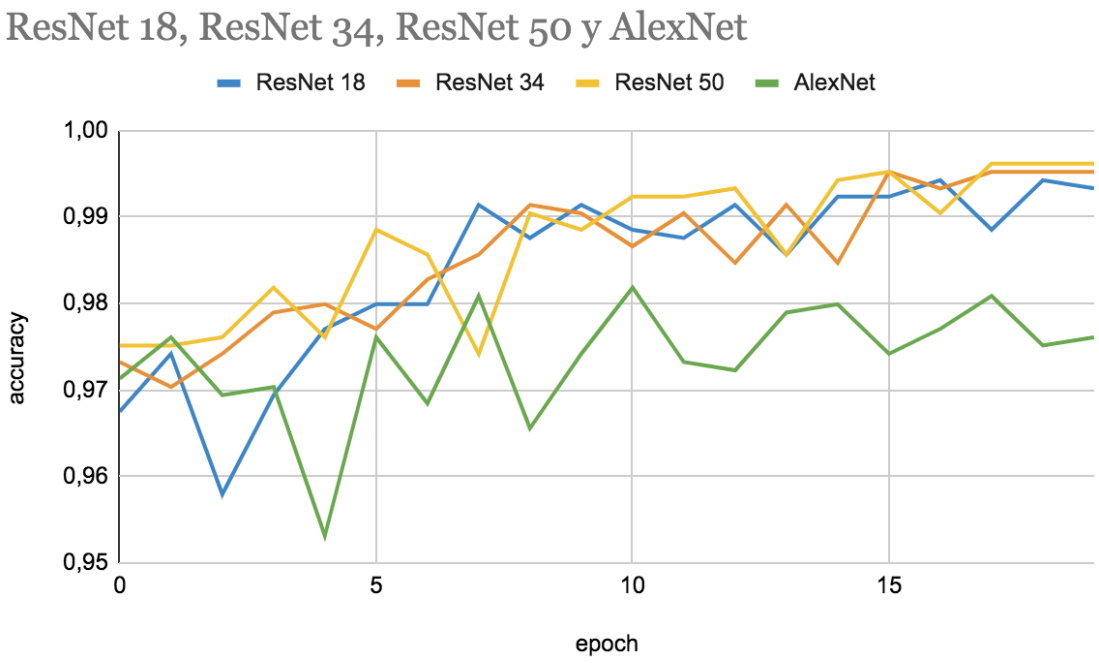


Figura 49: Evolución de accuracy frente los epochs en el modelo ResNet 18, ResNet 34, ResNet 50 y AlexNet.

MODELO	train_loss	valid_loss	accuracy
ResNet 18	0,00281	0,03105	0,99331
ResNet 34	0,00276	0,01952	0,99522
ResNet 50	0,00135	0,02583	0,99618
AlexNet	0,01294	0,10771	0,97610

Tabla 4: Resumen de los resultados de todos los modelos.

Los resultados obtenidos han superado las expectativas, sobretodo con las redes reducidas que, en las tres, se ha alcanzado un *accuracy* de más del 99%. Si se comparan estas tres, se observa como cuantas más capas y más parámetros tenga el modelo, mayor precisión se obtendrá y, por lo tanto, tendrá una mayor capacidad de aprendizaje.

Igualmente, si se realiza una comparación general de todos los modelos, teniendo en cuenta los resultados obtenidos y las matrices de confusión, el mejor modelo ha sido ResNet 50. Le siguen ResNet 34 y ResNet 18. Como se ha comentado previamente, AlexNet es el modelo con el que se ha obtenido el peor rendimiento.

5.4 Programa clasificador

Una vez se ha seleccionado el modelo que da mejores resultados, es el momento de crear el clasificador. Para ello, se ha efectuado un programa con Colab capaz de detectar la neumonía en cualquier radiografía torácica que se quiera. En el clasificador, se carga el modelo realizado con ResNet 50 y se utiliza todo lo aprendido. Asimismo, el programa se basa en realizar una predicción sobre si en la imagen se detecta la patología o no. En las siguientes figuras se podrá ver, con más claridad, de qué forma hace esta predicción con dos imágenes extraídas de la carpeta *Test* del *dataset*. Este programa se puede encontrar en el mismo repositorio de GitHub donde también se encuentran los modelos (61).

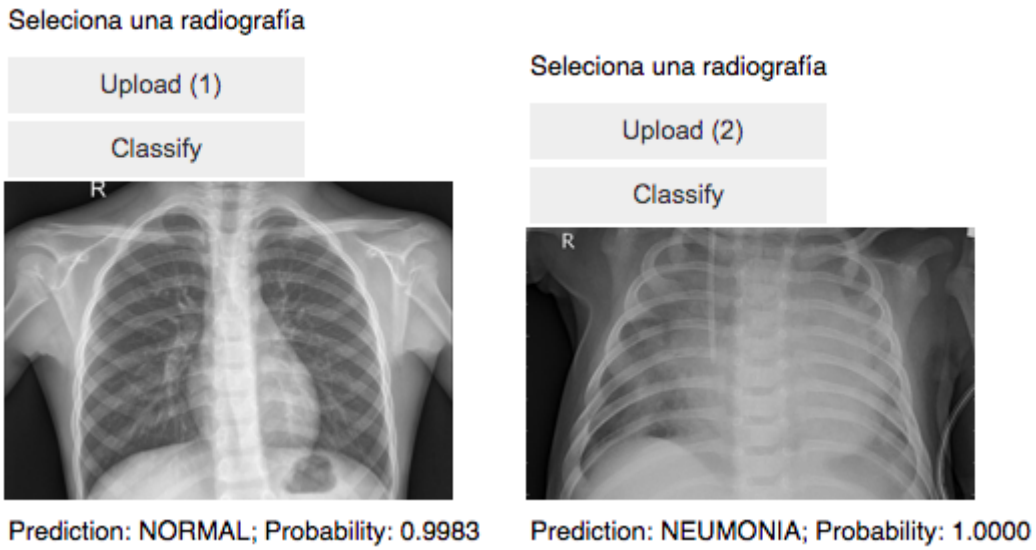


Figura 51: Programa que detecta la neumonía.

6. PLANIFICACIÓN TEMPORAL

El trabajo se ha llevado a cabo principalmente desde el mes de febrero hasta el mes de junio del año 2021. Se ha realizado un diagrama de Gantt donde se puede observar con mayor facilidad la planificación que se ha seguido, dividiéndolo en diferentes actividades.

Actividad	Febrero'21				Marzo'21				Abril'21				Mayo'21				Junio'21			
Semanas	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Actividad #1																				
Actividad #2																				
Actividad #3																				
Actividad #4																				
Actividad #5																				
Actividad #6																				
Actividad #7																				
Actividad #8																				

Tabla 5: Diagrama de Gantt para la planificación del trabajo.

Las diferentes actividades que se han realizado son las siguientes:

1. La primera actividad se basó en el estudio y aprendizaje de deep learning. Se realizó el curso de Fast.ai; *Practical Deep Learning for Coders,* se leyó libros sobre el tema y se buscó información en diferentes artículos.
2. La segunda actividad fue la elección de la patología que se iba a estudiar y clasificar. También se buscó un *dataset* de calidad.
3. En la tercera actividad se realizó la parte del código que reestructura el dataset según las necesidades.
4. En la cuarta, se realizó el código que entrenaba el modelo de ResNet 18.

5. La quinta, se basó en entrenar los modelos restantes, ResNet 34, ResNet 50 y AlexNet.
6. La sexta fue la actividad donde se realizó las actividades relacionadas con la validación de los resultados y el estudio de estos.
7. La séptima actividad es redactar la memoria.
8. En la octava actividad se corrigió y mejoró los códigos.

7. IMPACTO AMBIENTAL

En este capítulo, se analizará y estudiará el impacto ambiental que ha provocado el proyecto de este trabajo.

En el ámbito medioambiental, se prevé que el proyecto en sí tenga un impacto mínimo o prácticamente nulo. Aun así, el desarrollo de nuevas tecnologías de AI y deep learning sí que implican un gran gasto medioambiental. Estas requieren un alto consumo eléctrico, que conlleva emisiones contaminantes. Un estudio realizado por la Universidad de Massachusetts demostró que cuando se entrena un robot de AI de alta capacidad, se emite a la atmósfera una huella de carbono muy parecida a las emisiones que generan entre cuatro y cinco vehículos estadounidenses, durante toda su vida útil e incluyendo el impacto generado por la fabricación de estos (54).

Como la inteligencia artificial es claramente el futuro, se considera necesario investigar, con urgencia, nuevos métodos para solucionar el problema debido a que el calentamiento global se está convirtiendo en una de las principales crisis de los últimos tiempos.

Aun así, se insiste en que la realización de este modelo clasificatorio no ha tenido un impacto medioambiental significativo. También, es necesario comentar que el problema no yace en la existencia de la inteligencia artificial, sino en el uso excesivo de esta.

Por otro lado, con el clasificador, se busca fomentar la realización de estudios sobre la neumonía creando un modelo que ayude a simplificar el trabajo de los investigadores y médicos. El tiempo de dedicación que emplearía el experto disminuiría considerablemente con la ayuda de este modelo, lo que conlleva a una disminución del gasto económico.

Asimismo, el modelo se podría utilizar como herramienta para automatizar el diagnóstico de un paciente. De esta manera, además de disminuir el tiempo de trabajo del profesional, se conseguiría realizar el diagnóstico con más antelación. En ciertas enfermedades esto podría llegar a ser decisivo.

8. ANÁLISIS ECONÓMICO

Seguidamente, se realizará el análisis económico que ha supuesto la realización de este proyecto. Los únicos gastos que han habido están relacionados con el personal que ha realizado el trabajo de estudio y programación. Todos los servicios utilizados para la realización del modelo han sido gratuitos.

A continuación se observa la tabla de los costes de personal según el cargo de este, la tabla de gastos de servicios utilizados para la realización del modelo y la tabla del presupuesto total.

TAREA	COSTE (€/h)	TIEMPO(h)	TOTAL(€)
Desarrollo de los modelos	10	300	3000
Memoria	10	80	800
			3800 €

Tabla 6: Presupuestos de personal.

SERVICIOS	COSTE (€/año)	TIEMPO(meses)	TOTAL(€)
Amortización del ordenador	286	5	119,16 €

Tabla 7: Presupuestos de servicio.

TIPO DE COSTE	COSTE (€)
Personal	3800
Servicios	119,16
TOTAL	3919,16 €

Tabla 8: Presupuestos totales.

CONCLUSIONES

En este proyecto se han estudiado y analizado conceptos teóricos de deep learning, como las redes neuronales convolucionales. Este es un mundo que no deja de crecer y que cada vez se va fomentando más su notoriedad. Gracias a estas nuevas técnicas, se han podido crear tecnologías que facilitan la vida de las que lo utilizan, como lo que ha pretendido hacer el clasificador realizado en este trabajo.

A partir de la realización de las prácticas profesionales que se efectuaron unos meses antes de empezar este trabajo, se aprendió la metodología de ejecución del algoritmo clasificatorio. Asimismo, ha sido en este trabajo donde se ha llegado a comprender, de una forma más completa, la teoría que hay detrás de los modelos de deep learning.

El objetivo final del trabajo era crear un modelo clasificatorio que permita detectar la neumonía. La motivación nació porque se quería que el modelo fuera capaz de actuar como auxiliar en un estudio médico sobre dicha patología, sobre todo dirigido hacia niños menores de cinco años. En un futuro, incluso podría sustituirse como método de diagnóstico.

Utilizando la metodología de transfer learning, se ha diseñado, con éxito, dicho modelo. Previamente, se decidió realizar y comparar cuatro modelos realizados a partir de cuatro modelos pre entrenados con el objetivo de encontrar los mejores resultados. Al final, se ha seleccionado al modelo ResNet 50 como el modelo que ha obtenido el mejor rendimiento entre todos. Con este modelo se ha creado un programa que consigue detectar la neumonía.

Si se quisiese continuar con el trabajo, se podría realizar una página web donde el usuario podría clasificar, con mayor facilidad, las radiografías que busca estudiar a partir del programa clasificador. Lógicamente, se utilizaría el modelo entrenado a partir de ResNet 50 para crear dicha página web. También, se podría crear un modelo más avanzado que pueda identificar los diferentes tipos de neumonía.

AGRADECIMIENTOS

Me gustaría agradecer en primer lugar a la empresa Napptilus Tech por darme la oportunidad de realizar las prácticas con ellos. Me permitieron recabar la información necesaria para este proyecto y conocer más acerca de la realidad de deep learning.

En segundo lugar a Jordi Fonollosa Magrinya y Samir Kanaan Izquierdo por acompañarme en este camino y por su nivel de implicación durante todo el trabajo.

Por último a la universidad y a mi círculo cercano por confiar en mí durante todos estos años y por apoyarme en todas las decisiones que he tomado.

BIBLIOGRAFÍA

1. Chollet, F. (2018). *Deep Learning with Python*. Manning Shelter Island. ISBN 9781617294433.

2. Artificial Intelligence. (s.f). *What is Artificial Intelligence (AI)?* [en línea] Builtin. Recuperado de: https://builtin.com/artificial-intelligence . [consulta: 2 de abril de 2021].

3. Copeland, B. J. (12 de junio de 2017). *Artificial intelligence.* Britannica. Recuperado de: https://www.britannica.com/technology/artificial-intelligence [consulta: 2 de abril de 2021].

4. Wu, J. (1 de julio de 2019). *AI, Machine Learning, Deep Learning Explained Simply.* [en línea] Toward data science. Recuperado de: https://towardsdatascience.com/ai-machine-learning-deep-learning-explained-simply-7b553da5b960 . [consulta: 2 de abril de 2021].

5. Burkov A. (2019) *The Hundred-Page Machine Learning.* Quebec, Canadá

6. Ng A. (s.f). *Aprendizaje Automático.* [en línea] Coursera. Recuperado de: https://www.coursera.org/learn/machine-learning . [consulta: 2 de abril].

7. Tresp V., Ławrynowicz A., (junio 2014). *Introducing Machine Learning* [figura]. Researchgate. Recuperado de: https://www.researchgate.net/publication/268804320_Introducing_Machine_Learning [consulta: 2 de abril de 2021].

8. Equipo de Expert.ai (6 de mayo de 2020). *What is Machine Learning? A Definition.* [en línea]. Expert.ai. Recuperado de: https://www.expert.ai/blog/machine-learning-definition/ . [consulta: 4 de abril de 2021].

9. Arrabales, R. (2016). *Deep Learning: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial*. [en línea]. Webedia. Recuperado de: https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial . [consulta: 6 de mayo].

10. Ng, A. (8 de junio de 2019). *Part-1 Neural Networks and Deep Learning.* [notebook] Github. Recuperado de: https://github.com/ashishpatel26/Andrew-NG-Notes/blob/master/andrewng-p-1-neural-network-deep-learning.md#what-is-a-neural-network-nn . [consulta: 3 de abril de 2021].

11. Di Ciaccio, A. (abril de 2021) *Introduction to Neural Networks.* [PowerPoint de clase] Laboratory of machine learning, Sapienza Universidad de Roma. Recuperado de: https://elearning.dss.uniroma1.it/pluginfile.php/38326/mod_resource/content/2/LAB%20-%20Neural%20Networks.pdf .

12. Bejerano, P. (2017). *Diferencias entre machine learning y deep learning.* [en línea]. Blogthinkbig.com. Recuperado de: https://blogthinkbig.com/diferencias-entre-machine-learning-y-deep-learning . [consulta: 28 de abril].

13. Villanueva, J. D. (23 de octubre de 2020) *Redes neuronales desde cero (I) - Introducción.* [en línea] Iartificial. Recuperado de: https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/#La_Neurona_Artificial . [consulta: 8 de abril].

14. Ujjwalkarn, K. (2016). *A Quick Introduction to Neural Networks.* [en línea]. The data science blog. Recuperado de: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ . [consulta: 8 de abril de 2021].

15. Yiu, T. (2019). *Understanding Neural Networks.* [en línea]. Towards data science. Recuperado de: https://towardsdatascience.com/understanding-neural-networks-19020b758230 . [consulta: 8 de abril de 2021].

16. Freire, E. (2019). *Redes neuronales.* [en línea]. Bootcamp Al. Recuperado de: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb . [consulta: 8 de abril].

17. Olivier, E. (2020). *Qué es el Deep Learning y Cómo Nos Beneficia.* [en línea]. Genwords. Recuperado de: https://www.genwords.com/blog/deep-learning . [consulta: 9 de abril de 2021].

18. Gengiskanhg. (14 de diciembre de 2004). *Red Neuronal Artificial* [Figura]. Commons. Recuperado de: https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa . [consulta:8 de abril de 2021],

19. Malik, F. (18 de mayo de 2019). *Neural Networks Bias And Weights* [en línea]. Medium. Recuperado en: https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da . [8 de abril de 2021].

20. Fox, P. (2020). *Algoritmos de aprendizaje automático.* [en línea]. Khan Academy. Recuperado de: https://es.khanacademy.org/computing/ap-computer-science-principles/data-analysis-101/x2d2f703b37b450a3:machine-learning-and-bias/a/machine-learning-algorithms . [consulta: 16 de mayo].

21. Biswal, A. (12 de febrero de 2021). *Top 10 Deep Learning Algorithms You Should Know in 202*. [Notas del curso]. Deep learning tutorial. Recuperado en: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm . [8 de abril de 2021] .

22. Durán, J. (2018). *Técnicas de Regularización Básicas para Redes Neuronales*. [en línea]. MetaDatos. Recuperado de: https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1sicas-para-redes-neuronales-b48f396924d4 . [consulta: 9 de abril de 2021].

23. Di Ciaccio, A. (2019) *Deep learning and convolutional neural networks for image recognition.* [PowerPoint de clase] Laboratory of machine learning, Sapienza Universidad de Roma. Recuperado de: https://elearning.dss.uniroma1.it/pluginfile.php/34513/mod_resource/content/6/Image%20recognition.pdf .

24. Karn, U. (11 de agosto de 2016) *An Intuitive Explanation of Convolutional Neural Networks.* [en línea]. Recuperado de: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ . [consulta: 12 de abril de 2021].

25. Bagnato, J. I. (17 Diciembre 2020) *Aprende Machine Learning en español.* La Coruña. ISBN: 8409258161.

26. Silva, S. Freire, E. (23 de noviembre de 2019). *Intro a las redes neuronales convolucionales*. [En línea]. Bootcamp Al. Recuperado de: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8 . [consulta: 15 de abril de 2021].

27. Sumit, S. (15 de diciembre de 2018). *A Comprehensive Guide to Convolutional Neural Networks*. [en línea]. Toward data science. Recuperado de: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 . [consulta: 13 de abril de 2021].

28. MedlinePlus, Biblioteca Nacional de Medicina de EE. UU. (09 junio 2021) *Neumonía en adultos, adquirida en la comunidad.* [Figura]. Medline Plus. Recuperado de:

https://medlineplus.gov/spanish/ency/article/000145.htm . [consulta: 14 de mayo de 2021].

29. Personal clínica Mayo. (13 de junio 2020). Neumonía. [En línea]. Mayo Clinic. Recuperado de: https://www.mayoclinic.org/es-es/diseases-conditions/pneumonia/symptoms-causes/syc-20354204 . [consulta: 14 de mayo de 2021].

30. Moënne, K. (junio 2017). Neumonías adquiridas en la comunidad en niños: diagnóstico por imagenesCommunity-acquired pneumonia in children: diagnostic imaging. [en línea]. ScienceDirect. Recuperado en: https://www.sciencedirect.com/science/article/pii/S0716864013701263 . [consulta: 14 de mayo de 2021].

31. Instituto nacional de estadísticas. Defunciones según la Causa de Muerte. Recuperado de: https://www.ine.es/jaxiT3/Tabla.htm?t=7947 . [consulta: 14 de mayo de 2021].

32. Martínez, R., Vallés, J. M., Reyes, S., Menéndez, R. (s.f.) *NEUMONÍA ADQUIRIDA EN LA COMUNIDAD: EPIDEMIOLOGÍA, FACTORES DE RIESGO Y PRONÓSTICO*. [en línea] Neumomadrid.org. Recuperado de: https://www.neumomadrid.org/wp-content/uploads/monogix_4._neumonia_adquirida._epidemiol.pdf . [consulta: 14 de mayo de 2021].

33. Pfizer. (12 de noviembre de 2018). *Cada día son hospitalizadas 274 personas por neumonía en España.* [en línea]. Pfizer. Recuperado de: https://www.pfizer.es/noticia/cada_dia_son_hospitalizadas_274_personas_neumonia_espana.html . [consulta: 14 de mayo de 2021].

34. Mooney, P. (2018). *Chest X-Ray Images (Pneumonia)*. [en línea]. Kaggle. Recuperado de: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia [consulta: 5 de abril].

35. *¿Qué lenguaje de programación aprender primero?*. (2020). [en línea]. Tus clases particulares. Recuperado de: https://www.tusclasesparticulares.com/blog/que-lenguaje-programacion-aprender-primero . [consulta: 2 de junio].

36. Python. (6 de agosto de 2008). *Python logo* [Figura]. Wikipedia. Recuperado de: https://es.wikipedia.org/wiki/Archivo:Python-logo-notext.svg . [consulta:20 de mayo de 2021]

37. Fast.ai. (2018). *Practical Deep Learning for Coders, v3 | fast.ai course v3*. [en línea]. Recuperado de: https://course.fast.ai/ .

38. Protasiewicz, J. (2018). *Python AI: Why Is Python So Good for Machine Learning?*. [en línea]. Netguru. Recuperado de: https://www.netguru.com/blog/python-machine-learning . [consulta: 3 de junio].

39. Martinez, D. (2020). I*s Transfer Learning the final step for enabling AI in Aviation?*. [en línea]. Datascience.aero. Recuperado de: https://datascience.aero/transfer-learning-aviation/ . [consulta: 23 de mayo].

40. Donges, N. (2019). *WHAT IS TRANSFER LEARNING? EXPLORING THE POPULAR DEEP LEARNING APPROACH.* [en línea]. Built in. Recuperado de: https://builtin.com/data-science/transfer-learning . [consulta: 29 de mayo].

41. Brownlee, J. (2019). T*ransfer Learning in Keras with Computer Vision Models.* [en línea]. Machine Learning Mastery. Recuperado de: https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/ . [consulta: 31 de mayo].

42. Soria, E. Martin, J. D., Martinez, M., Magdalena, J. R., Serrano, A. J. (4 de septiembre de 2009). *Transfer Learning, Handbook of Research on Machine Learning Applications.* Information Science Reference. ISBN: 1605667668

43. Cs231n. (s.f) *CS231n Convolutional Neural Networks for Visual Recognition*. [en línea] Cs231n.github.io. Recuperado de: https://cs231n.github.io/transfer-learning/ . [consulta: 26 de mayo de 2021].

44. *ResNet: una comprensión simple de las redes residuales*. [en línea]. Ichi.pro Recuperado de: https://ichi.pro/es/resnet-una-comprension-simple-de-las-redes-residuales-210937146811463 . [consulta: 26 de mayo de 2021].

45. *Arquitectura de CNN: ¿Cómo funciona ResNet y por qué?*. [en línea]. Ichi.pro Recuperado de: https://ichi.pro/es/arquitectura-de-cnn-como-funciona-resnet-y-por-que-30895772711476 . [consulta: 26 de mayo de 2021].

46. Kaiming, H., Xiangyu, Z., Shaoqing, R. and Jian, S., (10 de diciembre de 2015). *Deep Residual Learning for Image Recognition*. [pdf] Recuperado de: https://arxiv.org/pdf/1512.03385.pdf . [consulta: 26 de mayo de 2021].

47. Wei, J., (3 de julio de 2019). *AlexNet: The Architecture that Challenged CNNs*. [en línea] Medium. Recuperado de: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951 . [consulta: 27 de mayo de 2021].

48. Nakanishi, K. (2020). *Exploring Convolutional Neural Network Architectures with fast.ai* [en línea]. Towards Data Science. Recuperado de: https://towardsdatascience.com/exploring-convolutional-neural-network-architectures-with-fast-ai-de4757eeeebf . [consulta: 27 de mayo de 2021].

49. Krizhevsky, A., Sutskever, I. and Hinton, G., (s.f). *ImageNet Classification with Deep Convolutional Neural Networks*. [pdf] Toronto. Rcuperado det: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf [consulta: 30 de mayo de 2021].

50. Khademi, G., (junio de 2019). *DIAGRAM OF TRANSFER LEARNING FROM A PRE-TRAINED CNN MODEL*. [figura]. Recuperado det: https://www.researchgate.net/figure/TOP-LEVEL-DIAGRAM-OF-TRANSFER-LEARNING-FROM-A-PRE-TRAINED-CNN-MODEL_fig4_333882146 . [consulta: 27 de mayo de 2021].

51. Vasani, D. (2019). *Mixed precision training using fastai*. [en línea]. Towards data science. Recuperado de: https://towardsdatascience.com/mixed-precision-training-using-fastai-435145d3178b [consulta: 15 de abril].

52. Howard, J; Gugger, S. (2020). *A Layered API for Deep Learning.* [en línea]. Fastai. Recuperado de: https://arxiv.org/pdf/2002.04688.pdf . [consulta: 19 de abril].

53. *IA, DEEP LEARNING E IMPACTO MEDIOAMBIENTAL*. (s.f.) Controlp. [en línea]. Recuperado de: https://www.controlp.es/deep-learning-ia/. [consulta: 4 de junio de 2021].

54. *Machine learning glossary.* [en línea]. Developers. Recuperado de: https://developers.google.com/machine-learning/glossary?hl=es_419 .

55. *Childhood pneumonia: Everything you need to know.* [en línea]. Unicef. Recuperado de: https://www.unicef.org/stories/childhood-pneumonia-explained. [consulta: 14 de junio].

56. *Python.* [en línea]. Recuperado de: https://www.python.org/.

57. *PyTorch*. [en línea]. Recuperado de: https://pytorch.org/.

58. *Colaboratory*. [en línea]. Recuperado de: https://colab.research.google.com .

59. *GitHub*. [en línea]. Recuperado de: https://github.com/.

60. *Google Drive*. [en línea]. Recuperado de: https://drive.google.com.

61. Blasi, A. (10 de junio de 2021). *Detector de neumonía*. [repositorio]. Recuperado de: https://github.com/anablasi/Detector-de-neumonia