

Aplicación web para la identificación del tipo de patología cancerígena en pulmones humanos mediante clasificación de imágenes a través de una red neuronal convolucional

Oriol López-Doriga Sagalés

Máster en Bioinformática y Bioestadística
Área 4

Romina Astrid Rebrij

Antoni Pérez Navarro

7/6/21



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](#)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Aplicación web para la identificación del tipo de patología cancerígena en pulmones humanos mediante clasificación de imágenes a través de una red neuronal convolucional</i>
Nombre del autor:	<i>Oriol López-Doriga Sagalés</i>
Nombre del consultor/a:	<i>Romina Astrid Rebrij</i>
Nombre del PRA:	<i>Antoni Pérez Navarro</i>
Fecha de entrega (mm/aaaa):	<i>06/2021</i>
Titulación:	<i>Máster Universitario en Bioinformática y Bioestadística</i>
Área del Trabajo Final:	<i>Computación e Inteligencia Artificial en problemas biológicos y clínicos</i>
Idioma del trabajo:	<i>Castellano</i>
Número de créditos:	<i>15</i>
Palabras clave	<i>Patología cancerígena, aprendizaje profundo, desarrollo web</i>
Resumen del Trabajo	
<p>La finalidad de este trabajo fin de máster es el desarrollo de una aplicación web que permita clasificar imágenes de pulmones humanos con alguno de los tres tipos de tumor siguientes: adenocarcinoma, carcinoma escamoso y tumor benigno.</p> <p>En el contexto de una enfermedad que es una de las más diagnosticadas en España entre los distintos cánceres existentes, se hace imprescindible poder reconocer este tipo de patología y diagnosticarla correctamente de forma eficaz en sus etapas iniciales, puesto que eso permite elegir el correcto tratamiento a seguir por parte del paciente y eso es vital para la superación exitosa de la enfermedad.</p> <p>La elección de la red neuronal convolucional, desarrollada en lenguaje Python mediante la librería Keras, combinado con el framework Flask para este lenguaje, han permitido integrar una aplicación para el diagnóstico del tipo de tumor en imágenes en una web desarrollada en HTML y JavaScript. El</p>	

funcionamiento de esta aplicación consiste en la subida de una imagen a través de un formulario web que devuelve el resultado del diagnóstico y la fiabilidad del resultado obtenido.

Esta herramienta podría servir de apoyo para cualquier profesional, para la toma de decisiones médicas, de libre acceso y que devuelve un resultado determinante. Aunque la red neuronal tiene margen de mejora, se pueden clasificar imágenes con alguno de estos tumores y se obtendrán resultados con un 89,33% de precisión y las líneas de mejora tendrían que ir en relación a conseguir una aplicación que abarque más tipo de clases de tumor y que consiga unos resultados infalibles.

Abstract

The purpose of this end of master project is the development of a web application that allows to classify human lung images that suffer one of the three tumors next: adenocarcinoma, squamous carcinoma or non-malignant tumor.

In the context of an illness that is one of the most diagnosed in Spain between the different existing cancers, it is extremely important to be able to identify and diagnose correctly the type of tumor in its early stages, because that would allow to choose the adequate treatment for the patient from the beginning and consequently be able to overcome it successfully.

The choice of a convolutional neural network written in Python through the framework Keras, combined with another framework like Flask for Python, have allowed me to integrate an application for the diagnosis of the tumor type in lung cancer images inside a webpage made by HTML and JavaScript. The application runs following an upload of an image through a web form that returns the diagnosis result and the confidence of the obtained result.

This tool could be useful for any professional, at making decisions, is access free and brings back a determining result. Despite the the neural network still has margin of improvement, images with one of those lung tumors can be classified with a 89.33% of precision and the ways of improving the application might be related with obtaining an application that can even classify more tumor types and that reaches levels of accuracy and confidence that are almost perfect.

Índice

1	<i>Resumen</i>	2
2	<i>Introducción</i>	4
2.1	Contexto y justificación del Trabajo	4
2.2	Objetivos del Trabajo	4
2.3	Enfoque y método seguido	5
2.4	Planificación del Trabajo	6
2.5	Breve sumario de contribuciones y productos obtenidos	8
2.6	Breve descripción de los otros capítulos de la memoria	9
3	<i>Estado del arte</i>	11
3.1	Cáncer de pulmón en humanos	11
3.2	La inteligencia artificial y el aprendizaje de máquina	14
3.3	Red neuronal artificial	16
3.4	Aprendizaje profundo	17
3.5	Redes neuronales convolucionales	21
3.6	Aprendizaje profundo y cáncer de pulmón	23
4	<i>Metodología</i>	26
4.1	Conjunto de datos (dataset)	26
4.2	Programación con Python	26
4.3	Desarrollo del algoritmo de clasificación	27
4.4	Aplicación web con Python Flask, HTML y JavaScript	33
4.5	Alojamiento web en Heroku	34
5	<i>Resultados</i>	35
6	<i>Discusión</i>	39
7	<i>Conclusiones</i>	40
7.1	Conclusiones	40
7.2	Líneas de futuro	40
7.3	Seguimiento de la planificación	41
8	<i>Glosario</i>	42
9	<i>Bibliografía</i>	44
	<i>Anexos</i>	47

Lista de figuras

Figura 1. Diagrama de gantt	8
Figura 2. Adenocarcinoma. Tipos histológicos más frecuentes. A: Patrón lepídico que muestra revestimiento de los alvéolos por neumocitos neoplásicos (40X). B: Patrón acinar medianamente diferenciado donde se advierte estructura tubular (40X). C: Patrón papilar, donde se observan las proyecciones papilares intraluminales (10X). D: Patrón sólido que no presenta túbulos, ni estructuras papilares o patrón lepídico (40X). [10]	12
Figura 3. Adenocarcinoma. Tipos histológicos más frecuentes. A: Adenocarcinoma invasivo mucinoso, que evidencia mucina intracelular (10X). B: Imagen a mayor aumento que muestra en detalle la mucina intracitoplasmática. C: Adenocarcinoma papilar con expresión de TTF1 nuclear (10X). D: El mismo caso anterior, expresando NAPSIN-A (40X). [10]	13
Figura 4. A: Carcinoma de células escamosas que muestra foco de queratinización (40X). B: El mismo caso anterior, que muestra expresión de P40. [10]	14
Figura 5. Esquema del proceso de aprendizaje de máquina.[5].....	15
Figura 6. Relación entre una neurona biológica (izquierda) y una neurona artificial (derecha). [12].....	16
Figura 7. Ejemplo de red neuronal artificial profunda, con las distintas capas de entrada, escondidas y de salida.[13].....	17
Figura 8. Diagrama de flujo de las capas que las características de una imagen atraviesan en una red neuronal convolucional, desde las capas de convolución, hasta las max-pooling que finalizan en tres capas de conexión completa para obtener el resultado de la característica final.....	18
Figura 9. Diagrama de flujo de las capas que atraviesan las características de una imagen en una red completamente convolucional. La flecha azul muestra la conexión de salto donde el tensor de salida y de la segunda capa de convolución es concatenado al tensor de salida de la última capa convolucional a través de la capa de deconvolución. La k denota el número de clases posibles a la que el pixel puede pertenecer.....	19
Figura 10. Un AE típico, que tiene una estructura simple, que generalmente tiene 3 capas: una capa de entrada, una capa escondida y una capa de salida.[14]	20
Figura 11. Ajuste de una imagen mediante una red neuronal de autocodificación (AE). [15]	20
Figura 12. Los distintos pasos de una DBN, que van consiguiendo una mejor representación (h) de las características a medida que se van apilando las RBM.	21
Figura 13. Las imágenes pueden ser despiezadas en capas de convolución en pequeños patrones locales, como ejes, texturas, etc. [8]	21
Figura 14. Ejemplo de jerarquía de las características de cada parte de un gato reconocidas por una red neuronal convolucional. [8].....	22
Figura 15. Diagrama de flujo de la convolución. [10].....	23
Figura 16. Esquema del workflow seguido para la creación del algoritmo de clasificación de imágenes de pulmón con Python y la librería Keras.....	27

Figura 17. Arquitectura de la red neuronal creada para este proyecto.[13]..	28
Figura 18. Evaluación del modelo definitivo gear classifier 4.	30
Figura 19. Evaluación del modelo lung_2.....	32
Figura 20. Evaluación del modelo lung_3.....	32
Figura 21. Evaluación del modelo check_lung.	33
Figura 22. Interacción entre las distintas partes de la aplicación web.....	34
Figura 23. Página principal de la aplicación web.....	35
Figura 24. Desplegable con el menú de opciones de la web.	36
Figura 25. Apartado ¿Cómo funciona? de la aplicación web.	36
Figura 26. Apartado Tipos de cáncer de pulmón de la aplicación web.	37
Figura 27. Parte de la página web de “Subir imagen” con el formulario de envío de la imagen que se desea clasificar.	37
Figura 28. Página del resultado obtenido de la clasificación de la imagen con la aplicación integrada en la web.	38

Lista de tablas

Tabla 1. Timing previsto para las tareas del proyecto.....	6
Tabla 2. Resumen de algunos proyectos realizados con aprendizaje profundo para la detección de cáncer de pulmón basado en imágenes.	24
Tabla 3. Parámetros del aprendizaje de la red neuronal.	29
Tabla 4. Comparativa de los distintos modelos creados.....	31
Tabla 5. Precisión alcanzada por la red neuronal convolucional.	35

1 Resumen

El cáncer de pulmón es la enfermedad tumorosa más diagnosticada en España [1], pero no siempre se ha clasificado correctamente, entre los profesionales médicos, el tipo de cáncer sufrido por el paciente. Gran parte de este posible error viene dado por las imágenes de los distintos tejidos pulmonares según el tipo de cáncer, que pueden ser imágenes muy abstractas y difíciles de diferenciar para el ojo humano. [2]

La inteligencia artificial, a través del aprendizaje profundo, ha tenido como uno de sus retos en la última década, poder desarrollar herramientas para ayudar a los profesionales médicos a diagnosticar el tipo de tumor mediante la clasificación de estas imágenes.

Aunque aún no existe ningún método infalible, parece que las redes neuronales convolucionales están siendo unas de las más utilizadas para desarrollar este tipo de proyectos y existen ya proyectos iniciados por distintos grupos de investigadores profesionales dedicados únicamente a conseguir una red neuronal que permita clasificar imágenes de distintos tumores con precisiones muy elevadas. [14]

En este proyecto, se pretende construir una red neuronal convolucional mediante lenguaje Python, que permita clasificar imágenes de tres tipos de tumor de cáncer pulmonar en humanos: adenocarcinoma, carcinoma escamoso y tumor benigno.

Además, se ha elaborado una interfaz web mediante Python Flask para que cualquier persona interesada en la clasificación de imágenes de este estilo, pueda subir la suya a la aplicación para que le devuelva el resultado de la clasificación de su imagen.

Con la aplicación creada, se han obtenido resultados de clasificación de imágenes con alguno de estos tres tipos de tumor con un 89,5% de precisión.

La idea y objetivos de este proyecto han sido realizados de forma satisfactoria, pero para que su desarrollo pueda alcanzar los mínimos de precisión para servir profesionalmente en la realidad, se deberían estudiar más metodologías y tipos de redes. También otros parámetros requieren de un más profundo estudio y determinación, tal como la arquitectura de la red a aplicar o la cantidad de imágenes dedicadas para entrenamiento, validación o testeo, para evitar que se produzca un *overfit/underfit*. Al final, la dedicación y el empleo a fondo de estas cuestiones, debería poder conseguir una estructura que permita

clasificar perfectamente todo tipo de imágenes y tener un porcentaje de error que se acerque al cero.

Aunque existen trabajos mucho más profundos y complejos en esta materia, llevados a cabo por investigadores profesionales, hoy en día no hay muchas redes neuronales convolucionales que clasifiquen imágenes de estos tres tipos de tumor en pulmones humanos y, mucho menos, disponibles en Open Source. Por lo tanto, este trabajo es, por una parte, un inicio de un posible trabajo a desarrollar más profundamente por investigadores interesados en el tema y, por otra parte, puede ser un ejemplo de red neuronal a utilizar para otros fines y que además demuestra que la clasificación de las imágenes de este tipo de tumores con buena precisión es posible. Este trabajo implica un gran aporte porque no existen muchas aplicaciones web de este tipo y menos para cáncer de pulmón.

2 Introducción

2.1 Contexto y justificación del Trabajo

En cuanto al cáncer de pulmón, las principales decisiones con el tratamiento a seguir, se toman al distinguir claramente los diferentes tipos de carcinoma y por eso es fundamental un diagnóstico preciso de la enfermedad [1]. Actualmente, aún persiste el error de pasar por alto la detección de cáncer de pulmón, siendo el error de observación humana de imágenes y toma de decisión del tipo de cáncer de los más significantes [2].

La Inteligencia Artificial puede ser de gran ayuda para paliar estas deficiencias, tal y como se ha demostrado en otros estudios biomédicos. Gracias a un método de clasificación por aprendizaje profundo, el diagnóstico con éxito de cáncer de pulmón en sus etapas de inicio fue del 94% según el algoritmo por aprendizaje profundo de Etemadi et al., superando a 6 médicos veteranos en el diagnóstico de esta enfermedad [3].

En este trabajo, se pretende crear un algoritmo que permita identificar, a través de redes neuronales por aprendizaje profundo, las diferentes imágenes relacionadas con un Carcinoma escamoso, un Adenocarcinoma pulmonar o un tumor benigno en pulmones humanos, a través de imágenes de tejido pulmonar humano. Para ello, se usarán los datos e imágenes facilitadas por un grupo de investigadores de la Cornell University de Nueva York, con su proyecto Lung and Colon Cancer Hispathological Image Dataset [4],. De este modo, el experto en medicina podría tener una herramienta que le sirva de soporte para tomar la decisión médica en cuanto al diagnóstico y tratamiento a seguir.

2.2 Objetivos del Trabajo

Objetivo general:

Clasificar imágenes de tejido pulmonar humano afectados por alguno de los tres tipos de tumor siguientes: Adenocarcinoma pulmonar, Carcinoma escamoso o tumor benigno utilizando una red neuronal convolucional.

Objetivos específicos:

- Elaborar un algoritmo en lenguaje Python que permita entrenar una red neuronal convolucional para la clasificación de imágenes de tejido pulmonar y reconocer de qué tumor se trata con un nivel de precisión superior al 70% en la identificación de los tres tipos de tumor.
- Desarrollar una interfaz web en lenguaje HTML/JavaScript que incluya este algoritmo de forma subyacente que permita a un usuario poder

clasificar imágenes de tejido pulmonar afectados por uno de los tumores mencionados.

2.3 Enfoque y método seguido

Actualmente, en el área del aprendizaje de máquina o aprendizaje profundo, existen varias técnicas que permiten predecir un valor a través de la clasificación de un conjunto de muestras por entrenamiento. Las técnicas que usan un modelo predictivo y lo entrena para luego analizar una muestra similar y lo clasifican dentro de las categorías identificadas en el modelo, reciben el nombre de técnicas de clasificación mediante aprendizaje automático, que es la metodología que se va a seguir en este proyecto.

Dentro de las técnicas de clasificación, existen varias entre las que destacan la kNN o vecinos más cercanos, la naive Bayes, los árboles de decisión o los modelos Hidden Markov. Además, existen técnicas como las redes neuronales y las máquinas de soporte vectorial que son duales y además de la clasificación, permiten realizar predicciones numéricas. [5]

Para este proyecto, se ha elegido, por haber demostrado ser eficientes al trabajar con grandes cantidades de datos y en procesos de identificación de imágenes, simplificando la extracción de características previos de forma intrínseca, además de una fiabilidad contrastada [6], el método de clasificación de imágenes mediante una red neuronal convolucional.

Aunque lenguajes de programación en informática hay varios, el nivel de conocimiento del autor de este proyecto reduce el abanico de opciones a elegir a dos, R o Python.

A priori, el lenguaje en R ha sido el más utilizado en este máster y seguramente el que tenga más dominio por parte del autor, pero la importancia de Python en el mundo informático cada vez es mayor [7], con lo cual, en vistas al futuro, parece el más adecuado para poder ser integrado en proyectos más ambiciosos y por lo tanto este proyecto se desarrollará mediante Python íntegramente.

Por otra parte, la existencia de documentación y paquetería enfocada a la creación de estas redes para procesamiento de imágenes, como las librerías Keras de Tensorflow o MXNet, también ha sido decisivo a la hora de elegir este lenguaje para el desarrollo del proyecto. Para el proyecto se ha optado por usar Keras, por la facilidad de su uso, su flexibilidad y su soporte en Python, lo cual la hacen la más utilizada para los que trabajan con redes neuronales en Python.[8]

Para el desarrollo de la interfaz web, además, la existencia de paquetes como Django o Flask para Python [9], permiten crear una web desarrollada en

HTML y JavaScript pero servida por Python, que además es el mismo lenguaje del algoritmo. Para este trabajo se ha elegido Flask por su simplicidad en interfaces web que no son muy extensas [9]. Como cada vez son más los servidores web que trabajan con el lenguaje Python para servir interfaces web, no resulta difícil conseguir un alojamiento en alguna de las posibilidades existentes, que, en principio, Heroku, ya que permite subir una interfaz web de forma gratuita siempre que no se superen los 512 MB de espacio ocupado en la nube. Las demás opciones no disponen de Python o de espacio suficiente y de forma gratuita.

El entrenamiento de la red neuronal se llevó a cabo mediante las imágenes facilitadas por un grupo de investigadores de la Cornell University de Nueva York, con su proyecto Lung and Colon Cancer Hispathological Image Dataset [4], que contiene 5000 imágenes de 768 x 768 píxeles de cada uno de los tres tipos de tumor a clasificar.

2.4 Planificación del Trabajo

Recursos necesarios

Los recursos necesarios para realizar el trabajo son los siguientes:

- Ordenador con tarjeta gráfica Radeon Pro 560X 4 GB.
- Sistema operativo macOS Big Sur con procesador 3 GHz Intel Core i5 de seis núcleos.
- Software Miniconda3 versión py38_4.9.2 para ambiente virtual creado con la paquetería y el lenguaje Python en versión 3.8.8 totalmente integrados
- La paquetería requerida con sus versiones se puede consultar en el archivo anexo: requirements.txt.
- Repositorio GitHub con el código y archivos de la web.
- Cuenta con alojamiento web Heroku.

1.1. Tareas

Tabla 1. Timing previsto para las tareas del proyecto.

Tarea	Fecha inicio	Fecha final
Definir contenidos TFM	01/02/2021	17/02/2021
Elaborar propuesta inicial (PEC0)	17/02/2021	01/03/2021
Proponer un plan de trabajo (PEC1)	01/03/2021	16/03/2021
Configuración del ambiente virtual	10/03/2021	16/03/2021
Elaborar el documento de desarrollo de trabajo (PEC2)	17/03/2021	19/04/2021
Redacción de la memoria	17/03/2021	08/06/2021

Tarea	Fecha inicio	Fecha final
Desarrollar un algoritmo para crear una red neuronal	17/03/2021	24/03/2021
Preprocesamiento de imágenes	25/03/2021	29/03/2021
Crear el modelo	30/03/2021	01/04/2021
Entrenar el modelo	02/04/2021	04/04/2021
Validar el modelo	05/04/2021	08/04/2021
Testear la red neuronal	09/04/2021	13/04/2021
Mejorar la red neuronal	14/04/2021	19/04/2021
Elaborar el documento de desarrollo del trabajo (PEC3)	20/04/2021	17/05/2021
Diseño inicial de la web: creación del HTML	20/04/2021	23/04/2021
Incorporar módulos de JavaScript a la web	24/04/2021	25/04/2021
Desarrollar el servidor web mediante Python Flask	26/04/2021	29/04/2021
Integrar la red neuronal con la web y el servidor web	30/04/2021	02/05/2021
Servir la web localmente para su testeo	03/05/2021	06/05/2021
Crear un repositorio GitHub con el código de la web	07/05/2021	08/05/2021
Alojar la web en Heroku	09/05/2021	11/05/2021
Realizar las pruebas de la interfaz web	12/05/2021	13/05/2021
Mejorar las prestaciones de la interfaz web	14/05/2021	15/05/2021
Servir la web definitiva	16/05/2021	17/05/2021
Analizar los resultados obtenidos y redactar la memoria (PEC4)	18/05/2021	08/06/2021
Elaborar la presentación (PEC5a)	10/06/2021	13/06/2021
Realizar la defensa pública del trabajo (PEC5b)	16/06/2021	23/06/2021

1.2. Calendario

Gantt Chart

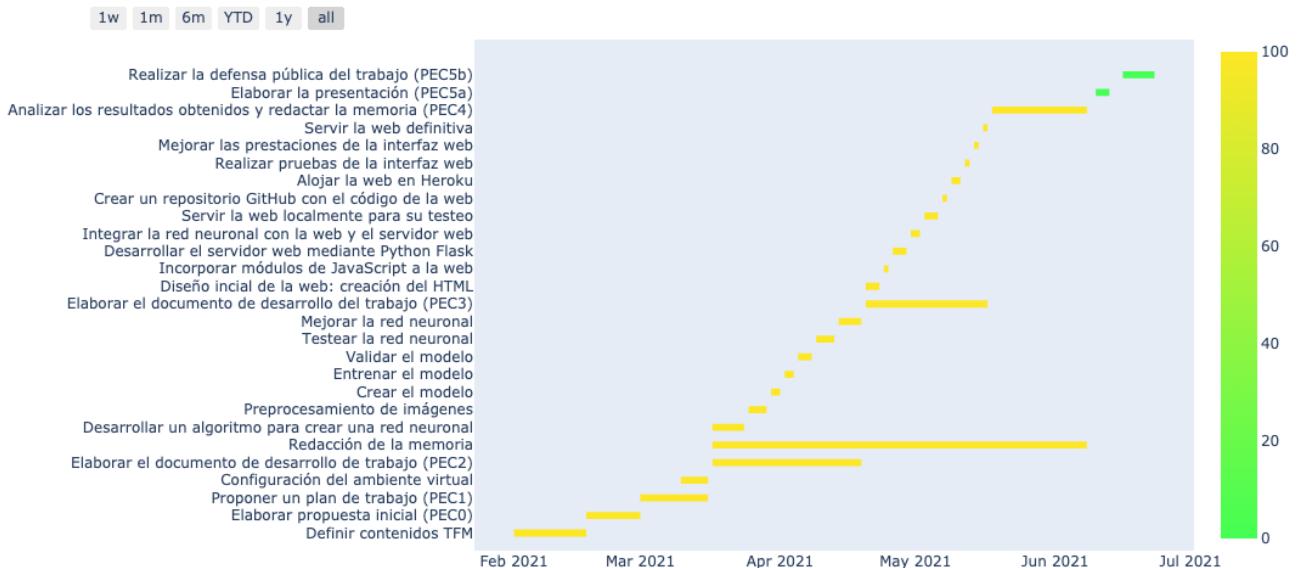


Figura 1. Diagrama de gantt

El diagrama de Gantt muestra las tareas ejecutadas, siendo color verde en estado inicial y amarillo en estado acabado.

2.5 Breve resumen de contribuciones y productos obtenidos

Plan de trabajo

- Documento que contiene la metodología de trabajo y de planificación de proyectos de clasificación de imágenes a seguir.

Memoria

- Documento que integra la descripción general, metodología y todos los resultados obtenidos del estudio.

Producto

- Interfaz web que integra la red neuronal para poder servir a usuarios que quieran identificar el tipo de tumor de sus imágenes con cáncer de pulmón (<https://dcapi2.herokuapp.com>).
- Código fuente de desarrollo para permitir su réplica, alojado en un repositorio GitHub (<https://github.com/costero-e/dcapi2>).

Presentación virtual

- Presentación audiovisual que permita recoger todos los hitos conseguidos.

2.6 Breve descripción de los otros capítulos de la memoria

3. Estado del arte

En este apartado se pretende poner en conocimiento del lector el estado en que se encuentran las distintas materias en las que se basa este proyecto.

En este capítulo se explicará cuál es la situación actual en el mundo y en España del cáncer de pulmón y de los distintos tumores existentes, profundizando en los que van a ser objeto de estudio de este proyecto.

Se pretende hacer una pequeña introducción a la inteligencia artificial y el aprendizaje de máquina, explicando qué son y para qué sirven, enumerando los distintos tipos a grandes rasgos de aprendizaje de máquina existentes.

El objetivo de este capítulo es explicar en qué consisten las redes neuronales artificiales, el motivo de su nombre y su base científica.

También, se enumeran y explican los distintos tipos de redes neuronales artificiales que usan técnicas de aprendizaje profundo.

Por último, se da a conocer los distintos trabajos realizados sobre la materia de este proyecto por parte de investigadores reconocidos y se hace una enumeración de las características de las redes empleadas en ellos, así como su grado de éxito.

4. Metodología

En este apartado el objetivo es dar a conocer los distintos pasos que se han seguido para dar a luz este proyecto.

Para empezar, se explica cuál ha sido el origen de los datos empleados para poder desarrollar la red neuronal convolucional de este proyecto.

Posteriormente, se detallan los distintos pasos seguidos para crear la aplicación en Python que permite identificar y clasificar las imágenes con cáncer de pulmón.

También, se detallan los distintos pasos a seguir para crear la red neuronal y la estructura informática a crear para su entrenamiento, validación y testeo y se indica cuál ha sido la red neuronal convolucional creada y se dan detalles sobre su creación y la arquitectura de la misma.

Además, se explican los distintos pasos que se han seguido para crear la aplicación web e integrar la aplicación en Python de reconocimiento de imágenes con Python Flask, HTML y JavaScript.

Por último, se describe cómo se ha llevado a cabo el alojamiento web o de la aplicación web creada para este proyecto en el servicio webhosting de Heroku.

5. Resultados

Aquí se da a conocer el resultado de la aplicación web, cómo acceder a ella y su funcionamiento.

6. Discusión

En esta parte del trabajo se dan a conocer las posibles mejoras de la aplicación creada o la opinión sobre los resultados obtenidos.

7. Conclusiones

Aquí se hará un resumen final sobre los hitos alcanzados y la relación de satisfacción respecto a los objetivos iniciales del trabajo.

8. Glosario

Aquí se describen las palabras principales y necesarias para entender la memoria del trabajo.

9. Bibliografía

Relación de las distintas fuentes consultadas para la elaboración del proyecto.

Anexos

Documentos generados durante el trabajo para consulta adicional.

3 Estado del arte

3.1 Cáncer de pulmón en humanos

En España cada año son diagnosticados 29.638 nuevos casos de cáncer de pulmón (cifras en 2020), siendo el tercer tumor en frecuencia tanto en varones (con 21.847 casos) como en mujeres (con 7.791 casos). El cáncer de pulmón es la primera causa de muerte por cáncer, estimándose más de 22.000 fallecimientos por esta causa cada año. [1]

El cáncer de pulmón ha experimentado una serie de cambios epidemiológicos en las últimas décadas. Ha disminuido la frecuencia de carcinomas fuertemente asociados al tabaco (carcinoma escamosos y carcinoma de células pequeñas), y se han incrementado los adenocarcinomas no asociados a este elemento nocivo, de forma importante, en la población femenina, lo que está ligado con mutaciones genéticas descubiertas en los últimos años. [10]

Dado el importante y rápido avance del conocimiento de los mecanismos genéticos y moleculares en la patogénesis del cáncer de pulmón y su gran relevancia en el tratamiento de esta enfermedad, reflejado en diversas y múltiples publicaciones y reuniones académicas desde 2004, la OMS se propuso hacer una revisión de la clasificación sobre todo en relación con adenocarcinoma. Tres sociedades auspiciaron esta revisión, la ASLC (Association for the Study of Lung Cancer), la ERS (European Respiratory Society) y la ATS (American Thoracic Society). El equipo de investigadores estuvo liderado por William D. Travis, Elisabeth Brambilla y Masayuqui Noguchi (27). [10]

La clasificación actual de estos dos carcinomas, según la OMS, hecha en 2015, es la siguiente [10]:

- ADENOCARCINOMA

Neoplasia maligna epitelial con diferenciación glandular (figuras 1 y 2).

- a. Lepídico: constituido por neumocitos tipo II. Crece a lo largo de la superficie de las paredes alveolares, con área invasiva de más de 5 mm.
- b. Acinar: estructura glandular con lumen central rodeado por células tumorales.
- c. Papilar: crecimiento papilar de células neoplásicas glandulares a lo largo de un núcleo fibrovascular.
- d. Micropapilar: crecimiento en pequeños nidos papilares sin núcleo fibrovascular.
- e. Sólido: patrón predominante sin evidencia de patrón lipídico, acinar, papilar o micropapilar. Si el patrón sólido es del 100 %, deben haber, al

menos, 5 o más células productoras de mucina por cada dos campos de alto poder, comprobadas con tinción de histoquímica.

- f. Invasivo mucinoso: corresponde al antes denominado BAC mucinoso con morfología columnar o de células caliciformes con abundante mucina intracitoplasmática. Además del patrón lipídico, también puede presentarse con otros patrones.
- g. Coloide: muestra abundante mucina reemplazando los espacios aéreos.
- h. Fetal: estructura histológica semejante a tejido pulmonar fetal. Puede ser de alto o bajo grado.
- i. Entérico: estructura histológica semejante al adenocarcinoma colorrectal.
- j. Adenocarcinoma mínimamente invasivo: Adenocarcinoma solitario, de tamaño igual o menor de 3 cm, con patrón lipídico, no mucinoso predominante y con invasión de hasta 5 mm en dimensión máxima.
- k. Lesiones preinvasivas: Hiperplasia adenomatosa atípica: proliferación atípica localizada de neumocitos tipo II o células de Clara, de hasta 0,5 cm en dimensión máxima.
- l. Adenocarcinoma *in situ*: adenocarcinoma localizado, usualmente no mucinoso, de hasta 3 cm en dimensión máxima, que crece a lo largo de estructuras alveolares preexistentes, en un patrón lipídico puro, sin invasión estromal ni vascular.

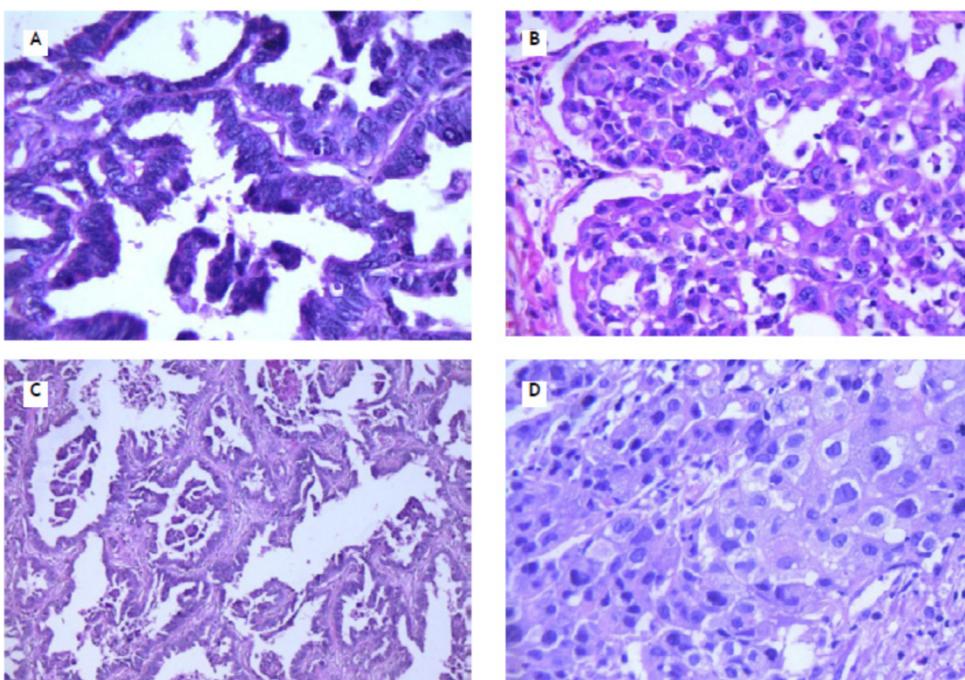


Figura 2. Adenocarcinoma. Tipos histológicos más frecuentes. **A:** Patrón lepídico que muestra revestimiento de los alvéolos por neumocitos neoplásicos (40X). **B:** Patrón acinar medianamente diferenciado donde se advierte estructura tubular (40X). **C:** Patrón papilar, donde se observan las proyecciones papilares intraluminales (10X). **D:** Patrón sólido que no presenta túbulos, ni estructuras papilares o patrón lepídico (40X). [10]

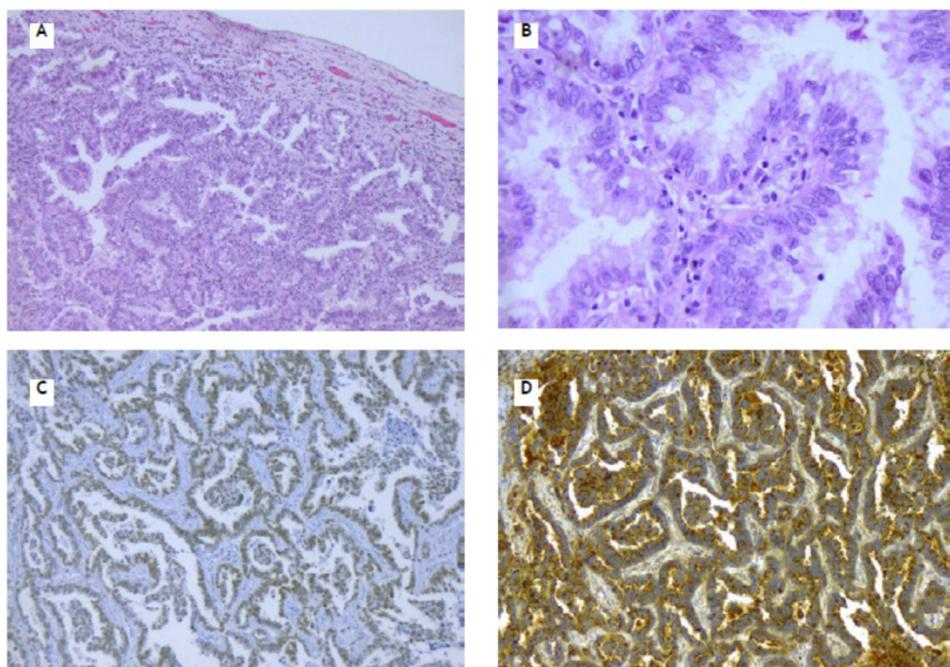


Figura 3. Adenocarcinoma. Tipos histológicos más frecuentes. **A:** Adenocarcinoma invasivo mucinoso, que evidencia mucina intracelular (10X). **B:** Imagen a mayor aumento que muestra en detalle la mucina intracitoplasmática. **C:** Adenocarcinoma papilar con expresión de TTF1 nuclear (10X). **D:** El mismo caso anterior, expresando NAPSIN-A (40X). [10]

- CARCINOMA DE CÉLULAS ESCAMOSAS

Neoplasia epitelial con presencia de queratinización o puentes intercelulares o con morfología indiferenciada, pero con marcadores de IHQ que indican diferenciación escamosa (figura 3).

- a. Patrón queratinizante: contiene abundantes células polimórficas con citoplasmas densos, núcleos irregulares y escamas queratinizadas.
- b. Patrón no queratinizante: aparece como nidos irregulares y dentados de células poligonales hinchadas que invaden el estroma cervical. Puede haber disqueratosis y puentes intercelulares. El polimorfismo celular y nuclear es más obvio y las figuras mitóticas son muy numerosas.
- c. Basaloide: neoplasia poco diferenciada, con arquitectura lobular, con periferia en empalizada y pérdida de morfología escamosa, pero con marcadores de IHQ que indican dicho linaje. Puede haber casos con morfología escamosa queratinizante o no queratinizante, pero deben tener más de 50% de componente escamoso. Son de mal pronóstico.
- d. Lesión preinvasiva: Carcinoma escamoso *in situ*: neoplasia de células escamosas, no invasiva, originada en el epitelio bronquial y que se origina en lesiones displásicas previas.

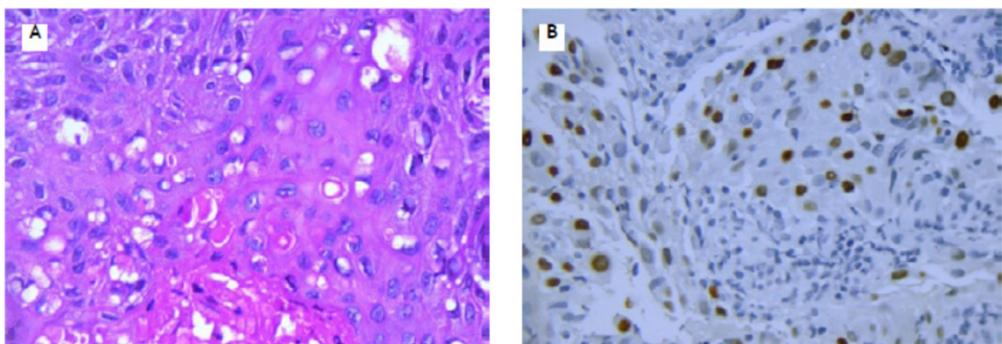


Figura 4. **A:** Carcinoma de células escamosas que muestra foco de queratinización (40X). **B:** El mismo caso anterior, que muestra expresión de P40. [10]

3.2 La inteligencia artificial y el aprendizaje de máquina

La inteligencia artificial es la forma en la que las máquinas pueden, a través de sus algoritmos, estructuras físicas y conexiones en las que se basan, obtener soluciones y dar fin a unos objetivos como si de seres vivos se trataran.[11]

Por ejemplo, a través de sensores que captan movimiento, pueden permitir abrir o cerrar una puerta, como si de un controlador de barreras humanos se tratara.

En muchas aplicaciones, para conseguir que una máquina procese unos datos de entrada, y ejecute unas acciones en función de estos datos, tal y como se desea, se debe entrenar un modelo, y esto es lo que se llama el aprendizaje de máquina.

Un científico de máquinas, Tom M. Mitchell, dijo en su libro *Machine Learning: A Guide to Current Research* [11] que una máquina aprenderá si puede coger experiencia y utilizarla para que las actuaciones mejoren en experiencias similares, tal y como hacen los humanos. [5]

Indistintamente si el aprendiz es un humano o una máquina, el proceso de aprendizaje es similar y se puede basar en tres componentes, como sigue:

- **Datos de entrada:** a través de la observación, memoria y recuperación de las experiencias transitadas.
- **Abstracción:** incluye la traducción de los datos de entrada en representaciones más amplias.
- **Generalización:** es la capacidad de obtener buenos resultados con datos nuevos.

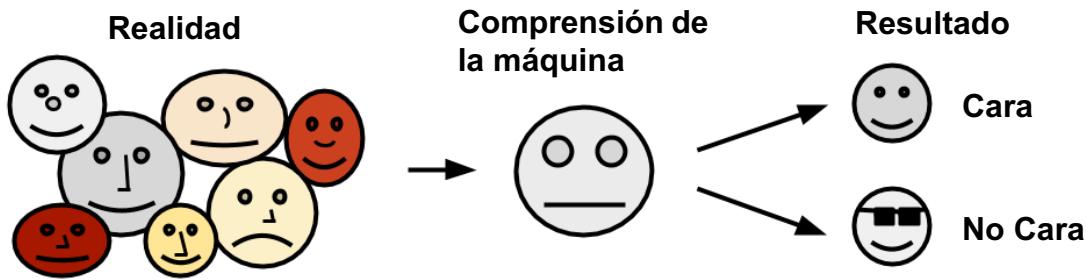


Figura 5. Esquema del proceso de aprendizaje de máquina.[5]

Para conseguir que una máquina haga el proceso de abstracción correctamente, primero se deberá elegir un algoritmo que le permita obtener una buena traducción de los datos. Para ello, es importante tener en cuenta cómo y de qué forma se presentan los datos de entrada. El paso de abstracción a generalización será el resultado de entrenar un modelo mediante el algoritmo elegido.

Para elegir correctamente el algoritmo, hay que saber que existen dos grandes grupos de aprendizaje, el aprendizaje supervisado, que se usan para construir modelos predictivos, y el aprendizaje no supervisado, que se usan para construir modelos descriptivos.[5]

La supervisión se refiere a los modelos que aprenden funciones, relaciones que asocian entradas con salidas, por lo que se ajustan a un conjunto de ejemplos de los que se conocen la relación entre la entrada y la salida. Es decir, dados unos datos de entrada, el algoritmo de aprendizaje intentará optimizar una función (el modelo) para encontrar la combinación de valores característicos que resulten en el objetivo de salida.

Mientras que los modelos predictivos se usan para tareas que implican predecir un valor a través de unos valores de entrada, el modelo descriptivo se usa para obtener asociaciones a grandes rasgos entre distintos tipos de datos.

En el caso de la clasificación de imágenes estamos, por lo tanto, en un caso de modelo predictivo, ya que tratan de predecir un valor final (tipología de la imagen, con un valor entre 0 y 1 que indican la mínima y máxima probabilidad con que correspondan a un tipo de imagen), respecto de unos valores de entrada (mapa de características de la imagen).

Este modelo predictivo, además, usará como el título indica, un tipo de red neuronal artificial llamada red neuronal convolucional.

3.3 Red neuronal artificial

Una red neuronal artificial establece la relación entre un conjunto de datos de entrada y un conjunto de datos de salida usando un modelo parecido a como un cerebro biológico responde a la percepción de estímulos sensoriales, de ahí, red neuronal.

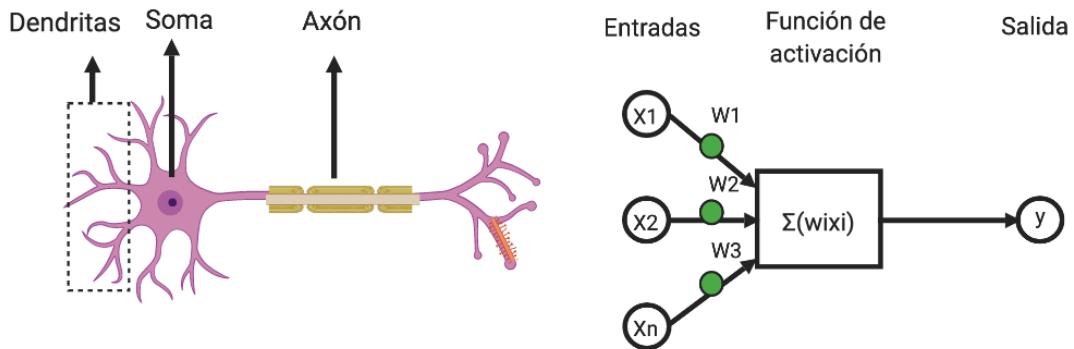


Figura 6. Relación entre una neurona biológica (izquierda) y una neurona artificial (derecha). [12]

Las neuronas biológicas disponen de un cuerpo celular (soma), que sintetiza la mayoría de las proteínas que se encuentran en la neurona. A través de las Dendritas, que son prolongaciones del cuerpo celular, reciben y procesan los impulsos que llegan de otras neuronas para transmitirlas al cuerpo celular y, el impulso es conectado por un Axón, que es otra prolongación, pero única y más larga que conecta con otra neurona. Esta conexión entre neuronas se llama sinapsis.

El modelo de los nodos artificiales puede entenderse en términos muy parecidos a los de una neurona biológica. Como se observa en la figura 5, un conjunto de datos de entrada (x_1, x_2, x_3) entran al nodo, y, según la función de activación, la función de pérdida y la métrica definida para definir la precisión del aprendizaje, sale un único dato de salida (y). Análogamente, si trasladamos este concepto a las neuronas biológicas, las dendritas aportarían los datos de entrada y el soma sería el encargado de proporcionar un dato de salida (axón).

Estos nodos se agrupan en capas, de modo que todos los nodos de entrada forman una capa, que será la capa de entrada, el nodo de salida forma otra capa o incluso hay nodos escondidos, que procesan y almacenan los datos de entrada previo a obtener el nodo de salida, que forman parte de las capas escondidas.

Cuando existen varias capas escondidas en una red neuronal artificial, se dice que se trata de una red que neuronal profunda. Estas capas escondidas no siguen un solo sentido de flujo de la información, sino que pueden iterar sobre sí mismas para entrenarse y conseguir un mejor nodo o datos de salida.

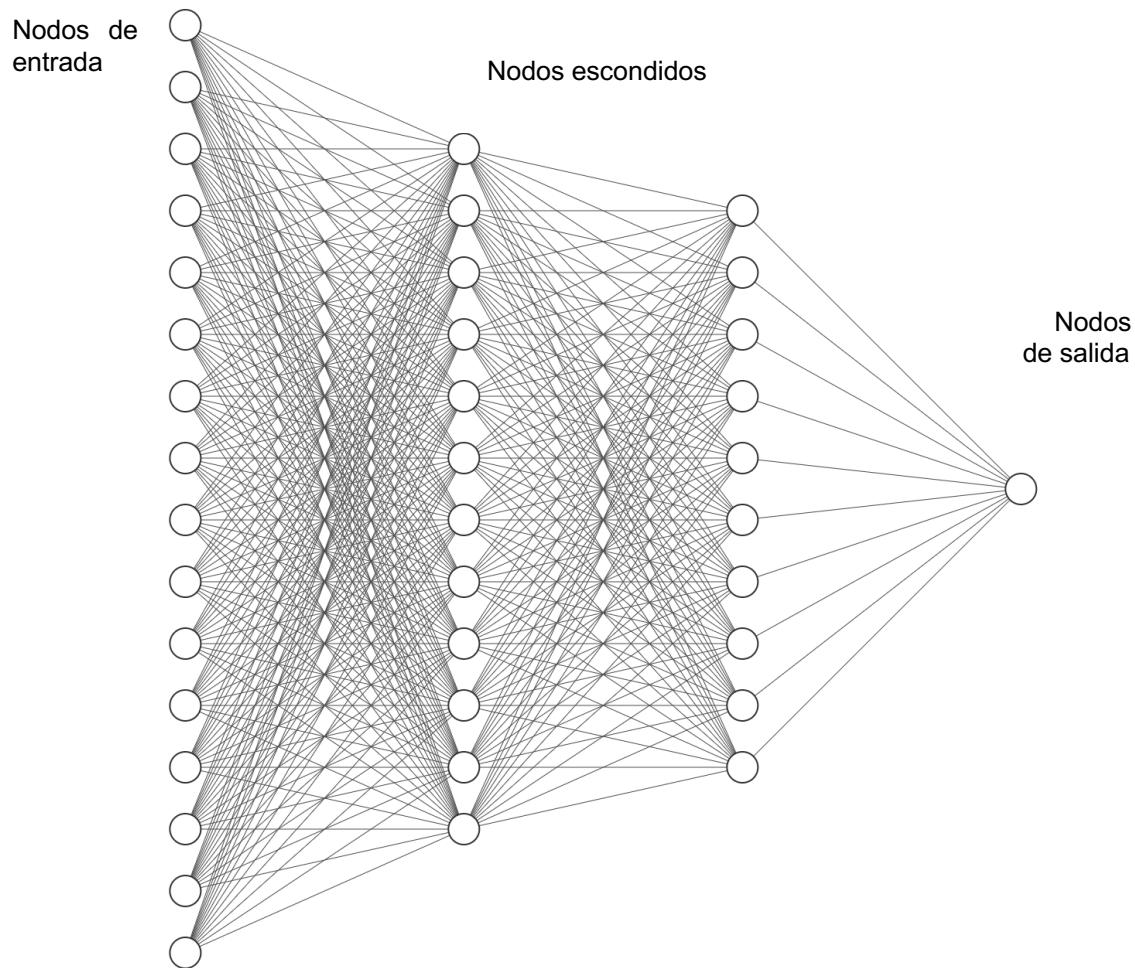


Figura 7. Ejemplo de red neuronal artificial profunda, con las distintas capas de entrada, escondidas y de salida.[13]

3.4 Aprendizaje profundo

El aprendizaje profundo es el conjunto de técnicas de aprendizaje de representación que aprenden características jerárquicas de los datos de entrada[14], mediante múltiples capas escondidas que van siendo modeladas con cada paso de entrenamiento hasta conseguir una función de pérdida mínima [12].

Las cuatro arquitecturas básicas de modelos de aprendizaje profundo utilizadas en el procesamiento de imágenes son las siguientes: las redes neuronales convolucionales (CNN), las redes completamente convolucionales (FCN), los Autocodificadores (AE) y las redes de profunda creencia (DBN). [14] Cada una de estas arquitecturas funciona de la siguiente manera:

- REDES NEURONALES CONVOLUCIONALES (CNN)

Las redes neuronales convolucionales son un tipo de red neuronal artificial avanzada, donde una señal fluye por esta red formando iteraciones, que pueden ser expresadas como:

$$F(x) = f_N(f_{N-1}(\dots(f_1(x))))$$

donde la N denota el número de capas escondidas y f_i representa la función en la capa correspondiente i.

Las capas presentes en esta técnica son las siguientes:

- **Capas convolucionales:** son las capas iniciales de la red, que se utilizan para extraer varias características de las imágenes de entrada. Los datos de salida de estas capas son un mapa de características que nos proporcionan información general como, por ejemplo en el caso de procesamiento de imágenes, las formas encontradas y sus dimensiones. Los datos de salida son pues, la respuesta a varias funciones y la convolución de las mismas. La convolución es una operación matemática donde dos funciones (en este caso, de activación) dan un resultado (conocido como la transformada de Fourier) producto de las transformadas de las funciones, punto por punto.
- **Capas pooling:** son las capas que siguen a las convolucionales, que lo que hacen básicamente es reducir el tamaño del mapa de características procedente de las capas convolucionales para disminuir el coste de procesamiento y de computación.
- **Capas completamente conectadas:** son las capas de redes neuronales en la que todos los resultados de la capa anterior están conectados a todos los nodos de la capa siguiente.
- **Capas dropout:** son las capas que se quitan de una red cuando se ha acomodado un modelo a una serie de datos de modo que cuando se ingresan datos radicalmente distintos, el resultado no es el esperado. Quitando una parte de estos nodos o capas desestimadas al azar, se permite volver a una red neuronal con capacidad de dar resultados precisos para datos de entrada distintos.

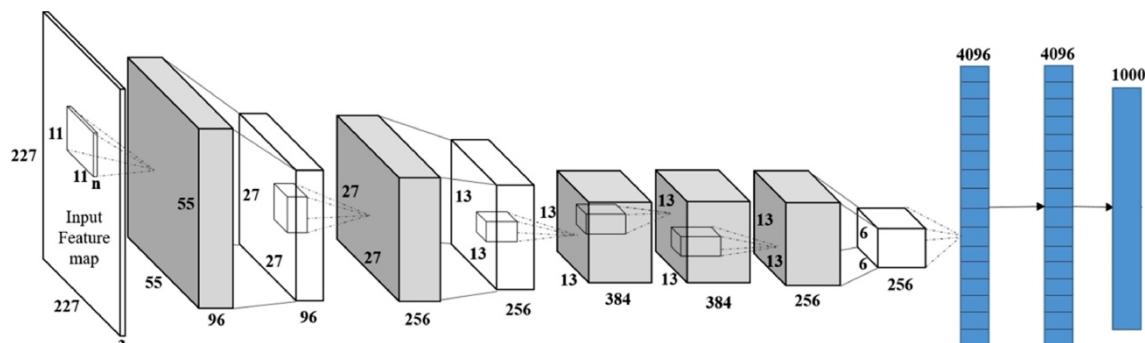


Figura 8. Diagrama de flujo de las capas que las características de una imagen atraviesan en una red neuronal convolucional, desde las capas de convolución, hasta las max-pooling que finalizan en tres capas de conexión completa para obtener el resultado de la característica final.

Ejemplos de uso: son útiles en una variedad de aplicaciones pero donde han conseguido revolucionar el estado del arte es en la visión artificial y el procesado de imágenes.

- REDES COMPLETAMENTE CONVOLUCIONALES (FCN)

La mayor diferencia entre las FCN y las CNN es que en las redes completamente convolucionales se reemplazan las capas completamente conectadas por capas de deconvolución. Como su nombre indica, la capa de deconvolución devuelve las capas que se han empequeñecido a su tamaño original, antes de conseguir los datos de salida.

Las FCN, básicamente, permiten crear un mapa que, en el caso de procesar una imagen, tiene el mismo tamaño que la imagen analizada para cada clase y clasifican cada píxel de la imagen.

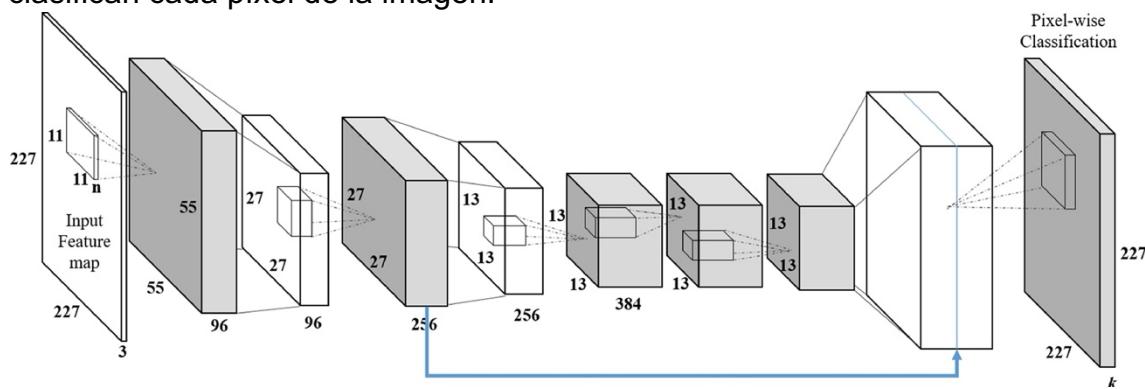


Figura 9. Diagrama de flujo de las capas que atraviesan las características de una imagen en una red completamente convolucional. La flecha azul muestra la conexión de salto donde el tensor de salida y de la segunda capa de convolución es concatenado al tensor de salida de la última capa convolucional a través de la capa de deconvolución. La k denota el número de clases posibles a la que el pixel puede pertenecer.

Ejemplos de uso: clasificación de imágenes para la actualización de los mapas que cubren la superficie de la Tierra a través de las series de imágenes espaciales y espectrales (SITS).

• AUTOCODIFICADORES (AE)

Este tipo de aprendizaje generativo es usado para el aprendizaje sin supervisión. El aprendizaje generativo es un tipo de aprendizaje que, por ejemplo, en el procesamiento de imágenes, se basa en la comparación entre la imagen a modificar, antes y después de su modificación, para establecer si la imagen ha mejorado no respecto a su forma anterior. La comparación de datos y de la obtención de parámetros de pérdida (loss) mayores o menores respecto unos parámetros a evaluar, indicarán esas mejoras generadas. Su mayor meta es aprender una representación con menor dimensión que los datos de entrada.

aprender una representación con menor dimensión que los datos de entrada. El entrenamiento de AEs normalmente contiene dos etapas: la etapa de codificación y la etapa de decodificación.

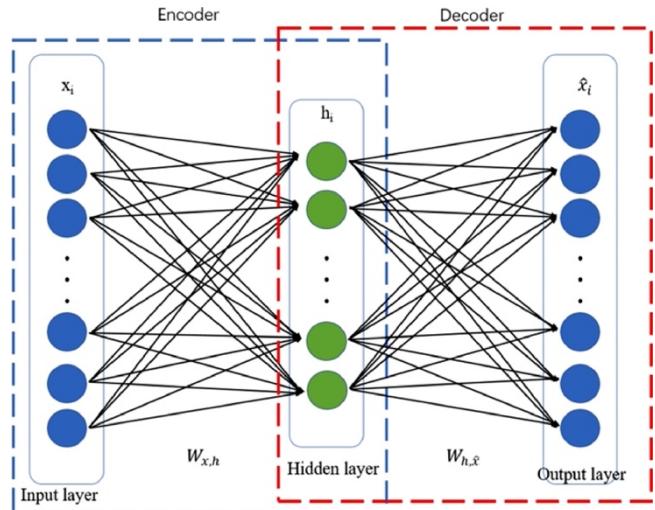


Figura 10. Un AE típico, que tiene una estructura simple, que generalmente tiene 3 capas: una capa de entrada, una capa escondida y una capa de salida.[14]

Ejemplos de uso: usadas para compresión de datos o para mejora de datos, como en el procesamiento de imágenes, eliminando todo el ruido o degradaciones de éstas, como, por ejemplo, el desenfoque, para mejorarla. Por ejemplo, en la imagen que se muestra en la Figura 10, se elimina el flash existente.

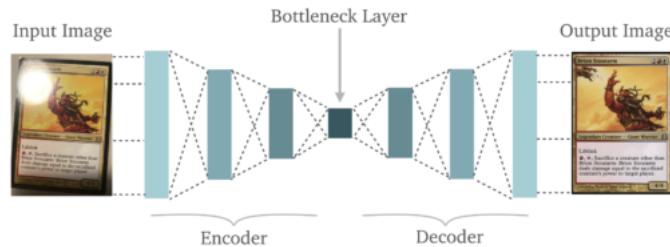


Figura 11. Ajuste de una imagen mediante una red neuronal de autocodificación (AE). [15]

- REDES DE PROFUNDA CREENCIA (DBN)

Las DBN son un modelo probabilístico generativo que están construidos por un apilamiento de Máquinas de Boltzmann Restringidas (RBMs) [14] en vez de AEs. Las DBN sólo contienen dos capas, una visible y una escondida y su funcionamiento es muy parecido al de los AE.

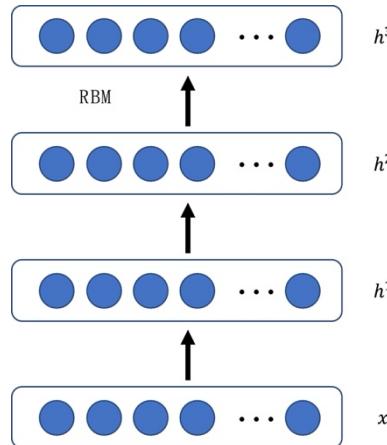


Figura 12. Los distintos pasos de una DBN, que van consiguiendo una mejor representación (h) de las características a medida que se van apilando las RBM.

Ejemplo de uso: mecanismos de precisión en aplicaciones industriales e inspecciones de calidad para manufactureros que quieren asegurar que el producto que sale de la fábrica tiene la calidad deseada.

3.5 Redes neuronales convolucionales

En general, de todas las técnicas descritas, las redes neuronales convolucionales (CNN) son las que han demostrado un rendimiento superior en las tareas de reconocimiento de imágenes. [14]

La diferencia fundamental entre una capa completamente conectada y una capa de convolución es que las capas completamente conectadas normalmente aprenden patrones globales dentro del espacio de los datos de entrada (por ejemplo, en un dígito MNIST [16], patrones relacionados con los píxeles), mientras que las capas de convolución aprenden los patrones en pequeñas ventanas 2D, como se puede ver en la siguiente figura.

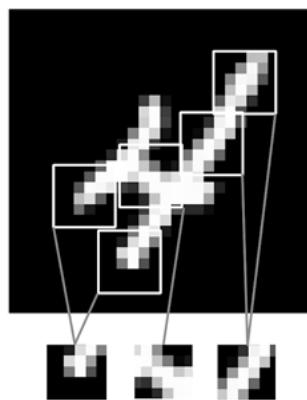


Figura 13. Las imágenes pueden ser despiezadas en capas de convolución en pequeños patrones locales, como ejes, texturas, etc. [8]

Esta característica clave confiere a las redes neuronales convolucionales dos propiedades interesantes:

- Los patrones que aprenden son una traducción inmutable. Por ejemplo, una red neuronal convolucional, una vez aprende una característica, como una oreja de gato, la reconoce en cualquier imagen.
- Pueden aprender jerarquías de patrones espaciales. Cada una de las características las puede despiezar en distintas partes, tal y como hace con la imagen general.

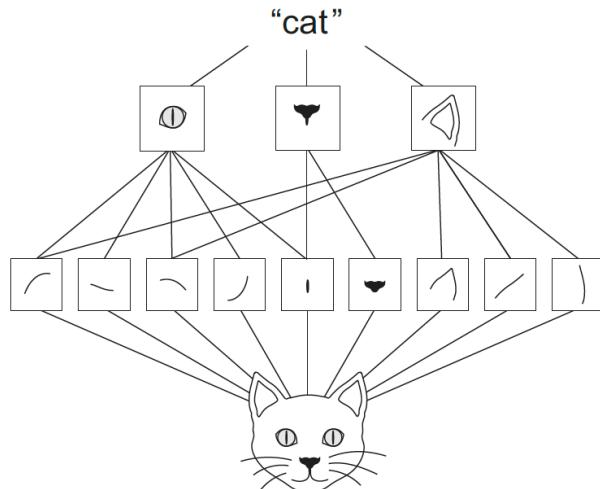


Figura 14. Ejemplo de jerarquía de las características de cada parte de un gato reconocidas por una red neuronal convolucional. [8]

Generalmente, todos los sistemas de aprendizaje de máquina utilizan tensores como su estructura básica de datos. Un tensor es un contenedor de datos, casi siempre numéricos. Por lo tanto, es un contenedor de números. Las matrices, por ejemplo, son tensores de 2 dimensiones.

Un tensor es definido por tres atributos clave:

- Número de ejes – un tensor de 3D por ejemplo, tiene 3 dimensiones. Una matriz tiene 2 dimensiones. Esto también es conocido como la variable *ndim* en librerías de Python como Numpy.
- Shape – es una tupla de enteros que describe cuántas dimensiones tiene el tensor en cada eje. Por ejemplo, una matriz con 3 filas y 5 columnas tiene forma (3,5).
- Tipo de datos (normalmente llamado *dtype* en librerías Python) – es la tipología de datos que hay en el tensor. Por ejemplo, podrían ser decimales, enteros, etc.

Las redes neuronales de convolución operan en tensores, llamados mapas de características, con dos ejes espaciales (altura y amplitud) así como un eje de profundidad (también llamado el eje de canales). Para una imagen RGB, la dimensión del eje de profundidad es 3, debido a que los canales de colores son el rojo, el verde y el azul.

Así pues, las redes de convolución son definidas por dos parámetros clave:

- Tamaño de los parches extraídos de los datos de entrada – Normalmente 3x3 o 5x5.
- Profundidad del mapa de características resultante – el número de filtros computados por la convolución.

Una red de convolución trabaja corriendo estas ventanas sobre el mapa de características de entrada, parando en cada posible ubicación y extrayendo la ventana de características, normalmente 3D (altura, anchura y profundidad). Cada parche 3D es transformado en un vector 1D, el vector de salida. Este proceso se puede ver en la siguiente figura:

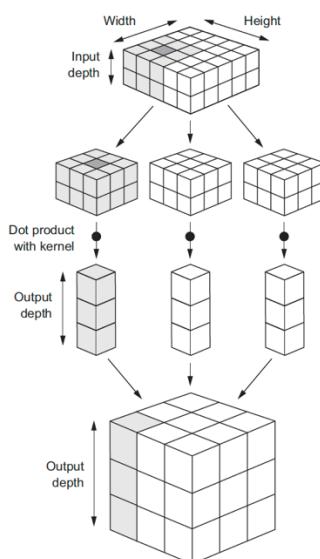


Figura 15. Diagrama de flujo de la convolución. [10]

Para acabar, las redes neuronales convolucionales pasan de la convolución al max-pooling. En general, se llama max-pooling a la extracción de ventanas de los mapas de características de entrada para dar un valor de salida máximo para cada canal, en otras palabras, se reduce el número de coeficientes del mapa de características para simplificar las convoluciones, es decir, si un ojo tiene un diámetro posible entre 1,5 o 2,5 cm, max-pooling se quedaría con 2,5 cm. [8]

3.6 Aprendizaje profundo y cáncer de pulmón

La detección en fases iniciales del diagnóstico de cualquier cáncer juega un papel clave para la supervivencia del paciente y está demostrado que puede mejorar las ratios de supervivencia a largo plazo. Las imágenes médicas son muy importantes para esta pronta detección y diagnóstico correcto. Recientemente, se han ido investigando técnicas bastante fiables relacionadas con el aprendizaje profundo de imágenes.

En cuanto al aprendizaje profundo de detección de cáncer de pulmón, existen varios estudios realizados a través de aprendizaje profundo, que se muestran a continuación:

Tabla 2. Resumen de algunos proyectos realizados con aprendizaje profundo para la detección de cáncer de pulmón basado en imágenes.

Referencia	Aplicación	Modalidad	Técnica empleada	Precisión alcanzada
Zhu et al. [17]	Análisis de supervivencia	Histopatología	CNN	62,9%
Hua et al. [18]	Clasificación de nódulos	Tomografía computacional en rodajas	DBN & CNN	73,4% & 73,3%
Hussein et al. [19]	Caracterización de nódulos	Tomografía computacional volumétrica	CNN	92,3%
Setio et al. [20]	Detección de nódulos	Tomografía computacional volumétrica	CNN	63,7%
Dou et al. [21]	Detección de nódulos	Tomografía computacional volumétrica	CNN	82,7%
Shen et al. [22]	Clasificación de nódulos malignos sospechosos	Tomografía computacional volumétrica	CNN	87,1%
Paul et al. [23]	Predicción de supervivencia	Tomografía computacional en rodajas	CNN	77,5%
Wang et al. [24]	Clasificación de nódulos	Tomografía computacional en rodajas	CNN	69,3%
Hirayama et al. [25]	Extracción de la opacidad del suelo de cristal de la región candidata	Tomografía computacional en rodajas	CNN	93%
Tajbakhsh et al. [26]	Detección y clasificación de nódulos	Tomografía computacional en rodajas	MTANN & CNN	88,1% & 77,5%
Kim et al. [27]	Clasificación de nódulos	Tomografía computacional en rodajas	AE	95,5%
Vas et al. [28]	Clasificación de regiones	Tomografía computacional volumétrica	FCN	94%

Revisando los trabajos que aparecen en la lista anterior, la mayoría están profundamente desarrollados y con unos resultados de alta precisión. Respecto la clasificación de nódulos que se han realizado en estos proyectos, se centran en extraer la conclusión de si el tumor es cancerígeno o no, pero no clasifican el

tipo de tumor, que en este proyecto se pretende clasificar el tipo de cáncer de pulmón además de clasificar si el tumor es benigno o no.

4 Metodología

4.1 Conjunto de datos (dataset)

El entrenamiento de la red neuronal se ha llevado a cabo mediante las imágenes facilitadas por un grupo de investigadores de la Cornell University de Nueva York, con su proyecto Lung and Colon Cancer Hispathological Image Dataset [4], que contienen 5000 imágenes de 768 x 768 píxeles de cada uno de los tres tipos de tumor a clasificar.

Este dataset es el elegido para el trabajo porque al disponer de 5000 imágenes por cada tipo de tumor, facilita mucho el entrenamiento de la red neuronal convolucional. Además, al disponer de más de 1000 imágenes por tipo de tumor, es una garantía de que se podrá analizar y clasificar correctamente las imágenes, y además de no tener que realizar un aumento adicional al disponer de suficientes imágenes para poder entrenar la red, lo cual facilita y reduce el número de tareas a realizar para conseguir una red neuronal exitosa.

4.2 Programación con Python

La elección de Python para este proyecto se explica básicamente por la existencia de la librería Keras para Python, que ofrece una vía muy útil para definir y entrenar casi cualquier tipo de modelo de aprendizaje profundo, y por ser uno de los lenguajes más intuitivos de usar y que actualmente usan muchos bioinformáticos.

Esta librería Keras posee específicamente soporte para redes neuronales convolucionales para clasificación de imágenes, lo que hacen que la programación de la red en Python sea inmejorable.

Para programar en Python, se ha configurado previamente un ambiente virtual usando miniconda, que permite utilizar una versión de Python posterior a la que se posee para que todas las librerías necesitadas puedan funcionar correctamente. En versiones muy nuevas o muy antiguas, algunas librerías pueden no ser compatibles o no estar actualizadas correctamente.

En los anexos, se adjunta un listado con todas las versiones de las librerías, ambiente virtual y versión de Python utilizados para este proyecto.

Una vez realizada esta configuración, se ha elaborado el algoritmo mediante Jupyter Notebook, por su facilidad para recorrer todas las funciones y obtener los resultados y fallos del algoritmo.

4.3 Desarrollo del algoritmo de clasificación

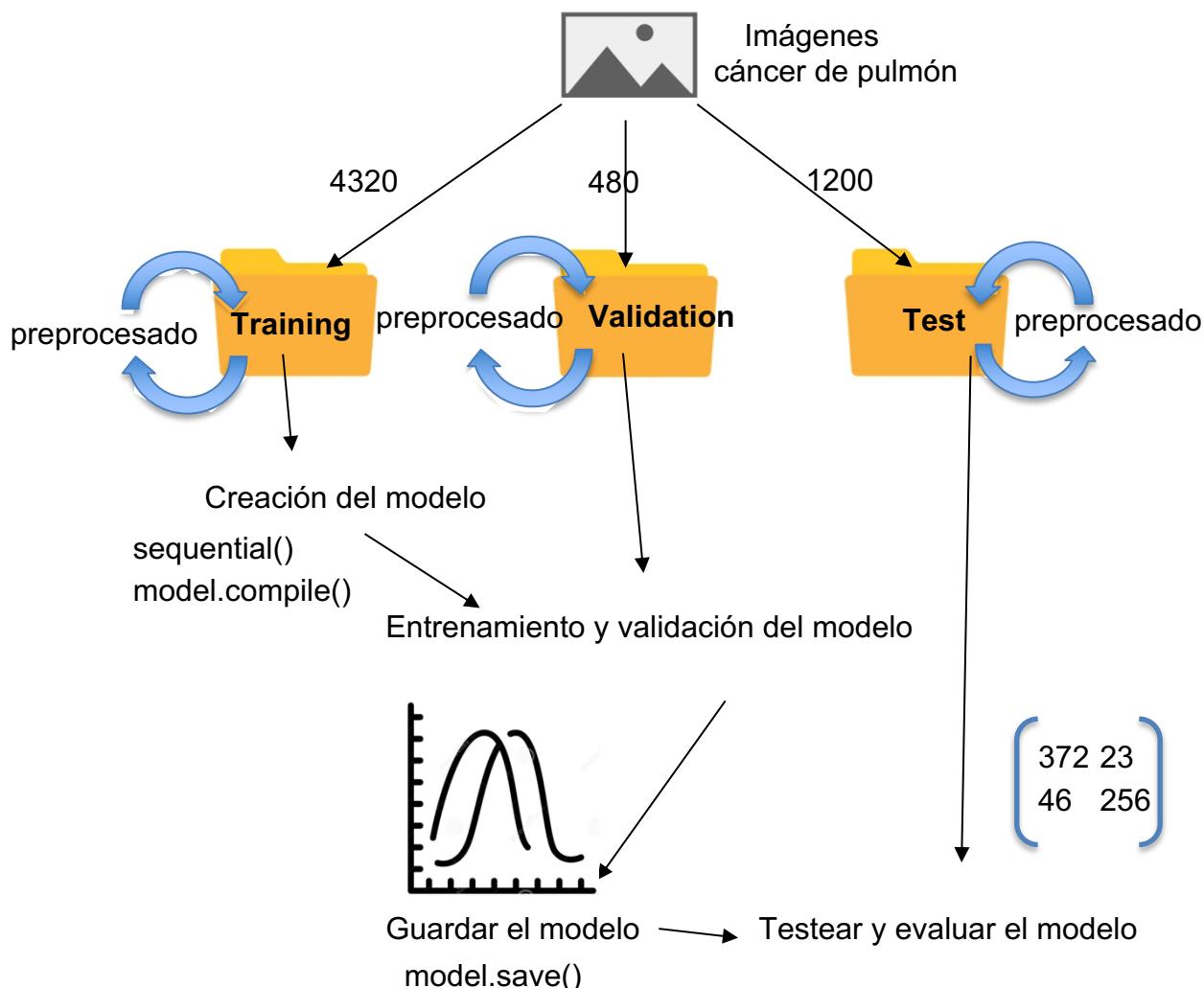


Figura 16. Esquema del workflow seguido para la creación del algoritmo de clasificación de imágenes de pulmón con Python y la librería Keras.

Los pasos a seguir para implementar un modelo a través de Keras son los siguientes:

1. Preparación de los datos

Se han asignado imágenes, dividiendo en carpetas de entrenamiento, validación y testeо, con 4320 para entrenamiento, 480 para validación y 1200 para testeо.

Las carpetas deben llevar el nombre de las etiquetas con las que se identificarán el nombre de clases en el algoritmo para su posterior clasificación. Estos nombres son:

aca = Adenocarcinoma

n = Tumor benigno

scc = Carcinoma escamoso

Por último, se ha definido un código para el preprocesado de las imágenes, que incluye la normalización de los píxeles, el redimensionamiento de imágenes y la homogeneización de colores.

2. Diseñar una red neuronal convolucional que permita conducir los datos de entrada a los objetivos definidos.

La red neuronal de convolución es la mejor elección posible para este proyecto. La razón es que las representaciones aprendidas por una red convolucional son más genéricas y, por lo tanto, más enfocadas a clasificar imágenes abstractas o en las que las diferencias son mínimas, lo que hace que un ordenador con una red neuronal convolucional sea una herramienta potentísima para diferenciar tumores que, a simple vista, son difíciles de clasificar. Estas características típicas que diferencian unas imágenes de otras, un tumor de otros, se deben etiquetar mediante las clases y eso otras redes que no pueden desgranar las imágenes de esta forma, no tienen la capacidad para clasificar imágenes del mismo modo que lo hace una red neuronal convolucional.

La arquitectura de la red neuronal convolucional creada consiste en tres bloques de convolución con una capa max-pooling para cada bloque. Hay una capa completamente conectada con 128 unidades al final, que es activada por una función de activación relu.

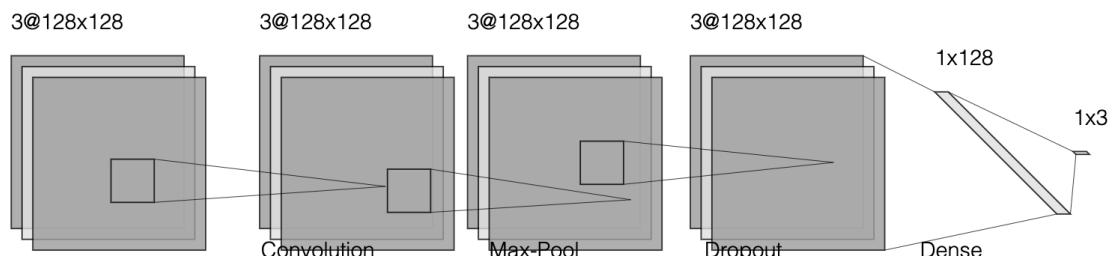


Figura 17. Arquitectura de la red neuronal creada para este proyecto.[13]

Se ha decidido elegir el diseño de la arquitectura de este modo bastante simple porque así se puede conseguir una buena aproximación a la clasificación correcta de imágenes que se posee sin tener que emplear mucha memoria ni tiempo para conseguir un modelo aceptable.[29] Evidentemente, una red neuronal fiable para todo tipo de imágenes de pulmón y de distinta índole requeriría una arquitectura más compleja y más costosa de entrenar para conseguir una precisión alta.

3. Configurar el proceso de aprendizaje eligiendo una función de pérdida, un optimizador, y algunas métricas a monitorizar.

Se han elegido los siguientes parámetros para el aprendizaje la red neuronal:

Tabla 3. Parámetros del aprendizaje de la red neuronal.

Función de pérdida	Categorical crossentropy
Optimizador	Adam
Métricas a monitorizar	Precisión, Pérdida
Función de activación	relu
Función de salida	softmax

La función de pérdida elegida ha sido la denominada categorical crossentropy porque es la que permite obtener una probabilidad para un problema de múltiples clases a identificar.

El optimizador es Adam porque es un algoritmo indicado también por la mayoría de expertos para problemas de clasificación de imágenes. Su nombre se debe a las palabras en inglés Adaptive Moment Estimation, que mide dos momentos, uno como la media de los gradientes a lo largo del tiempo y otro con la varianza.

La primera métrica monitorizada es la precisión, que calcula con cuánta frecuencia las predicciones se ajustan a la clase de la imagen.

Por otro lado, se ha monitorizado la pérdida, que es el valor total obtenido por la función de pérdida y, eso significa, como más grande este valor, más alejadas son las predicciones de los resultados reales. Definie, por tanto, las distancias entre predicciones y realidad y si obtenemos outliers, esto dará una función de pérdida muy elevada.

La función de activación elegida ha sido relu, que es de las más usadas en redes neuronales convolucionales, ya que eso permite conseguir algún patrón positivo, pues esta función no es continua en 0 y los valores negativos, que no son importantes en el preprocesado de imágenes, se establecen en cero.

Por último, la función de salida elegida ha sido softmax , y esta calcula la exponencial del valor de entrada dado y la suma de los valores exponenciales de todos los valores en la entrada y la relación entre estos datos da la salida de la función softmax.

4. Entrenar el modelo.

Se ha entrenado el modelo usando 20 épocas o pasos de evaluación de las métricas.

El tamaño del lote usado ha sido de 128, ya que generalmente toman una potencia de 2 y como más grande sea, más uso de memoria necesitará la computadora ya que más espacio ocuparán las imágenes, así que se ha utilizado este valor estándar.

Se ha utilizado un total de 4800 imágenes de las cuales el 10% han servido para validación y el resto para entrenamiento.

5. Evaluar el modelo entrenado.

Para comprobar si la clasificación es correcta y obtener la precisión alcanzada para las distintas clasificaciones, se ha comparado las métricas de precisión y valor de pérdida obtenidos en cada paso de clasificación de imágenes (época), tanto para las imágenes de entrenamiento como de validación. En función de estos valores de pérdida y precisión, el modelo ha ido cambiando los parámetros de entrada para obtener unos valores de salida más ajustados hasta llegar a la mejor precisión y pérdida posibles.

La evolución en cada paso de entrenamiento de la red neuronal se puede ver en los siguientes gráficos:

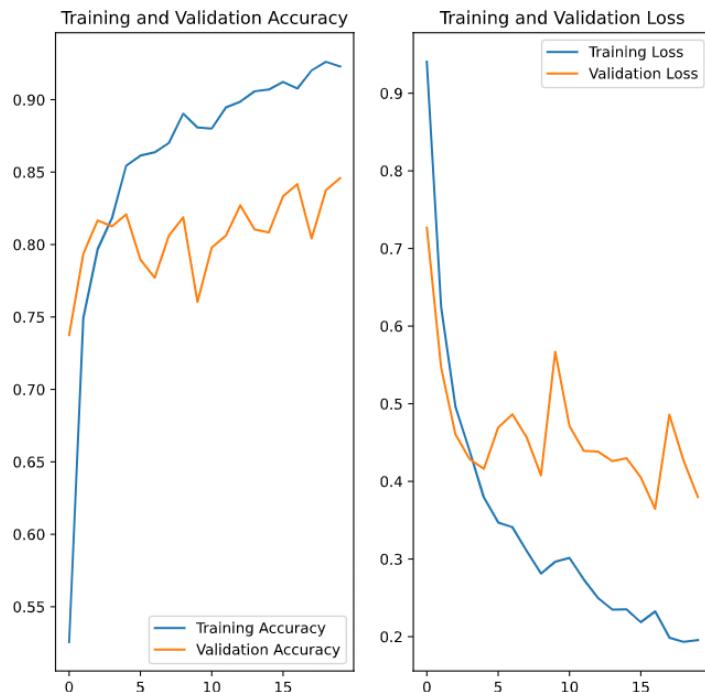


Figura 18. Evaluación del modelo definitivo gear classifier 4.

Estos gráficos nos muestran que la función de pérdida ha ido disminuyendo, pero se ha hecho constante ya a partir de los 10-15 pasos de entrenamiento, con lo cual no es aconsejable seguir entrenando para evitar el sobreajuste (overfit).

6. Mejorar el modelo.

Para mejorar el modelo se han utilizado imágenes distintas a las usadas para el entrenamiento, validación y testeo, de modo que se comprueba si se ha sobreentrenado o no el modelo y si se obtienen buenas predicciones en caso de testear imágenes que proceden de fuentes distintas y pueden ser bastante diferentes a las que se ha acostumbrado el modelo.

Para mejorar el modelo, se ha añadido un preprocessado de imágenes más potente que el inicial. Además, la arquitectura de la red neuronal ha ido

variando, seleccionando más o menos capas, también cambiando el número de imágenes de entrada y se han ajustado las distintas características para cada modelo. A continuación, se presenta una tabla comparativa de los distintos modelos creados con sus características principales hasta llegar al modelo final:

Tabla 4. Comparativa de los distintos modelos creados.

Modelo	Arquitectura	Características	Precisión	Pérdida
lung_2	<ul style="list-style-type: none"> • 3 Max-Pooling • 3 Conv • 1 Dense 	<ul style="list-style-type: none"> • Adam • Epochs = 10 • 6000 imgs • Relu • Relu 	<ul style="list-style-type: none"> • 80,16% 	<ul style="list-style-type: none"> • 0,13
lung_3	<ul style="list-style-type: none"> • 3 Max-Pooling • 3 Conv • 1 Dense 	<ul style="list-style-type: none"> • Adam • Epochs = 10 • 2000 imgs • Relu • Relu 	<ul style="list-style-type: none"> • 86% 	<ul style="list-style-type: none"> • 0,29
check_lung	<ul style="list-style-type: none"> • 5 Max-Pooling • 5 Conv • 5 Dropout • 1 Dense 	<ul style="list-style-type: none"> • Optimizer • Epochs = 5 • 5100 imgs • Relu • softmax 	<ul style="list-style-type: none"> • 82% 	<ul style="list-style-type: none"> • 0,27
gear classifier 4	<ul style="list-style-type: none"> • 1 Max-Pooling • 2 Conv • 2 Dropout • 2 Dense 	<ul style="list-style-type: none"> • Adam • Epochs = 20 • 4320 imgs • Relu • softmax 	<ul style="list-style-type: none"> • 89,33% 	<ul style="list-style-type: none"> • 0,35

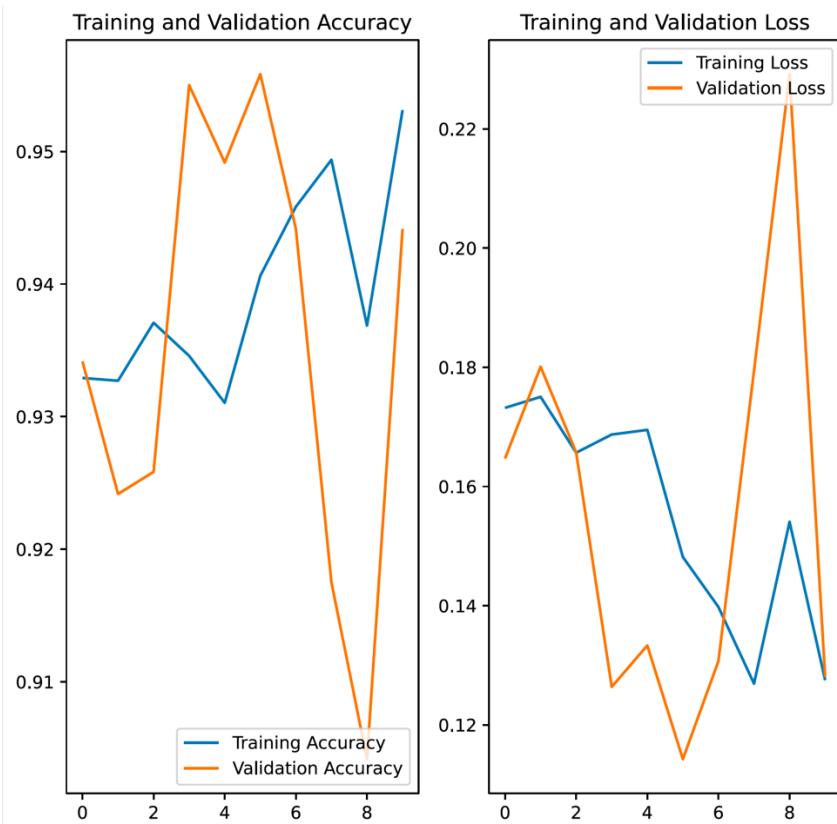


Figura 19. Evaluación del modelo lung_2.

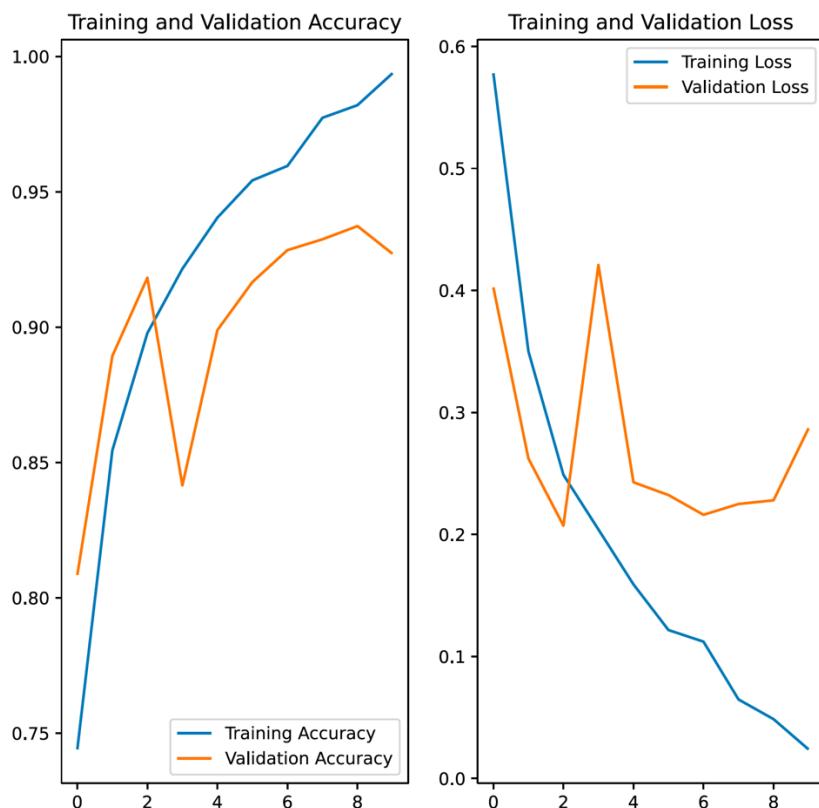


Figura 20. Evaluación del modelo lung_3.

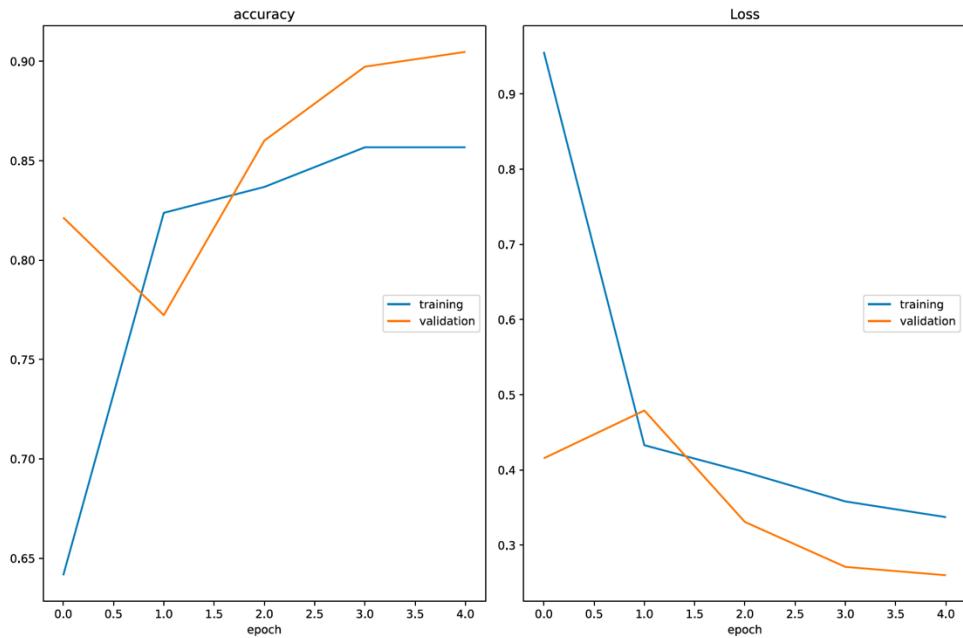


Figura 21. Evaluación del modelo check_lung.

4.4 Aplicación web con Python Flask, HTML y JavaScript

La elección de Python Flask para servir como aplicación web se explica por su facilidad a la hora de trabajar con webs pequeñas y hacer de conexión entre el código de un algoritmo a ejecutar en lenguaje Python (elegido para elaborar la red neuronal) en el backend y la respuesta a un frontend, que será una combinación entre HTML y JavaScript.

Python Flask es un framework o estructura en lenguaje Python que permite hacer de servidor web para una app básica o que se quiera desarrollar de forma ágil y rápida.

Los pasos para la creación de una aplicación web con Python Flask son los siguientes:

1. Crear un directorio con una estructura inmutable de carpetas y archivos
2. Crear un archivo app.py (backend) que incluya el código en Python llamando al algoritmo que procese la imagen por la red neuronal y creando la interconexión entre frontend y backend.
3. Crear los archivos .html que servirán de frontend.

La web se lanza a través de un HTML que conecta con la aplicación en Python a través de Flask, del siguiente modo:

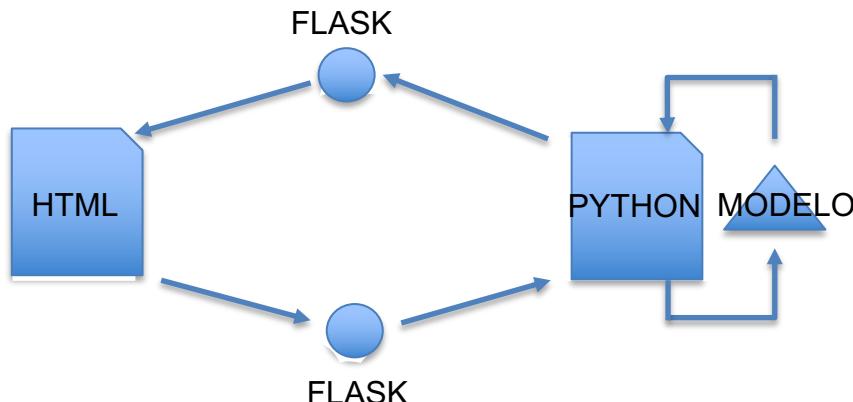


Figura 22. Interacción entre las distintas partes de la aplicación web.

4.5 Alojamiento web en Heroku

Heroku ha sido el sitio web elegido para alojar la web porque reúne las condiciones que se necesitaban para que funcionase correctamente:

- Es gratuito
- Funciona con openSource (GitHub)
- Permite lanzar apps con código hecho en Python
- El *backend* se puede manejar fácilmente desde el terminal
- En la versión gratuita, tiene espacio en la nube para un límite de 512 MB

Quizás, la única condición un poco limitante, es el espacio ocupado máximo permitido en la nube, de 512 MB, que para una aplicación pequeña es suficiente, pero para webs más grandes, no serviría. Para paliar esta condición limitante, se ha ejecutado un código que borre la memoria cada vez que se suba una nueva imagen a la aplicación web.

La siguiente tarea ha sido crear los archivos que permitirán al sitio alojamiento web, reconocer las versiones del lenguaje y de las librerías empleadas para el correcto funcionamiento de nuestra aplicación y su subida a Heroku a través del terminal. En el anexo se detallan los pasos seguidos para realizar estas tareas.

5 Resultados

La red neuronal convolucional ha sido capaz de clasificar correctamente las imágenes para testeo con un 89,33% de acierto. Esto significa que es un método de aprendizaje profundo bastante adecuado para llevar a cabo este tipo de tareas.

Tabla 5. Precisión alcanzada por la red neuronal convolucional.

Clase de tumor	Aciertos	Fallos	Precisión	Función de pérdida
Adenocarcinoma (aca)	1029	172	85,75%	
Benigno (n)	1154	46	96,17%	
Escamoso (scc)	1033	167	86,08%	
TOTAL	3216	385	89,33%	0,35

La función de pérdida alcanzada ha sido de 0,35.

La creación de la aplicación web se puede visitar libremente desde el siguiente enlace:

<http://dcapi2.herokuapp.com/>

En la página principal, se observa el título en la cabecera y un ícono de 3 barras que permite abrir un desplegable a la izquierda al hacer click en el botón.



Figura 23. Página principal de la aplicación web.

Al abrir el desplegable se pueden visitar las subpáginas de instrucciones de funcionamiento de la web en **¿Cómo funciona?**, una pequeña explicación de los tres tipos de cáncer de pulmón que detecta nuestra aplicación en **Tipos de cáncer de pulmón** y directamente clasificar una imagen que se desee en **Subir imagen**.



Figura 24. Desplegable con el menú de opciones de la web.

En **¿Cómo funciona?**, cualquier usuario puede entender para qué sirve la web, cuál es su finalidad y se detallan los pasos a seguir para poder clasificar una imagen de cáncer de pulmón:



DCAP-i² es una aplicación web que posee una herramienta para clasificar imágenes de pulmón y devolver el tipo de tumor existente en el pulmón, en caso de que exista.

Para utilizar el clasificador de imágenes DCAP-i², sigue los siguientes pasos:

1. Dirígete a la pestaña **Subir imagen** dentro del menú que se desplegará a tu izquierda al apretar el botón con tres rayas
2. A través del formulario, sube tu imagen seleccionándola desde el explorador de archivos de tu ordenador (debe acabar en .jpg., jpeg o .png) y haz click en el botón "Envía"
3. Espera que se procese tu imagen y en unos segundos aparecerá el tipo de cáncer de pulmón y su % de fiabilidad de la respuesta.

DCAP-i² no permite clasificar aún todos los tipos de cáncer de pulmón. Para conocer qué tipos de cáncer de pulmón puede detectar, dirígete a la pestaña **Tipos de cáncer de pulmón**.

Figura 25. Apartado **¿Cómo funciona?** de la aplicación web.

Para conocer qué tipos de cáncer de pulmón se clasifican en la aplicación web, hay que dirigirse a la pestaña **Tipos de cáncer de pulmón**, que se muestra en la siguiente imagen:

DCAP-i²

Diagnóstico de Cáncer de Pulmón imagen-inteligente

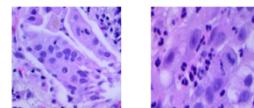
Tipos de tumor en pulmón humano analizados por DCAP-i²

DCAP-i² clasifica imágenes procedentes de alguna de las tres imágenes que tenéis en pantalla.

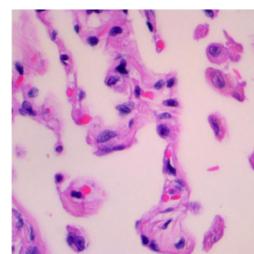
El adenocarcinoma es un tipo de cáncer de pulmón no microcítico, cuyas células se asemejan a las células de las glándulas, tal como las glándulas que secretan moco en los pulmones. (www.cancer.org)

El Carcinoma escamoso o carcinoma de células escamosas es el nombre de un tipo de cáncer de pulmón no microcítico, cuyas células se asemejan a las células planas (llamadas células escamosas) que revisten las vías respiratorias. (www.cancer.org)

Más de la mitad de todos los nódulos pulmonares solitarios no son cancerosos (benignos). Los nódulos benignos tienen muchas causas, por ejemplo, cicatrices e infecciones pasadas. Los granulomas infecciosos (que son formados por células como reacción a una infección pasada) causan la mayoría de las lesiones benignas. (medlineplus.gov)



Adenocarcinoma pulmonar Carcinoma escamoso



Tumor benigno

Figura 26. Apartado Tipos de cáncer de pulmón de la aplicación web.

Para subir la imagen deberemos elegir el archivo y enviarlo con este formulario que nos aparece:

DCAP-i²

Diagnóstico de Cáncer de Pulmón imagen-inteligente

Sube la imagen que deseas clasificar:

No s'ha triat cap fitxer

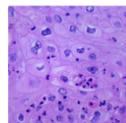
Nota: La imagen debe ser un archivo .jpeg, .jpg o .png.

Figura 27. Parte de la página web de “Subir imagen” con el formulario de envío de la imagen que se desea clasificar.

Y a continuación, cuando presionamos el botón de enviar, realiza la clasificación automáticamente e informa del resultado obtenido y su % de fiabilidad.

DCAP-i²

Diagnóstico de Cáncer de Pulmón imagen-inteligente



Esta imagen pertenece a un tumor del tipo scc con una precisión de 58.00 sobre 100.

aca = Adenocarcinoma

n = tumor benigno

scc = Carcinoma escamoso

[Volver](#)

Figura 28. Página del resultado obtenido de la clasificación de la imagen con la aplicación integrada en la web.

6 Discusión

El modelo obtenido cumple con los objetivos establecidos inicialmente de conseguir clasificar las imágenes de tejido pulmonar con tumor con una precisión superior al 70% (89,33% obtenido).

Respecto a la aplicación web, la creación mediante Python Flask ha sido óptima para las condiciones de espacio que estaban disponibles, al elegir el *web hosting* gratuito de Heroku. La simplicidad de Flask permite que se puedan conectar aplicaciones en Python con estructuras web en HTML ocupando muy poco espacio y transmitiendo datos a una buena velocidad.

El resultado es que las clasificaciones se consiguen entre 1 y 2 segundos de media tras introducir la imagen en el formulario y es un tiempo más que aceptable teniendo en cuenta que se trata de una aplicación web sin coste económico alguno.

La aplicación web parece que es una buena herramienta de soporte para la toma de decisiones para cualquier profesional médico y lo que se requeriría es de añadir más complejidad a la red neuronal convolucional para que los resultados se acerquen al 100% de precisión.

Seguramente, se podría dar uso a aplicaciones parecidas en ambientes profesionales que requieran de una segunda opinión y la red neuronal podría incluso determinar más tipos de tumor ya que, como demuestra, funciona bien en la clasificación multiclase, lo cual permitiría en un futuro, poder convertirse en una buena herramienta para el mundo y los profesionales de la medicina.

7 Conclusiones

7.1 Conclusiones

Una red neuronal para clasificar imágenes, puede funcionar bien inicialmente, pero alcanzar una buena precisión y que tenga una estructura adecuada para clasificar imágenes con más o menos diferencias, no es tan sencillo. La necesidad de realizar un preprocesado de imágenes, de elegir un número adecuado para el entrenamiento que no sobreentrene la red, de escoger las capas de nodos adecuados y ajustar la red añadiendo capas dropout, entre otras, es una ardua tarea que requiere de un gran número de simulaciones. La prueba está en que la red neuronal ha tenido un gran número de cambios desde la idea inicial y, muchas otras ideas de red neuronal se han quedado por el camino.

El logro de clasificar imágenes con cierta precisión sí que se ha obtenido pero la precisión alcanzada aún podría mejorarse, experimentando con otras arquitecturas de redes y técnicas de optimización. No obstante, esta herramienta tiene una buena base como punto de partida para llegar a una red neuronal de detección de cáncer de pulmón.

Como se ha demostrado, el diagnóstico de cáncer de pulmón, a través de esta aplicación web, se realiza de forma rápida y sencilla y, como herramienta para un profesional médico puede ser de gran utilidad. En el ámbito médico, si se empiezan a emplear estas herramientas de soporte que utilizan una aplicación con inteligencia artificial, puede significar la reducción del porcentaje de error en el diagnóstico de este tipo de enfermedades y eso traducirse en ayudar a las personas que sufren esta enfermedad de poder tener un diagnóstico más certero ya desde las etapas iniciales de la enfermedad.

7.2 Líneas de futuro

En el futuro, en cuanto a la mejora de la aplicación, se dividirían en las referentes a la red neuronal y por otro lado, a la aplicación web:

- Por lo que hace a la red neuronal, las líneas a seguir serían las de implementar nuevos recursos informáticos para alcanzar una red neuronal perfecta, probar con computadoras más rápidas y así poder aumentar el número de simulaciones y pruebas con arquitecturas de redes más complejas.
- Respecto a la aplicación web, el trabajo seguramente se tendría que enfocar en añadir más prestaciones de velocidad, de diseño y de dar más parámetros de salida evaluando las imágenes y detallando los procesos que han llevado a una u otra clasificación de la imagen, de manera que el profesional médico tenga información del porqué de la clasificación. Esto, en un espacio tan pequeño como 512 MB no sería posible y, por lo tanto,

se debería ampliar el espacio de alojamiento web y quizás debería pensarse en crear la aplicación con librerías más avanzadas como Django en vez de Flask.

7.3 Seguimiento de la planificación

La planificación se ha respetado en casi todos los puntos menos en los apartados de mejora de la red neuronal. La validación de resultados de una red neuronal, que más allá de ser un código bien escrito, requiere de mucho tiempo para optimizar y mejorar una buena red neuronal que consiga los fines deseados.

El tiempo, por lo tanto, aunque se ha planificado correctamente, y dividido en todas las tareas necesarias, seguramente lo asignaría ahora de otro modo, dedicando más tiempo a las tareas de mejora de la red neuronal, que seguramente sea la tarea más ardua y compleja de este proyecto.

8 Glosario

Adenocarcinoma, tipo de cáncer de pulmón que comienza en las células que forman las glándulas que produce el moco.

Backend, o parte de la aplicación web a la que no puede acceder directamente un usuario, responsable de almacenar y manipular los datos procedentes del Frontend.

Cáncer de pulmón, patología que se inicia en las células del órgano pulmonar por deficiencias en el código genético de las células.

Capas completamente conectadas, capas de una red neuronal encargadas de conectar cada nodo de una capa con los nodos de la siguiente capa de la red.

Carcinoma escamoso, tipo de cáncer de pulmón que se desarrolla en las células escamosas que componen las capas media y externa de la piel del pulmón.

Data Augmentation, estrategia que permite aumentar la diversidad de los datos para entrenar un modelo, a partir de datos ya existentes, sin tener que añadir nuevos datos.

Dense, es la capa de una red neuronal que está conectada profundamente, es decir, que recibe datos de entrada de todos los nodos de la capa anterior.

Django, framework de Python que permite el desarrollo rápido y pragmático de aplicaciones web.

Dropout, capa de una red neuronal que permite desechar parte de los datos retenidos por las redes neuronales durante su entrenamiento, para evitar el sobreentrenamiento u overfit de la red.

Framework, librería o estructura de un lenguaje que contiene funciones desarrolladas para conseguir una finalidad específica.

Frontend, es la parte de la aplicación web que interactúa con los usuarios y que puede ser manejada por estos.

Heroku, servicio de alojamiento web, en parte gratuito, que acepta multitud de lenguajes informáticos como ahora Python.

HTML, o HyperText Markup Language, es un lenguaje de programación para desarrollar páginas web.

JavaScript, o JS, es un lenguaje de programación que permite añadir todo tipo de funciones, objetos y elementos web a páginas en HTML.

Keras, framework que tienen algunos lenguajes informáticos como Python o R enfocado a la creación y entrenamiento de redes neuronales convolucionales.

Max-Pooling, capa de una red neuronal que permite calcular los valores máximos y mínimos para cada característica extraída de un mapa de características.

Modelo, tipo de red neuronal que permite extraer representaciones de una imagen para conseguir una clasificación de la misma.

Overfit, o sobreentrenamiento, en estadística, es el resultado de un análisis que da por resultado un alineamiento con un mismo subconjunto de datos.

Python, lenguaje informático diseñado para todo tipo de propósitos.

Python Flask, microframework de Python que permite el desarrollo minimalista de una aplicación web.

Red neuronal convolucional, es una clase de red neuronal de aprendizaje profundo, comúnmente utilizada para analizar imágenes.

Tumor benigno, también denominado sugar tumor, es una neoplasia infrecuente, por presencia de glucógeno en las células pulmonares, generalmente asintomática y detectada de forma incidental.

Underfit, o infraentrenamiento, en estadística, es el resultado de un análisis que da no se alinea con ningún conjunto de datos por falta de experiencia en el reconocimiento de las características de los datos de entrada.

9 Bibliografía

[1] (23 de febrero de 2021). Asociación Española Contra el Cáncer. Cáncer de pulmón. Recuperado el 23 de febrero de 2021 de <https://www.aecc.es/es/todo-sobre-cancer/tipos-cancer/cancer-pulmon>

[2] Del Ciello, A., Franchi P., Contegiacomo A., Cicchetti G., Bonomo L., Larici, A. (2017). Missed lung cancer: when, where and why?

[3] Etemadi, M., et al. (2019) End-to-end cancer screening with three-dimensional deep learning on low-dose chest computed tomography. 25, 954-961. *PubMed*.

[4] Borkowski, A., Bui, M., Brannon Thomas, L., Wilson, C., DeLand, L., Mastorides, S. (Online). Lung and Colon Cancer Histopathological Image Dataset (LC2500). Disponible: https://github.com/tampapath/lung_colon_image_set/blob/master/README.md

[5] Lantz, B. (2013) Machine Learning with R.

[6] Chang, P., et al. (2018) Deep-Learning Convolutional Neural Networks Accurately Classify Genetic Mutations in Gliomas. 1201-1207. *PubMed*.

[7] Vosoglou, C. (5 de mayo de 2017). *What is the best programming language for Machine Learning?*. Towards Data Science. Recuperado el 9 de marzo de 2021 de

<https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7>

[8] Chollet, F. (2018). Deep Learning with Python.

[9] Vijay, S. (7 de enero de 2021). *Flask vs Django in 2021: Which Framework to Choose?*. Hackr.io. Recuperado el 9 de marzo de 2021 de

<https://hackr.io/blog/flask-vs-django>

[10] Barrionuevo, C. (2019). Clasificación actual del carcinoma de pulmón. Consideraciones histológicas, inmunofenotípicas, moleculares y clínicas

[11] Mitchell et al. (1986) Machine Learning: A Guide to Current Research

[12] (25 de enero de 2020). García, U. Future Lab. Introducción a las Redes Neuronales Pt. I. Recuperado el 14 de abril de 2021 de <https://medium.com/futurelabmx/introducci%C3%B3n-a-las-redes-neuronales-pt-i-a73f87933f8e>

[13] alexlenail.me Recuperado el 22 de abril de 2021 de <http://alexlenail.me/NN-SVG/index.html>

[14] Zilong et al. (2018). Deep learning for image-based cancer detection and diagnosis – A survey

[15] (12 de octubre de 2018). Sayantini, D. Edureka! How to perform data compression using autoencoders? Recuperado el 14 de abril de 2021 de <https://medium.com/edureka/autoencoders-tutorial-cfdcebde37>

[16] (14 de abril de 2021) Unipython. Reconocimiento de dígitos (números) manuscritos. Recuperado el 14 de abril de 2021 de <https://unipython.com/reconocimiento-de-digitos-numeros-manuscritos/>

[17] Zhu et al. (2016). Deep convolutional network for survival analysis with pathological images

[18] Hua et al. (2015). Computer-aided classification of lung nodules on computed tomography via deep learning technique

[19] Hussein et al. (2017). TumorNet: Lung Nodule Characterization Using Multi-View Convolutional Neural Network with Gaussian Process

[20] Setio et al. (2016). Pulmonary nodule detection in CT images: false positive reduction using multi-view convolutional networks

[21] Dou et al. (2017). Multilevel contextual 3-D CNNs for false positive reduction in pulmonary nodule detection

[22] Shen et al. (2017). Multi-crop convolutional neural networks for lung nodule malignancy suspiciousness classification

[23] Paul et al. (2016). Combining deep neural network and traditional image features to improve survival prediction accuracy for lung cancer patients from diagnostic CT

[24] Wang et al. (2017). Lung nodule classification using deep feature fusion in chest radiography

[25] Hirayama et al. (2016). Extraction of GGO candidate regions from the LIDC database using deep learning

[26] Tajbakhsh et al. (2017). Comparing two classes of end-to-end machine-learning models in lung nodule detection and classification: MTANNs vs CNNs

[27] Kim et al. (2016). Deep feature learning for pulmonary nodule classification in a lung CT

[28] Vas et al. (2017). Lung Cancer detection system using lung CT image processing

[29] (21 de abril de 2021) Bourniq, G. GitHub. CNN multi class classification gear. Recuperado el 21 de abril de 2021 de <https://github.com/gbourniq/cnn-multiclass-classification-gear>

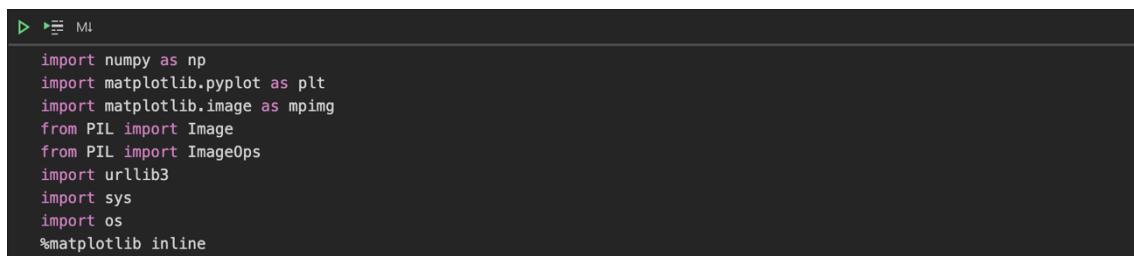
Anexos

Para elaborar este proyecto se ha tenido que crear un ambiente virtual que permita trabajar con una versión estable de Python que incluya toda la paquetería necesaria para lanzar la red neuronal y entrenarla.

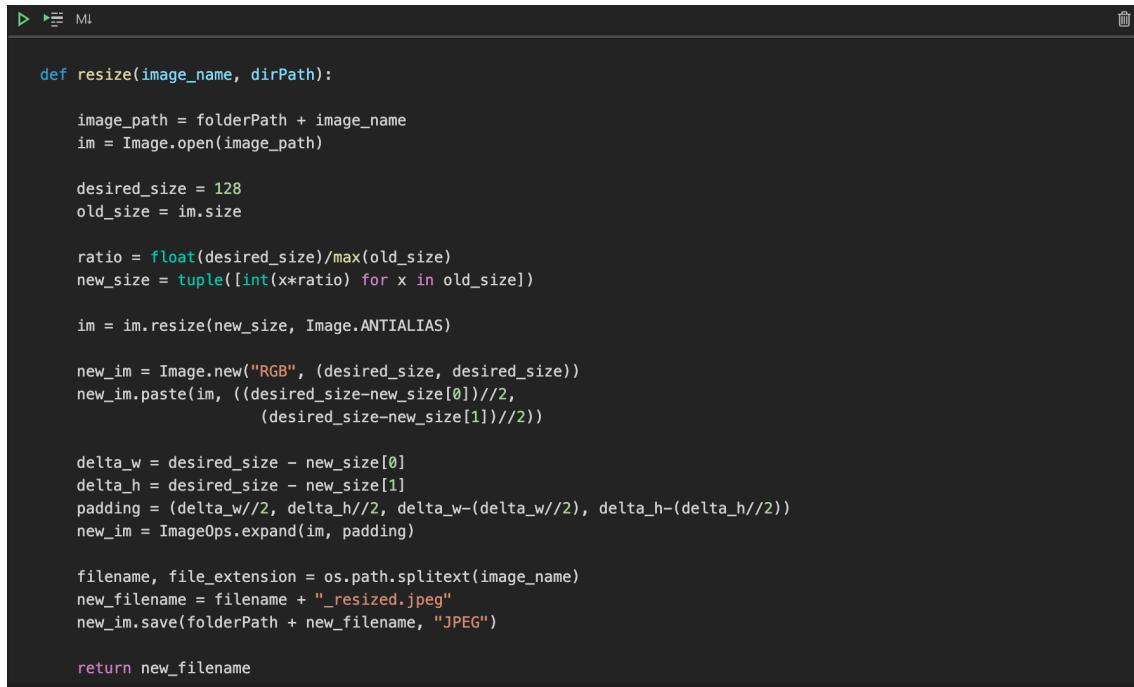
Para ello, se ha instalado miniconda y se ha creado el ambiente virtual e instalado la paquetería requerida siguiendo los siguientes pasos en el terminal del macOS:

```
~ % conda create --name lastenv python=3.8.8
~ % conda activate lastenv
~ % pip install matplotlib
~ % pip install keras
```

Una vez cargadas todas las librerías deseadas, se han definido las funciones para preprocesar las imágenes, dándoles un tamaño equitativo con la función `resize` y un mismo código de colores con la función `equalize_image`.



```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from PIL import ImageOps
import urllib3
import sys
import os
%matplotlib inline
```



```
def resize(image_name, dirPath):
    image_path = folderPath + image_name
    im = Image.open(image_path)

    desired_size = 128
    old_size = im.size

    ratio = float(desired_size)/max(old_size)
    new_size = tuple([int(x*ratio) for x in old_size])

    im = im.resize(new_size, Image.ANTIALIAS)

    new_im = Image.new("RGB", (desired_size, desired_size))
    new_im.paste(im, ((desired_size-new_size[0])//2,
                      (desired_size-new_size[1])//2))

    delta_w = desired_size - new_size[0]
    delta_h = desired_size - new_size[1]
    padding = (delta_w//2, delta_h//2, delta_w-(delta_w//2), delta_h-(delta_h//2))
    new_im = ImageOps.expand(im, padding)

    filename, file_extension = os.path.splitext(image_name)
    new_filename = filename + "_resized.jpeg"
    new_im.save(folderPath + new_filename, "JPEG")

    return new_filename
```

```

def equalize_image(image_name, dirPath):

    image_path = folderPath + image_name
    im = Image.open(image_path)

    im_out = ImageOps.equalize(im)

    filename, file_extension = os.path.splitext(image_name)
    new_filename = filename + "_equalized.jpeg"
    im_out.save(folderPath + new_filename, "JPEG")

return new_filename

```

A continuación, se han escrito las funciones para dirigir a Python al directorio de entrada de datos, que contiene tres carpetas, una para entrenamiento, una para validación y una para testeo. Hay 6000 imágenes en la carpeta de entrenamiento, 2000 para cada tumor, para validación, hay 3600 imágenes, 100 para cada tumor y para el testeo hay 6000, 2000 para cada tumor.

Del modo siguiente se han preprocessado todas las imágenes para el entrenamiento:

```

rootDir = "/Users/barnatasa/Desktop/Màster_Bioinformàtica/TFM/lung_colon_image_set/train"
directories = ['aca', 'n', 'scc']

number_files = 0

progress = 0

for directory in directories:
    folderPath = rootDir + '/' + directory + '/'
    filelist = os.listdir(folderPath)
    number_files += len(filelist)
    print('Folder: {}'.format(folderPath))

    for fname in os.listdir(folderPath):

        try:
            resized_image_name = resize(fname, folderPath)
            equalized_image_name = equalize_image(resized_image_name, folderPath)
        except Exception as e:
            print('Following exception occurred during the processing of {}: {}'.format(fname, str(e)))

        try:
            os.remove(folderPath + fname)
        except Exception as e:
            print('Exception occurred when removing {} : {}'.format(fname, str(e)))
        try:
            os.remove(resized_image_name)
        except Exception as e:
            print('Exception occurred when removing {} : {}'.format(resized_image_name, str(e)))

    progress += 1

    if progress % 100 == 0:
        print('Current progress : {}/{}'.format(str(progress), str(number_files)))

Output was trimmed for performance reasons.
To see the full output set the setting "jupyter.textOutputLimit" to 0.
...
sized.jpeg : [Errno 2] No such file or directory: 'lungsc1548_resized.jpeg'
Exception occurred when removing lungsc171_resized.jpeg : [Errno 2] No such file or directory: 'lungsc171_resized.jpeg'
Exception occurred when removing lungsc1661_resized.jpeg : [Errno 2] No such file or directory: 'lungsc1661_resized.jpeg'

```

A continuación, se han cargado las librerías que servirán para crear la red neuronal, acabar de normalizar los datos de entrada a la red y evaluar su capacidad de clasificar imágenes:

```
▶ ▷ ML
import os
import pandas as pd
import numpy as np
np.random.seed(42)
from PIL import Image
from scipy.stats import randint

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.utils import normalize
```

El siguiente paso es dirigirse a los directorios para obtener los datos que entrarán a la red neuronal, especificando cada una de las clases en que se pueden clasificar las imágenes con la función *label_encoder*.

```
▶ ▷ ML
def label_encoder(key):
    label_mapping = {
        "aca" : "1",
        "n" : "2",
        "scc" : "3",
    }
    return int(label_mapping[key])
```

```
▶ ▷ ML
rootDir = "/Users/barnatasa/Desktop/Màster_Bioinformàtica/TFM/lung_colon_image_set/train"
directories = ['aca', 'n', 'scc']

df = pd.DataFrame()
category = []
pixel_array = []

for directory in directories:
    folderPath = rootDir + '/' + directory + '/'
    print('Folder: {}'.format(folderPath))
    for fname in os.listdir(folderPath):
        if fname.endswith('resized_equalized.jpeg'):
            im = Image.open(folderPath + fname)
            im_array = np.array(im, dtype=float)

            category.append(label_encoder(directory))
            pixel_array.append(im_array)

pd_dict = {
    'pixel_array' : pixel_array,
    'category' : category,
}
df = pd.DataFrame(pd_dict)
df['category'] = pd.to_numeric(df['category'])
df = shuffle(df)

Folder: /Users/barnatasa/Desktop/Màster_Bioinformàtica/TFM/lung_colon_image_set/train/aca/
Folder: /Users/barnatasa/Desktop/Màster_Bioinformàtica/TFM/lung_colon_image_set/train/n/
Folder: /Users/barnatasa/Desktop/Màster_Bioinformàtica/TFM/lung_colon_image_set/train/scc/
```

```

▶ ▶ ML
print(df.shape)
df.info()
df.head()

(6000, 2)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 1782 to 860
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   pixel_array  6000 non-null   object  
 1   category     6000 non-null   int64  
dtypes: int64(1), object(1)
memory usage: 140.6+ KB

      pixel_array  category
1782  [[78.0, 113.0, 106.0], [95.0, 126.0, 128.0], ...
3917  [[47.0, 44.0, 27.0], [42.0, 25.0, 80.0], [165...
221   [[136.0, 127.0, 182.0], [162.0, 154.0, 203.0]...
2135  [[209.0, 194.0, 235.0], [203.0, 158.0, 189.0]...
5224  [[114.0, 126.0, 104.0], [166.0, 177.0, 163.0]...

```

El siguiente paso consiste en dividir los datos de entrada en distintos arrays de entrenamiento y testeo, según los datos de entrada y las categorías a las que pertenecen cada conjunto de datos.

```

▶ ▶ ML
X_train, X_test, y_train, y_test = train_test_split(df['pixel_array'].values.tolist(),
                                                    df['category'].values.tolist(),
                                                    test_size = 0.2,
                                                    random_state=42,
                                                    stratify=df['category'])

X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

```

```

▶ ▶ ML
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(4800, 128, 128, 3)
(1200, 128, 128, 3)
(4800,)
(1200,)

```

```

▶ ▶ ML
X_train = X_train.reshape(X_train.shape[0], X_train.shape[3], X_train.shape[1], X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[3], X_test.shape[1], X_test.shape[2])

```

```

▶ ▶ ML
print(X_train.shape)
print(X_test.shape)

(4800, 3, 128, 128)
(1200, 3, 128, 128)

```

La siguiente tarea ha sido normalizar estos datos para que todos los datos estén reescalados y entren en una base de renglo [0,1] con *normalize()*.

```
▶ ▶ ML
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = normalize(X_train)
X_test = normalize(X_test)
```

```
▶ ▶ ML
Y_train = np_utils.to_categorical(y_train, 4)[:,1:]
Y_test = np_utils.to_categorical(y_test, 4)[:,1:]
print(Y_train.shape)
for i in range(0,10):
    print('image ' + str(i+1) + ': ' + str(Y_train[i]))
(4800, 3)
image 1: [0. 0. 1.]
image 2: [0. 0. 1.]
image 3: [0. 1. 0.]
image 4: [1. 0. 0.]
image 5: [0. 0. 1.]
image 6: [1. 0. 0.]
image 7: [0. 1. 0.]
image 8: [1. 0. 0.]
image 9: [0. 0. 1.]
image 10: [0. 0. 1.]
```

Por último, ahora sí, se ha creado la arquitectura del modelo, que se muestra a continuación:

```
▶ ▶ ML
model = Sequential()

▶ ▶ ML
model.add(Convolution2D(32, 1, 1, activation='relu', input_shape=(3,128,128)))

model.add(Convolution2D(32, 1, 1, activation='relu'))
model.add(MaxPooling2D(pool_size=(1,1)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

```
▶ ▶ ML
print(model.input_shape)
print(model.output_shape)

(None, 3, 128, 128)
(None, 3)
```

Para entrenar el modelo, primero ha habido que determinar el tipo de función de pérdida, en nuestro caso, *categorical_crossentropy*, al tratarse de un modelo multicase, un optimizador, *adam*, y una métrica, que será precisión (*accuracy*).

Con la función *model.fit* procedemos a entrenar el modelo:

```
▶ ▶ ML
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```

  model.fit(X_train, Y_train, epochs=20)

Epoch 8/20
150/150 [=====] - 3s 19ms/step - loss: 0.2089 - accuracy: 0.9210
Epoch 9/20
150/150 [=====] - 3s 19ms/step - loss: 0.1838 - accuracy: 0.9245
Epoch 10/20
150/150 [=====] - 3s 19ms/step - loss: 0.1420 - accuracy: 0.9495
Epoch 11/20
150/150 [=====] - 3s 19ms/step - loss: 0.1530 - accuracy: 0.9439
Epoch 12/20
150/150 [=====] - 3s 19ms/step - loss: 0.1256 - accuracy: 0.9519
Epoch 13/20
150/150 [=====] - 3s 19ms/step - loss: 0.1198 - accuracy: 0.9556
Epoch 14/20
150/150 [=====] - 3s 18ms/step - loss: 0.1089 - accuracy: 0.9609
Epoch 15/20
150/150 [=====] - 3s 19ms/step - loss: 0.0841 - accuracy: 0.9738
Epoch 16/20
150/150 [=====] - 3s 18ms/step - loss: 0.0601 - accuracy: 0.9789
Epoch 17/20
150/150 [=====] - 3s 18ms/step - loss: 0.0752 - accuracy: 0.9743
Epoch 18/20
150/150 [=====] - 3s 18ms/step - loss: 0.0681 - accuracy: 0.9793
Epoch 19/20
150/150 [=====] - 3s 18ms/step - loss: 0.0605 - accuracy: 0.9812
Epoch 20/20
150/150 [=====] - 3s 17ms/step - loss: 0.0613 - accuracy: 0.9797

<tensorflow.python.keras.callbacks.History at 0x7fa69ac290d0>

```

Por último, se ha accedido a evaluar el modelo entrenado, para ver si la función de pérdida y precisión ha sido la deseada:

```

  val_loss, val_acc = model.evaluate(X_test, Y_test)
  print(model.metrics_names)
  print(val_loss, val_acc)

38/38 [=====] - 0s 4ms/step - loss: 0.4850 - accuracy: 0.8558
['loss', 'accuracy']
0.48499807715415955 0.8558333516120911

```

Como se puede observar, la precisión de entrenamiento y validación está por encima del 85% y la función de pérdida es aceptable.

El modelo se da por válido y se guarda para utilizarlo posteriormente en el test o clasificación de imágenes final.

```

  model.save('gear_classifier_2.model')

INFO:tensorflow:Assets written to: gear_classifier_2.model/assets

```

Se ha probado si la clasificación de imágenes de los directorios de entrada de test funciona correctamente y con qué precisión el modelo es capaz de relacionar cada imagen con su clase o tipo de tumor.

Para ello, se ha creado una matriz de confusión, en función de las predicciones obtenidas para las imágenes testeadas y las predicciones reales para cada tipo de tumor, luego de procesar un poco los datos:

```

  from keras.models import load_model
  from sklearn import metrics
  from sklearn.metrics import confusion_matrix
  model = load_model('gear_classifier_2.model')
  predictions = model.predict(X_test)
  y_pred = (predictions > 0.5)

```

```
▶ ▶ ML
a = []

for i in y_test:
    if i == 1:
        a.append([1,0,0])
    elif i == 2:
        a.append([0,1,0])
    else:
        a.append([0,0,1])
```

```
▶ ▶ ML
import numpy as np
from sklearn.metrics import multilabel_confusion_matrix
multilabel_confusion_matrix(y_pred.astype(int), a)

array([[724,  81,
       [ 76, 319]],

       [[776,   15],
        [ 24, 385]],

       [[729,   80],
        [ 71, 320]]])
```

Como podemos comprobar, el algoritmo parece que funciona correctamente.

De las 3600 imágenes testeadas se han conseguido los siguientes parámetros:

Clase de tumor	Aciertos	Fallos	Precisión
Adenocarcinoma (aca)	1043	157	86,92%
Benigno (n)	1161	39	96,75%
Escamoso (scc)	1049	151	87,42%
TOTAL	3253	347	90,36%

Como se puede observar, se ha acertado en la predicción del tumor en 3253 imágenes y se ha fallado en 347 casos.

Como se puede comprobar con los resultados obtenidos, la precisión obtenida para el modelo creado es del 90,36%.

Para acometer mejoras en la red neuronal, se ha introducido una nueva capa de eliminación de outliers en el modelo para conseguir la reducción de la pérdida del modelo.

```

from keras.constraints import maxnorm

model.add(Convolution2D(32, 1, 1, activation='relu', input_shape=(3,128,128)))

model.add(Convolution2D(32, 1, 1, activation='relu'))
model.add(MaxPooling2D(pool_size=(1,1)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_constraint=maxnorm(2)))
model.add(Dropout(0.5))

model.add(Dense(3, activation='softmax'))

```

Además, se ha añadido un validation_split para acometer el entrenamiento a la vez que la validación de las imágenes.

```

model.fit(X_train, Y_train, epochs=20, validation_split=0.1)

135/135 [=====] - 3s 22ms/step - loss: 0.3499 - accuracy: 0.8491 - val_loss: 0.4465 - val_accuracy: 0.8125

```

Con eso, se ha reducido el val_loss hasta 0.35.

```

val_loss, val_acc = model.evaluate(X_test, Y_test)
print(model.metrics_names)
print(val_loss, val_acc)

38/38 [=====] - 0s 4ms/step - loss: 0.3578 - accuracy: 0.8383
['loss', 'accuracy']
0.3578004837036133 0.8383333086967468

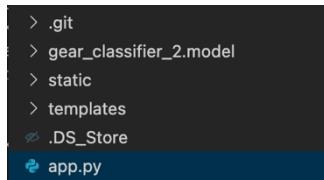
```

De las 3600 imágenes testeadas nuevamente se han conseguido los siguientes parámetros:

Clase de tumor	Aciertos	Fallos	Precisión
Adenocarcinoma (aca)	1029	172	85,75%
Benigno (n)	1154	46	96,17%
Escamoso (scc)	1033	167	86,08%
TOTAL	3216	385	89,33%

Aunque se ha bajado la precisión, se ha conseguido que el modelo se adapte mejor a imágenes distintas a las usadas para el entrenamiento del modelo con la reducción del val_loss.

Para configurar una web con Python Flask se necesita una estructura de directorios que incluya la aplicación en Python (algoritmo con la red neuronal) en la carpeta principal (*main*), una carpeta *static* donde quedarán almacenadas las imágenes subidas y una carpeta *templates* donde se incluya el frontend con los documentos HTML.



Dentro de la aplicación app.py, se conecta a través del código @app.route a los archivos .htm que nos interesan y es imprescindible tener creada la clase *main* (nuestra página web) configurar un código secreto de seguridad para el funcionamiento correcto de la aplicación.

A través del terminal de nuestro MAC OS, se ha servido la web localmente mediante el siguiente código:

python app.py

Esto ha producido un enlace interno al localhost del PC que ha permitido comprobar errores y testear la página web.

El repositorio en GitHub se ha subido creando una carpeta independiente con todos los archivos necesarios para el funcionamiento de la aplicación.

A continuación, en el terminal del pc, se han introducido las líneas de código siguientes:

1. Dirigirse al directorio donde se encuentran los archivos de la web, en este caso, la carpeta se llama “Nou”:

cd /Users/barnatasa/Desktop/Nou

2. Crear un archivo “README” e iniciar la subida de archivos al repositorio GitHub:

```
echo "# dcapi2" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/costero-e/dcapi2.git
git push -u origin main
```

Este repositorio se puede visitar de forma libre a través de este enlace web:

<https://github.com/costero-e/dcapi2>

La siguiente tarea ha sido crear los archivos que permitirán al sitio alojamiento web, reconocer las versiones del lenguaje y de las librerías empleadas para el correcto funcionamiento de nuestra aplicación. Para ello se han seguido los siguientes pasos:

1. Crear un archivo Procfile (sin extensiones) que llame a un servidor que sea compatible con Python Flask. En este caso, se ha utilizado gunicorn. Este archivo Procfile sólo contiene la información siguiente:

web: gunicorn app:app

2. Cargar el ambiente virtual con el que se ha trabajado la aplicación. En nuestro caso, lo creamos primeramente con miniconda así:

conda create – name lastenv python=3.8.8

Y luego lo activamos así:

conda activate lastenv

3. Crear el archivo que contendrá todas las librerías con las versiones exactas que se requieren para nuestra aplicación y que se fueron instalando progresivamente a medida de que eran requeridas en nuestro ambiente virtual cerrado *lastenv*. Para ello, escribimos el código siguiente:

pip freeze > requirements.txt

Esto permite crear un archivo con todas las librerías requeridas y su versión. Este archivo contiene la siguiente información:

**absl-py==0.11.0
astunparse==1.6.3
cachetools==4.2.1
certifi==2020.12.5
chardet==4.0.0
click==7.1.2
cyclere==0.10.0
Flask==1.1.2**

Flask-WTF==0.14.3
flatbuffers==1.12
gast==0.3.3
google-auth==1.27.0
google-auth-oauthlib==0.4.2
google-pasta==0.2.0
grpcio==1.32.0
gunicorn==20.0.4
h5py==2.10.0
idna==2.10
itsdangerous==1.1.0
Jinja2==2.11.3
joblib==1.0.1
Keras==2.4.3
Keras-Preprocessing==1.1.2
kiwisolver==1.3.1
Markdown==3.3.4
MarkupSafe==1.1.1
matplotlib==3.3.4
numpy==1.19.5
oauthlib==3.1.0
opt-einsum==3.3.0
packaging==20.9
Pillow==8.1.1
protobuf==3.15.3
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparsing==2.4.7
python-dateutil==2.8.1
PyYAML==5.4.1
requests==2.25.1
requests-oauthlib==1.3.0
rsa==4.7.2
scikit-learn==0.24.1
scipy==1.5.4
six==1.15.0
tensorboard==2.4.1
tensorboard-plugin-wit==1.8.0
tensorflow-cpu==2.4.1
tensorflow-estimator==2.4.0
termcolor==1.1.0

```
threadpoolctl==2.1.0
typing-extensions==3.7.4.3
urllib3==1.26.3
Werkzeug==1.0.1
wheel==0.35.0
wrapt==1.12.1
WTForms==2.3.3
```

Una vez creados estos archivos, se vuelven a subir al repositorio GitHub, desde el terminal, con el siguiente código:

```
git add .
git commit -am "second commit"
git push -u origin main
```

4. Crear el dominio en Heroku y subir el repositorio al dominio. Una vez creada la cuenta en Heroku nos hemos dirigido a crear la nueva app y le hemos dado el nombre *dcapi2*.

Una vez realizado este paso, en el terminal de nuestro pc, hemos escrito el siguiente código:

```
heroku login
cd /Users/barnatasa/Desktop/Nou
git init
heroku git:remote -a dcapi2
git add .
git comit -am "adding files"
git push heroku master
```

5. Crear el dominio en Heroku y subir el repositorio al dominio. Una vez creada la cuenta en Heroku nos hemos dirigido a crear la nueva app y le hemos dado el nombre *dcapi2*.

Create New App

App name

Choose a region

 Europe



[Add to pipeline...](#)

[Create app](#)