Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA
DE VALLADOLID

Grado en Ingeniería Informática
Mención Computación

Clasificación de Imágenes Médicas de Rayos-X mediante Redes Neuronales Convolucionales

---

Alumno: Miguel Toquero Barón

Tutor: Teodoro Calonge Cano

Índice general

Índice de figuras

Índice de cuadros

Dedicado a
mi familia

Agradecimientos

Muchas gracias a todos los que me habéis apoyado y me habéis hecho ser mejor durante estos años.

Resumen

Hace varias décadas, la Inteligencia Artificial (IA) se convirtió en un paradigma, el cual fue la base de muchos proyectos informáticos para ser aplicados a muy diversos campos de nuestra vida. Uno de ellos fue el de la Salud, donde la influencia de la IA es cada vez mayor. Es más, hoy en día nadie conoce el límite en esta área.

Debido a la actual situación de pandemia en todo el mundo, la IA se ha aplicado también al tratamiento de enfermedades Covid-19. Precisamente, uno de los síntomas más preocupantes es la neumonía, ya que puede provocar la muerte del paciente. En este trabajo se propone un sistema de clasificación de imágenes de tórax mediante Aprendizaje Automático. En particular, se ha implementado un prototipo de Aprendizaje Profundo para realizar el correspondiente reconocimiento de imágenes. En concreto, se compone de varias capas de Neuronas Artificiales Convolucionales, así como de un conjunto de capas de neuronas densas (Perceptrón Multicapa). La precisión de la clasificación obtenida fue superior al 95 % utilizando imágenes que nunca fueron introducidas en nuestro sistema.

Además, se ha probado una reciente interpretación de imágenes perteneciente a las técnicas de Visión Artificial, en particular, Grad-CAM, que trata de retornar las áreas de imagen más influyentes utilizadas por una Red Neuronal Convolucional en un problema de clasificación. Por el momento, no se ha comprobado si las áreas obtenidas por Grad-CAM son similares a las que los médicos especialistas en pulmón utilizan para el diagnóstico de la neumonía.

Palabras clave: Aprendizaje Profundo, Covid-19, Redes Neuronales Convolucionales, Clasificación de Imágenes, Grad-CAM.

Abstract

Several decades ago, the Artificial Intelligence (AI) became a paradigm which was the basis of a lot of computing projects to be applied to a very different fields of our life. One of them was the Health, where the IA influence is growing up everyday. Even more, nowadays no body knows limit in this area.

Due to the present pandemic worldwide situation, the IA has been also applied to Covid-19 disease treatment. Precisely, one the most worrying symptoms is the pneumonia, because it could lead to the patient dead. In this work, an X-ray thorax image classification system is proposed using Machine Learning. In particular, a Deep Learning prototype was implemented to carry out the corresponding image recognition. More precisely, it is made up of several Convolutional Artificial Neurons layers, as well as set of dense neurons layers (Multiplayer Perceptron). The classification accuracy obtained was greater than 95 % using images never input to our system.

In addition, a recent image interpretation belonging to Vision Artificial techniques has been proved, in particular, Grad-CAM, that tries to return the most influence image areas used by a Convolutional Neural Network in a classification problem. As for now, it is not verified if the areas obtained by Grad-CAM are similar to the lung specialist physicians use to consider for the pneumonia diagnostic.

Keywords: Deep Learning, Covid-19, Convolutional Neural Networks, Image classification, Grad-CAM.

Parte I

Memoria del Proyecto

Capítulo 1

Introducción

El día 11 de Marzo de 2020, la Organización Mundial de la Salud (OMS) declaró que la enfermedad COVID-19 pasaba a ser una pandemia. Con esto y la situación vivida a nivel mundial, se desarrollaron muchas técnicas para la diagnosis de esta enfermedad en pacientes sospechosos de contraerla , así como de sus contactos directos. De la necesidad de obtener un rápido diagnóstico, nace este trabajo, explotando la posibilidad de detectar a nuevos pacientes, mediante el análisis de imágenes radiológicas de tórax. Con todo ello, se podrá plantear un campo de investigación multidisciplinar, con el objeto de contrastar la validez del modelo propuesto en este TFG, con el de la realidad clínica/médica.

La inclusión de la Inteligencia Artificial en todos los ámbitos de nuestra vida está a la orden del día. Una de las acusas que motivan este hecho es, sin duda, la explotación de los datos como fuente de conocimiento, tal y como propone el Aprendizaje Automático. La gran cantidad de datos generada diariamente, en la actualidad, constituye un gran valor de mercado que se puede explotar en cualquier ámbito: desde análisis para mejorar las infraestructuras de transportes [49], hasta predicciones deportivas sobre la Eurocopa de fútbol [25].

Un campo muy conocido, y cada vez más aprovechado de la Inteligencia Artificial es la visión artificial o *Computer Vision*. Se trata de una disciplina del Aprendizaje Automático, que propone tratamientos computerizados de imágenes digitales, con el fin de clasificarlas, realizar tareas de reconocimiento, etc.

En el ámbito de la salud, destaca el proyecto de Microsoft InnerEye [46], que es un software capaz de visualizar e identificar tumores u otras anomalías en las radiografías. A partir de la correspondiente imagen tridimensional, el software colorea las zonas que presentan anomalías, con el fin de prestar mayor atención a esa parte del cuerpo humano.

Los términos Inteligencia Artificial (Artificial Intelligence - AI), Aprendizaje Automático (*Machine Learning* - ML) y Aprendizaje Profundo (*Deep Learning* - DL) están realmente extendidos, muchas veces, sin conocerse muy bien la diferencia entre ellos. Pues

bien, la Inteligencia Artificial es el campo de estudio que abarca la explicación y emulación de la conducta inteligente, en función de procesos computacionales basados en la experiencia y el conocimiento continuo del ambiente [13]. Refiriéndonos a conducta inteligente, como la capacidad que tienen las máquinas para realizar tareas que, hasta el momento, son realizadas por seres humanos [53]. El Aprendizaje Automático se define como el estudio de algoritmos de computación que mejoran automáticamente su rendimiento gracias a la experiencia. Por ello, se puede emplear una medida de rendimiento que, conforme va tratando los datos (experiencia), se espera que esta mejore [44].

Existe una gran variedad de técnicas de *Machine Learning* capaces de resolver el problema de clasificación de imágenes Podríamos mencionar, por ejemplo, el algoritmo de k Vecinos Más Próximos o la Regresión Logística. Sin embargo, nos decantaremos por un algoritmo de Aprendizaje Profundo. Dentro del Aprendizaje Automático, encontramos un subgrupo de algoritmos conocidos como Aprendizaje Profundo. Consiste en la agrupación de un conjunto de procedimientos de aprendizaje automático, que intenta modelar abstracciones de alto nivel en datos usando arquitecturas computacionales, que admiten transformaciones no lineales múltiples e iterativas de datos expresados en forma matricial o tensorial [3]. Es decir, una combinación de múltiples algoritmos de Aprendizaje Automático estructurados en forma de muchas capas.

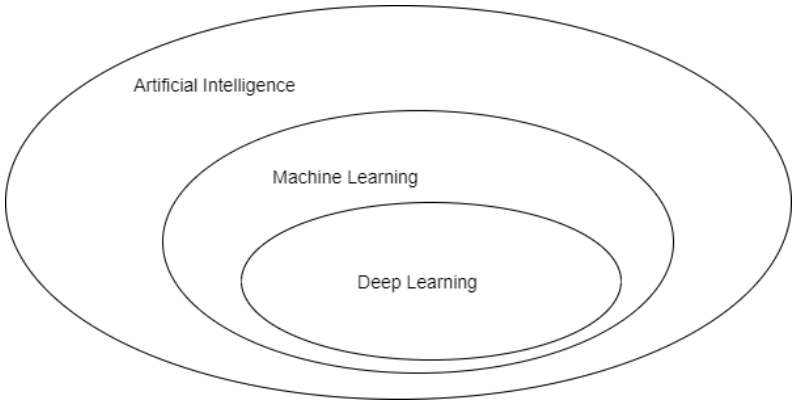


Figura 1.1: Relación entre los términos de Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo

El objetivo de este proyecto es hacer un estudio sobre técnicas de Aprendizaje Automático y una adaptación de estas al problema que tratamos. En particular, elaboraremos algunos modelos de Aprendizaje Profundo sobre un conjunto de datos, que se describirá en el capítulo 6, y el desarrollo de una aplicación fácilmente usable. Nos centraremos en los modelos de Redes Neuronales Convolucionales para tratar de predecir la clase de las imágenes. Este tipo de redes se ha utilizado históricamente, precisamente, para dicho propósito, como por ejemplo, para el famoso conjunto de número manuscritos [57].

1.1. Objetivos del trabajo

Se detallarán, en esta sección, los objetivos generales del TFG. En este caso, tiene tanto parte teórica o de estudio, como parte de desarrollo de aplicaciones *software*. Como hitos del presente TFG se podría contar con:

- Encontrar y adaptar los datos de forma adecuada para tratar el problema.
- Desarrollar un modelo predictivo con alta tasa de acierto.
- Construir una aplicación software que permita a los usuarios hacer uso del modelo. Esta aplicación también tiene sus requisitos:
  - El usuario debe poder elegir la imagen que desea predecir.
  - La aplicación debe mostrar de forma clara el resultado de la predicción.
  - Se presentará de forma gráfica cómo ha obtenido la predicción.
- Este proyecto se debe llevar a un entorno de desarrollo local fácilmente migrable.

1.2. Motivación

La clasificación de imágenes es una técnica cada vez más utilizada en diversas tareas: reconocimiento de fallos, identificación de personas, clasificación de imágenes médicas, etc. En este trabajo, nos vamos a centrar en las imágenes médicas obtenidas mediante rayos X, radiografías, en particular, de tórax para el disgnóstico de la Covid-19.

Durante este último año, la pandemia vivida a nivel mundial ha impulsado a la tecnología a avanzar en el campo de la medicina, para facilitar las labores técnicas a los especialistas en la materia. La diagnosis de la enfermedad COVID-19 ha resultado clave hasta ahora, para lo cual, la capacidad de realizar pruebas y tener un diagnóstico con alto grado de fiabilidad ha sido uno de los grandes objetivos de la Medicina. La prueba de reacción en cadena polimerasa o *Polymerase Chain Reaction* (PCR) han servido para detectar la presencia o ausencia de esta enfermedad. Estas han sido muy escasas y se han realizado de manera muy controlada para mantener una reserva de pruebas disponibles. Sin embargo, sería muy útil contar con una técnica fiable de detección de la enfermedad que no tuviese límite de uso.

Con el análisis de radiografías de tórax, una técnica relativamente económica y sin un claro límite de utilización más que las restricciones temporales, permitiría obtener una gran cantidad de muestras para clasificar y obtener el resultado de la prueba diagnóstica.

Por ello, en este trabajo, se presenta la inquietud de diseñar un método eficaz de clasificación de imágenes radiológicas de tórax para diagnosticar de forma rápida la enfermedad COVID-19. Podría constituir una manera más de diagnosis de la enfermedad para liberar, así, la situación de estrés con las pruebas PCR y su análisis en los laboratorios.

1.3. Estructura

En la presente memoria se seguirá el siguiente esquema:

- Capitulo 1. Introducción. Con una explicación del contexto, motivación y, finalizando, con esta estructura del documento.
- Capítulo 2. Fundamento Teórico. Se explican las bases de las Redes Neuronales, especialmente de las Convolucionales, pilar fundamental de este proyecto.
- Capítulo 3. Metodología. Metodología de trabajo.
- Capítulo 4. Marco de trabajo. Se comentan las herramientas utilizadas para el desarrollo del proyecto.
- Capítulo 5. Planificación. Se abordarán los temas relacionados con la gestión y planificación del proyecto.
- Capítulo 6. Datos. En este capítulo veremos la recogida, descripción y procesamiento de los datos.
- Capítulo 7. Construcción del sistema. Se tratará una pequeña introducción a las librerías utilizadas y la implementación del sistema.
- Capítulo 8. Resultados. Se comentarán los resultados obtenidos para el modelado de los datos.
- Capítulo 9. Aplicación. En este se abordará el proceso de desarrollo de una aplicación web capaz de implementar la funcionalidad propuesta en los objetivos. Se trata como un proyecto de Ingeniería de *Software*.
- Capítulo 10 Conclusiones. encontraremos tanto las conclusiones como futuras aplicaciones y trabajos a desarrollar que surgen de este proyecto.

Capítulo 2

Fundamento Teórico

2.1. Redes Neuronales Artificiales

Su origen se remonta a 1943, cuando McCulloch y Pitts presentaron un modelo matemático que pretendía simular el comportamiento de una neurona biológica [43], que, en principio: a partir de una o varias entradas binarias, se obtenía una salida con respuesta también binaria.

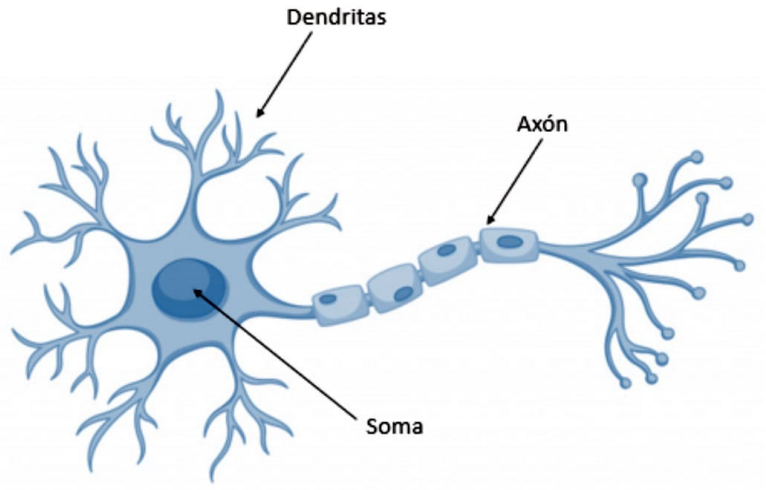


Figura 2.1: Representación de una neurona biológica. Imagen de [47].

Utilizando el trabajo anterior, Frank Rosenblantt ideó, en 1958, lo que conocemos hoy como perceptrón simple [51], representado en la Figura 2.2.

En el perceptrón simple, se relajaban las restricciones: las entradas ahora son valores

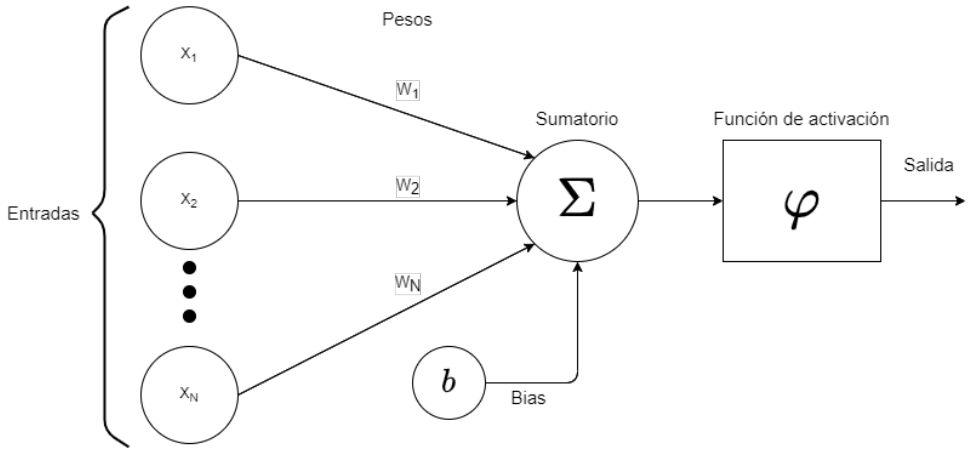


Figura 2.2: Modelo de Frank Rosenblatt (perceptrón simple).

numéricos y juegan el papel de las dendritas. La suma ponderada de las entradas y el valor umbral o *bias* simularían el soma y, por último, la función de activación haría las veces del axón. Podemos ver que las entradas tienen unos pesos, de esta forma cada entrada tendrá una diferente contribución a la neurona.

Fue en esta época, cuando comenzó un auge de la Inteligencia Artificial, con diversos modelos que simulaban el comportamiento humano. Se crearon redes con muchas neuronas organizadas en capas, lo que dio lugar a las Redes Neuronales. Sin embargo, esto duró poco tiempo debido a la escasa capacidad del cómputo y al método rudimentario de entrenamiento. Este método de entrenamiento consistía en realizar varias pasadas hacia delante en la red neuronal, es decir, ajustar los pesos de manera aleatoria y observar su resultado. A continuación, se cambiaban estos pesos de manera intuitiva y se realizaba, de nuevo, una pasada hacia delante.

No es hasta 1986 [52], cuando se empieza a utilizar el algoritmo de *Backpropagation*, que consiste en un procedimiento de retropropagación del error para cambiar el método de calcular los pesos. Si bien es cierto, que el punto de partida es el mismo, la asignación de unos pesos iniciales aleatorios y la forma de actualizarlos difiere completamente. Este algoritmo trata de repartir el error entre las neuronas o, dicho de otro modo, asignar a cada neurona un porcentaje del error que ella ha provocado, siguiendo un procedimiento similar al de actualización de pesos como el que se llevaba a cabo en el perceptrón simple, utilizando en este caso las derivadas y la regla de la cadena.

La agrupación de estas neuronas artificiales en capas y la unión de varias capas de neuronas es lo que da lugar a las Redes Neuronales actuales, llamadas también perceptrón multicapa o *multi layer perceptron* (MLP).

Nos referimos a topología o arquitectura cuando hablamos de la forma en que se organizan las neuronas dentro de una red neuronal. En este trabajo, se desarrollarán las redes densamente conectadas con propagación hacia delante. En este tipo de redes, cada

neurona toma como entrada las salidas de las neuronas de la capa inmediatamente anterior. A su vez, esta salida se propaga a las todas las neuronas de la capa inmediatamente posterior. Las neuronas se agrupan en capas según su función y su situación dentro de la red. Podemos diferenciar tres tipos de capas:

- Capa de entrada (*input layer*): Conecta los datos con la red neuronal. Cada uno de sus nodos tiene tantas entradas como variables tiene el conjunto de datos utilizado.
- Capa de salida (*output layer*): Recibe como entrada las salidas de las neuronas de la capa anterior y produce la respuesta o salida final del sistema. Juega un papel crucial ya que el número de neuronas en esta capa y el tipo de salida que se desea obtener tiene que adaptarse al problema tratado.
- Capas ocultas (*hidden layers*): Las capas que se sitúan entre la capa de entrada y la capa de salida se denominan capas ocultas. Suelen llevar el peso computacional por su gran cantidad de neuronas y porque el número de capas es ilimitado.

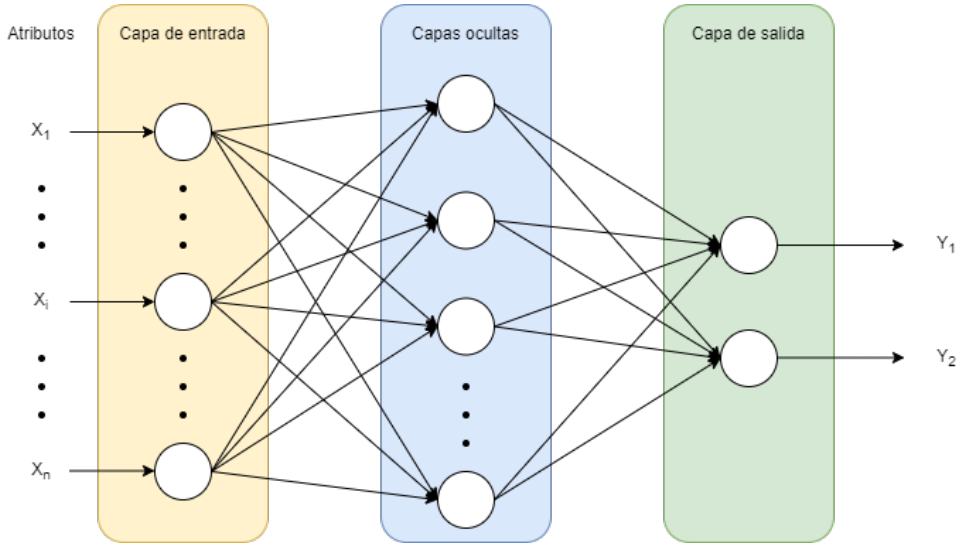


Figura 2.3: Arquitectura de una Red Neuronal.

En el funcionamiento de esta red, se suele hablar de dos fases:

- Propagación hacia delante: a partir de unos pesos asignados a las conexiones entre neuronas y unos datos iniciales, se calcula la salida de la red neuronal con una pasada hacia delante.
- Propagación hacia atrás o retropropagación: la función de pérdida o de coste se propaga hacia atrás, asignando una responsabilidad del error a cada neurona y actualizando, de forma proporcional al error, los valores de las conexiones entre neuronas utilizando el descenso del gradiente.

Esto es lo que dió lugar a las Redes Neuronales o *Artificial Neural Network* (ANN) con relativo éxito. Sin embargo, hay más arquitecturas o formas de agruparse basadas en estas, que utilizan diferentes funcionalidades. Esto da lugar a las redes recurrentes, las cuales guardan, de alguna manera, la influencia de instantes de tiempo pasados, o a las Redes Convolucionales, históricamente utilizadas en el tratamiento de imágenes.

A continuación, veremos unos conceptos básicos que se consideran necesarios en las Redes Neuronales.

2.1.1. Funciones de activación

Podemos decir que una función de activación es una función matemática aplicada a la salida de la neurona para modificar y añadir deformaciones no lineales [45]. De esta forma, permite aprender y representar funciones mucho más complejas. Estas son algunas de las propiedades deseables en las funciones de activación:

- No lineal: introduce irregularidades en el modelo para poder aproximar una gran variedad de funciones.
- Diferenciable: para poder aplicar métodos del descenso del gradiente en la optimización de parámetros.
- Rango: los valores de salida deben estar acotados.
- Monótona: son funciones siempre crecientes. Si la función de activación es monótona, podemos garantizar que la superficie de error asociada a un modelo monocapa es convexa y, por tanto, el óptimo local coincide con el global.
- Simplicidad: debemos usar funciones con bajo coste computacional, pues suele presentarse un uso masivo de ellas.
- Se aproximan a la identidad en el origen: esto es necesario para evitar la influencia de los pesos iniciales, los cuales son aleatorios.

Función identidad

La función identidad s una función lineal que retorna el mismo valor que su entrada. Sería equivalente a no aplicar ninguna función de activación.

$$f(x) = x \tag{2.1}$$

Aunque puede ser útil en otros problemas, no se adapta a las Redes Neuronales, ya que no introduce no-linealidades. Por tanto, no nos permitiría adaptarnos a problemas que no sean linealmente separables. Si tuviésemos un modelo de red neuronal muy complejo con muchas capas y neuronas pero todas ellas tuviesen esta función de activación, sería equivalente a tener un modelo con una sola neurona.

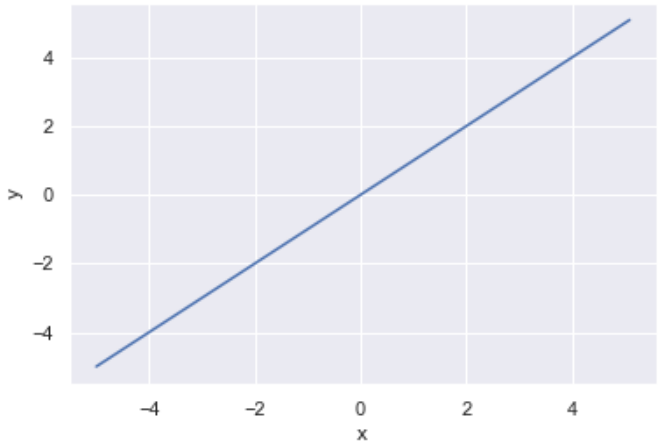


Figura 2.4: Función identidad.

Función signo

$$sgn(x) = \begin{cases} +1 & si\ x \geq 0 \\ -1 & en\ otro\ caso. \end{cases} \tag{2.2}$$

Transforma la entrada en una salida binaria en función del signo. Combinado con el *bias*, valor límite, nos permitiría establecer un límite de separación entre clases.

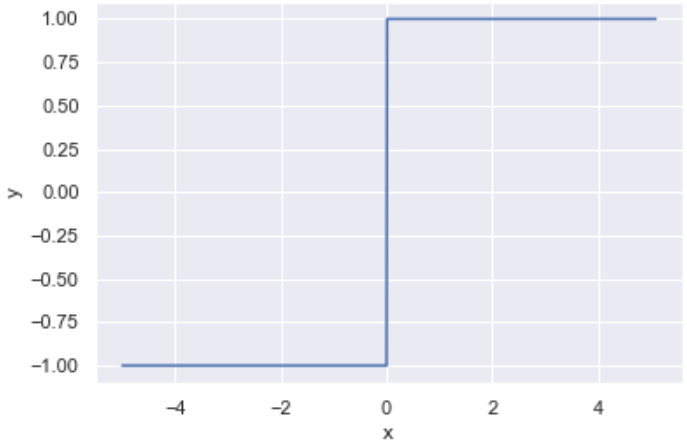


Figura 2.5: Función signo.

Función ReLU

El rectificador lineal(*Rectified Linear Unit*) es la función de activación más utilizada [50]. Su salida es igual a cero si su entrada es negativa. En caso de que sea positiva coincide con la entrada. Es una función estrictamente creciente.

$$f(x) = \begin{cases} x & si\ x \geq 0 \\ \\ 0 & en\ otro\ caso. \end{cases} \tag{2.3}$$

Una reformulación habitual es:

$$f(x) = max(0, x) \tag{2.4}$$

Supone una gran ventaja, en cuanto a la velocidad de entrenamiento o construcción del modelo respecto a otras funciones de activación. Tiene activación dispersa, es decir, si las neuronas toman valores iniciales aleatorios, solo la mitad de ellas se activarían. Algunas de sus desventajas podrían ser que no está centrada en cero y no es diferenciable en ese punto, aunque sí lo es en todos los demás puntos. Otro inconveniente es que todos los puntos a la izquierda del cero toman el mismo valor, esto nos impide encontrar diferencias entre ellos. Por desgracia, las unidades $ReLU$ pueden ser frágiles durante el entrenamiento y pueden "morir". Por ejemplo, un gran gradiente que se propaga a través de una neurona $ReLU$ puede hacer que los pesos se actualicen de tal manera que la neurona no vuelva a activarse en ningún punto de datos. Si esto ocurre, el gradiente que se propaga a través de la unidad será siempre cero a partir de ese punto. Cuando las neuronas se vuelven inactivas y sólo dan salida a cero para cualquier entrada [41] estamos ante el problema conocido como *Dying ReLu*. La función $ReLU$ es lineal a trozos.

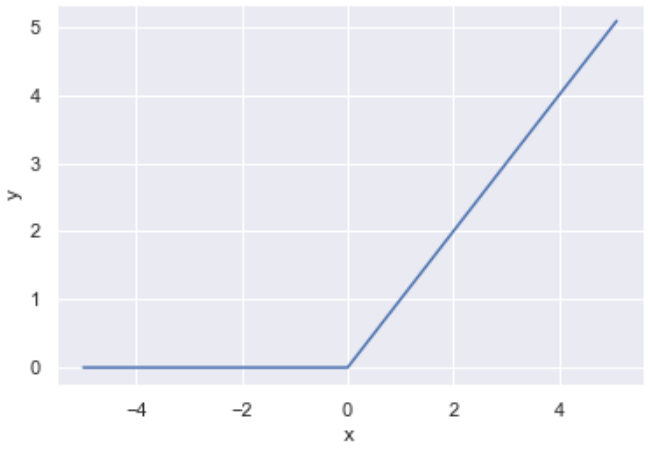


Figura 2.6: Función ReLU

Uno de sus grandes inconvenientes es que no está acotada. Esta función suele ser útil en el reconocimiento de imágenes y Redes Convolucionales.

Función LeakyReLU

Partiendo de la idea anterior del rectificador lineal nace esta variante. La función *LeakyReLU* proporciona una corrección para los valores negativos de 2.3, con esto obtendríamos que los valores ya no serían cero sino un valor muy pequeño dependiente de la entrada.

$$f(x) = \begin{cases} x & si\ x \geq 0 \\ \\ 0{,}01x & en\ otro\ caso. \end{cases} \tag{2.5}$$

Con ello, logramos solventar el problema de que todos los valores negativos fuesen igual a cero, así como el problema de *Dying ReLu*.

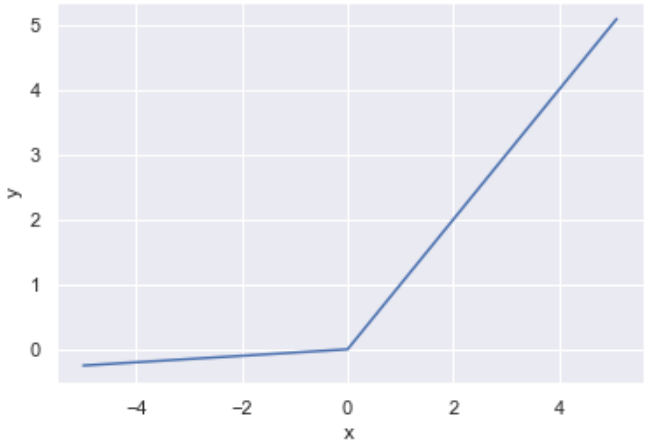


Figura 2.7: Función Leaky ReLU

También, existe una versión paramétrica de esta función en la que el coeficiente para la x, cuando la entrada negativa, se sustituye por un valor muy pequeño denominado α. Como sucedía con la función ReLU, *LeakyReLU* no está acotada. Igualmente, se comporta bien con imágenes y Redes Convolucionales.

Función sigmoide

Denominada en el mundo matemático como función logística, la función sigmoide es especialmente útil en problemas de clasificación binaria ya que su rango de valores es $(0, 1)$ y siempre toma valores positivos. Por todo ello, la salida de esta función podría interpretarse como una probabilidad. Es una función estrictamente creciente y derivable, con derivada no nula en cada punto, diferenciable. El método del descenso del gradiente puede lograr un mejor resultado en cada paso de la optimización por estos motivos.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

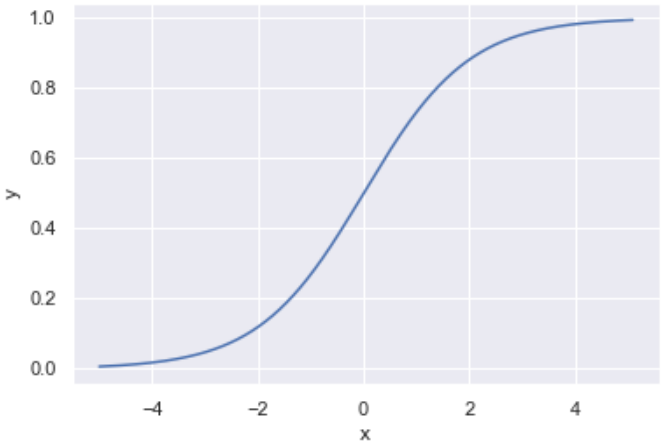


Figura 2.8: Función sigmoide

En contraposición, esta función tiene lenta convergencia, es decir, satura o mata el gradiente. Sin embargo, tiene un buen rendimiento en la última capa por su alta interpretabilidad.

Función *softmax*

La función*softmax* es realmente útil cuando tratamos con problemas de clasificación en varias clases. Con una base matemática muy similar a la de la función sigmoide vista en 2.6, la función *softmax* puede tener tantas salidas como se deseen, extendiendo, de esta manera, la interpretación de probabilidad como la de pertenencia a cada clase. Todas las salidas están en el rango de valores $(0, 1)$. La suma de estos es igual a uno. Para k clases, tenemos la siguiente formulación:

$$f_i(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{k} e^{y_i}} \tag{2.7}$$

Función tangente hiperbólica

Con un alto grado de semejanza a la función sigmoide, la función tangente hiperbólica toma valores en el rango $(-1, 1)$. Está centrada en torno al cero y es estrictamente creciente. En la práctica, la optimización es más fácil que para la función logística, pero sufre del problema del desvanecimiento del gradiente.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.8}$$

Podemos utilizar una reformulación equivalente, pero más eficiente computacional-

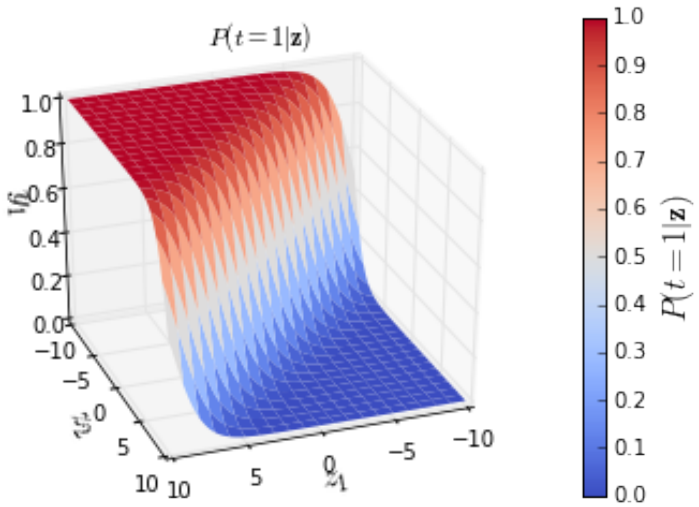


Figura 2.9: Función de activación *softmax*. Imagen de [58]

mente, al reducir el cálculo de exponenciales.

$$f(x) = tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.9}$$

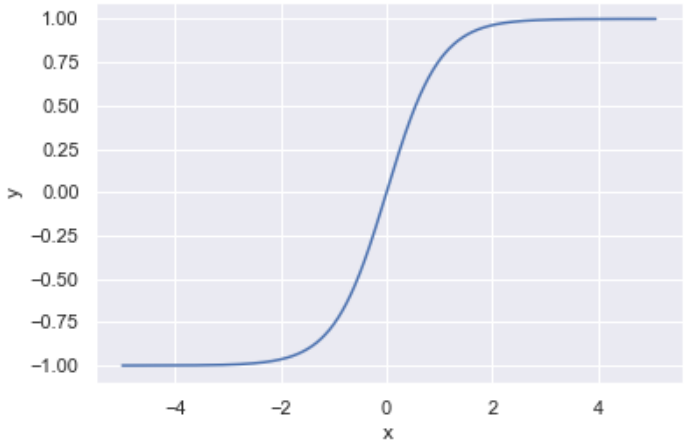


Figura 2.10: Función tangente hiperbólica

Como sucedía en la función sigmoide, tiene lenta convergencia y satura o mata el gradiente. Se utiliza para decidir entre una opción y la contraria por su rango de valores. Tiene buen desempeño en Redes Recurrentes.

Más allá de las funciones de activación presentadas en este trabajo, existen muchas otras variantes. Sin embargo, usaremos las ya mencionadas por su suficiencia, utilidad y popularidad.

2.1.2. Función de coste o pérdida

Representa la suma del error: la diferencia entre el valor predicho y el real. Se emplea en problemas supervisados, es decir, con la variable respuesta conocida. Su misión es medir cómo de bien se comporta nuestra red neuronal frente al problema establecido. Sea $\hat{y}$ = valor predicho, y = valor real, w = pesos y x = variables de entrada. Calculamos $\hat{y} = w \cdot x$.

Destacamos las siguientes funciones de coste:

- MAE(*Mean Absolute Error*): Se utiliza cuando la salida es un valor escalar. $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- MSE(*Mean Squared Error*): igual que la anterior, pero $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- *Binary Crossentropy*: Se utiliza cuando estamos ante un problema de clasificación binaria. Sea p la etiqueta real y q la etiqueta predicha.
  $H(x) = \sum_{i=1}^{n} p_i(x) log(q_i(x))$
- *Categorical Crossentropy*: Se usa en problemas de multi-clasificación. Para m clases y n muestras, sea y_{ij} la pertenencia de la muestra i a la clase j, valor real. Y su valor predicho $\hat{y_{ij}}$. $L(y, \hat{y}) = \sum_{j=0}^{m}\sum_{i=1}^{n}(y_{ij} \cdot log(\hat{y_{ij}}))$

Las funciones de coste también se deben elegir cuidadosamente según el problema tratado [21].

2.1.3. Optimizadores

Con el conocimiento obtenido acerca de la tarea de optimización de los parámetros en la etapa del entrenamiento, veremos, a continuación, qué técnicas se van a utilizar en la práctica. No debemos olvidar que todas ellas hacen uso del descenso del gradiente y del algoritmo de retropropagación.

- Descenso del gradiente estocástico o SGD [35]: optimizador con descenso de gradiente y momento. Puede incluirse la aceleración de Nesterov [60].
- RMSprop [34]: mantiene una media móvil del cuadrado de los gradientes y divide el gradiente por la raíz de esta media. Esta implementación de RMSprop utiliza el impulso simple, no el impulso Nesterov. Utiliza esa media móvil para estimar la varianza.
- Adam (*Adaptative moment estimation*) [33]: la optimización de Adam es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden. Se ha demostrado que el método es eficiente desde el punto de vista computacional [37], tiene pocos requisitos de memoria, es invariable al reescalado diagonal de los gradientes y se adapta bien a los problemas que son grandes en términos de datos/parámetros.

Es interesante destacar que aunque la convergencia con Adam es más rápida, el algoritmo SGD generaliza mejor [65].

Para la elaboración de esta sección 2.1, se ha utilizado [62].

2.2. Redes Neuronales Convolucionales

Convolutional Neural Network (CNN) son un tipo de Redes Neuronales que constituye una variación de un perceptrón multicapa. Debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones.

2.2.1. Un poco de historia

Encontramos sus inicios en 1959, en el trabajo realizado por Hubel y Wiesel [27], que permitió mejorar la comprensión sobre cómo funciona la corteza visual, particularmente, las células responsables de la selectividad de orientación y detección de bordes en los estímulos visuales. En dicho trabjao, se determinó que las neuronas del córtex visual eran sensibles a la posición y a la orientación.

Posteriormente, en 1980, se asentaron los fundamentos de las Redes Neuronales Convolucionales basados en el Neocognitron [18]. Yann LeCun, en 1998 [39], fue capaz de mejorar este modelo introduciendo un método de aprendizaje basado en la propagación hacia atrás, *backpropagation*, para poder entrenar el modelo de forma eficiente y actualizar sus pesos automáticamente.

La red neuronal convolucional (CNN) es el estado del arte para la tarea de clasificación de imágenes [59]. Son consideradas el modelo base para el tratamiento y clasificación de imágenes, así como el punto de partida para otros modelos más complejos.

La Redes Neuronales convolucionales se consideran un caso particular de las redes densamente conectadas o perceptrón multicapa, como hemos denominado previamente, ya que todas las neuronas se encuentran conectadas entre sí. Su principal aportación es la operación de convolución, realizada en las neuronas que veremos en detalle en la sección 2.2.2.

La diferencia fundamental entre una capa densamente conectada y una capa convolutiva es la siguiente: las primeras aprenden patrones globales en su espacio de características de entrada, mientras que, las segundas aprenden patrones locales; en el caso de las imágenes, patrones encontrados en pequeñas ventanas 2D de las entradas. Esta característica clave confiere a las Redes Convolucionales dos propiedades interesantes [9]:

- Los patrones que aprenden son invariables a la traslación. Después de aprender un determinado patrón en una zona de una imagen, una red convolucional puede

reconocerlo en cualquier lugar. Una red densamente conectada (MLP) tendría que aprender el patrón de nuevo si apareciera en una nueva ubicación.

- Pueden aprender jerarquías espaciales de patrones. Por ejemplo, una primera capa de convolución aprenderá pequeños patrones locales, como los bordes, una segunda capa de convolución una segunda capa de convolución aprenderá patrones más grandes, compuestos por las características de las primeras capas, y así sucesivamente. Este permite a las Redes Convolutivas aprender eficazmente conceptos visuales cada vez más complejos y abstractos.

Las Redes Convolucionales reciben este nombre porque aplican la operación matemática de la convolución.

2.2.2. Operación de convolución

En su forma más general, la convolución es una operación sobre dos funciones con un argumento de valor real, en otras palabras, es una operación matemática que transforma dos funciones, f y g, en una tercera función que, en cierto sentido, representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) \cdot w(t-a). \tag{2.10}$$

En nuestro caso, el primer argumento, x, se refiere a la entrada Esta suele ser un array multidimensional, y el segundo argumento, w, se refiere a la función *kernel* o núcleo, que suele ser un array multidimensional de parámetros que acepta el algoritmo de aprendizaje de los pesos. La salida que se obtiene se conoce como *feature map* o mapa de características.

De ahora en adelante, nos referiremos como I a la función de entrada, que antes habíamos llamado x y, como K, a la función *kernel* que habíamos llamado w.

Podemos adaptar esta fórmula de convolución 2.10 a un caso bidimensional y un a instante de tiempo fijo. Sería el caso de imágenes estáticas, como las que vamos a tratar en este trabajo.

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum nI(m,n)K(i-m, j-n). \tag{2.11}$$

La convolución es conmutativa, lo que quiere decir que podemos escribirla de forma equivalente:

$$S(i,j) = (K * I)(i,j) = \sum_{m} \sum nI(i-m, j-n)K(m,n). \tag{2.12}$$

Normalmente, esta última fórmula 2.12 es más sencilla de implementar, porque hay menos variación en el rango de valores válidos de m y n.

La propiedad conmutativa de la convolución se debe a que hemos invertido el núcleo en relación con la entrada, en el sentido de que, a medida que m aumenta, el índice de la entrada crece, pero el índice del núcleo disminuye. La única razón para dar la vuelta al núcleo es obtener la propiedad conmutativa. Aunque esta es útil para escribir demostraciones, no suele ser una propiedad importante en la implementación de red neuronal. En su lugar, muchas bibliotecas de Redes Neuronales implementan una función aislada llamada correlación cruzada, que es lo mismo que la convolución, pero sin invertir el núcleo:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum nI(i+m, j+n)K(m,n). \tag{2.13}$$

En la Figura 2.11, podemos ver un ejemplo de convolución bidimensional sin inversión del núcleo. Restringimos la salida sólo a las posiciones en las que el núcleo se encuentra completamente dentro de la imagen, lo que en algunos contextos se denomina convolución válida. Dibujamos recuadros con flechas, para indicar cómo se forma el elemento superior izquierdo del tensor de salida, aplicando el núcleo a la correspondiente región superior izquierda del tensor de entrada. Se aprecia cómo el tamaño del núcleo es menor que el de los datos de entrada y, por ello, la ventana del núcleo se va deslizando.

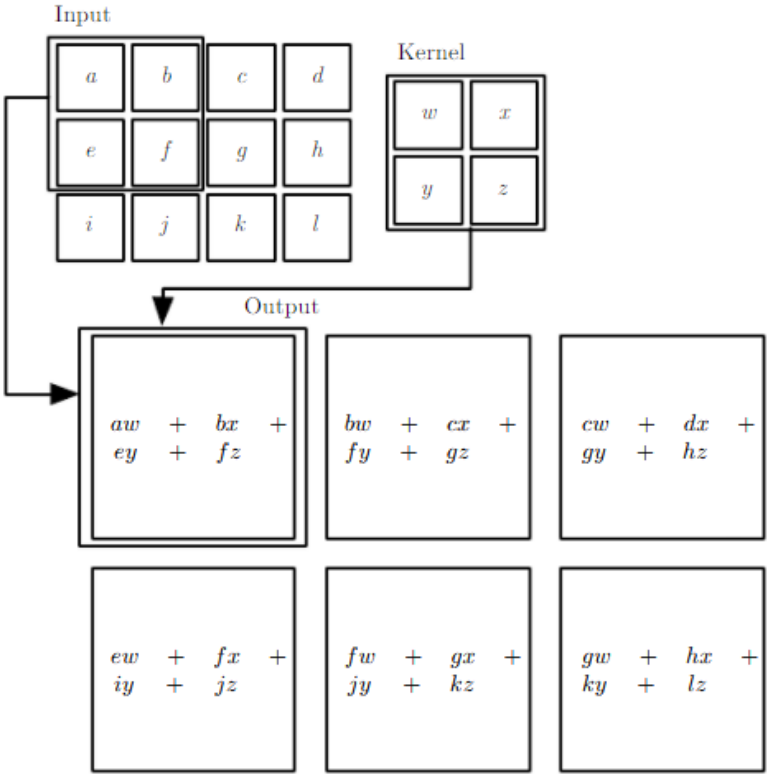


Figura 2.11: Ejemplo de convolución 2D. Imagen de [21].

La convolución aprovecha tres ideas importantes que pueden ayudar a mejorar un sistema de aprendizaje automático:

- Conectividad dispersa. Solo un grupo pequeño de neuronas se activan en un cierto momento. Se consigue haciendo el kernel más pequeño, dimensionalmente hablando, que la entrada. Podemos procesar una imagen de entrada, que puede tener miles o millones de píxeles, y, sin embargo, detectar características pequeñas y significativas, como los bordes, con núcleos que sólo ocupan decenas o cientos de píxeles. Esto significa que necesitamos almacenar menos parámetros, lo que reduce los requisitos de memoria del modelo y mejora su eficiencia estadística. También el cálculo de su salida requiere de menos operaciones.

  Si hay m entradas y n salidas, la multiplicación de matrices requiere $m \cdot n$ parámetros y el algoritmo tiene un coste temporal de ejecución del orden de $\mathcal{O} = (m \cdot n)$. Si limitamos el número de conexiones que puede tener cada salida a k, entonces la aproximación de conectividad dispersa solo requiere $k \cdot n$ parámetros y tiene coste del orden $\mathcal{O} = (k \cdot n)$, que mejora cuando k es más pequeño que m.

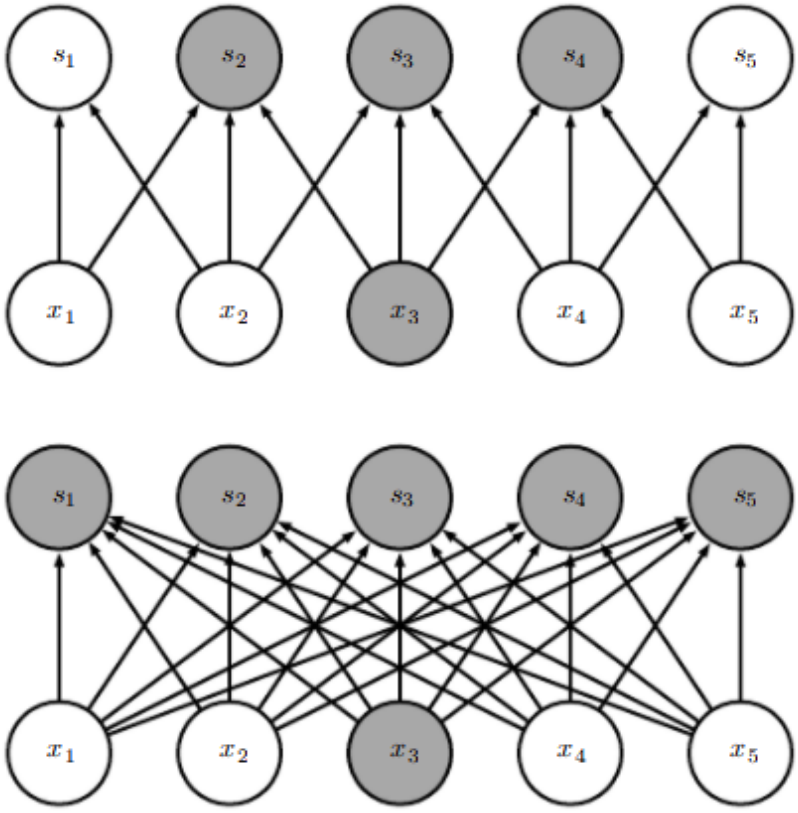


Figura 2.12: Ejemplo de conectividad dispersa -convolución (arriba) frente a conectividad densa - mutiplicación matricial (abajo). Imagen de [21].

- Parámetros compartidos. Se refiere a la utilización de un mismo parámetro para más de una función en el modelo. En una red neuronal habitual, MLP, cada parámetro se usa exactamente una vez para calcular la salida de una neurona de una capa.

Se dice que las Redes Convolucionales tienen pesos ligados por este motivo: un mismo peso o parámetro se utiliza varias veces. Cada miembro del núcleo se usa en todas las posiciones, aunque puede haber excepciones en los bordes. Esto reduce el número de parámetros del modelo a k, que solo tenemos que aprender una vez. Siendo k más pequeño que m, los requisitos de memoria se vuelven insignificantes en la convolución, esto es, k mucho menor frente a la multiplicación de matrices $n \cdot m$. Resulta mucho más eficiente en términos de memoria.

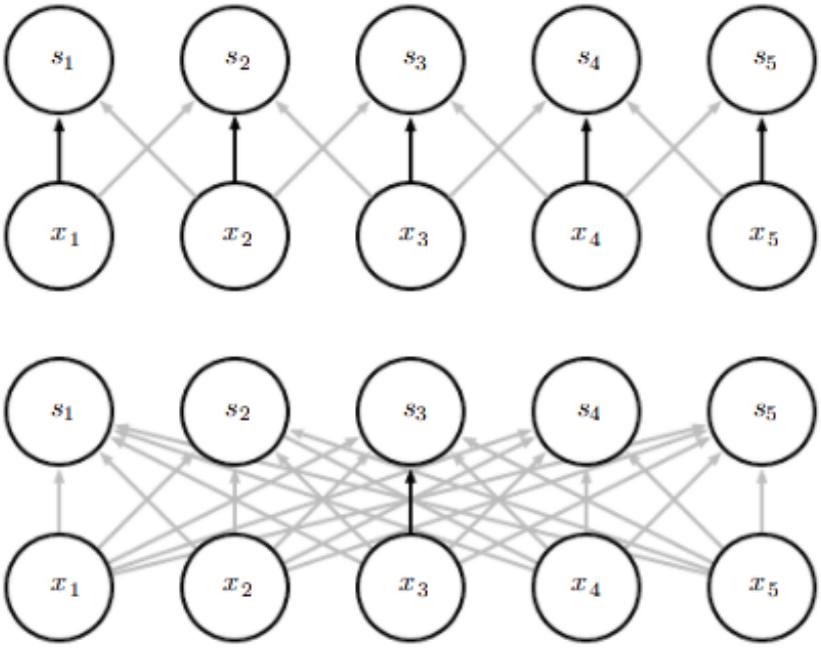


Figura 2.13: Ejemplo de compartición de parámetros -convolución (arriba) frente a multiplicación densa de matrices (abajo). Imagen de [21].

En la Figura 2.13, podemos ver, en negro, cómo el núcleo se comparte en las Redes Convolucionales. Sin embargo, en MLP, cada peso afecta a una sola entrada (píxel).

- Representaciones equivalentes. Debido a que los parámetros son compartidos, se da la propiedad de equivalencia dentro de la capa. Una función es equivalente si: dado un cambio en la entrada, la salida cambia de la misma manera. Una función, $f(x)$, es equivalente a otra, g, si:

$$f(g(x)) = g(f(x)). \tag{2.14}$$

La convolución es equivalente ante la traslación, es decir, sería capaz de detectar un patrón en diferentes lugares de la imagen. Sin embargo, no es equivalente a otros cambios, como la rotación y el escalado. Son necesarios otros métodos para manejar este tipo de transformaciones.

2.2.3. Estructura de una capa convolucional

Cada una de las capas convolucionales tiene tres partes [26]:

- Etapa de convolución. Se utiliza para extraer los patrones y características de la imagen. Se realiza la operación de convolución, vista en la sección 2.2.2, sobre la imagen de entrada con un tamaño de filtro, núcleo, determinado MxM. Se desliza el filtro sobre la imagen de entrada y se realiza el producto entre las partes de la imagen y el filtro. La salida se denomina mapa de características. Nos proporciona información sobre la imagen, como las esquinas y los bordes. Más tarde, este mapa de características alimenta a otras capas para aprender otras, a menudo de mayor complejidad. La capa realiza varias convoluciones en paralelo, para producir un conjunto de activaciones lineales. Cada activación lineal, se ejecuta a través de una función de activación no lineal, como la función de activación ReLU.

- En la mayoría de los casos, cada capa densa, va seguida de una reducción de dimensionalidad. Esto es, un agrupamiento del mapa de características para disminuir su tamaño. De esta manera, también se reduce el coste computacional en futuras operaciones. Esta operación se llama *Pooling*, agrupación, y la veremos más adelante en la sección 2.2.5.

- Capa densa o totalmente conectada, *Fully Connected*(FC). Cada una de las salidas de la capa anterior tiene una conexión con cada neurona de esta capa, contiene los pesos de todas estas conexiones. Sería equivalente a añadir un MLP al final de las etapas convolutivas. Suelen situarse antes de la capa de salida y forman las últimas capas de una arquitectura CNN. Requiere de un aplanado, *Flatten*, de la imagen o de la salida de las capas anteriores para obtener un vector de una dimensión. Llevan a cabo el proceso final de la tarea de clasificación.

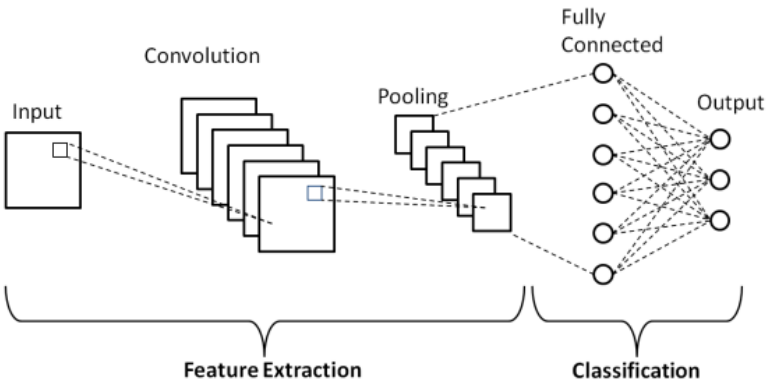


Figura 2.14: Esbozo de CNN. Imagen de [2].

2.2.4. Dimensionalidad

Debemos destacar que las Redes Convolucionales no solo se utilizan para imágenes. Según el número de dimensiones que tenga la estructura de datos, dada como entrada, y la arquitectura de la red, puede tratarse uno u otro problema [36].

- Una dimensión (1D): para datos temporales, vectores.
- Dos dimensiones (2D): para datos espaciales, matrices. Aquí entrarían las imágenes.
- Tres dimensiones (3D): para datos espaciales-temporales. Estaríamos hablando de una mezcla de los dos anteriores, se utiliza especialmente en el tratamiento de vídeo, que no deja de ser una secuencia temporal de imágenes.

2.2.5. Pooling

Una función de agrupación sustituye la salida de la red en un lugar determinado por un estadístico de resumen de las salidas cercanas. Esto supone una reducción de la dimensionalidad, de forma que, el número de celdas consideradas cercanas se reduce a una sola con un único valor, el estadístico utilizado como resumen. La operación *pooling* ayuda a que la representación sea aproximadamente invariable frente a pequeñas traslaciones de la entrada.

Podemos destacar las siguientes funciones de agrupación [36], haciendo referencia a dos dimensiones pero extensible a 1 y 3:

- MaxPooling: submuestrea la entrada a lo largo de sus dimensiones espaciales, altura y anchura, tomando el valor máximo en de la ventana de entrada.
- AveragePooling: submuestrea la entrada a lo largo de sus dimensiones espaciales, altura y anchura, tomando la media como valor de salida en la ventana de entrada.
- GlobalMaxPooling: en esencia, es igual que MaxPooling, pero ya tiene definido el tamaño de la ventana, que coincide con el tamaño del mapa de características.
- GlobalAveragePooling: en esencia es igual que AveragePooling pero ya tiene definido el tamaño de la ventana, que coincide con el tamaño del mapa de características.

Se podría definir cualquier función matemática, con el fin de reducir el tamaño de la ventana a un solo valor, comom por ejemplo, el uso de estadísticos como el mínimo o la mediana. Sin embargo, los mostrados en esta sección son los más utilizados en la práctica.

En todos los casos, el pooling ayuda a que la representación sea aproximadamente invariable a pequeñas traslaciones de la entrada. Esto significa que si se traslada la entrada en una pequeña cantidad, los valores de la mayoría de las salidas agrupadas no cambian. Puede ser una propiedad útil si nos importa más si alguna característica está presente, que dónde está exactamente. Por ejemplo, cuando se determina si una imagen contiene

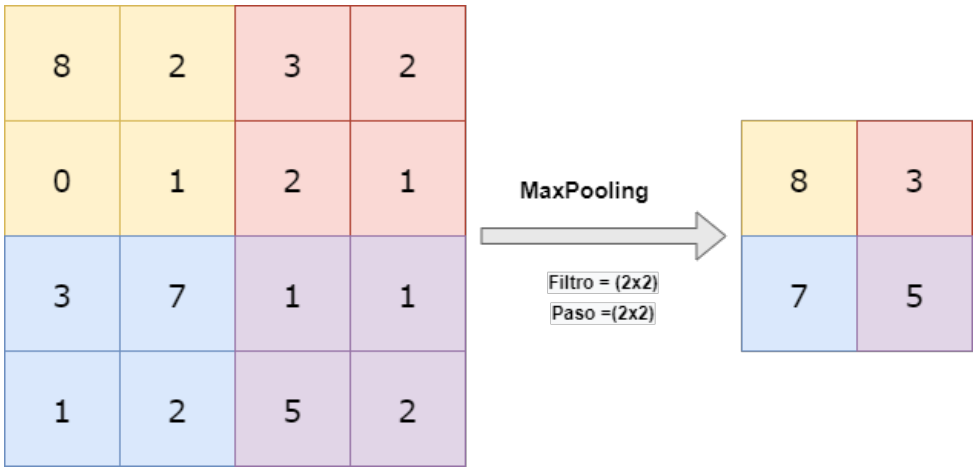


Figura 2.15: Ejemplo MaxPooling con filtro de 2 y paso de 2.

una cara, no es necesario conocer la ubicación de los ojos con una precisión de píxeles perfecta, sólo necesitamos saber que hay un ojo en el lado izquierdo de la cara y otro, en el lado derecho de la cara. En otros contextos, es más importante conservar la ubicación de un rasgo.

Dado que la agrupación resume las respuestas de un conjunto de celdas del mapa de características, es posible utilizar menos unidades de agrupación que de detección, informando de las estadísticas de resumen de las regiones, en lugar de por un valor por píxel. Esto mejora la eficiencia computacional de la red, ya que la siguiente capa tiene un número menor de entradas que procesar, lo que también puede mejorar la eficiencia estadística y reducir los requisitos de memoria para almacenar los parámetros.

En [5], podemos encontrar una guía sobre qué función pooling se debería utilizar en cada situación. Por lo general, se suele recomendar utilizar MaxPooling, ya que hace uso del máximo y, teóricamente, sería capaz de extraer la característica más relevante. AveragePooling también es muy usada en la práctica. Esto puede suponer que la información sea demasiado plana y no recoja suficiente variabilidad para aprender los patrones necesarios, por lo que resultaría menos informativa.

2.3. Redes Convolucionales en la práctica

Al hablar de Redes Neuronales Convolucionales, explicamos la función matemática de convolución, sección 2.2.2. Sin embargo, la fórmula matemática difiere ligeramente de las funciones utilizadas en la práctica, las cuales veremos en esta sección, así como algunas de las propiedades útiles de estas funciones.

El primer punto que debemos aclarar es que, cuando hablamos de Redes Neuronales, la función de convolución se refiere a muchas convoluciones matemáticas realizadas en

paralelo. Esto es, la convolución con un solo filtro o *kernel*, solo es capaz de extraer un tipo de característica o patrón en las localizaciones que se aplica. Sin embargo, el objetivo es encontrar cualquier tipo de patrón en cualquier posible localización.

También, debemos destacar que las imágenes pueden tener dos formatos:

- Escala de grises: solo tendríamos una malla bidimensional con los valores de los píxeles y, por tanto, un canal.
- *Red Green Blue*(RGB): tenemos imágenes a color, cuyo resultado es la suma de los tres colores, rojo, verde y azul. Serían tres mallas bidimensionales, una por cada color y, por tanto, tres canales de entrada.

En el caso de tener varios canales, situación muy habitual, se realiza la etapa de convolución con diferentes filtros en paralelo sobre cada canal. A continuación, se realiza la suma de sus salidas tras la activación en los píxeles coincidentes. Por último, se llevaría a cabo la etapa de agrupamiento o *pooling*.

Observamos, a continuación, en la Figura 2.16, el procedimiento que se realizaría en una imagen con tres canales.

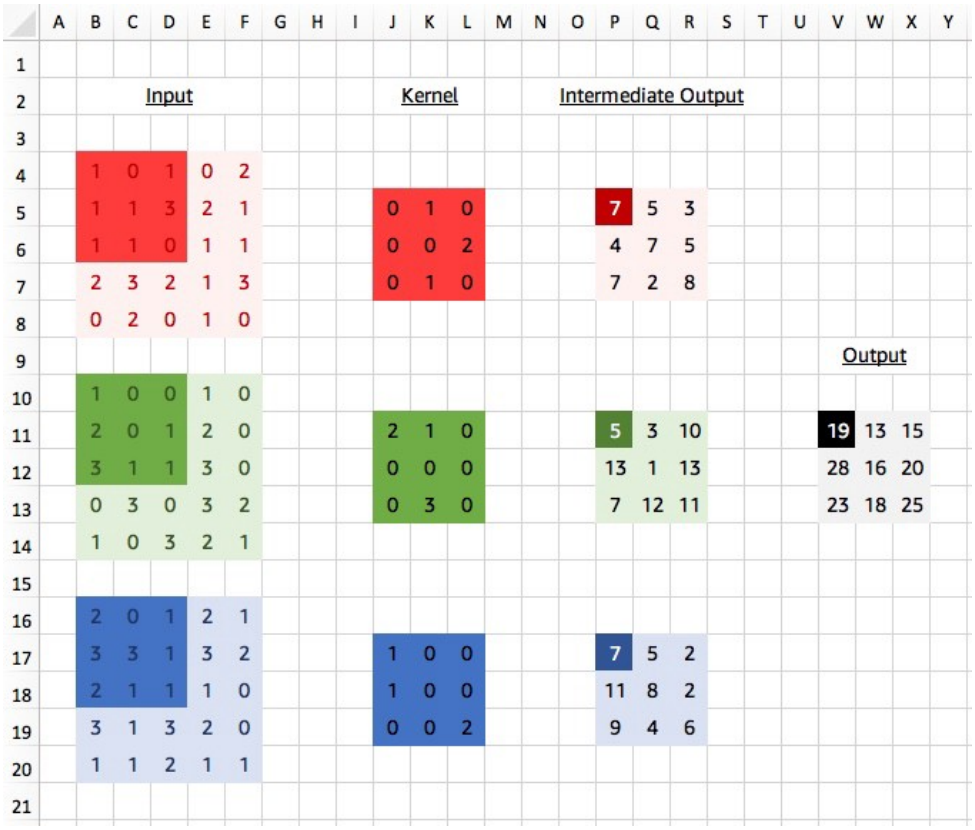


Figura 2.16: Ejemplo de convolución en imagen RGB (3 canales) con filtro 3x3. Imagen de [38].

Podemos ver, en la figura 2.16, un ejemplo de cómo se aplica un filtro para cada canal. La salida es la suma de los tres canales después de pasarlos por el filtro. A partir de ese momento, se tendrán tantos canales como núcleos se quieran aplicar, de igual manera que con las imágenes en escala de grises.

2.3.1. Variantes de la operación convolución

Cuando hablamos de convolución en el contexto de las Redes Neuronales, habitualmente, no nos referimos a la operación de convolución estándar según la bibliografía matemática. Las funciones que se utilizan en la práctica difieren ligeramente.

Debemos recordar, como ya dijimos en la sección 2.3, que la operación de convolución en este contexto se refiere, no solo a una convolución, sino a la aplicación de esta en paralelo a lo largo de los diferentes canales y del número de núcleos o filtros definido. Con esto, se obtiene más de un mapa de características, sobre el que se realizarán las operaciones.

Por todo ello, representaremos el *kernel* o núcleo como un tensor 4-D. Donde, el elemento $K_{i,j,k,l}$ nos proporciona la fuerza en la conexión entre la unidad del canal i de salida y, la unidad del canal j de entrada, desplazado k filas y l columnas entre las unidades de salida y entrada.

Representamos los datos observados con el tensor V, donde $V_{i,j,k}$ nos da el valor en el canal i, la posición i, j, se refiere a fila y columna. Para calcular la salida Z, con el mismo formato que V, a partir de la convolución de K, obtenemos la Fórmula 2.15.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} \cdot K_{i,l,m,n} \tag{2.15}$$

El sumatorio se extiende sobre todos los valores en los que la operación es válida. Así, se obtiene los resultados tras aplicar la convolución en paralelo sobre los diferentes canales.

Para entender las variantes que se presentan a continuación, debemos explicar previamente algunos conceptos clave.

- Stride o paso: El *stride* es un parámetro del filtro de la red neuronal que modifica la cantidad de movimiento sobre la imagen o el vídeo. Por ejemplo, si el *stride* de una red neuronal se establece en 1, el filtro se moverá un píxel, o unidad, cada vez, ya sea movimiento en filas o en columnas. El tamaño del filtro afecta al volumen de salida codificado, por lo que el *stride* suele establecerse en un número entero.

  Podemos observar cómo funciona el *stride* de uno en la Figura 2.17, en la que el desplazamiento es de una unidad. En la Figura 2.18, podemos ver el desplazamiento

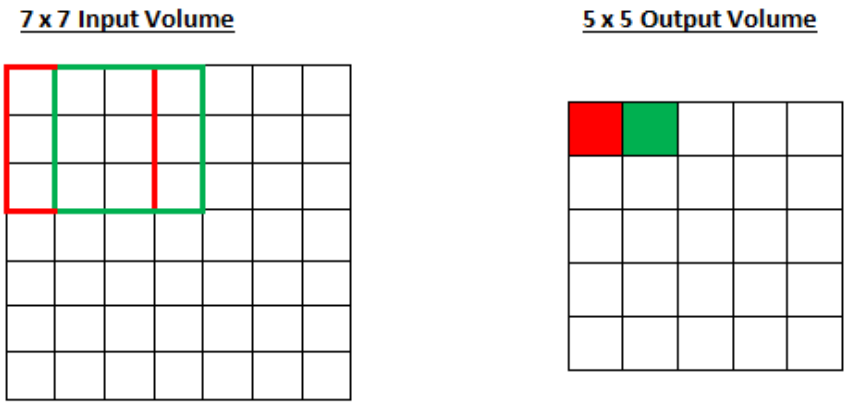


Figura 2.17: Ejemplo de convolución con *stride* de (1,1).

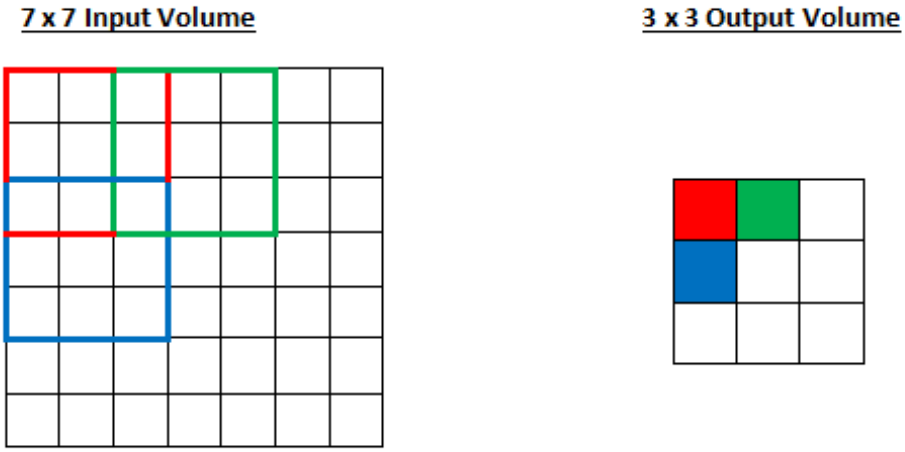


Figura 2.18: Ejemplo de convolución con *stride* de (2,2). Imagen de [14].

de dos unidades. Cuando variamos el *stride* la fórmula resultante es:

$$Z_{i,j,k} = c(K, V, s)_{i,j,k} = \sum_{l,m,n} [\, V_{l,(j-1)\cdot s+m,(k-1)\cdot s+n} \cdot K_{i,l,m,n}] \tag{2.16}$$

Donde s hace referencia al *stride*.

- *Padding* o rellenado. Es un término que tiene que ver con el rellenado de píxeles en los bordes. Extiende la imagen con píxeles invisibles para dar más importancia a los píxeles que se encuentran en los bordes de la imagen. Hasta ahora, hemos utilizado lo que se conoce como *valid*, que consiste en no añadir ningún píxel. El número de píxeles de salida, después de aplicar el *padding*, será el mismo que el número de píxeles de entrada. Esto hace que, tras cada etapa de convolución, las dimensiones de entrada en la siguiente capa se reduzcan, de manera que, si la imagen

de entrada tiene anchura m y el filtro anchura k, la salida tendrá anchura $m-k+1$. Esto produce una drástica reducción de dimensionalidad, que acaba convergiendo a dimensión $1x1$. Si la técnica de *padding* que utilizamos añade algún píxel, que deben ser ceros, se les llama *zero-padding* o rellenado de ceros. Esta técnica de añadir píxeles de 0, en filas y columnas, aumentando las dimensiones de la imagen para que la ventana de convolución del mapa de características sea del mismo tamaño, se conoce como *same*. De esta manera, no se reduce la dimensión de la salida después de la convolución. Sin embargo, aun utilizando *padding same*, los píxeles de los bordes tienen menos influencia que los píxeles centrados, porque se han utilizado menos veces en la etapa de convolución. Existe una última variante, denominada *full* o completa, que consiste en añadir tantos ceros como sea necesario para que cada píxel sea visitado k veces. Esta da como resultado una salida de tamaño $m+k-1$. Con esto, se consigue que todos los píxeles tengan la misma influencia, ya que se han utilizado el mismo número de veces. Por contrapartida, utilizar una imagen con tantos ceros puede dificultar la elección de un filtro, que sea capaz de aprender correctamente los patrones en todo el mapa de características.

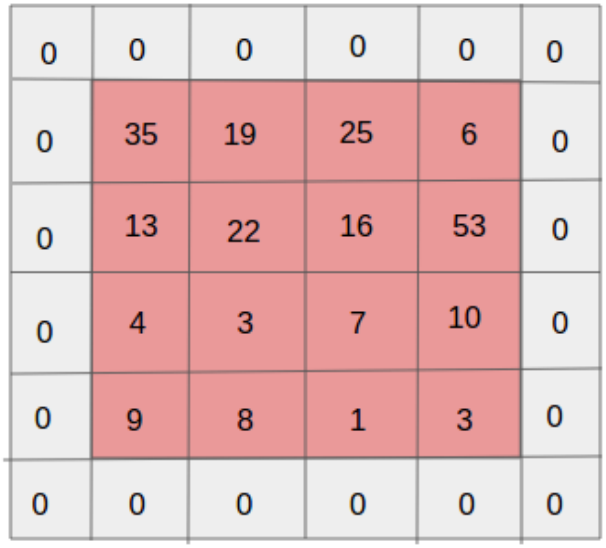


Figura 2.19: Ejemplo de padding same. Imagen de [55]

Asimismo, encontramos dos variantes menos comunes pero útiles en algunos casos. Las comentamos, brevemente, a contiunación.

- Parámetros no compartidos. Consiste en relajar la propiedad de parámetros compartidos, descrita en la sección 2.2.2. En este caso, existe una conexión local entre capas, asignando un peso diferente a cada uno de los parámetros sin necesidad de que sean iguales.

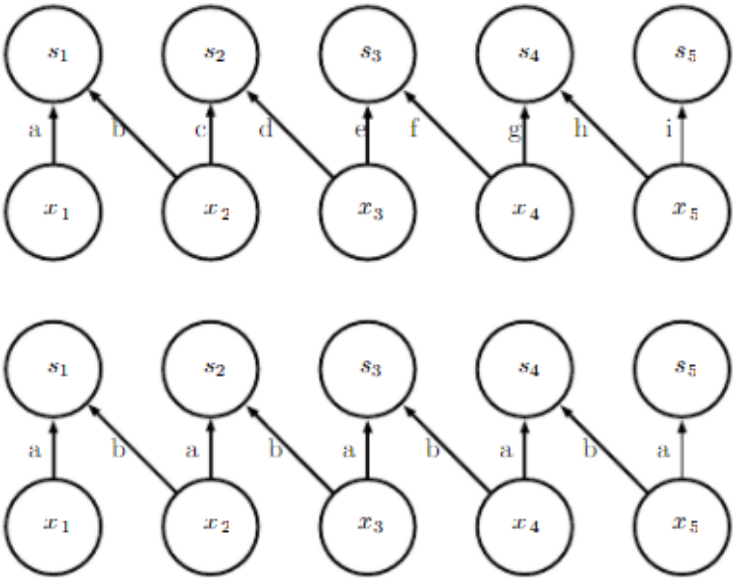


Figura 2.20: Ejemplo de parámetros no compartidos (arriba) frente a una convolución habitual (abajo). Imagen de [21].

En la Figura 2.20, cada borde está etiquetado con una letra única para mostrar que está asociado con su propio parámetro de peso en el gráfico superior. En el gráfico inferior, se observa cómo los pesos sí son compartidos. Puede ser especialmente útil, si sabemos que hay patrones muy específicos en pequeñas zonas o espacios de la imagen.

- Convolución en mosaico [24]. Supone un compromiso entre la convolución global, con parámetros compartidos, y local, con parámetros no compartidos. Consiste en aprender un conjunto de filtros que se van rotando a lo largo del espacio del mapa de características.

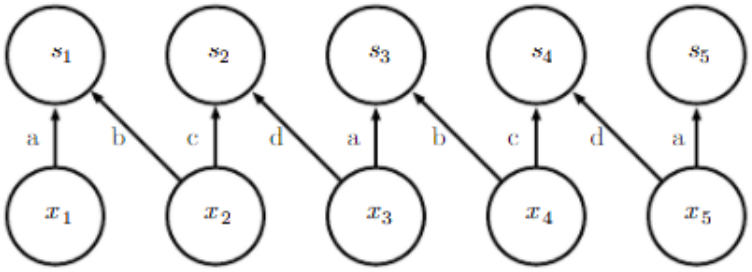


Figura 2.21: Ejemplo de convolución en mosaico. Imagen de [21].

Podemos ver, en la Figura 2.21, como hay varios filtros aprendidos que se van rotando en el mapa de características.

2.4. Grad-CAM

Gradient-weighted Class Activation Mapping [56] (Grad-CAM) es una técnica para producir explicaciones visuales en la toma de decisiones de una gran variedad de modelos basados en CNN. Utiliza los gradientes de cualquier concepto objetivo, que se propagan hacia la capa convolucional final, para producir un mapa de localización grueso, destacando, así, las regiones importantes en la imagen para predecir el concepto.

En el contexto de los modelos de clasificación de imágenes, las visualizaciones obtenidas con Grad-CAM aportan información sobre sus modos de fallo. Son resistentes a las imágenes adversas. Superan a los métodos anteriores en cuanto a localización. Son más fieles al modelo subyacente y ayudan a lograr la generalización mediante la identificación de los sesgos del conjunto de datos.

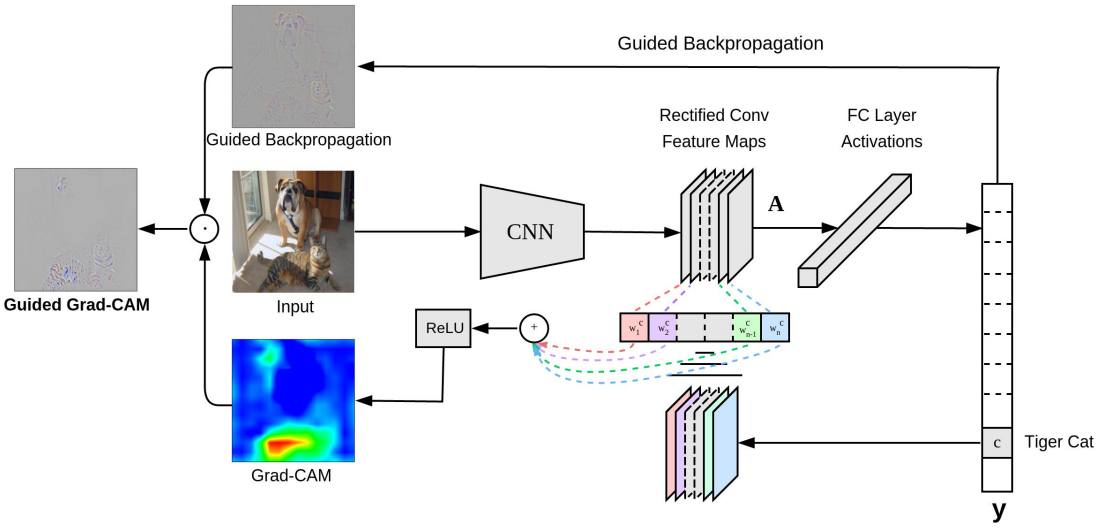


Figura 2.22: Arquitectura de Grad-CAM. Imagen de [56].

Las Redes Convolucionales retienen mucha información espacial que se pierde en MLP, por lo que podemos esperar, que las últimas capas convolucionales tengan el mejor compromiso entre la semántica de alto nivel y la información espacial detallada. Las neuronas de estas capas buscan información semántica específica de la clase en la imagen, en partes del objeto. Grad-CAM utiliza la información del gradiente, que se propaga hacia la última capa convolucional de la CNN, para asignar valores de importancia a cada neurona para una decisión concreta de interés. Para cuestiones matemáticas y más metodológicas, referimos a la publicación original [56].

Grad-CAM se utiliza para la diagnosis de la clasificación de imágenes en Redes Neuronales convolucionales. Podemos detectar los siguientes problemas.

- Análisis de los fallos del modelo. Podemos listar las imágenes mal clasificadas por el modelo y visualizar la contribución de zonas de la imagen a la predicción de la clase

predicha y las zonas que contribuyen dicha predicción de la clase real. La ventaja que proporciona Grad-CAM es su alta resolución y capacidad de distinguir clases.

- Efecto del ruido adverso. Se ha demostrado la vulnerabilidad actual de las Redes Neuronales a ejemplos adversos, que son ligeras perturbaciones imperceptibles de las imágenes de entrada, que engañan a la red para que las clasifique erróneamente. sin embargo, con las visualizaciones de Grad-CAM, se encuentra la categoría real de la imagen aunque la red no la clasifique así. Esto demuestra que Grad-CAM es bastante robusto al ruido.

- Identificar sesgos. Los modelos pueden estar sesgados y no generalizar correctamente a los datos del mundo real. Vemos un ejemplo muy ilustrativo de cómo se emplea esta técnica para detectar sesgos en la Figura 2.23.

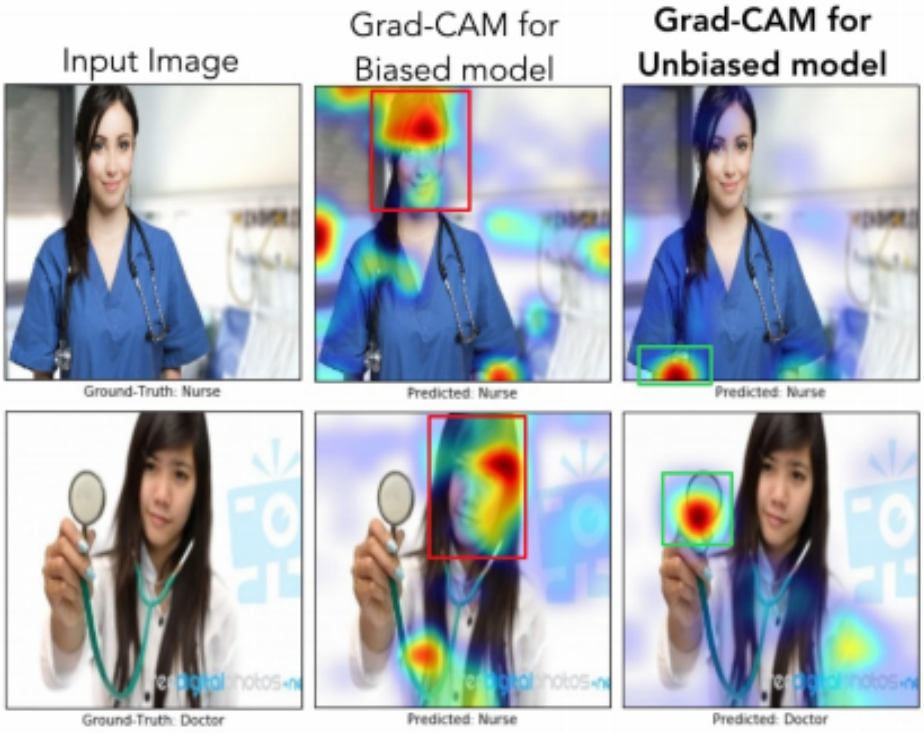


Figura 2.23: Detección de sesgo con Grad-CAM. Imagen de [56].

Grad-CAM revela que, el modelo central utiliza características como el pelo o la cara para predecir la imagen como enfermera. En el modelo de la derecha, vemos como no hay sesgos relacionados con el sexo y consigue detectar el fonendoscopio para realizar las predicciones, así como el color de la ropa en la imagen inferior.

Capítulo 3

Metodología

La metodología de trabajo habitual en proyectos de desarrollo de software no es óptima para llevar a cabos proyectos de investigación, o estudios con incertidumbre sobre los resultados y los entregables. Según Bob Hughes y Mike Cotterell [28], merece la pena realizar la planificación en proyectos inciertos, como los de investigación, siempre que los planes resultantes se consideren provisionales. Por ello, el capítulo 5 presenta una programación y planificación ligeramente ambigua, pero flexible, muy adecuada para los proyectos de ciencia de datos.

3.1. Proceso de desarrollo

Se opta por el modelo *Cross Industry Standard Process for Data Mining* (CRISP-DM). Se trata de un estándar abierto del proceso que describe los enfoques comunes más usados por los expertos en Minería de Datos [17]. Consta de las siguientes etapas:

1. Comprensión del negocio. Se debe fijar y entender los objetivos del proyecto.
2. Comprensión de los datos. Se lleva a cabo la recogida de datos y la familiarización con estos.
3. Preparación de los datos. Pre-procesamiento de los datos para su utilización en la siguiente etapa de modelización.
4. Modelización. En este punto se ajustan y entrenan los modelos.
5. Evaluación. Se consideran los mejores modelos y se comprueba si satisfacen los objetivos del proyecto.
6. Despliegue. Se entregan los resultados finales, ya sea en un informe o mediante una aplicación.

Podemos ver, en la Figura 3.1, la representación gráfica del ciclo CRIPS-DM. Como se puede observar, es un ciclo iterativo ya que, al avanzar en las diferentes etapas, puede ser necesario revisar las anteriores.

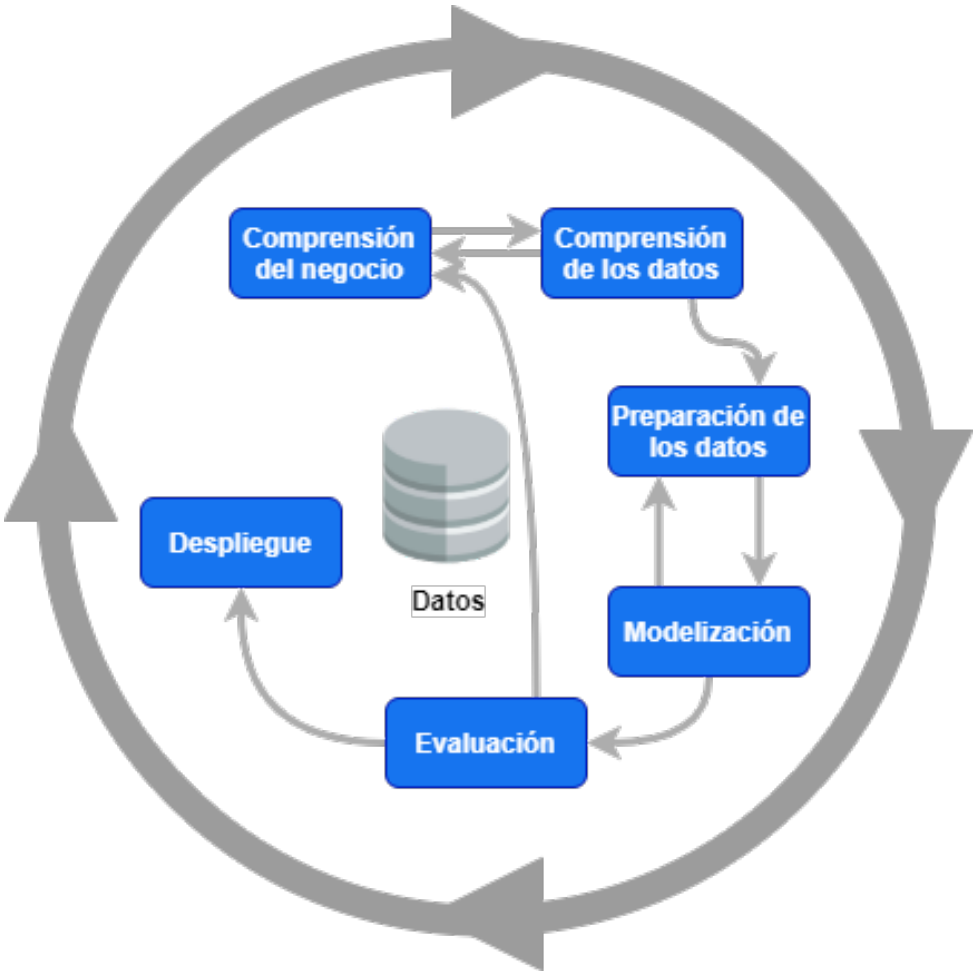


Figura 3.1: Ciclo CRISP-DM.

3.1.1. Entregables del proyecto

Podemos nombrar las siguientes etapas como hitos del proyecto o considerarlos como objetivos en la metodología de trabajo descrita.

- Entregable 1. Red Neuronal funcional adaptada al problema tratado.
- Entregable 2. Red Neuronal mejorada con alta precisión en la clasificación.
- Entregable 3. Visualización Grad-CAM para una predicción concreta con un modelo especificado.
- Entregable 4. Aplicación funcional, que implemente los requisitos que se describirán en la sección 9.1.1.

Debemos destacar, que cada entregable consiste en una mejora en la implementación o añadido de funcionalidad a lo anterior. Por ello, se presenta también un proceso de

desarrollo incremental, en el que es necesario haber cumplido con el entregable anterior para el desarrollo del siguiente.

3.2. Evaluación

Se decide mantener reuniones periódicas con el tutor cada semana o dos semanas, como máximo, para evaluar la evolución del proyecto. En cada etapa de las descritas en CRISP-DM, se realizan comprobaciones de las etapas anteriores en estas reuniones. Se valora el curso del proyecto, así como su tarea más inmediata, de acuerdo a la planificación establecida (ver sección 5). En cada tarea, se aborda el planteamiento y la forma, o posibles formas, de llevarla a cabo, ya sea uso de bibliotecas, decisiones sobre la implementación o el proyecto, etc. De forma coordinada con el tutor, se toman las decisiones clave para el correcto desarrollo del proyecto.

Durante las últimas semanas, se tiene un contacto casi a diario entre alumno y tutor, para garantizar el éxito del proyecto, tanto la implementación y modelos, como la documentación y memoria. Se reciben las correspondientes correcciones, que se van incorporando para una entrega satisfactoria del presente TFG.

Se utiliza el repositorio GitLab de la UVa para llevar un control de versiones del trabajo realizado. Podemos ver la estructura del repositorio en el Anexo B. **Aquí** el enlace al repositorio.

Capítulo 4

Marco de trabajo

En esta sección, se entrará más en detalle en las tecnologías específicas que se han empleado para el desarrollo del proyecto y de la aplicación.

Asimismo, se defenderán los criterios por los que se han seleccionado las herramientas elegidas, comparándolas con otras alternativas, haciendo referencia a posibles ventajas e inconvenientes.

4.1. Herramientas utilizadas

La clasificación supervisada consiste en asignar la etiqueta o clase correcta a una observación, tras haber procesado un conjunto de ellas ya etiquetadas. En particular, en este trabajo, se aborda un problema de clasificación supervisada, cuyos datos son imágenes.

Para llevar a cabo esta tarea, se pueden emplear multitud de herramientas. Existe software específico como WEKA [64], que tiene una sencilla interfaz gráfica de usuario para la construcción de modelos predictivos con algunos hiperparámetros configurables. Otra opción sería MATLAB [23], que es una plataforma de programación y cálculo numérico utilizada para analizar datos, desarrollar algoritmos y crear modelos. Sin embargo, utilizaremos como base el lenguaje de programación Python, que es interpretado, cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. El motivo principal para optar por este lenguaje de programación es la multitud de bibliotecas disponibles que facilitan el código de algoritmos de Aprendizaje Automático. Podemos ver, a continuación, una comparativa entre las distintas posibilidades y las decisiones tomadas.

4.1.1. Hardware

Para la elaboración de este proyecto se ha utilizado una máquina virtual prestada por el Departamento de Informática (ACT,CCIA y LSI) de la UVa con 4 núcleos virtuales,

con Intel 440FX. Respecto a la RAM, constaba de 16GB. Teniendo 50GB de tamaño de disco.

4.1.2. Sistema Operativo

Antes de empezar a trabajar debemos elegir el sistema operativo de la Máquina Virtual. Solo hay disponibilidad de Linux y Windows. Cabe destacar que, el proyecto puede realizarse en cualquier sistema operativo, pero se toma la decisión de utilizar Linux por los siguientes motivos.

- Es el SO de referencia en la Escuela de Ingeniería Informática.
- Es gratis frente a Windows, que no lo es.
- Es más seguro en cuanto a la intrusión de virus y *malware*.
- Es de código abierto. Cualquiera puede acceder al código del SO y modificarlo, si así lo desea, en su máquina.
- Los servidores basados en Linux ofrecen un mejor rendimiento que los de Windows, aparte de la ingente documentación para su implementación, hecho que no tiene parangón con los construidos en Windows.

Se ha utilizado [6] para realizar esta comparación entre Sistemas Operativos.

4.1.3. Lenguaje de programación

En este punto, valoramos varios lenguajes de programación, tanto para la parte de Aprendizaje Automático, como para la parte de desarrollo de software. Los lenguajes considerados son los siguientes.

- Python.
- Julia.
- Matlab.
- R.
- PHP.
- HTML.
- JavaScript.

A continuación, expondremos las ventajas y desventajas de cada uno de ellos, así como una breve comparativa y las decisiones tomadas.

Como comentario adicional, mencionamos que se decide utilizar un lenguaje de programación, frente al software específico, por la necesidad de crear modelos complejos muy adaptados al problema y por la capacidad de extensión de esto a una aplicación.

En primer lugar, debemos elegir un lenguaje para crear nuestros modelos de Aprendizaje Automático, así como para gestionar y procesar los datos. Como primera opción, pensamos en los lenguajes de programación capaces de resolver este problema. Al ser una cuestión puramente matemática, aunque compleja, se podría resolver con cualquier lenguaje de programación. Sin embargo, existen muchos con bibliotecas que implementan la funcionalidad requerida por este proyecto. Julia es un lenguaje puramente pensado para proyectos de *Machine Learning* y, por tanto, consideramos difícil su extensión a una aplicación, a parte de ser un lenguaje nuevo con relativamente poca documentación publicada, lo que dificultaría su aprendizaje. A pesar de ofrecer grandes ventajas como la velocidad de cómputo [4], decidimos descartar este lenguaje.

PHP, aunque es un gran lenguaje en el desarrollo web, apenas proporciona bibliotecas para la elaboración del proyecto, en la parte de modelado, siendo estas unas implementaciones de las cuestiones más simples sobre la metodología que se quiere utilizar, por tanto, también se deshecha este lenguaje.

A continuación, queremos comparar los lenguajes más populares en el Aprendizaje Automático: R, Python y Matlab [48]. Este último está orientado al campo matemático, con el que apenas hemos trabajado a lo largo de la carrera y con una curva de aprendizaje muy lenta, por ello, se decide descartarlo. Entre Python y R, se encuentra un considerable equilibrio en las técnicas y en la multitud de bibliotecas orientadas al Aprendizaje Automático. No obstante, Python es un lenguaje mucho más amplio que no solo está orientado a la Minería de Datos, también al desarrollo de aplicaciones web. Asimismo proporciona una más fácil interacción con el paradigma de orientación a objetos, que resulta complejo en R. Por todo ello, tomamos la decisión de utilizar Python para llevar a cabo el desarrollo de este proyecto.

Python nos da la libertad de configurar los parámetros, salidas y resultados tanto como queramos en comparación con WEKA, que tiene una interfaz bien definida y solo puedes hacer uso de ella. Igualmente, Python proporciona bibliotecas útiles para cálculo matemático, emulando así la funcionalidad de MATLAB. Todo ello, sumado al previo uso de Python durante los estudios y trabajos académicos, fomenta la decisión de utilizar este lenguaje de programación en este TFG.

Como herramienta, también se utilizará Anaconda, una aplicación que configura un entorno virtual, con el lanzamiento de cuadernos Jupyter, que permiten la ejecución de código por celdas y la impresión de resultados por pantalla.

Debemos comentar que, la aplicación web necesitará de lenguajes como HTML y JavaScript para la realización de las vistas y la interacción de estas. Se desconoce la posibilidad de utilizar otros lenguajes, pero se opta por estos, ya que son los más populares y los que se han trabajado durante el grado.

4.1.4. Bibliotecas

Uno de los motivos principales para tomar la decisión de utilizar Python es la multitud de bibliotecas disponibles que facilitan el código de algoritmos de Aprendizaje Automático (TensorFlow , Keras , scikit-learn...), así como su forma de almacenar y tratar los datos (Numpy , Pandas...). En resumen, proporciona muchas facilidades para este tipo de tareas. Asimismo, permite ejecutar comandos en el interior del código (OS , shutil) y otros (matplotlib , Pillow) para manejar imágenes o gráficos y mostrarlos. Además, tiene bibliotecas, como Flask y Gunicorn, para adaptar el código a aplicaciones web. En la parte del desarrollo de la aplicación, se utilizan los lenguajes HTML y JavaScript para poder crear las vistas y manejarlas. Igualmente, se utiliza CSS para proporcionar una hoja de estilo a los elementos de la web.

Se considera de utilidad dar más detalles sobre cada biblioteca, así como su versión.

- OS: facilita el uso de comandos desde los cuadernos o scripts de Python.
- shutil: provee de operaciones de alto nivel.
- numpy: (versión 1.19.2) da soporte para crear estructuras de datos como matrices y vectores, con una gran variedad de funciones matemáticas.
- Pandas: (versión 1.1.3) nació como una extensión de numpy. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- scikit-learn: (versión 0.23.2) es para Aprendizaje Automático de software libre para el lenguaje de programación Python, que incluye varios algoritmos de clasificación, regresión y análisis de grupos.
- TensorFlow: (versión 2.2.0) es una plataforma de código abierto especialmente utilizada para Aprendizaje Profundo.
- Keras: (versión 2.4.3) está diseñada para experimentar con Redes Neuronales. Puede ejecutarse sobre TensorFlow.
- Flask: (versión 1.1.2) está basado en la especificación *Web Server Gateway Interface* WSGI de Werkzeug y el motor de *templates* o plantillas Jinja2. Es una biblioteca basada en otras que facilita el desarrollo web.
- Gunicorn: (versión 20.1.0) proporciona un servidor http en Unix.

Existen muchas otras bibliotecas capaces de implementar las mismas funcionalidades, pero optamos por estas, ya que logran unos buenos resultados y de las que se tiene conocimientos previos, que facilitarán la labor y la etapa de aprendizaje.

Capítulo 5

Planificación

El presente Trabajo de Fin de Grado constituye 12 *European Credit Transder System* (ECTS) de la formación académica de la titulación de Grado en Ingeniería Informática. Teniendo en cuenta el actual marco de traducción a horas de la UVA, implicaría una dedicación estimada de 300 horas. El inicio data del 15 de febrero de 2021 y su fecha estimada de entrega es el 28 de junio. Por lo que, tendríamos un marco temporal de 19 semanas con una carga promedio de trabajo de 16 horas semanales.

5.1. Tareas

Se presentan los pasos clave para la consecución del objetivo del proyecto. Tras la etapa de introducción, destacamos:

- Iniciación, donde se fijan los objetivos y la metodología.
- Preparación de los datos: se realizará la obtención y exploración de los datos, así como el procesamiento de los mismos.
- Modelización: investigación e implementación de modelos capaces de resolver nuestro problema.
- Prueba: se comprobarán los resultados obtenidos en la tarea anterior y se elegirá un modelo que constituirá la base del producto final.
- Despliegue: se elaborará una aplicación de despliegue local con fácil usabilidad por cualquier usuario.
- Documentación: de forma continua, a lo largo del proyecto, se irá confeccionando la documentación y esta memoria.

5.2. Estimación de coste

Con las tareas presentadas y el tiempo disponible, se realiza una estimación del esfuerzo en horas de trabajo para desarrollar la planificación del mismo.

Tarea	Subtareas	Tiempo estimado en horas	Estimación total
Iniciación	Fijar los objetivos del proyecto	5	35
	Planificación	5	
	Revisión bibliográfica	25	
Preparación de los datos	Búsqueda de datos	5	20
	Almacenamiento y formato de imágenes	5	
	Tratamiento y procesamiento previo de los datos	10	
Modelización	Aprendizaje de librerias	15	75
	Selección de modelos candiatos	25	
	Implementación de modelos	35	
Evaluación	Prueba de los modelos	20	40
	Selección de modelos	20	
Despliegue	Aprendizaje sobre despliegue de aplicaciones	25	50
	Implementación de la aplicación	25	
Documentación	Elaboración del informe	80	80
Total			300

Tabla 5.1: Estimación de costes

De acuerdo con la metodología presentada en la sección 3.1, las etapas de comprensión son englobadas en la tarea iniciación; las demás se pueden identificar inequívocamente.

Debemos recordar lo que se mencionó en la sección 3.1: merece la pena hacer la planificación siempre y cuando se considere provisional y sea flexible. De esta manera, presentamos la estimación inicial de los costes de las tareas del proyecto.

Podemos ver, en la Figura 5.1, una estimación de las fechas en relación a los costes previstos en la Tabla 5.1. Las fechas surgen en la planificación inicial y en el cómputo de esfuerzos. Empero, el desarrollo de trabajo no siempre cumple con esta planificación, pues la elaboración de las tareas ha supuesto un proceso continuo y simultaneo en vez de secuencial. Hay que tener en cuenta que, las fechas se estiman con una jornada de trabajo de 8 horas diarias, sin descanso en sábado y domingo. La situación real supone que las primeras semanas de proyecto no se satisfacen los objetivos, pues no se puede dedicar esa cantidad de tiempo por diferentes motivos, como la elaboración de otro TFG, las prácticas externas y el curso de una asignatura. En la parte central y final del proyecto, las horas de trabajo superan a las estimadas para compensar el déficit anterior.

5.3. Variaciones respecto la planificación inicial

Como ya se ha comentado, las primeras semanas de trabajo, por diferentes motivos, no cumplieron con los tiempos establecidos. Sin embargo, la planificación de estas sí se llevó

a cabo. Respecto de la planificación inicial, podemos observar dos grandes variaciones:

- Inclusión de técnicas de interpretabilidad de modelos. En los objetivos iniciales, no se contemplaba incluirla, pero a medida que se fue desarrollando el proyecto y conociendo sus características, resultó realmente interesante investigar sobre estas técnicas e incluirlas en el modelado posterior de la aplicación. Esto supuso una cantidad de tiempo y esfuerzo, que no está contemplada en la planificación inicial.
- Falta de tiempo para Dockerización. En la etapa de despliegue de la aplicación, estaba previsto incluir la aplicación en un contenedor Docker, para facilitar su transporte entre máquinas y su ejecución. Al final, esto no se ha llevado a cabo por falta de tiempo, ya que las versiones de Docker realizadas no funcionaban correctamente y, por tanto, no se incluyen en la entrega final.

Finalmente, es justo mencionar que la elaboración de este informe tiene su grueso de trabajo en el último mes, ya que las primeras semanas supusieron una introducción en la materia, así como las primeras pruebas. En ellas, solo se realizó la planificación correspondiente a la memoria.

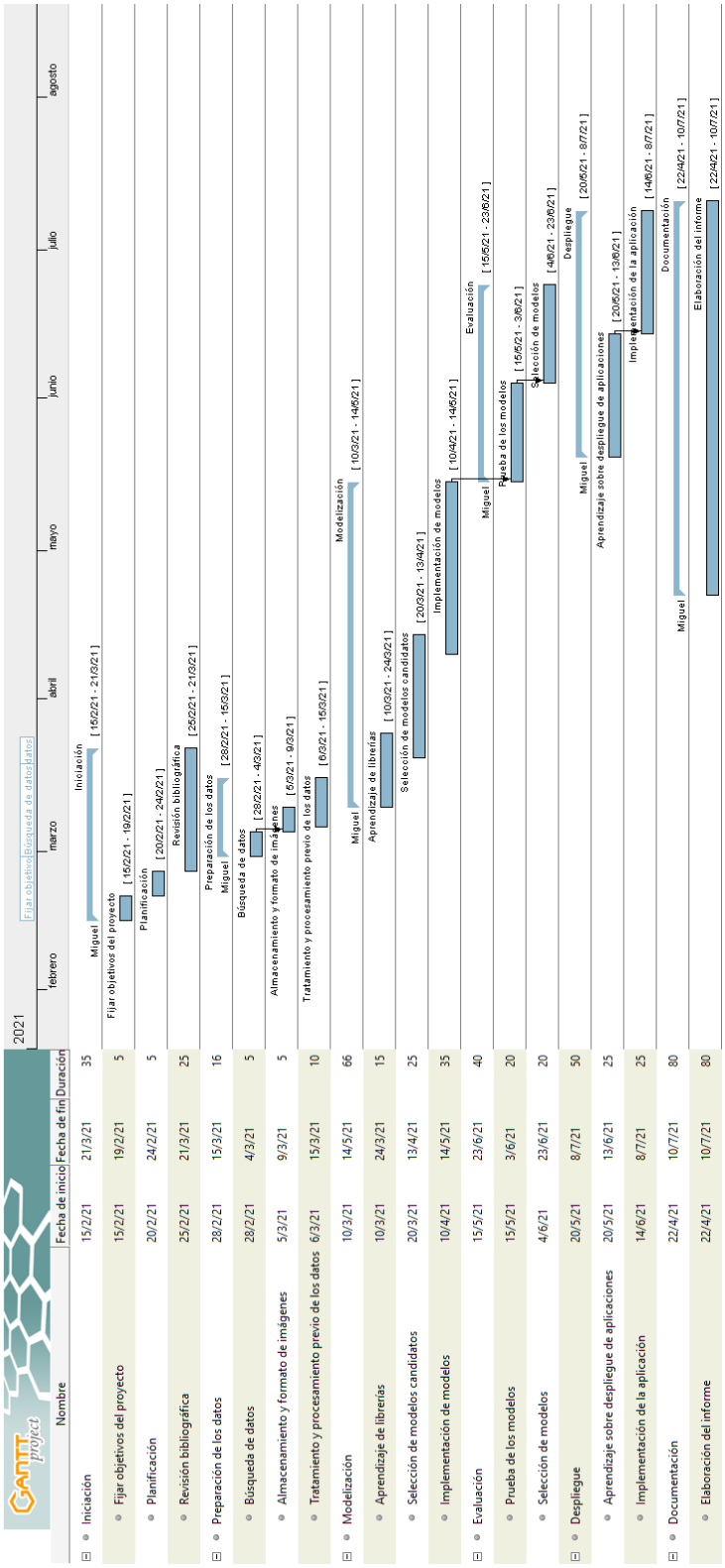


Figura 5.1: Diagrama de Gantt para el proyecto.

Capítulo 6

Datos

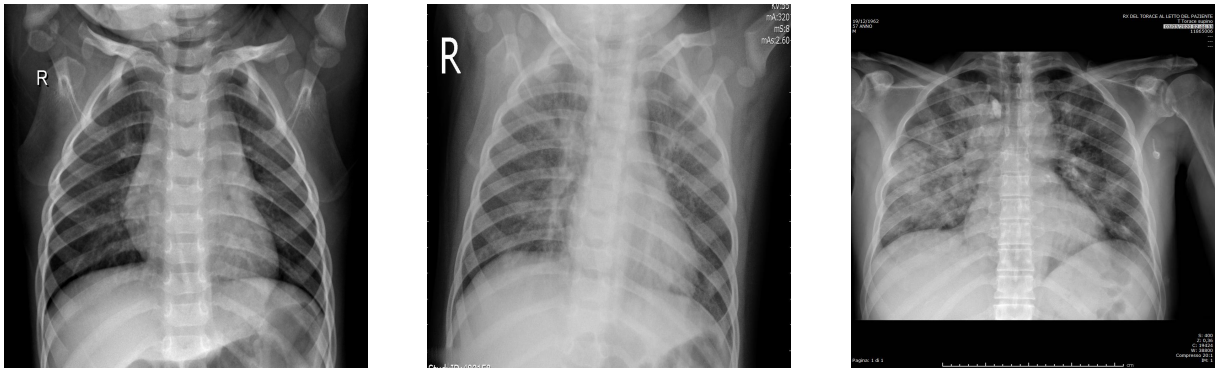
6.1. Descripción de los datos

Contamos con un conjunto de imágenes correspondientes a radiografías de tórax procedentes de una competición publicada en Kaggle. Está compuesto por 2905 imágenes con un tamaño de 1024x1024 píxeles repartidas en tres clases o categorías.

- Normal. Sin ninguna enfermedad o sano.
- Neumonía. Con neumonía, pero negativo Covid. En el conjunto de datos lo llaman *Viral Pneumonia* o neumonía vírica.
- Covid-19. Positivo en Covid-19.

Las imágenes de las clases Normal y Viral Pneumonia están en escala de grises y, por lo tanto, solo tienen un canal. Las de la clase Covid-19 están en formato RGB y, por ello, tienen tres canales. Se toma la decisión inicial de tratar todas las imágenes de la misma manera, entonces se transforman las imágenes en color a escala de grises, para poder tratarlas de manera uniforme en nuestros modelos de Aprendizaje Automático.

Las imágenes normales, junto con las de neumonía vírica, proceden de la base de datos de Kaggle de Paul Moore, *Chest X-Ray Images (Pneumonia)* [11]. Mientras que, las imágenes positivas de COVID-19 se recogen de varias fuentes abiertas: la base de datos COVID-19 de la Sociedad Italiana de Radiología Médica e Intervencionista, *Società italiana di Radiologia Medica e Interventistica*(SIRM), del conjunto de datos Novel Corona Virus 2019 (nCOVID-19) de Joseph Paul Cohen, Paul Morrison y Lan Dao, y de otras 43 publicaciones diferentes [54].



(a) Normal (b) Pneumonía vírica (c) Covid-19

Figura 6.1: Muestra de imágenes de cada clase.

6.2. Preprocesado

6.2.1. División del conjunto

Los datos originales se dividen en dos grupos: entrenamiento y prueba. De esta manera, se pretende ajustar un modelo capaz de generalizar a datos que nunca ha visto. Esto lo conseguimos reservando un conjunto de observaciones, que el modelo nunca ve en su etapa de entrenamiento y solamente se usan para obtener una estimación del error que no sea optimista, esto es, una estimación justa del error.

Esta técnica se conoce, en metodología experimental, como *hold-out* o método de resorte. Reservamos un 33.3 % de observaciones del total para construir el conjunto de prueba. El 66.6 % restante constituirá el conjunto de entrenamiento. El reparto de observaciones entre ambos grupos se realiza con un muestreo de forma estratificada, para que todas las clases tengan un mismo porcentaje de observaciones representativo de la distribución inicial, es decir, se mantiene la distribución inicial de clases en los dos subconjuntos de entrenamiento y de prueba. Podemos ver, de forma gráfica, en qué consiste este método en la Figura 6.2.

La situación ideal, en estos casos, es contar también con un conjunto de validación que se usa para elegir el modelo óptimo y, posteriormente, se emplea el conjunto de prueba para obtener una estimación del error verdadero. En este trabajo, por la escasez de observaciones, se prescinde del conjunto de validación.

En la Tabla 6.1, podemos observar la distribución de clases de los datos.

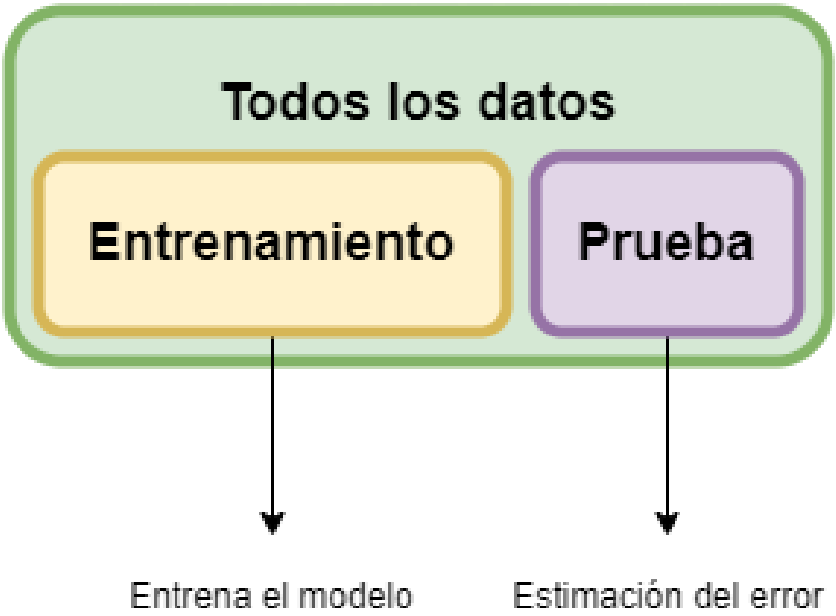


Figura 6.2: Metodología *Hold-out*.

Clases	Conjunto de entrenamiento	Conjunto de prueba	Total
Covid-19	146	73	219
Pneumonia	896	449	1345
Normal	894	447	1341

Tabla 6.1: Distribución de clases.

En la Figura 6.3, podemos ver la distribución de clases en el conjunto de datos original.

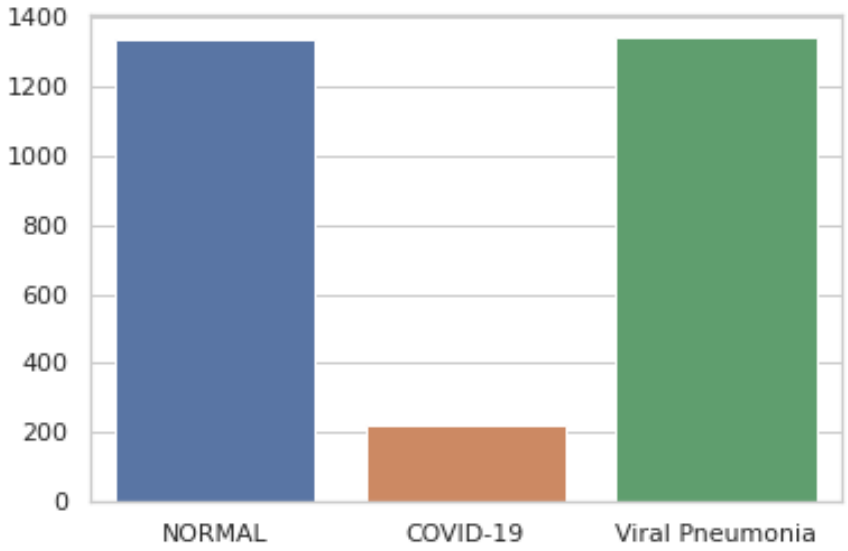


Figura 6.3: Gráfico de barras para la clase de los datos originales.

En la Figura 6.4, apreciamos la distribución de clases en el conjunto de entrenamiento tras el muestreo de estos con la técnica, previamente mencionada: *hold-out*.

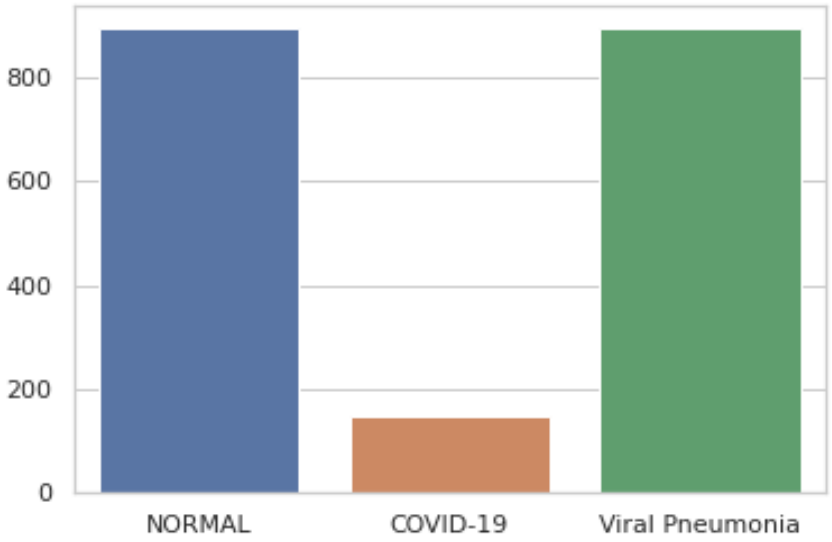


Figura 6.4: Gráfico de barras para la clase de los datos de entrenamiento.

Finalmente, observamos, en la Figura 6.5, la distribución de clases en el conjunto de prueba.

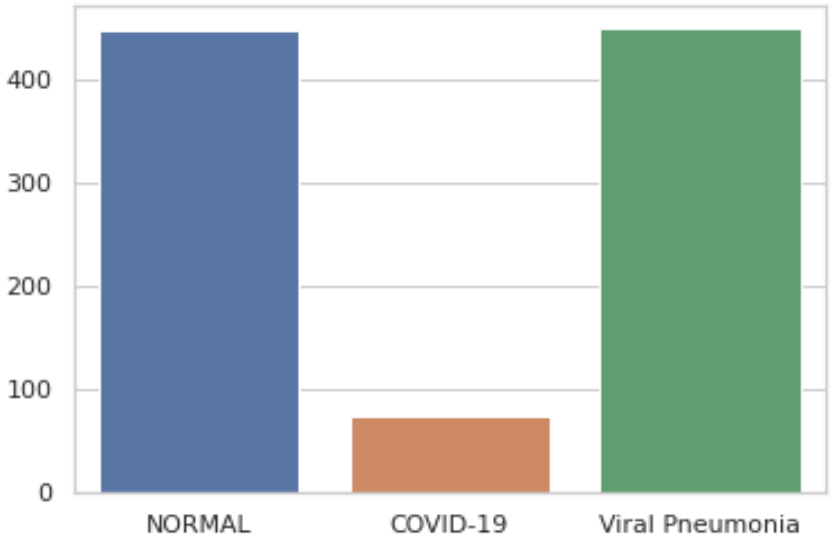


Figura 6.5: Gráfico de barras para la clase de los datos de prueba.

6.2.2. Normalización

Las imágenes son sometidas a un proceso de normalización. Se escalan para que todos los valores de los píxeles se encuentren en el entorno $[0, 1]$. Podemos realizar esta operación con la Fórmula 6.1.

$$X_i = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{6.1}$$

En nuestro caso, el valor mínimo de un píxel es 0 y el máximo 255. Por tanto, en la práctica, podemos efectuar esta operación dividiendo el valor de cada píxel entre 255.

Capítulo 7

Construcción del sistema

Como ya se ha comentado, se ha elegido Python como lenguaje de programación para el desarrollo del proyecto, tanto los modelos de Aprendizaje Profundo, como el desarrollo de la aplicación.

Utilizaremos, principalmente, el paquete Keras para desarrollar estos modelos de Aprendizaje Profundo. Keras es una API de alto nivel construida sobre TensorFlow, quien proporciona el *back-end* o motor de ejecución. Habitualmente, estas dos bibliotecas se usan de manera simultánea, ya que TensorFlow proporciona los medios para la realización de procesos. Por su parte, Keras construye una abstracción de alto nivel, para que estos procesos sean fácilmente usables. Tanto es así, que el propio TensorFlow incorpora los módulos de Keras. Es más, esta implicación es en ambos sentidos, ya que si importamos la librería de Keras, esta trae consigo sus dependencias, entre ellas, TensorFlow.

7.1. Lectura de datos

Contamos con un cierto número N de imágenes de $1024x1024$ píxeles. Podríamos almacenarlas en memoria con una matriz de dimensiones $Nx1024x1024$. Sin embargo, esto rompe con cualquier almacenamiento de memoria, en tiempo de ejecución, y no nos permite el tratamiento de estas imágenes de la manera habitual.

Buscamos una forma de cargar estas imágenes en memoria, en el momento que vayan a ser utilizadas. En cada época de entrenamiento, el número de imágenes utilizadas no es el total del conjunto de *train*, sino un pequeño subconjunto de estas llamado *batch* o lote. Idealmente, este número debe ser lo suficientemente pequeño para que todas las imágenes tengan una gran influencia y los pesos se actualicen de forma adecuada, pero también suficientemente grande para no ralentizar de forma notable el proceso de entrenamiento. Como indica Yan LeCun, conocido como el padre de las Redes Convolucionales, el tamaño ideal de *batch* sería de 1, si las condiciones computacionales nos lo permiten [40, 42] y, en cualquier caso, este tamaño no debería superar las 32 muestras.

Aprovechamos este hecho para cargar únicamente un número fijado de imágenes en memoria al mismo tiempo, así, podemos satisfacer las restricciones de memoria de la máquina en la que estamos trabajando. Fijamos un tamaño de lote múltiplo del número de observaciones tanto para el conjunto de prueba, como para el de entrenamiento logrando, de esta forma, que todas las imágenes se utilicen el mismo número de veces. Esto nos hace no respetar la recomendación de Yan LeCun, ya que utilizamos 44 muestras en el conjunto de entrenamiento y 19 en el de prueba como lote.

Utilizamos un generador de imágenes proporcionado por la librería Keras. Tiene la función principal de crear una estructura de datos iterable con las imágenes de un directorio sin cargar su totalidad en memoria, es decir, las almacena dinámicamente de la manera indicada anteriormente: en la etapa de entrenamiento o prueba se guardan las imágenes correspondientes al lote actual. Este generador permite también utilizar técnicas de *Data Augmentation*.

Utilizamos dos generadores, uno para el conjunto de entrenamiento y, otro, para el de prueba. Ambos realizan una lectura de imágenes en escala de grises, como se ha descrito previamente en la sección 6. Igualmente, son capaces de aplicar la normalización descrita en la sección 6.1.

En ciertos puntos de la etapa de entrenamiento, se producen unos picos de uso de memoria que colapsan el proceso. Se toma la decisión de aumentar el área de memoria *swap* o de intercambio, para soportar esos picos y poder continuar con la ejecución del proceso con normalidad. Otra posibilidad podría haber sido la reducción del tamaño de lote, a cambio de aumentar considerablemente el tiempo de ejecución de cada época del entrenamiento.

Data Augmentation

Es una técnica que consiste en realizar ligeras modificaciones a imágenes ya existentes para obtener otras nuevas, las cuales comparten etiqueta. Se basa en la aplicación de transformaciones tales como rotación o escalado. Su objetivo principal es aumentar el número de imágenes disponibles, cuando la muestra es pequeña y reducir, así, el sobreajuste al introducir más variabilidad.

En nuestro trabajo, se ha tomado la decisión de no utilizar esta técnica, pues las imágenes radiológicas de tórax son muy concretas, precisas y no tienen rotaciones ni escalados, puesto que se toman siempre de la misma manera. Es más, si una prueba no tiene un resultado satisfactorio, en cuanto a la imagen obtenida se refiere, el técnico de Radiología repite la prueba hasta obtener el resultado esperado. Por ello, consideramos que incluir variabilidad en las imágenes, puede no resultar beneficioso en el proceso de clasificación.

7.1.1. Lectura de datos alternativa

Para evitar la utilización del área de memoria de intercambio, se propone utilizar otra forma de lectura de datos proporcionada por la biblioteca TensorFlow. Se opta por el formato de archivo recomendado para los conjuntos de datos de TensorFlow que es TFRecord. Este es un formato binario simple orientado a registros que contiene mensajes de búfer de protocolo de tf.train.Example, donde cada registro contiene uno o más atributos. Estos se convierten en tensores, cuando se ingresan en el modelo con fines de entrenamiento [12].

En la sección de 7.6, se proporciona, además, la versión adaptada a esta lectura de datos.

7.2. Creación de modelos

Para la redacción de este capítulo, utilizaremos el libro de F. Chollet [8].

Un modelo de Aprendizaje Profundo es un grafo acíclico y dirigido de capas, como vimos en la sección 2.1. La praxis más común es el apilado de capas con un mapeado individual de cada entrada a cada salida. La API de Keras proporciona dos posibilidades igual de válidas para la implementación de estos modelos.

1. *Sequential*: se utiliza cuando las conexiones entre las neuronas de una capa y la siguiente son completas o densas, es decir, todas las neuronas de una capa tienen como entrada la salida de todas las neuronas de la capa anterior. Por supuesto, su salida se propaga a todas las neuronas de la capa siguiente. Podemos ver un ejemplo de esta estructura de red en la Figura 7.1.

2. *functional API*: se denomina API funcional a la estructura abierta del grafo en la organización de capas y neuronas. Esta permite, como ventaja sobre *Sequential*, un añadido de complejidad estructural en cuanto a las conexiones de neuronas entre capas. Asimismo, permite una conexión densa jugando el mismo papel que *Sequential*. Podemos ver un ejemplo de esta estructura de red en la Figura 7.2.

En este trabajo, utilizaremos los modelos secuenciales, ya que trataremos de implementar Redes Neuronales de pocas capas. Por tanto, las arquitecturas secuenciales nos permiten explorar el desempeño de esta metodología.

7.2.1. Capas convolucionales

El siguiente paso en la creación de un modelo sería el apilado de capas. Como hemos decidido utilizar el modelo secuencial, la manera de apilar una capa a la secuencia ya existente es con la función *add()*, cuyo argumento será la siguiente capa. Automáticamente

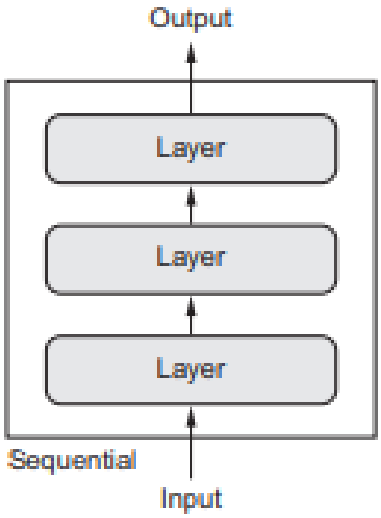


Figura 7.1: Ejemplo de estructura de una red neuronal secuencial. Imagen de [8].

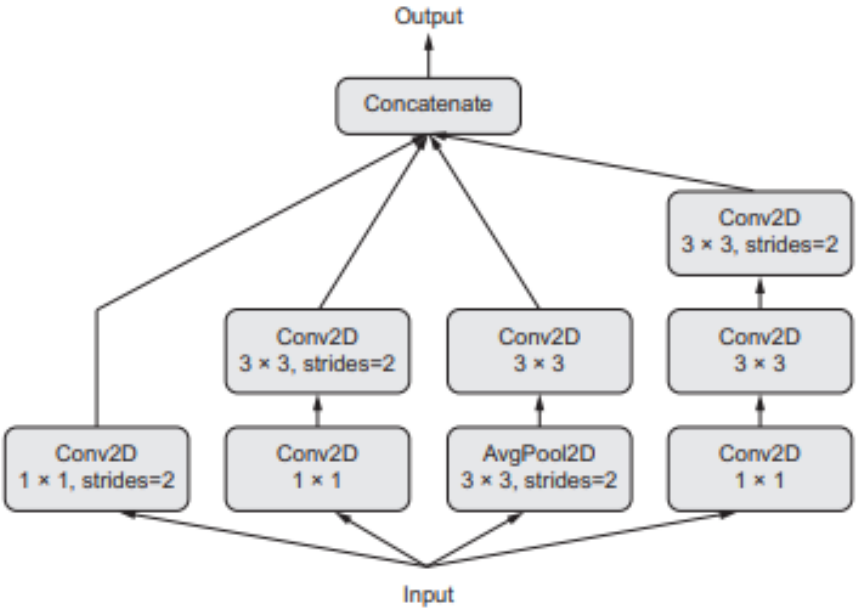


Figura 7.2: Ejemplo de estructura de una red neuronal con API funcional. Imagen de [8].

se crearan las conexiones correspondientes a las salidas de la capa anterior con las entradas de la nueva capa.

La primera capa, de entrada o *input*, se crea por defecto en Keras al incluir la primera capa oculta en el modelo mediante el argumento *input_ size*. Especificando este argumen-

to, el modelo implementa esta primera capa de entrada y sus conexiones, 1 a 1, ya que la capa de entrada no tiene pesos. Debemos dar a este argumento el valor correspondiente a las dimensiones de las imágenes de entrada, en nuestro caso 1024 x 1024, y el número de canales, como están en escala de grises, tendrán un único canal. Luego, debemos completarlo de la siguiente manera: *input_size = (1024, 1024, 1)*.

Este proceso de apilado de capas se utiliza, tanto para las convolucionales como para las de conexiones densas. En esta sección, presentaremos las capas convolucionales. Al tratarse de imágenes, utilizaremos capas con dos dimensiones o convolucionales 2D, como ya se explicó en la sección 2.2.4.

La capa convolutiva utilizada en la API de Keras [29] para dos dimensiones tiene los siguientes argumentos:

- *filters*: número, entero, de filtros de salida de la capa.
- *kernel_size*: puede ser un entero o una tupla que especifique las dimensiones, alto y ancho de la ventana de convolución, es decir, el tamaño del filtro.
- *strides*: puede ser un entero o una tupla para dar las dimensiones, alto y ancho, del stride o paso, salto, de la ventana de convolución. Por defecto, este valor es igual a 1.
- *padding*: podria ser *valid* o *same*, para referirnos al padding explicado en la sección 2.3.1.
- *data_format*: es el orden de las dimensiones en los datos de entrada, puede ser el canal primero, *channel_first*, o al final, *channel_last*. Por defecto, se utiliza el canal al final.
- *dilation_rate*: entero o tupla que especifica las dimensiones de la tasa de dilatación. Es incompatible con el stride solo uno de ellos puede tener valor distinto de 1. Por defecto, este valor es igual a 1.
- *groups*: entero positivo que indica el número de grupos en los que se divide la entrada a lo largo del canal. Cada uno de ellos tiene convoluciones y filtros individuales. Por defecto, este valor es igual a 1.
- *activation*: función de activación aplicada.
- *use_bias*: si se usa, o no, el vector de *bias*. Por defecto, sí que se utiliza.
- *kernel_initializer*: valores iniciales para la matriz de pesos. Por defecto, es una matriz de numeros aleatorios generados con la técnica *glorot_uniform*.
- *bias_initializer*: valores iniciales para el vector de *bias*. Por defecto, es un vector de ceros.

- *kernel_regularizer*: función de regularización aplicada a la matriz de pesos. Por defecto, no se usa ninguna.
- *bias_regularizer*: igual, pero aplicado al vector de *bias*. Por defecto, tampoco se aplica ninguna.
- *activity_regularizer*: función de regularización para la salida de activación. Por defecto, ninguna.
- *kernel_constraint*: función de restricciones sobre la matriz de pesos. Por defecto, no se aplica ninguna.
- *bias_constraint*: función de restricciones aplicadas al vector de *bias*. Por defecto, no se aplica ninguna.

En la capa de reducción de dimensionalidad o *pooling*, también se debe especificar la dimensionalidad y tiene que coincidir con la de la capa convolucional. La utilizada en la API de Keras [31] para dos dimensiones contempla los siguientes argumentos:

- *pool_size*: número entero o tupla de números enteros que especifican el tamaño de la ventana en la cual se aplica la función de reducción elegida. Si solo se especifica un entero, tendrá dimensiones cuadradas con ese valor. Por defecto, es $(2, 2)$.
- *strides*: número entero o tupla de números enteros que indica el tamaño del salto, cuánto se mueve entre una y otra ventana. Por defecto, no hay stride.
- *padding*: *valid* o *same* especifica el padding. Por defecto, es *valid*, o lo que es lo mismo, no hay padding.
- *data_format*: es el orden de las dimensiones en los datos de entrada, puede ser el canal primero, *channel_first*, o al final, *channel_last*. Por defecto, se utiliza el canal al final.

7.2.2. Perceptrón multicapa

Tras aplicar todas las convoluciones, se plantea añadir alguna capa densamente conectada que recoja la información procedente de las convoluciones y la procese. Para añadir una última capa con tantas neuronas, como clases queramos distinguir, en nuestro caso 3, con una función de activación *softmax* basta, de forma que la salida de cada neurona juegue el papel de probabilidad de pertenencia a esa clase.

El primer paso para añadir estas capas es el aplanado, es decir, transformar la salida bidimensional procedente de los filtros de la etapa de convolución en un vector unidimensional fácilmente tratable por un MLP. Se consigue con la función *Flatten()* incorporada también en Keras.

Con ello, obtenemos la entrada para nuestra segunda parte de la red. En esta debemos añadir, de la misma manera, con el método *add()*, las capas que deseemos. En nuestro caso, se incorporan dos capas densamente conectadas llamadas *Dense* [30], con los siguientes argumentos:

- *units*: número de neuronas.
- *activation*: función de activación de las neuronas de esa capa. Por defecto, no se usa ninguna o lo que es lo mismo, la identidad.
- *use_bias*: si se utiliza el parámetro de *bias*. Por defecto, este valor es verdadero.
- *kernel_initializer*: proporciona o calcula unos pesos iniciales para la matriz de pesos. Por defecto, es *glorot_uniform* que suministra unos valores aleatorios a partir de una distribución normal [20].
- *bias_initializer*: calcula unos valores iniciales para el vector de *bias*. Por defecto, estos son ceros.
- *kernel_regularizer*: función de regularización aplicada al núcleo de la matriz de pesos. Por defecto, no se usa ninguna regularización.
- *bias_regularizer*: función de regularización del núcleo del vector de *bias*. Por defecto, no se contempla ninguna regularización.
- *activity_regularizer*: función de regularización de la salida o activación de la capa. Por defecto, no se usa ninguna regularización.
- *kernel_constraint*: función de restricciones del núcleo de la matriz de pesos. Por defecto, no se emplea ninguna regularización.
- *bias_constraint*: función de restricciones aplicada al núcleo del vector de *bias*. Por defecto, no se aplica ninguna regularización.

7.3. Compilación del modelo

Tras la construcción de la topología del modelo, este debe ser compilado antes de la etapa de entrenamiento. Esto se consigue con otra función provista por Keras llamada *compile()* [32]. En ella, debemos especificar los siguientes argumentos:

- *optimizer*: hace referencia al optimizador que se desea utilizar. Se explican algunos optimizadores en la sección 2.1.3. Por defecto, utiliza *RMSProp*.
- *loss*: o función de pérdida. Se explican algunas funciones de pérdida en la sección 2.1.2. Por defecto, no hay ninguna elegida, luego es un hiperparámetro que hay que especificar obligatoriamente. Se pueden construir funciones de pérdida manualmente utilizando código TensorFlow.

- *metrics*: es la lista de métricas con las que evaluar el modelo durante la etapa de entrenamiento. Podemos destacar las más comunes: *accuracy* o precisión para problemas de clasificación, *mse* o error cuadrático medio para salidas numéricas como en problemas de regresión.

- *loss_ weights*: lista o diccionario opcional que especifica los coeficientes escalares para ponderar las contribuciones de pérdidas de las diferentes salidas del modelo. Por defecto, no se aplica.

- *weighted_ metrics*: lista de métricas para evaluar y ponderar mediante el muestreo de pesos durante el entrenamiento y la evaluación. Por defecto, no se aplica.

- *run_ eagerly*: valor booleano, con valor falso por defecto. Si es verdadero, no se envuelve en una función de TensorFlow. Se recomienda dejar como *None*, a menos que no pueda ejecutarse.

- *steps_ per_ execution*: valor entero, por defecto a 1, que indica el número de lotes durante cada llamada a la función de TensorFlow. Ejecutar varios lotes simultáneamente puede favorecer una reducción del tiempo de entrenamiento en TPU (*tensor processing unit* o unidad de procesamiento tensorial).

7.4. Entrenamiento

Tras la complicación del modelo, abordamos la etapa de ajuste o entrenamiento. Consiste en calcular los pesos de las conexiones entre neuronas, así como los valores de los filtros.

Podemos realizar esta tarea con la función *fit()* proporcionada por Keras [32], que tiene los siguientes argumentos:

- x: son los datos de entrada. Podría ser una array de Numpy, un Tensor o un dataset de TensorFlow, como es nuestro caso.

- y: son los datos objetivo, la clase de cada imagen.

- *batch_ size*: un número entero que represente el tamaño del lote. En nuestro caso, este ya se ha especificado en el generador del dataset.

- *epochs*: entero que representa el número de épocas. Por defecto, vale 1.

- *verbose*: 'auto', 0, 1, o 2. Hace referencia a la información de salida del proceso de entrenamiento: 0 = silencioso, 1 = barra de progreso, 2 = una linea para cada época. Por defecto, es 'auto'.

- *callbacks*: lista de *callbacks* aplicadas durante el proceso de entrenamiento. Por defecto, no se aplica ninguna. Algunos ejemplos de *callbacks* pueden ser la detención temprana o *early stop* o la reducción de la tasa de aprendizaje, utilizada por el optimizador, a medida que avanza el entrenamiento.
- *validation_split*: float entre 0 y 1, que representa el porcentaje de datos de entrenamiento utilizados para validación del modelo. Por defecto, es 0. El porcentaje de datos de validación se selecciona de los últimos datos, por ejemplo, 0.1 nos haría seleccionar 10 % de observaciones desde el final, antes de muestrear los datos de entrenamiento en lotes.
- *validation_data*: datos sobre los que se evalúa la función de pérdida y las métricas del modelo al final de cada época.
- *shuffle*: valor de verdadero, o falso, que indica si se permutan aleatoriamente, o no, el orden de los ejemplos de entrenamiento antes de cada época. Por defecto, este valor es verdadero.
- *class_weigth*: Diccionario opcional que asigna índices de clase a un valor de peso, utilizado para ponderar la función de pérdida. Esto puede ser útil para indicar al modelo que preste más atención a las muestras de una clase poco representada. Por defecto, no se aplica.
- *sample_weight*: matriz de pesos para ponderar la función de pérdida con las muestras de entrenamiento. Por defecto, no se aplica.
- *initial_epoch*: época en la que se comienza el entrenamiento. El valor por defecto es 0.
- *steps_per_epoch*: número de lotes que se utilizarían en cada época, al contrario que tamaño de lote, este fija el número de lotes y reparte las muestras entre estos. El valor por defecto es None.
- *validation_steps*: solo para cuando se provee un conjunto de validación, es el número de lotes usados en la validación.
- *validation_batch_size*: número entero que representa el tamaño del lote en el conjunto de validación, es decir, la cantidad de muestras en cada lote.
- *validation_frec*: número de épocas que pasa entre dos validaciones consecutivas. Por defecto, vale 1, es decir, se realiza en todas las épocas.
- *max_queue_size*: tamaño máximo de la cola del generador, si no se especifica vale 10.
- *workers*: número máximo de procesos, que se pueden ejecutar en paralelo, cuando se usa computación multihilo. Si no se especifica, será por defecto 1.

- *use_multiprocessing*: valor booleano que indica si se utiliza o no procesamiento en paralelo. Por defecto, es falso, no se utiliza.

La función de ajuste retorna un objeto de tipo *History*, en el cual se almacena, entre otras cosas, la información correspondiente a la función de pérdida y las métricas en cada época del proceso de entrenamiento.

7.5. Evaluación

Tras construir el modelo y entrenar sus parámetros, interesa comprobar su funcionamiento con los datos reservados para evaluación, básicamente, estimando su tasa de error.

La función *evaluate()*, disponible en la API de Keras [32], nos proporciona una herramienta útil para el desarrollo de esta tarea. Esta función necesita los siguientes argumentos:

- x: son los datos de entrada. Podría ser una array de Numpy, un Tensor o un dataset de TensorFlow, como es nuestro caso.
- y: son los datos objetivo: la clase de cada imagen.
- *batch_size*: un número entero que represente el tamaño del lote. En nuestro caso, ya se ha especificado en el generador del dataset.
- *verbose*: 'auto', 0, 1, o 2.Hace referencia a la información de salida del porceso de entrenamiento: 0 = silencioso, 1 = barra de progreso, 2 = una linea para cada época. Por defecto, es 1.
- *sample_weight*: matriz de pesos para ponderar la función de pérdida con las muestras de entrenamiento. Por defecto, no se aplica.
- *steps*: número total de pasos, lotes, antes de declarar terminada la ronda de evaluación. Se ignora con el valor por defecto de None, en cuyo caso se ejecutará hasta que se agote el conjunto de datos.
- *callabacks*: lista de *callabacks* aplicadas durante la evaluación.
- *max_queue_size*: tamaño máximo de la cola del generador, si no se especifica vale 10.
- *workers*: número máximo de procesos que se pueden ejecutar en paralelo cuando se utiliza un proceso basado en hilos. Por defecto será 1.
- *use_multiprocessing*: valor booleano que indica si se utiliza, o no, procesamiento en paralelo.

- *return_dict*: booleano que establece si los resultados se devuelven como un diccionario, *True*, o como una lista, *False*.

Retorna un objeto con los resultados de la evaluación, tanto para la función de coste, como para las métricas establecidas.

7.6. Implementación

En esta sección, veremos paso a paso la implementación del modelo completo, desde el tratamiento de las imágenes de entrada, hasta la topología de la red y la evaluación de esta. Por simplicidad, se omiten las introducciones en las que se importan los paquetes y bibliotecas utilizados.

7.6.1. División de datos

Obtenemos los nombres de los ficheros para almacenarlos, en una etapa posterior, divididos en conjunto de entrenamiento y de prueba.

```
mypath = "/home/usuario/Escritorio/TFG/datos/COVID-19"
covid_files = [f for f in os.listdir(mypath)]

mypath = "/home/usuario/Escritorio/TFG/datos/NORMAL"
normal_files = [f for f in os.listdir(mypath)]

mypath = "/home/usuario/Escritorio/TFG/datos/Viral Pneumonia"
pneumonia_files = [f for f in os.listdir(mypath)]

clase_n = ['NORMAL' for i in range(0,len(normal_files))]
clase_p = ['Viral Pneumonia' for i in range(0,len(pneumonia_files))]
clase_c = ['COVID-19' for i in range(0,len(covid_files))]

clase = clase_n+clase_p+clase_c

df = pd.DataFrame(list(zip(files, clase)), columns =['FILENAME', 'CLASS'
    ])

train, test = train_test_split(df, test_size=1/3, stratify=df.CLASS)

for i in range(train.shape[0]):
    #train.iloc[i,1] = CLASE
    #train.iloc[i,0] = FILE NAME
    src = os.path.join("/home/usuario/Escritorio/TFG/datos",str(train.
    iloc[i,1]),str(train.iloc[i,0]))
    dst = os.path.join("/home/usuario/Escritorio/TFG/datos/train",str(
    train.iloc[i,1]),str(train.iloc[i,0]))
    shutil.copyfile(src,dst)

for i in range(test.shape[0]):
    #train.iloc[i,1] = CLASE
```

```
29      #train.iloc[i,0] = FILE NAME
30      src = os.path.join("/home/usuario/Escritorio/TFG/datos",str(test.
        iloc[i,1]),str(test.iloc[i,0]))
31      dst = os.path.join("/home/usuario/Escritorio/TFG/datos/test",str(
        test.iloc[i,1]),str(test.iloc[i,0]))
32      shutil.copyfile(src,dst)
```

Con esto, hemos logrado leer los nombres de todos los ficheros, de imágenes, en la carpeta de datos, separarlos de forma aleatoria y estratificada en dos subconjuntos: entrenamiento, 66.67 %, y, prueba, 33.33 %. Con esta distribución, se copian las imágenes en dos nuevos directorios independientes para cada subconjunto.

7.6.2. Lectura

Vemos cómo, con los subconjuntos creados, utilizamos el generador de datos proporcionado por la API de Keras, para confeccionar un lector de datos, que usaremos en la etapa de prueba y entrenamiento. Serán dos datasets independientes, uno para los datos de entrenamiento y otro para los datos de prueba. El tamaño se elige múltiplo del número total de observaciones, para que todas las imágenes se utilicen igual cantidad de veces. Se omiten las importaciones de paquetes y bibliotecas.

En la lectura de los datos se aplica el procesamiento descrito: normalización y transformación a escala de grises.

```
1 img_gen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255.,
    dtype=float)
2
3 train_dataset = keras.preprocessing.image.DirectoryIterator(directory='
    ../datos/train/', image_data_generator=img_gen, target_size
    =(1024,1024), batch_size=44, color_mode="grayscale")
4
5 test_dataset = keras.preprocessing.image.DirectoryIterator(directory='
    ../datos/test/', image_data_generator=img_gen, target_size
    =(1024,1024), batch_size=19, color_mode="grayscale", shuffle=False)
```

7.6.3. Lectura alternativa

Esta lectura no necesitaría hacer uso del área swap de memoria, ya que sigue las recomendaciones de TensorFlow en el tratamiento de conjuntos de datos [61]. Se realiza el mismo preprocesado que en la lectura anterior.

```
1 class_dict = {'NORMAL': 0, 'COVID-19':1, 'Viral Pneumonia':2}
2 num2label = {0:'NORMAL', 1:'COVID-19', 2:'Viral Pneumonia'}
3
4 TRAIN_LABELS = []
5 TEST_LABELS = []
6
7 for filename in TRAIN_IMGS:
8     label = str(filename.split('/')[4])
```

```
    #print (label)
    assert label in ['NORMAL', 'COVID-19', 'Viral Pneumonia']
    label = int (class_dict[label])
    TRAIN_LABELS.append (label)

for filename in TEST_IMGS:
    label = str(filename.split('/')[4])
    #print (label)
    assert label in ['NORMAL', 'COVID-19', 'Viral Pneumonia']
    label = int (class_dict[label])
    TEST_LABELS.append (label)

NCLASSES = 3
SIZE = (1024,1024)

def decode_image(filename, label = None, image_size= SIZE ):

    bits = tf.io.read_file(filename)
    image = tf.image.decode_png(bits, channels=1)
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.resize(image, image_size)

    if label is None:
        return image
    else:
        return image, label

AUTO = tf.data.experimental.AUTOTUNE

train_dataset = (
    tf.data.TFRecordDataset
    .from_tensor_slices((TRAIN_IMGS, TRAIN_LABELS))
    .map(decode_image, num_parallel_calls=AUTO)
    .repeat()
    .shuffle(1024)
    .batch(44)
    )

test_dataset = (
    tf.data.TFRecordDataset
    .from_tensor_slices((TEST_IMGS, TEST_LABELS))
    .map(decode_image, num_parallel_calls=AUTO)
    .batch(19)
    )
```

7.6.4. Construcción del modelo

Planteamos el modelo óptimo que hemos logrado obtener, en cuanto a tasa de acierto se refiere. Su topología es como sigue. La última capa requiere 3 neuronas y una función de activación *softmax*, ya que estamos ante un problema de clasificación con tres categorías.

La salida de cada una de ellas juega el papel de probabilidad de pertenencia a la clase que hace referencia esa neurona.

```
model = models.Sequential()
model.add(layers.Conv2D(8, (3,3), activation="relu", input_shape=(1024, 1024, 1)))
model.add(layers.Conv2D(8, (3,3), activation="relu"))
model.add(layers.MaxPooling2D((3, 3)))
model.add(layers.Conv2D(16, (3,3), activation="relu"))
model.add(layers.Conv2D(16, (3,3), activation="relu"))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(32, (3,3), activation="relu"))
model.add(layers.Conv2D(32, (3,3), activation="relu"))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation="relu"))
model.add(layers.Conv2D(64, (3,3), activation="relu"))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(128, (3,3), activation="relu"))
model.add(layers.Conv2D(128, (3,3), activation="relu"))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dense(3, activation ="softmax"))
```

7.6.5. Compilación del modelo

Se compila el modelo para fijar su topología y estructura antes de entrenar sus parámetros. De esta forma, asigna la función de périda, las métricas y el optimizador.

```
model.compile(loss="categorical_crossentropy", optimizer=optimizers.RMSprop(lr=1e-4),metrics=["acc"])
```

7.6.6. Entrenamiento

Calculamos el ajuste de parámetros para el conjunto de entrenamiento y el resto de ejemplos se reservan para validación. Con ello, se observará como se comporta el modelo, ante datos que nunca ha visto o que no ha utilizado en la etapa de actualización de pesos.

```
history = model.fit(
    train_dataset,
    epochs = 100,
    verbose = 1,
    validation_data=test_dataset)
```

7.6.7. Evaluación

Una vez que el modelo ya está entrenado, pasamos los datos del conjunto de prueba y observamos su comportamiento. Se obtiene una estimación del error de generalización, o

error verdadero, así como unos estadísticos resumen sobre el comportamiento del modelo con estos datos y la matriz de confusión. Podemos ver estos resultados en la sección 8

```
Y_pred = model.predict(test_dataset)
y_pred = np.argmax(Y_pred,axis=1)
target_names = ['COVID-19','NORMAL', 'Viral pneumonia']
c_matrix = confusion_matrix(test_dataset.classes, y_pred)
c_report = classification_report(test_dataset.classes, y_pred,
    target_names=target_names)
```

7.6.8. Grad-CAM

Para elaborar el mapa de calor de activación de la imagen utilizamos la referencia de la propia API de Keras [10].

```
model_builder = keras.applications.xception.Xception
preprocess_input = keras.applications.xception.preprocess_input
decode_predictions = keras.applications.xception.decode_predictions

model=tf.keras.models.load_model("modeloGrey9M.h5")
dims = model.input_shape[1:3]

image_uri = '/home/usuario/Escritorio/TFG/datos/test/COVID-19/COVID
    -19(135).png'

im = keras.preprocessing.image.load_img(image_uri, target_size=dims
                                        , color_mode="grayscale"
                                       )
last_conv_layer_name = "conv2d_9"

def get_img_array(img_path, size):
    # `img` is a PIL image of size 299x299
    img = keras.preprocessing.image.load_img(img_path, target_size=size,
    color_mode="grayscale")
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 299, 299, 3)
    array = np.expand_dims(array, axis=0)
    return array


def make_gradcam_heatmap(img_array, model, last_conv_layer_name,
    pred_index=None):
    # First, we create a model that maps the input image to the
    activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output,
    model.output]
    )
```

```
32
33      # Then, we compute the gradient of the top predicted class for our
        input image
34      # with respect to the activations of the last conv layer
35      with tf.GradientTape() as tape:
36          last_conv_layer_output, preds = grad_model(img_array)
37          if pred_index is None:
38              pred_index = tf.argmax(preds[0])
39          class_channel = preds[:, pred_index]
40
41      # This is the gradient of the output neuron (top predicted or chosen
        )
42      # with regard to the output feature map of the last conv layer
43      grads = tape.gradient(class_channel, last_conv_layer_output)
44
45      # This is a vector where each entry is the mean intensity of the
        gradient
46      # over a specific feature map channel
47      pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
48
49      # We multiply each channel in the feature map array
50      # by "how important this channel is" with regard to the top
        predicted class
51      # then sum all the channels to obtain the heatmap class activation
52      last_conv_layer_output = last_conv_layer_output[0]
53      heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
54      heatmap = tf.squeeze(heatmap)
55
56      # For visualization purpose, we will also normalize the heatmap
        between 0 & 1
57      heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
58      return heatmap.numpy()
59
60  # Prepare image
61  img_array = preprocess_input(get_img_array(image_uri, size=dims))
62
63
64  # Make model
65  model=tf.keras.models.load_model("modeloGrey9M.h5")
66  # Remove last layer's softmax
67  model.layers[-1].activation = None
68
69  # Print what the top predicted class is
70  preds = model.predict(img_array)
71  print("Predicted:", preds)
72
73  # Generate class activation heatmap
74  heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
75
76  def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg",
        alpha=0.4):
77      # Load the original image
```

```
78      img = keras.preprocessing.image.load_img(img_path, color_mode="
        grayscale")
79      img = keras.preprocessing.image.img_to_array(img)
80
81      # Rescale heatmap to a range 0-255
82      heatmap = np.uint8(255 * heatmap)
83
84      # Use jet colormap to colorize heatmap
85      jet = cm.get_cmap("jet")
86
87      # Use RGB values of the colormap
88      jet_colors = jet(np.arange(256))[:, :3]
89      jet_heatmap = jet_colors[heatmap]
90
91      # Create an image with RGB colorized heatmap
92      jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
93      jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
94      jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)
95
96      # Superimpose the heatmap on original image
97      superimposed_img = jet_heatmap * alpha + img
98      superimposed_img = keras.preprocessing.image.array_to_img(
        superimposed_img)
99
100     # Save the superimposed image
101     superimposed_img.save(cam_path)
102
103     # Display Grad CAM
104     display(Image(cam_path))
105
106
107 save_and_display_gradcam(image_uri, heatmap)
```

Capítulo 8

Resultados

En este capítulo, comentaremos cuáles han sido los resultados obtenidos, el mejor modelo que hemos encontrado, la tasa de error, matriz de confusión e interpretabilidad obtenida para futuras observaciones.

8.1. Evaluación del modelo

Tras ajustar los hiperparámetros manualmente y entrenar el modelo con los ejemplos, se efectúa el proceso de evaluación. Esto consiste en, con los datos reservados y no utilizados en la actualización de pesos de la red, pasar cada una de las observaciones por el modelo y obtener su clase predicha. Posteriormente, se compara esta clase con la real y, con esto, se obtiene una estimación de la tasa de error verdadero o, equivalentemente, del porcentaje de buena clasificación.

Cabe destacar, que los datos utilizados para la estimación del porcentaje de buena clasificación no han sido utilizados en la etapa de entrenamiento para actualizar los pesos del modelo, pues esto podría llevarnos a sobreajustar las observaciones existentes y obtener, así, una tasa muy elevada pero con una capacidad escasa de generalización. Igualmente, es deseable comentar, que la escasez de datos y potencia computacional nos lleva a realizar un experimento de *hold-out*, en el cual la partición inicial resulta claramente influyente, tanto en la construcción y entrenamiento del modelo, como en la estimación de su tasa de acierto.

Debemos destacar también, que el modelo se elige, de entre los modelos candidatos, el que menor error comente, tanto con los datos de entrenamiento, como con los datos de prueba. Esto podría suponer beneficios, pues es el que mejor se comporta con los datos que aún no ha visto el modelo. Sin embargo, no podemos garantizar plenamente su bondadporque sabemos que el modelo comete un error más bajo para estos datos de prueba, pues han sido los que se han utilizado para la selección del modelo. De esta manera, corremos los riesgos de que:

- Un mejor modelo, en términos de generalización, para datos que no están a nuestro alcance, sea más satisfactorio.
- El modelo elegido pueda estar sobreajustando, de alguna manera, a los datos de entrenamiento y validación de nuestra partición.

	Porcentaje de buena clasificación
Train	99.74 %
Test	96.18 %

Tabla 8.1: Porcentaje de buena clasificación para *Train* y *Test*.

En la Tabla 8.1, hemos obtenido las tasas de acierto para los conjuntos de entrenamiento y prueba. Hemos logrado tasas realmente altas, consiguiendo predecir casi la totalidad de los ejemplos correctamente.

Es interesante observar cómo se comporta dentro de cada grupo, es decir, comprobar si el porcentaje de buena clasificación es el mismo para todos los grupos, o varía entre ellos. Para esto, utilizamos la matriz de confusión: las filas indican la clase real y las columnas indican la clase predicha. Para que este resultado tenga sentido, utilizamos solo los datos de *test*.

		Predicho		
		COVID-19	NORMAL	Viral Pneumonia
Real	COVID-19	62	1	10
	NORMAL	1	430	15
	Viral Pneumonia	1	8	440

Tabla 8.2: Matriz de confusión.

En primer lugar, podemos observar, en la Tabla 8.2, de nuevo que la muestra está desbalanceada. Parece existir mayor confusión entre Covid-19 y Pneumonia, que con la clase Normal. Sin embargo, la clasificación resulta muy satisfactoria en todas las clases. Si queremos observar con más detalle la distribución de las predicciones sobre los datos de *test*, podemos calcular la matriz de confusión condicionada por filas. De esta manera, veremos el porcentaje de observaciones clasificado en cada grupo. Es decir, de los que tenían, por ejemplo, Covid-19, veremos, en la Tabla 8.3, qué porcentaje se clasificó como Covid-19, cuál se clasificó como Normal y como Pneumonia.

En la Tabla 8.3, podemos analizar específicamente el reparto de las nuevas predicciones. Vemos que la clase con peor tasa de aciertos es Covid-19. Esto resulta conflictivo,

		Predicho		
		COVID-19	NORMAL	Viral Pneumonia
Real	COVID-19	0.849	0.014	0.137
	NORMAL	0.002	0.964	0.034
	Viral Pneumonia	0.002	0.018	0.979

Tabla 8.3: Matriz de confusión condicionada por filas.

pues debería ser la clase que más interés tenemos en detectar. Debemos recordar, que la asignación a clases se realiza con la más probable, según la salida de la última capa del modelo, *softmax* con 3 neuronas. Nos planteamos en esta situación, si haber usado otras técnicas para determinar la asignación final, o incluso otras funciones de pérdida que diesen más importancia a la clase Covid-19, podrían haber incrementado la tasa de aciertos, sobre todo, en esta categoría.

Asimismo, hemos podido comprobar el buen desempeño en las clases Normal y Pneumonia vírica. Destacamos que el gran parecido entre las enfermedades Pneumonía y Covid a nivel pulmonar, pueden suponer este desliz en la clasificación de la clase Covid. En este caso, es lógico que, ante una situación confusa, se decante por la clasificación de Pneumonia, porque resulta ser el grueso de observaciones de este grupo y, de esta manera, se reduce la tasa de error. Es una cuestión digna de estudiar con más detenimiento y con distintas técnicas diferentes a las habituales.

8.2. Interpretabilidad del modelo

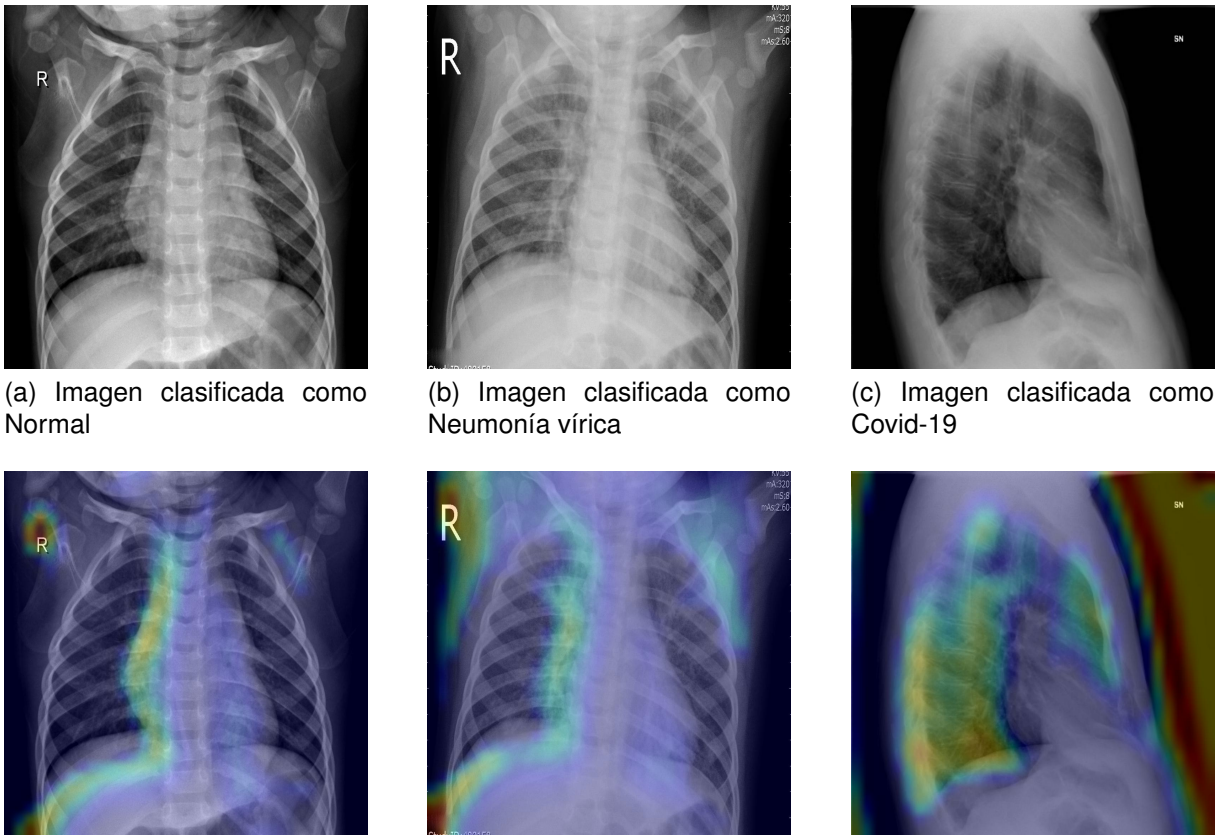
La gran crítica que se le hace a las Redes Neuronales es su falta de interpretabilidad, es decir, entender porqué se hace una determinada predicción. Nosotros utilizaremos la técnica Grad-CAM para tratar de interpretar estas predicciones.

Cada etiqueta proveniente de una imagen, se pasa por el algoritmo Grad-CAM, con el que obtenemos un mapa de calor, en función del gradiente de la activación de la última capa convolucional. De esta manera, el mapa de calor coloreará unas u otras zonas de la imagen. Obtendremos, por tanto, un mapa de temperaturas adaptado a la influencia en la predicción, el cual superpondremos a la imagen inicial para observar directamente qué zonas y píxeles han sido responsables de la clasificación obtenida. Con esta novedosa técnica, podremos explicar a cualquier persona, cómo se ha realizado la predicción fácilmente.

Esta técnica resulta realmente útil para detectar sesgos y ayudar a decidir si confiar, o no, en las predicciones de un algoritmo. En nuestro caso, la ausencia de un experto, que nos indique, si las zonas coloreadas son realmente influyentes en la toma de decisiones,

nos limitamos a mostrar y comentar nuestras ideas y conclusiones desde el punto de vista algorítmico, sin entrar en el apartado médico.

En la Figura 8.1, se muestran tres imágenes originales con su predicción y la imagen superpuesta con el mapa de calor obtenido con Grad-CAM.



(a) Imagen clasificada como Normal

(b) Imagen clasificada como Neumonía vírica

(c) Imagen clasificada como Covid-19

(d) Grad-CAM para la imagen Normal

(e) Grad-CAM para la imagen Neumonia vírica

(f) Grad-CAM para la imagen Covid-19

Figura 8.1: Ejemplos de imágenes clasificadas con su correspondiente Grad-CAM.

En la Figura 8.1, podemos ver tres imágenes del conjunto de prueba, las cuales han sido correctamente etiquetadas y, debajo de estas, vemos su mapa de calor superpuesto en la imagen.

Podemos observar claramente cómo para la imagen clasificada como Covid-19, la activación es mucho mayor. Muchas zonas de la imagen están resaltadas con colores muy fuertes. Para las imágenes con Neumonía y Normal hay menos zonas intensas, si bien es cierto, que la imagen con Neumonía tiene el contorno de los pulmones más azulado. Para la imagen Normal, se concentra en la zona izquierda de la columna vertebral.

Nos causa especial curiosidad, entender porqué las zonas exteriores a los pulmones resultan influyentes en la predicción, ya que son píxeles negros que, a priori, no deberían

contener información. De esta manera, sospechamos que el modelo está sobreajustando a estos ejemplos y utilizando zonas de la imagen sin información para clasificarlos correctamente.

Capítulo 9

Aplicación

Una vez tenemos definidos los modelos que vamos a utilizar, desarrollamos una aplicación que permita a los usuarios, en este caso, sería orientada para médicos, introducir sus propias imágenes y obtener la clasificación de estas, así como su explicación o interpretabilidad de la predicción.

Abordamos este capítulo como un proyecto de desarrollo de *software*, aunque únicamente sea una aplicación web simple, que nos permite acceder a las modelos implementados sin tener que acceder al código. Para esta aplicación nos inspiramos en el trabajo [16].

Queremos destacar, que esta aplicación supone una demostración del buen funcionamiento del modelo y es el culmen del estudio realizado para la clasificación de imágenes. No es un producto desarrollado para fines comerciales, ni para entregar a un cliente.

9.1. Análisis

9.1.1. Requisitos

En este apartado, se describen los requisitos del sistema que se va a desarrollar: requisitos funcionales, no funcionales y de información.

Requisitos funcionales

Definen las funciones del sistema de software o sus componentes, descritas como un conjunto de entradas, comportamientos y salidas. Pueden ser: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que, se supone, el sistema debe cumplir.

Identificador	Nombre	Descripción
RF-01	Selección de imagen	El sistema debe permitir al usuario seleccionar una imagen.
RF-02	Procesado de imagen	El sistema debe ser capaz de procesar la imagen seleccionada.
RF-03	Diagnóstico	El sistema debe ser capaz calcular el diagnóstico a partir de una imagen.
RF-04	Explicación	El sistema debe ser capaz de calcular la explicación para la predicción dada.
RF-05	Mostrar resultados	El sistema debe mostrar el diagnóstico obtenido y la explicación para este.
RF-06	Número de usos	El sistema debe permitir al usuario realizar tantos diagnósticos como desee.

Tabla 9.1: Requisitos funcionales.

Requisitos no funcionales

Los requisitos no funcionales también llamados atributos de calidad, en la ingeniería de software, son aquellos que especifican criterios que pueden usarse para juzgar la operación de un sistema, en lugar de sus comportamientos específicos, ya que estos corresponden a los requisitos funcionales. Dicho con otras palabras: se refieren a los requisitos que describen características de funcionamiento, por eso, suelen denominarse atributos de calidad de un sistema. Son las restricciones o condiciones que impone el cliente al programa que necesita, por ejemplo, el tiempo de entrega del programa, el lenguaje o la cantidad de usuarios. Se muestran en la Tabla 9.2

Requisitos de Información

Describen la información que debe almacenar y gestionar el sistema para dar soporte a los procesos de negocio. Se muestran en la Tabla 9.3

9.1.2. Casos de uso

Utilizamos los diagramas de caso de uso para especificar las acciones y la comunicación entre un usuario y nuestro sistema. Podemos observar el diagrama de casos de uso en la Figura 9.1.

Subir imagen

Identificador	Nombre	Descripción
RNF-01	Accesibilidad	El sistema debe permitir a cualquier usuario hacer uso del mismo.
RNF-02	Formato de imagen	El sistema debe permitir imágenes en varios formatos.
RNF-03	Tiempo	El sistema debe ser capaz de mostrar el resultado en un tiempo inferior a un minuto.
RNF-04	Seguridad	El sistema debe crear una clave para encriptar las cookies.
RNF-05	Plataforma	El sistema se debe desarrollar en Python 3.8.5.

Tabla 9.2: Requisitos no funcionales.

Identificador	Nombre	Descripción
RI-01	Modelos	El sistema debe almacenar los modelos utilizados para llevar a cabo el diagnóstico.
RI-02	Diagnóstico	El sistema debe guardar la imagen durante el proceso de diagnóstico.
RI-03	Resultados	El sistema debe almacenar el diagnóstico y la imagen obtenida en la explicación hasta finalizar la visualización de resultados.

Tabla 9.3: Requisitos de Información.

Nombre e ID del CU	CU-01. Subir imagen
Actor	Usuario
Descripción	El usuario selecciona la imagen de la cual desea obtener su diagnóstico desde su máquina local.
Precondiciones	1. La aplicación esta en proceso de ejecución y escuchando peticiones. 2. El usuario ha accedido a la dirección web donde está desplegada la aplicación.

Postcondiciones	1. La imagen queda almacenada en el sistema.
Flujo normal	1. El usuario introduce la imagen que quiere diagnosticar en sistema. 2. El sistema comprueba la extensión de la imagen. 3. El sistema almacena la imagen. 4. El sistema muestra el nombre de la imagen.
Flujo alternativo	2a. Si la extensión de la imagen no es correcta o no está contemplada en el sistema. El sistema muestra un mensaje de error y vuelve a la pantalla de inicio.

Tabla 9.4: CU-01. Subir imagen.

Diagnosticar

Nombre e ID del CU	CU-02. Diagnosticar
Actor	Usuario
Descripción	El usuario solicita el diagnóstico de la imagen previamente cargada.
Precondiciones	1. El sistema tiene la imagen almacenada.
Postcondiciones	1. El sistema obtiene y muestra los resultados del diagnóstico.
Flujo normal	1. El usuario solicita efectuar el diagnóstico. 2. El sistema realiza el caso de uso <Obtener diagnóstico>. 3. El sistema realiza el caso de uso <Obtener explicación>. 4. El sistema muestra por pantalla los resultados.
Flujo alternativo	2a. Se produce un error al procesar la imagen. El sistema muestra un mensaje de error.

Tabla 9.5: CU-02. Diagnosticar.

Obtener diagnóstico

Nombre e ID del CU	CU-03. Obtener diagnóstico
Actor	Usuario
Descripción	El sistema calcula la predicción para la imagen guardada.
Precondiciones	El sistema debe tener una imagen y el modelo guardados. 1.
Postcondiciones	1. El sistema almacena el resultado de la predicción.
Flujo normal	1. El sistema procesa la imagen. 2. El sistema carga el modelo. 3. El sistema calcula la predicción. 4. El sistema almacena el resultado.
Flujo alternativo	1a. Se produce un error al procesar la imagen. El sistema muestra un mensaje de error. 2a. Se produce un error al cargar el modelo. El sistema muestra un mensaje de error.

Tabla 9.6: CU-03. Obtener diagnóstico.

Obtener explicación

Nombre e ID del CU	CU-04. Obtener explicación
Actor	Usuario
Descripción	El sistema calcula la explicación para una predicción.
Precondiciones	El sistema debe tener una imagen, el modelo y la predicción guardados. 1.
Postcondiciones	1. El sistema almacena el resultado (imagen) de la explicación.

Flujo normal	1. El sistema procesa la imagen. 2. El sistema carga el modelo y la predicción. 3. El sistema calcula la explicación. 4. El sistema almacena el resultado.
Flujo alternativo	1a. Se produce un error al procesar la imagen. El sistema muestra un mensaje de error. 2a. Se produce un error al cargar el modelo o la predicción. El sistema muestra un mensaje de error.

Tabla 9.7: CU-04. Obtener explicación.

9.2. Diseño

En esta sección, se proporcionan los detalles acerca del diseño de la aplicación web desarrollada. Se lleva a cabo logrando el cumplimiento de los requisitos expuestos en la sección de análisis, sección 9.1.

9.2.1. Tecnologías utilizadas

Como ya se describió en la sección 4.1, para el desarrollo de la aplicación de utiliza Flask que es una biblioteca de Python para desarrollo web. Se busca que todas las herramientas sean compatibles con Python, pues es el lenguaje en el que hemos desarrollado los modelos.

Optamos por utilizar el servidor de despliegue Gunicorn, ya que es compatible con Flask y nos facilita el uso y despliegue de la aplicación, así como su ejecución multihilo para atender diferentes peticiones.

Para el desarrollo propio de las vistas, la web, se utiliza HTML y JavaScript. Estos, junto con CSS, nos permiten manipular el aspecto de la aplicación y sus acciones.

9.2.2. Arquitecturas

En esta sección, veremos los patrones de diseño, de acuerdo con [19], empleados en el desarrollo de la aplicación web.

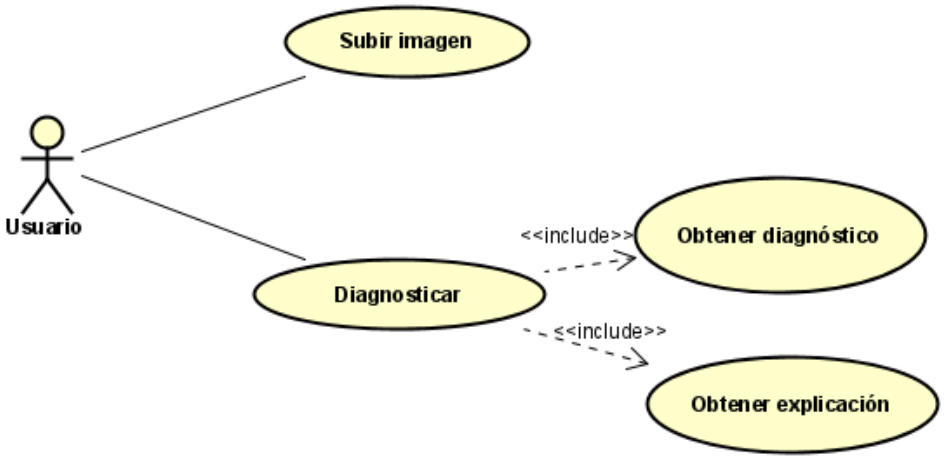


Figura 9.1: Diagrama de casos de uso.

Patrón MVC

El patrón Modelo-Vista-Controlador (MVC) es un estilo de arquitectura de software, que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

Se trata de un modelo muy utilizado, que ha demostrado su validez en todo tipo de aplicaciones y sobre multitud de lenguajes o plataformas de desarrollo [1].

- El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio y sus mecanismos de persistencia.
- La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

Se utilizará para el control de las vistas, documentos HTML, la gestión del modelo o datos y la interacciones o peticiones.

Patrón *Singleton*

Conocido como patrón de única instancia, este patrón de diseño permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Con esto, se garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella [63].

Patrón Fachada

Define una interfaz global de acceso para no acceder directamente a los métodos del modelo. Recibe las peticiones del sistema y delega las responsabilidades en los componentes o subsistemas.

9.2.3. Diagramas de clases

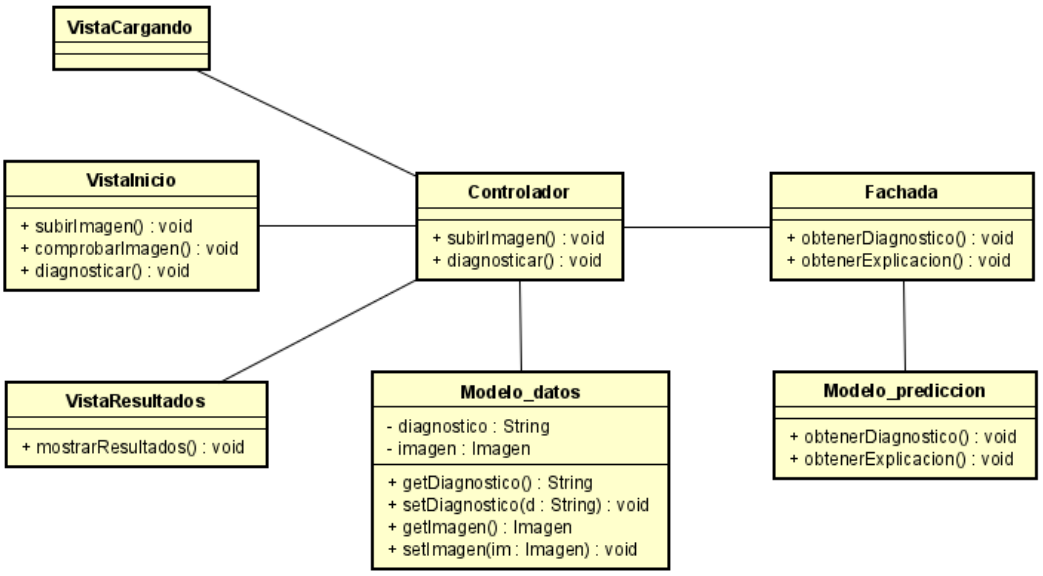


Figura 9.2: Diagrama de clases.

Debemos comentar que, la clase Modelo_datos hace referencia al modelo del MVC donde se almacenan los datos. Por su parte, VistaCargando, VistaInicio y VistaResultados son las vistas que mostrará la aplicación. La clase Controlador es la que gestiona este cambio de vistas, así como la consulta de información del Modelo e interactúa con Fachada, para realizar las operaciones de diagnóstico.

Se emplea la clase Fachada, como indica el patrón fachada, para proporcionar un punto de acceso único al modelo de predicción. Este, a su vez, sigue lo propuesto en el patrón Singleton ya que solo permite crear una instancia de él. El modelo de predicción realizará el diagnóstico, a partir de la imagen, que le envía el controlador.

9.2.4. Diagramas de secuencias

Presentamos los diagramas de secuencias correspondientes a los casos de uso desarrollados en la sección 9.1.2.

CU_01

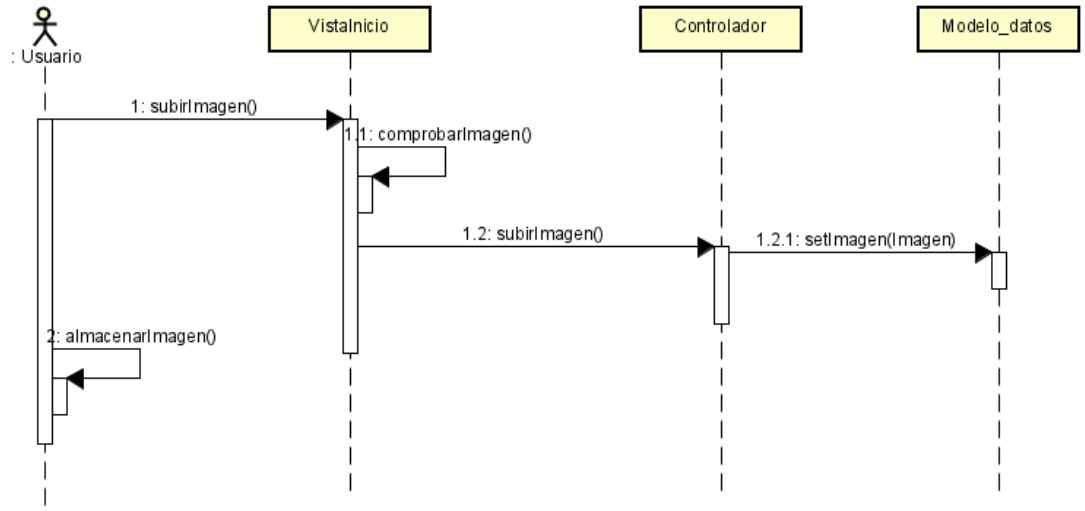


Figura 9.3: Diagrama de secuencias para CU-01 Subir imagen.

CU_02

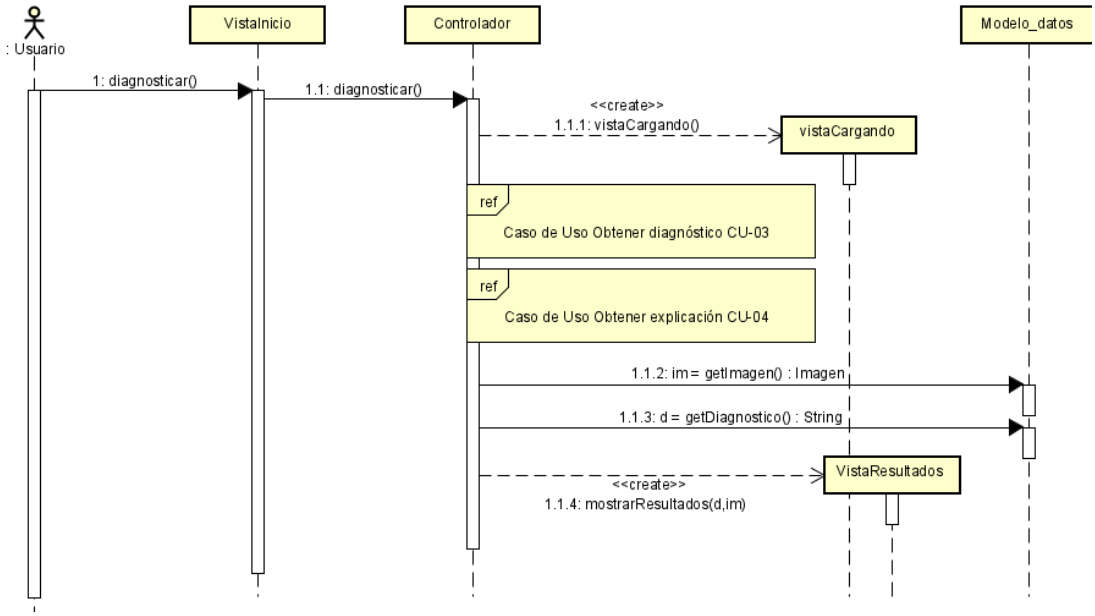


Figura 9.4: Diagrama de secuencias para CU-02 Diagnosticar.

CU_03

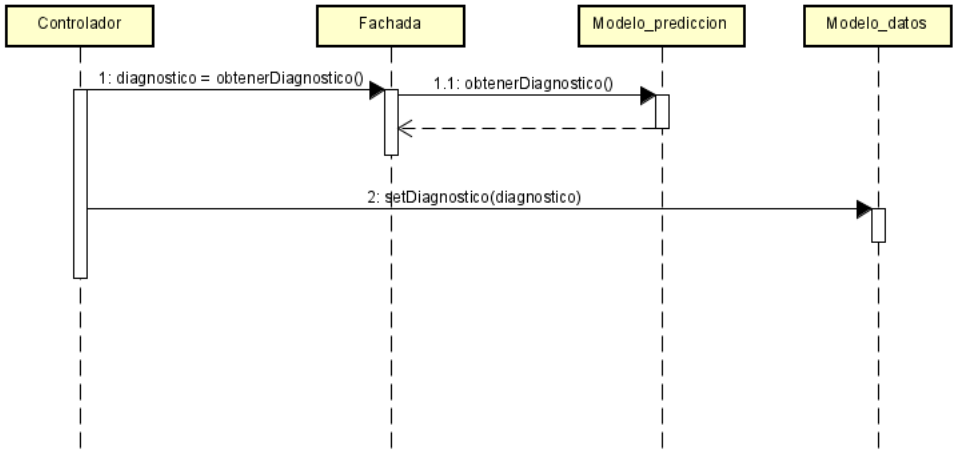


Figura 9.5: Diagrama de secuencias para CU-03 Obtener diagnóstico.

CU_04

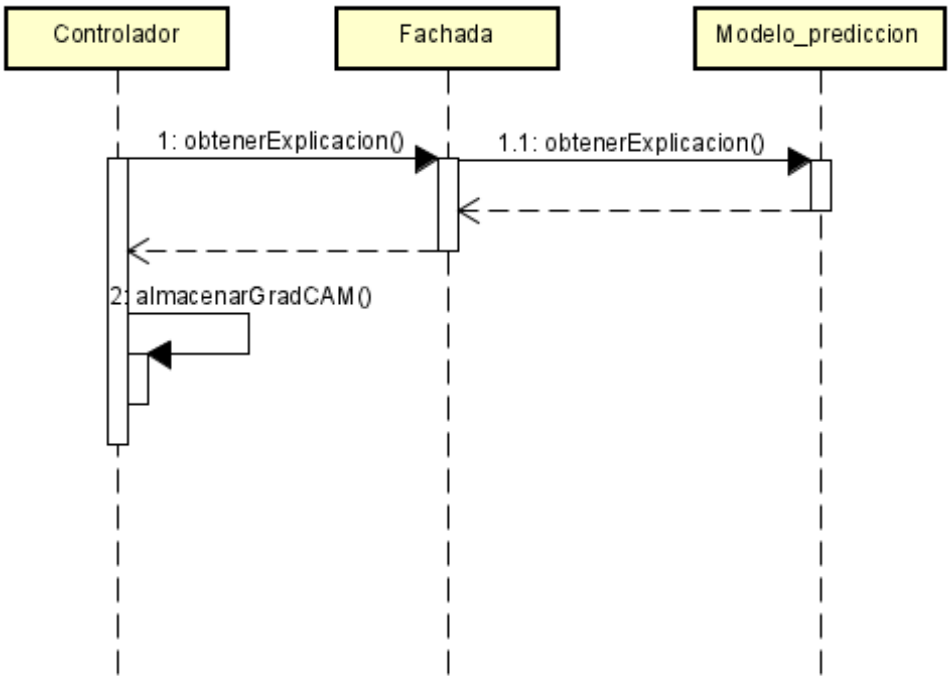


Figura 9.6: Diagrama de secuencias para CU-04 Obtener explicación.

9.2.5. Pruebas

Al tratar con una aplicación sencilla, no se elabora una amplia batería de pruebas. Se evalúan las situaciones habituales y se comprueba su funcionamiento de estas como batería de pruebas. Se comentan brevemente las pruebas realizadas, tanto positivas, como negativas.

- Imágenes o archivos con extensión incorrecta: reciben el mensaje de error.
- Imágenes apropiadas: obtienen el diagnóstico con el curso normal de ejecución.
- No se encuentra el modelo: se obtiene un mensaje de error.

9.2.6. Seguridad

Como el objetivo el lograr una aplicación sencilla, funcional, que dé soporte a los modelos de Aprendizaje Profundo construidos, y no desarrollar una aplicación completa con un proyecto de Ingeniería de Software. No consideramos tratar los temas de seguridad del sistema. Empero, se llevan a cabo las siguientes consideraciones para garantizar el buen funcionamiento del sistema y la buena praxis de la utilización de las bibliotecas:

- Se comprueba la extensión o formato de los archivos cargados en la aplicación. Con esto, garantizamos que los ficheros sean imágenes en uno de los formatos permitidos: jpg, jpeg o png. Puede que la API de Keras soporte más formatos de imágenes, pero solo hemos considerado estos por motivos de obtener garantías.
- Se cifran las sesiones de los usuarios. Se asigna una clave de 10 números y letras generada aleatoriamente al comienzo de la sesión para mantener las sesiones del cliente seguras. Con la clave, se encriptan las *cookies* del cliente para guardarlas en el navegador.

9.2.7. Implementación

Se incluyen, en el CD presentado junto a esta memoria, los scripts necesarios para el correcto funcionamiento de la aplicación. La implementación tiene base en [16]. En la sección B, se explica con detalle el contenido y organización del CD.

9.2.8. Despliegue

Se propone la utilización de contenedores Docker para realizar la migración y despliegue de la aplicación. Docker es una plataforma abierta, gratuita, para desarrollar, enviar y ejecutar aplicaciones [15] que permite separar la aplicación de la infraestructura.

Docker ofrece la posibilidad de empaquetar y ejecutar una aplicación en un entorno poco aislado llamado contenedor. El aislamiento y la seguridad permiten ejecutar muchos

contenedores simultáneamente en un determinado *host*. Estos son ligeros y contienen todo lo necesario para ejecutar la aplicación, por lo que no hay dependencias, en cuanto a instalación se refiere. Podríamos decir que Docker ofrece los mismos beneficios que trabajar en una máquina virtual, sin embargo, funcionan de manera completamente distinta, como ya hemos dicho, Docker contiene en sí mismo o en su empaquetado, contenedor, las aplicaciones y servicios necesarios para el desarrollo y despliegue de la aplicación.

Se pretende empaquetar nuestra aplicación en un contenedor Docker, para facilitar la migrabilidad entre máquinas, la compartición y el despliegue de la aplicación. Sin embargo, por falta de tiempo y errores en la implementación del Docker no se ha podido llevar a cabo.

Capítulo 10

Conclusiones

El desarrollo de este Trabajo de Fin de Grado ha supuesto una puesta en práctica más real de una gran cantidad de contenidos vistos en diferentes asignaturas, en particular:

- Fundamentos de Programación: básico para cualquier proyecto de *software*.
- Programación Orientada a Objetos, para la construcción de clases, como Modelo_predictivo.
- Fundamentos de Sistemas Operativos y Estructuras de Sistemas Operativos, pues hemos modificado las características del sistema, en este caso el área de memoria de intercambio, *swap*.
- Interacción Persona Computador y Diseño y Evaluación de Sistemas Interactivos, para la gestión de sistemas con interacción y aplicaciones, así como el Modelo-Vista-Controlador y el diseño gráfico de la apariencia de las vistas.
- Planificación y Diseño de Sistemas Computacionales y Fundamentos de Ingeniería de *Software*: se destaca la parte de planificación y gestión del proyecto y el diseño del sistema, así como los patrones empleados.
- Fundamentos de Inteligencia Artificial, Ingeniería del Conocimiento, Técnicas de Aprendizaje Automático y Minería de Datos: más relevantes en cuanto al desarrollo del proyecto, pues asientan las bases de los modelos predictivos, su funcionamiento, variedad de algoritmos, métricas, algoritmos de optimización, etc. los cuales ha resultado claves en este proyecto.

Para cumplir con los objetivos del proyecto, se han empleado técnicas que no se han visto durante la docencia o se han tratado de forma superficial. Como ampliación de materia, podemos encontrar los siguientes instrumentos.

- Redes Convolucionales: si bien es cierto que las Redes Convolucionales están en el programa de Minería de Datos, su uso y tiempo dedicado son muy escasos. La profundización en este tipo de Redes Neuronales ha resultado de gran utilidad para llevar a cabo la tarea de clasificación de imágenes que abordamos en este trabajo.

- Grad-CAM: la novedad que supone utilizar técnicas punteras de interpretabilidad de modelos de caja negra, nos resuelve el problema de la desconfianza que existe con las predicciones de las Redes Neuronales. Estas técnicas no se han visto durante los estudios del grado y suponen un campo muy amplio de investigación, a la par que sugerente.

- Aplicación: aunque hemos realizado varios proyectos de Ingeniería de Software, las aplicaciones realizadas hasta ahora, no suponían la integración de un sistema predictivo, como es el caso. Se ha necesitado explorar bibliotecas y opciones de desarrollo, para que la combinación entre los dos sistemas fuese satisfactoria.

Hemos abordado el problema de clasificación de imágenes radiológicas de tórax en la tarea de detección de la enfermedad Covid-19. Para ello, se ha trabajado con técnicas de Aprendizaje Profundo, en particular, las Redes Neuronales Convolucionales.

Hemos tratado y resuelto problemas típicos de la clasificación de imágenes, así como su manejo, preprocesado, almacenamiento... Se ha logrado obtener un algoritmo o modelo predictivo capaz de alcanzar una tasa de acierto del 96.18 % sobre el conjunto de prueba.

Con las técnicas de interpretabilidad implementadas, Grad-CAM, se consigue obtener una intuición sobre cómo se ha clasificado una cierta observación. Esta información puede resultar muy valiosa para detectar sesgos, tanto en los datos, como en el algoritmo, y para fijar el nivel de confianza del mismo. Destacamos que esta información podría haber resultado aún más valiosa, si se contrasta con la experiencia de personal médico especializado en la materia.

Se ha desarrollado una aplicación, en un entorno de desarrollo, que implementa la funcionalidad necesaria para hacer uso del modelo predictivo en tiempo real y obtener los resultados individuales para cada observación. Se han utilizado técnicas de diseño, como patrones, para la elaboración de la aplicación.

La planificación propuesta en un inicio supuso un soporte en la organización del proyecto, sin embargo, la estimación de tiempos resultó diferir ligeramente del tiempo real empleado, pues es el primer reto de grandes dimensiones al que nos enfrentábamos y los trabajos de investigación o estudio poseen incertidumbre, la cual somos incapaces de predecir. Las restricciones de velocidad de cómputo de la máquina utilizada no se tuvieron en cuenta en la planificación y podrían haber supuesto una estimación más realista, tanto en los tiempos, como en los objetivos.

No se ha conseguido el último de los objetivos propuestos al inicio del trabajo, correspondiente a la elaboración de un contenedor Docker para el proyecto, para que este fuese fácil de compartir y de ejecutar, pues la falta de tiempo y experiencia en esta tecnología nos lo ha impedido.

10.1. Trabajo futuro

Se propone como trabajos futuros y posible mejora.

- Uso de Docker para completar los objetivos y tener la aplicación en un entorno aislado.
- Emplear modelos de entrenamiento, ya conocidos y empleados, en otros problemas de clasificación de imágenes, *Transfer Learning*.
- Utilizar metodología experimental más exhaustiva, como validación cruzada, para obtener una estimación del error verdadero con menos varibialidad, sin dependencias de la partición inicial. Esto conllevará un notable aumento del tiempo de cómputo.
- Repetición y ampliación de modelos con máquinas más potentes capaces de llevar a cabo grandes procesos de entrenamiento en, relativamente, pequeñas cantidades de tiempo.
- Ponderar con más importancia la clasificación de Covid-19, para mejorar la precisión en dicha categoría.
- Utilización de técnicas de super-muestreo para dar más representación a clases desbalanceadas como *Synthetic Minority Over-sampling Technique* (SMOTE) [7].
- Desarrollo de una aplicación completa con mayor funcionalidad que pueda ser utilizada en entornos médicos reales.

Parte II

Apéndices

Apéndice A

Manuales de la Aplicación

A.1. Manual de instalación

Se requiere un entorno Python 3.8.5 para ejecutar la aplicación. Las especificaciones de librerías utilizadas, y por ello recomendadas, se encuentran en la sección 4.1. No se proporciona ninguna garantía en un entorno con otras versiones del lenguaje o de sus librerías.

En el CD, podemos encontrar un archivo llamado *requirements.txt* en el que se especifican las librerías que se necesitan instalar para el correcto funcionamiento de la aplicación. Se recomienda instalar las versiones de las librerias en un entorno virtual, como por ejemplo Anaconda, que ha sido el empleado en desarrollo, o directamente desde este fichero utilizando el comando pip.

```
$ pip install -r requirements.txt
```

Para la ejecución de la aplicación debemos situarnos dentro del directorio Aplicación. En él, se debe ejecutar el comando de lanzamiento para una aplicación Flask. directamente desde este fichero utilizando el comando pip.

```
$ python -m flask run
```

Tras lanzar el programa por consola, debemos acceder a un navegador y buscar una de las siguientes direcciones.

1. `localhost:5000`
2. `127.0.0.1:5000`

De esta manera, accedemos al puerto de la máquina local que lanza la aplicación web y ya está lista para su uso.

A.2. Manual de usuario

Una vez se accede a la web de la aplicación encontramos la pantalla de inicio, podemos verla en la Figura A.1.

Figura A.1: Pantalla de inicio.

En ella, debemos subir la imagen que queremos diagnosticar. Seleccionando el botón *Examinar* se despliega un explorador de archivos donde se puede seleccionar la imagen. También, se proporciona la posibilidad de arrastrarla al área de color azul. Si la imagen se carga de forma correcta, podremos observar su nombre en la pantalla, tanto en la zona superior, al lado del botón *Examinar* como en la zona central azul. Podemos observarlo en la Figura A.2.

Figura A.2: Pantalla de inicio con imagen seleccionada.

Tras subir la imagen al sistema, solicitaremos la obtención del diagnóstico utilizando el botón *Diagnosticar*. Con esto, pasamos a una pantalla de carga que se muestra mientras el sistema realiza la predicción, obtiene el diagnóstico y calcula la imagen Grad-CAM. Esta pantalla de carga se muestra en la Figura A.3.

Figura A.3: Pantalla de carga.

La última vista de la aplicación muestra los resultados obtenidos para la imagen seleccionada. En la zona derecha, podemos encontrar la clase predicha, en el ejemplo COVID-19. En la zona central, podemos ver la imagen obtenida con Grad-CAM que nos indica cómo se ha realizado la predicción. En este ejemplo, vemos que encuentra información relevante en la zona de ambos pulmones.

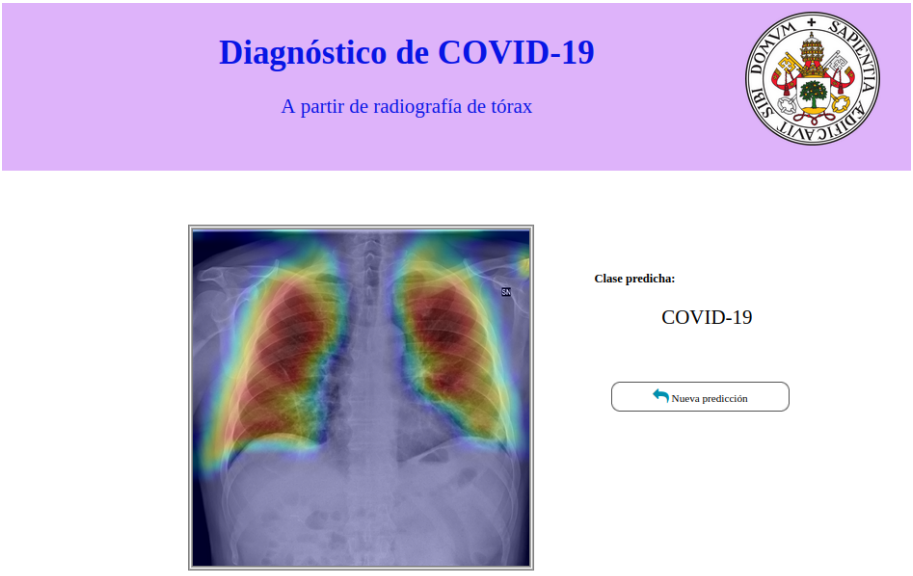


Figura A.4: Pantalla de resultados.

Apéndice B

Contenido del CD

Explicamos el contenido del CD, o repositorio, en este caso, que se entrega junto a la Memoria, junto con el árbol de directorios correspondiente, para facilitar la navegación por los archivos del mismo.

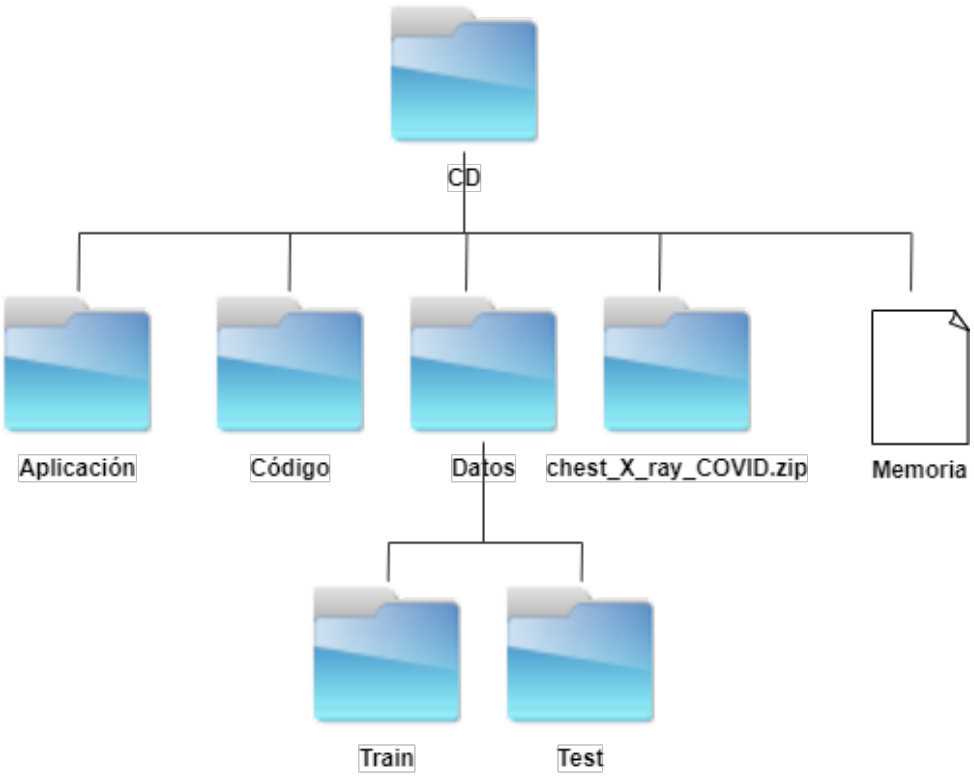


Figura B.1: Estructura de directorios.

En la Figura B.2, podemos ver la estructura de directorios del CD. En el directorio Aplicación encontraremos todas las clases, ficheros y estructura de directorios necesaria para el funcionamiento de la aplicación. En el directorio Código, están los cuadernos de

Python utilizados para el procesamiento de datos y la creación de modelos de Aprendizaje Profundo. Los directorios chest_X_ray.zip y Datos constan de los datos, en el primero los originales y en el segundo estos con la división en conjunto de entrenamiento y prueba. Por último, se encontrará esta memoria.

Los datos no se pueden adjuntar al repositorio por restricciones de almacenamiento. Por tanto, la estructura final del repositorio es la siguiente.

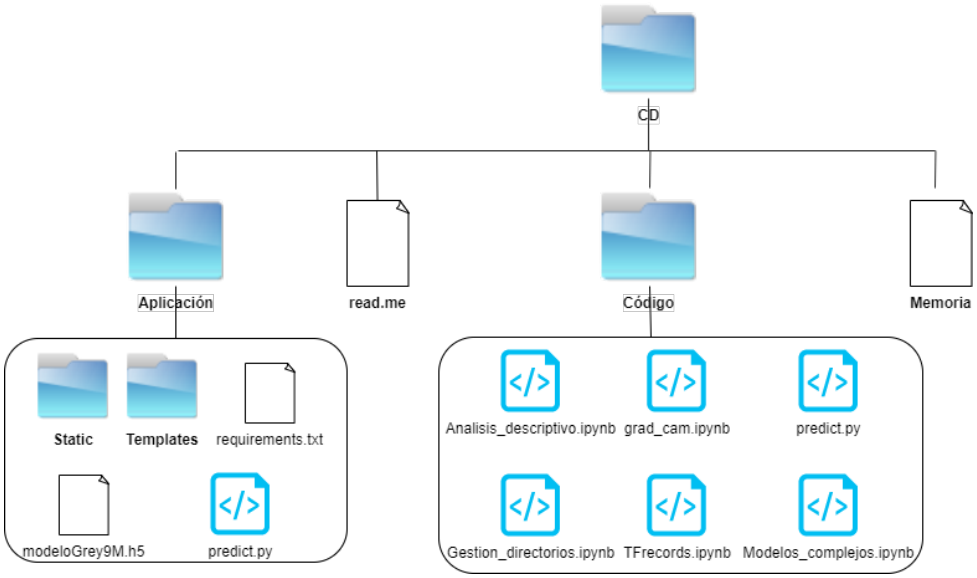


Figura B.2: Estructura de directorios final del repositorio.

En los directorios static y templates, podemos encontrar: los ficheros estáticos de la aplicación (hoja de estilo, imágenes guardadas para predecir...) y las plantillas HTML, respectivamente.

Bibliografía

[1] Universidad de Alicante. *Modelo Vista Controlador (MVC)*. URL: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html (visitado 09-07-2021).

[2] Sai Balaji. *Binary Image classifier CNN using TensorFlow*. 2020. URL: https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697 (visitado 29-06-2021).

[3] Y. Bengio, A. Courville y P. Vincent. "Representation Learning: A Review and New Perspectives". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (ago. de 2013), págs. 1798-1828. DOI: 10.1109/tpami.2013.50. URL: https://doi.org/10.1109/tpami.2013.50.

[4] Jeff Bezanson y col. "Julia: A fresh approach to numerical computing". En: *SIAM review* 59.1 (2017), págs. 65-98.

[5] Y-Lan Boureau y col. "Learning mid-level features for recognition". En: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, págs. 2559-2566.

[6] Joe Cabrera. "Windows vs. Linux: A comparative study". En: *Proposal, Technical Writing, Blinn College, Bryan, TX. Accessed January* 21 (2009), pág. 2019.

[7] Nitesh V Chawla y col. "SMOTE: synthetic minority over-sampling technique". En: *Journal of artificial intelligence research* 16 (2002), págs. 321-357.

[8] Francois Chollet. *Deep learning with Python*. Simon y Schuster, 2017.

[9] François Chollet. *Deep Learning with Python*. Manning, nov. de 2017. ISBN: 9781617294433.

[10] François Chollet. *Grad-CAM class activation visualization*. 2021. URL: https://keras.io/examples/vision/grad_cam/ (visitado 03-07-2021).

[11] Muhammad EH Chowdhury y col. "Can AI help in screening viral and COVID-19 pneumonia?" En: *IEEE Access* 8 (2020), págs. 132665-132676.

[12] Google Cloud. *Procesamiento previo de datos de aprendizaje automático con TensorFlow Transform*. URL: https://cloud.google.com/architecture/data-preprocessing-for-ml-with-tf-transform-pt2?hl=es-419 (visitado 03-07-2021).

[13] A. Delgado. *Inteligencia artificial y minirobots*. Textos universitarios. Ecoe Ediciones, 1998. ISBN: 9789586481557. URL: https://books.google.com.co/books?id=cmoaPwAACAAJ.

[14] Adit Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks*. 2017. URL: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/ (visitado 01-07-2021).

[15] *Docker overview*. URL: https://docs.docker.com/get-started/overview/ (visitado 12-07-2021).

[16] Silvia Duque. *Estudio y Aplicación de Redes Convolucionales a la Clasificación de Imágenes Estáticas*. Universidad de Valladolid: Trabajo de Fin de Grado, 2019.

[17] Forbes. *What IT Needs To Know About The Data Mining Process*. 2015. URL: https://www.forbes.com/sites/metabrown/2015/07/29/what-it-needs-to-know-about-the-data-mining-process/?sh=523106b3515f (visitado 25-06-2021).

[18] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". En: *Biological Cybernetics* 36.4 (abr. de 1980), págs. 193-202. DOI: 10.1007/bf00344251. URL: https://doi.org/10.1007/bf00344251.

[19] Erich Gamma y col. *Elements of reusable object-oriented software*. Vol. 99. Addison-Wesley Reading, Massachusetts, 1995.

[20] Xavier Glorot y Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". En: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Yee Whye Teh y Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, págs. 249-256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[21] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[22] Ian J Goodfellow, Jonathon Shlens y Christian Szegedy. "Explaining and harnessing adversarial examples". En: *arXiv preprint arXiv:1412.6572* (2014).

[23] Michael Grant y Stephen Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. http://cvxr.com/cvx. Mar. de 2014.

[24] Karol Gregor y Yann LeCun. "Learning fast approximations of sparse coding". En: *Proceedings of the 27th international conference on international conference on machine learning*. 2010, págs. 399-406.

[25] Andreas Groll y col. *Hybrid Machine Learning Forecasts for the UEFA EURO 2020*. 2021. arXiv: 2106.05799 [cs.LG].

[26] MK Guruchan. *Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network*. 2020. URL: https://www.upgrad.com/blog/basic-cnn-architecture/#Convolution_Layers (visitado 29-06-2021).

[27] D. H. Hubel y T. N. Wiesel. "Receptive fields of single neurones in the cat's striate cortex". En: *The Journal of Physiology* 148.3 (oct. de 1959), págs. 574-591. DOI: 10.1113/jphysiol.1959.sp006308. URL: https://doi.org/10.1113/jphysiol.1959.sp006308.

[28] Bob Hughes. *Software Project Management 5e*. McGraw Hill, 2009.

[29] *Keras API reference Convd2D layer*. URL: https://keras.io/api/layers/convolution_layers/convolution2d/ (visitado 02-07-2021).

[30] *Keras API reference Dense layer*. URL: https://keras.io/api/layers/core_layers/dense/ (visitado 01-07-2021).

[31] *Keras API reference MaxPooling2D layer*. URL: https://keras.io/api/layers/pooling_layers/max_pooling2d/ (visitado 02-07-2021).

[32] *Keras API reference Model training APIs*. URL: https://keras.io/api/models/model_training_apis/ (visitado 03-07-2021).

[33] *Keras API reference optimizers Adam*.

[34] *Keras API reference optimizers RMSprop*. URL: https://keras.io/api/optimizers/rmsprop/ (visitado 25-06-2021).

[35] *Keras API reference optimizers SGD*. URL: https://keras.io/api/optimizers/sgd/ (visitado 25-06-2021).

[36] *Keras API reference pooling layers*. URL: https://keras.io/api/layers/pooling_layers/ (visitado 21-06-2021).

[37] Diederik P. Kingma y Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[38] Thom Lane. *Multi-Channel Convolutions explained with... MS Excel!* 2018. URL: https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108 (visitado 01-07-2021).

[39] Y. Lecun y col. "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. DOI: 10.1109/5.726791. URL: https://doi.org/10.1109/5.726791.

[40] Yan LeCun. *Training with large minibatches is bad for your health*. 2018. URL: https://twitter.com/ylecun/status/989610208497360896 (visitado 03-07-2021).

[41] Lu Lu. "Dying ReLU and Initialization: Theory and Numerical Examples". En: *Communications in Computational Physics* 28.5 (2020), 1671–1706. ISSN: 1991-7120. DOI: 10.4208/cicp.oa-2020-0165. URL: http://dx.doi.org/10.4208/cicp.OA-2020-0165.

[42] Dominic Masters y Carlo Luschi. *Revisiting Small Batch Training for Deep Neural Networks*. 2018. arXiv: 1804.07612 [cs.LG].

[43] Warren McCulloch y Walter Pitts. "A Logical Calculus of Ideas Immanent in Nervous Activity". En: *Bulletin of Mathematical Biophysics* 5 (1943), págs. 127-147.

[44] Thomas M. Mitchell. *Machine Learning*. 1.ª ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[45] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[46] Ozan Oktay y col. "Evaluation of Deep Learning to Augment Image Guided Radiotherapy for Head and Neck and Prostate Cancers". En: *JAMA* (2020). URL: https://www.microsoft.com/en-us/research/publication/evaluation-of-deep-learning-to-augment-image-guided-radiotherapy-for-head-and-neck-and-prostate-cancers/.

[47] Oscar García-Olalla Olivera. *Redes Neuronales Artificiales*. 2019. URL: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i (visitado 18-06-2021).

[48] Ceyhun Ozgur y col. "MatLab vs. Python vs. R". En: *Journal of Data Science* 15.3 (2017), págs. 355-371.

[49] El País. *El INE seguirá la pista de los móviles de toda España durante ocho días*. 2019. URL: https://elpais.com/economia/2019/10/28/actualidad/1572295148_688318.html (visitado 12-06-2021).

[50] Prajit Ramachandran, Barret Zoph y Quoc V. Le. "Searching for Activation Functions". En: *CoRR* abs/1710.05941 (2017). arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

[51] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". En: *Psychological Review* (1958), págs. 65-386.

[52] David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams. "Learning Representations by Back-propagating Errors". En: *Nature* 323.6088 (1986), págs. 533-536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[53] Stuart Russell y P Norvig. "Inteligencia artificial: un enfoque moderno/Stuart J". En: *Russell y Peter Norvig; traducción [de la 2a ed. en inglés], Juan Manuel Corchado Rodríguez...[et al.]; revisión técnica, Juan Manuel Corchado Rodríguez...[et al.]; coordinación general de la traducción y revisión técnica, Luis Joyanes Aguilar* (2004).

[54] Shadman Sakib y col. "Detection of COVID-19 Disease from Chest X-Ray Images: A Deep Transfer Learning Framework". En: *medRxiv* (2020).

[55] Abhineet Saxena. *Convolutional Neural Networks (CNNs): An Illustrated Explanation*. 2016. URL: https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/ (visitado 01-07-2021).

[56] Ramprasaath R Selvaraju y col. "Grad-cam: Visual explanations from deep networks via gradient-based localization". En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 618-626.

[57] Fathma Siddique, Shadman Sakib y Md. Abu Bakr Siddique. "Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers". En: *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*. IEEE, sep. de 2019. DOI: 10.1109/icaee48663.2019.8975496. URL: https://doi.org/10.1109/icaee48663.2019.8975496.

[58] Akash Srivastava. *The softmax function*. URL: http://akashgit.github.io/2017/03/13/unsaturating_softmax.html (visitado 25-06-2021).

[59] Farhana Sultana, Abu Sufian y Paramartha Dutta. "Advancements in Image Classification using Convolutional Neural Network". En: *CoRR* abs/1905.03288 (2019). arXiv: 1905.03288. URL: http://arxiv.org/abs/1905.03288.

[60] Ilya Sutskever y col. "On the Importance of Initialization and Momentum in Deep Learning". En: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, III–1139–III–1147.

[61] *TensorFlow API tf.data.TFRecordDataset*. URL: https://www.tensorflow.org/api_docs/python/tf/data/TFRecordDataset (visitado 03-07-2021).

[62] Miguel Toquero. *Clasificación de Fallos en Motores de Inducción mediante Redes Neuronales*. Universidad de Valladolid: Trabajo de Fin de Grado, 2021.

[63] Wikipedia. *Singleton pattern*. URL: https://en.wikipedia.org/wiki/Singleton_pattern (visitado 09-07-2021).

[64] Ian H. Witten y col. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations*. 1999.

[65] Pan Zhou y col. *Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning*. 2020. arXiv: 2010.05627 [cs.LG].

[66] Y. Zhou y R. Chellappa. "Computation of optical flow using a neural network". En: *IEEE 1988 International Conference on Neural Networks* (1988), 71-78 vol.2.

Índice alfabético