Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE VALLADOLID

Trabajo de Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Detección y delimitación de hepatocarcinomas basándose en análisis de imagen mediante IA

Alumno: Diego Rodríguez Arroyo

Tutores: Teodoro Calonge Cano
Raúl García Pajares
Adrián Sánchez Zapico

Agradecimientos

Me gustaría expresar mi agradecimiento a mis tutores, Teodoro Calonge Cano, Raúl García Pajares y Adrián Sánchez Zapico, por haberme brindado la oportunidad de realizar este proyecto, así como por su paciencia y apoyo durante todo su desarrollo. Sus orientaciones y seguimiento han sido clave a lo largo del proceso.

También quiero agradecer a los profesionales del hospital su colaboración y su disposición para ayudarme y resolver mis dudas. Gracias a todos ellos, este trabajo ha sido una gran oportunidad de aprendizaje.

Finalmente, doy las gracias a quienes me han acompañado durante estos años de formación, tanto dentro como fuera del ámbito académico. Su apoyo ha sido fundamental para afrontar los momentos más exigentes.

Resumen

La Inteligencia Artificial (IA) ha experimentado un gran avance en los últimos años, siendo aplicada en distintos ámbitos y permitiéndoles evolucionar gracias a su capacidad para resolver tareas complejas.

Este proyecto estudia la clasificación automática de ecografías abdominales, prestando especial atención al hígado y al hepatocarcinoma, mediante modelos de Redes Neuronales Convolucionales. Para este propósito, se recopilaron y procesaron imágenes de pacientes reales, aplicándoles prepocesamiento y distintas técnicas de aumento de datos. Por otro lado, se ha desarrollado una aplicación para la clasificación de estas ecografías. Esta aplicación incluye visualizaciones de cómo los modelos toman las decisiones y pretende facilitar el acceso a usuarios sin experiencia.

Los resultados obtenidos tras la evaluación de los modelos, mostraron que estos lograron aprender para imágenes ya conocidas; sin embargo, demostraron tener aptitud más limitada para la generalización en muestras nunca antes vistas. Este sesgo está estrechamente relacionado con el desequilibrio en las muestras, destacando los insuficientes casos de hepatocarcinoma con respecto a hígados sanos.

Abstract

Artificial Intelligence (AI) has experienced significant progress in recent years, being applied across various fields and enabling their advancement thanks to its ability to solve complex tasks.

This project explores the automatic classification of abdominal ultrasound images, focusing particularly on the liver and the hepatocellular carcinoma, using Convolutional Neural Network (CNN) models. For this purpose, real patient images were collected and processed, applying preprocessing and different data augmentation techniques. Additionally, a web application has been developed for the classification of these ultrasounds. This application includes visualizations of how models make decisions and is intended to facilitate use for users without technical expertise.

The evaluation results showed that the models were able to learn effectively on images they had previously seen. However, their ability to generalize to unseen samples was more limited. This bias is closely related to the imbalance in the dataset, highlighting the insufficient number of hepatocellular carcinoma cases compared to healthy liver images.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La Inteligencia Artificial (IA) ha experimentado un gran desarrollo en las últimas décadas, transformando numerosos campos y permitiéndoles avanzar gracias a su capacidad para resolver tareas complejas de manera automatizada. Entre sus múltiples ramas, los modelos de Aprendizaje Profundo han demostrado un alto rendimiento en tareas de clasificación y detección de patrones, especialmente cuando son expuestos a grandes cantidades de datos y entrenamiento [1].

Dentro de este ámbito, la visión por ordenador permite a las máquinas interpretar imágenes con una precisión cada vez más cercana a la humana. En el campo de la Medicina, estas tecnologías han comenzado a desempeñar un papel crucial, asistiendo en diagnósticos, pronósticos y decisiones clínicas, especialmente en el análisis de imágenes médicas como radiografías, resonancias o ecografías [2].

1.1. Contexto

Para introducir el marco anatómico del proyecto, primero se debe presentar la región abdominal humana, la cual incluye diferentes órganos como el bazo, el colon, el estómago, el hígado, el intestino, el páncreas, la vesícula, el apéndice y la vejiga, figura 1.1.

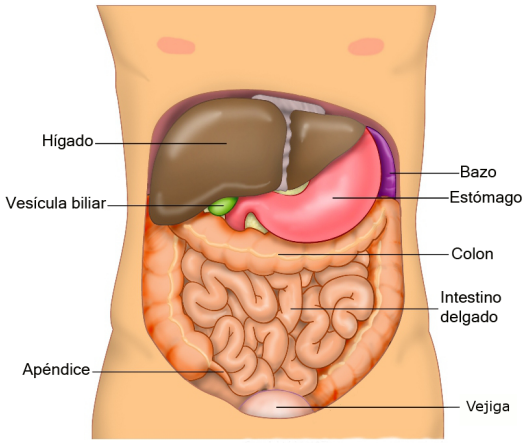

Figura 1.1: Anatomía del abdomen humano, de Ties van Brussel/tiesworks.nl
Fuente: `https://commons.wikimedia.org/w/index.php?curid=46003785`

A continuación, nos enfocaremos en el estudio del hígado, órgano de alta relevancia para este proyecto, puesto constituye el objetivo de estudio principal del mismo.

1.1.1. Hígado

Para contextualizar el proyecto, es importante destacar la relevancia del hígado como uno de los órganos vitales más importantes del cuerpo humano. Situado en la parte superior derecha del abdomen, justo debajo del diafragma, este órgano cumple funciones esenciales para el mantenimiento del equilibrio fisiológico.

Entre sus principales responsabilidades se encuentran la metabolización de nutrientes, la síntesis de proteínas, la producción de bilis y, especialmente, la desintoxicación de sustancias nocivas presentes en la sangre, como el alcohol u otros compuestos potencialmente perjudiciales. Su correcto funcionamiento es indispensable para la vida, y cualquier alteración significativa en su estructura o funcionalidad puede comprometer gravemente la salud del individuo. Dentro del ámbito de este proyecto, se prestará especial atención a dos patologías del hígado de gran relevancia clínica: la cirrosis hepática y el hepatocarcinoma.

Cirrosis Hepática

La cirrosis hepática es una de las enfermedades crónicas del hígado más graves y frecuentes. Se caracteriza por la progresiva sustitución del tejido hepático sano por tejido cicatricial como consecuencia de la muerte celular[3]. Este proceso deteriora la estructura del órgano y afecta severamente a su funcionalidad.

Las causas más comunes de esta afección incluyen el consumo excesivo de alcohol, infecciones virales como las hepatitis B y C, así como ciertas enfermedades metabólicas. Si no se detecta y trata a tiempo, la cirrosis puede avanzar hasta causar complicaciones potencialmente mortales. Constituye un importante factor de riesgo para el desarrollo de hepatocarcinoma, lo que acentúa la importancia de su diagnóstico temprano y seguimiento clínico.

Hepatocarcinoma

También conocido como carcinoma hepatocelular (CHC), es el tipo de cáncer de hígado más frecuente, representando entre el 80 % y el 90 % de los tumores hepáticos malignos. En aproximadamente el 90 % de los casos, su aparición está estrechamente relacionada con la cirrosis hepática, como resultado de la acumulación progresiva de tejido cicatricial en el hígado.

Una de las principales dificultades en su manejo clínico reside en que esta patología era casi indetectable en fases iniciales, lo que históricamente ha llevado a que muchos diagnósticos se realizaran en etapas avanzadas, cuando el tumor ya había alcanzado un tamaño considerable y las opciones terapéuticas curativas se habían reducido notablemente. No obstante, los avances en las técnicas de imagen, como la ecografía abdominal, y la inclusión de pacientes de alto riesgo en programas de seguimiento intencionado han mejorado significativamente la detección precoz del CHC. Este diagnóstico temprano resulta clave para permitir la aplicación de tratamientos con intención curativa y mejorar exponencialmente el pronóstico de los pacientes[4].

1.1.2. Ecografía

La ecografía es una técnica de diagnóstico por imagen no invasiva ampliamente utilizada en Medicina para examinar órganos y estructuras internas, entre ellos el hígado. Se basa en el uso de ultrasonidos (ondas sonoras de alta frecuencia) que, al rebotar en los tejidos del cuerpo, permiten generar imágenes en tiempo real.

A diferencia de otras pruebas como la radiografía o la tomografía computarizada, la ecografía no emplea radiación ionizante, lo que la convierte en una técnica inocua, accesible y de bajo coste. Por estas razones, es una de las principales herramientas para la evaluación hepática, especialmente útil en la detección de anomalías o signos de cirrosis [5]. A continuación, en la figura 1.2, se presenta un ejemplo.

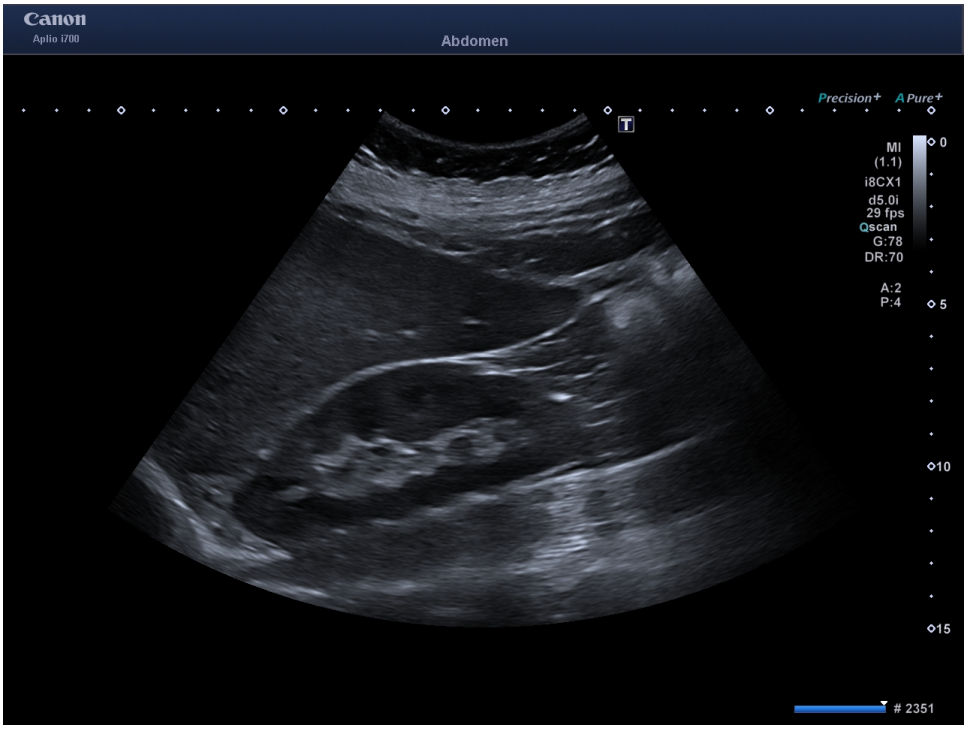

Figura 1.2: Ecografía abdominal donde se puede observar un riñón y a su izquierda, parcialmente, un hígado.

La ecografía es una prueba dinámica que depende en gran medida de la habilidad y experiencia del profesional que la realiza. El transductor, dispositivo que se coloca en contacto con el cuerpo del paciente, emite y recibe los ultrasonidos mientras se desplaza sobre la piel; por ello, la calidad y contenido de las imágenes obtenidas pueden variar considerablemente según la orientación, posición y movimiento de este, lo que introduce una variabilidad en la interpretación de los resultados. Este factor subjetivo representa una limitación importante, especialmente en contextos clínicos donde se requiere de una alta precisión diagnóstica.

Normalmente, la ecografía genera una secuencia de imágenes en escala de grises y generalmente en forma de sector circular, una de las modalidades más utilizadas se conoce como "*brightness mode*" (modo B). Sin embargo, existen otras más avanzadas como la ecografía dópler, que permite visualizar la circulación del flujo sanguíneo, incluyendo su velocidad y dirección, mediante codificación por colores. Esta técnica resulta particularmente útil para evaluar la vascularización del hígado y detectar posibles alteraciones asociadas a patologías como el hepatocarcinoma [6].

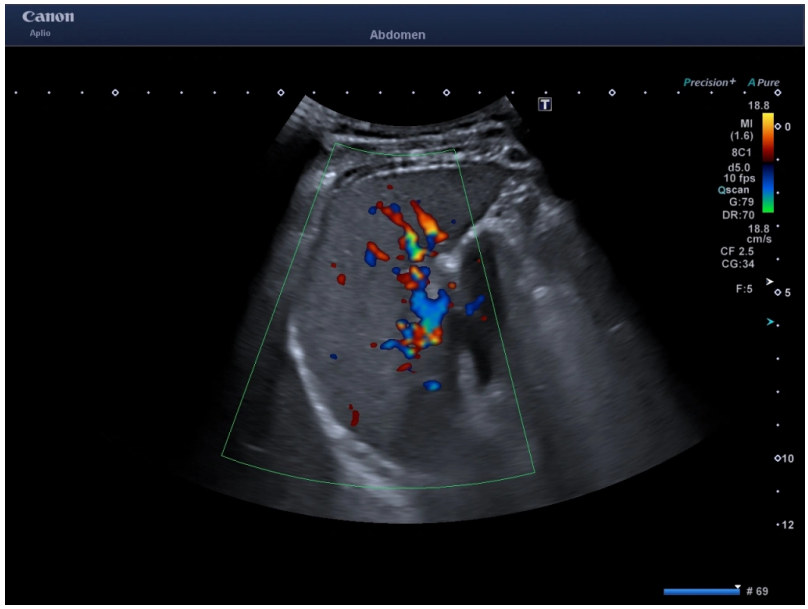

Figura 1.3: Ecografía abdominal de un bazo con técnica dópler.
Fuente `https://ecografiafacil.com/2023/06/21/estudio-ecografico-del-bazo-con-smi-de-canon-medical-sistem-los-colores-del-bazo/`

En este contexto, la aplicación de modelos de Inteligencia Artificial puede aportar un valor añadido significativo, al permitir una interpretación automatizada y estandarizada de las ecografías, reduciendo la dependencia del operador humano y mejorando la precisión diagnóstica.

1.2. Estado del Arte

Para llevar acabo este proyecto se han revisado diversos trabajos cuyo contenido aborda la aplicación de Redes Neuronales Convolucionales para el diagnóstico de enfermedades hepáticas mediante ecografías.

En el estudio de Mitrea et al. (2023) [7] se presenta un sistema híbrido que combina técnicas convencionales, como texturas, histogramas y transformaciones de ondícula con modelos de aprendizaje automático.

Emplearon imágenes de ecografías Modo B de dos ecógrafos diferentes, consiguiendo un total de 296 pacientes enfermos, ningún caso con hígado sano. Definieron dos clases, HCC y cirrótico. De esas imágenes, se seleccionaron manualmente trozos de tamaños 50x50 y 56x56 píxeles, de zonas con tumor o afectadas por cirrosis y se utilizaron para entrenar los

modelos. Todas estas imágenes se pueden encontrar en el siguinete enlace [8]. En su caso no se desarrolló ninguna aplicación para la clasificación de ecografías.

Otro estudio, Byra et al. (2019) [9], utiliza el modelo de Red Convolucional *Inception-ResNet-v2*, preentrenada en el conjunto de datos *ImageNet*, para determinar el nivel de esteatosis en hígados grasos. Utilizaron 550 ecografías Modo B, con resolución 434x636 píxeles de 55 pacientes. Estas imágenes contenían, en su totalidad, hígados junto a riñones, y se seleccionaron manualmente regiones de interés, finalmente no se realizó una aplicación de medición de esteatosis.

1.3. Planteamiento y Objetivos

Este proyecto se centra en el desarrollo de un sistema de clasificación automática de ecografías abdominales mediante el uso de técnicas de Inteligencia Artificial, específicamente Redes Neuronales Convolucionales (CNN). El objetivo principal es construir una aplicación capaz de clasificar imágenes de ecografías del hígado. Para esto será necesario:

- Conseguir un número suficiente de ecografías de la región abdominal.
- Desarrollar y entrenar modelos de visión por ordenador capaces de realizar tareas de clasificación.
- Poder visualizar la toma de decisiones del modelo mientras realiza la clasificación, pues esto es muy deseable en el diagnóstico médico.
- Desarrollar una aplicación que permita clasificar una ecografía, devolver el resultado y mostrar la visualización de dicha clasificación.

La elección de la ecografía, como técnica de imagen, se debe a su amplia disponibilidad, seguridad y bajo coste, lo que la convierte en una herramienta importante para el cribado de pacientes y el seguimiento de enfermedades hepáticas. No obstante, su realización e interpretación requiere personal especializado y puede resultar subjetivo y costoso en términos de tiempo, por lo que automatizar este proceso mediante modelos de aprendizaje profundo representa una solución tecnológica de gran valor clínico.

A diferencia de otros enfoques de segmentación, este proyecto se centra exclusivamente en una tarea de clasificación; es decir, no se busca identificar la localización exacta de una lesión o patología, sino simplemente detectar su presencia a partir de la imagen completa. De este modo, se explora el potencial de las CNN para apoyar el diagnóstico médico de manera eficiente, contribuyendo así al desarrollo de herramientas inteligentes en el ámbito de la salud.

1.4. Estructura

La estructura de capítulos de este trabajo de fin de grado es la siguiente:

- Capitulo 1: Introducción. Se contextualiza el problema, se presentan las patologías hepáticas abordadas y se define el objetivo general del trabajo.
- Capitulo 2: Gestión del proyecto. Se detalla la planificación, organización y control del proyecto, incluyendo la distribución temporal de tareas para asegurar su correcta ejecución.
- Capitulo 3: Fundamento Teórico. Se explican los conceptos fundamentales de la Inteligencia Artificial utilizados, haciendo especial énfasis en las redes convolucionales (CNN).
- Capitulo 4: Conjunto de datos. Se describe el proceso de obtención, organización, preprocesamiento y tratamiento del conjunto de datos ecográficos empleado para entrenar y evaluar los modelos.
- Capitulo 5: Diseño del sistema. Se expone la arquitectura general del sistema desarrollado, incluyendo el diseño de los modelos, la selección de parámetros y la implementación de la lógica de clasificación.
- Capitulo 6: Evaluación y resultados. Se presentan los resultados obtenidos tras el entrenamiento de los modelos, así como una evaluación cuantitativa de su rendimiento mediante métricas específicas de clasificación.
- Capitulo 7: Aplicación y despliegue. Se muestra cómo se ha integrado el modelo en una aplicación funcional, junto con su interfaz y características principales para el uso clínico o experimental.
- Capitulo 8: Conclusiones y trabajo futuro. Se resumen los principales logros del proyecto, se valoran las limitaciones encontradas y se proponen posibles líneas de mejora y continuación futura del trabajo.

Capítulo 2

Gestión del proyecto

En este capítulo se recoge todo lo relativo a la gestión del proyecto y se desarrollan aspectos como la metodología o los recursos utilizados para la correcta evolución de este TFG. Para tal fin se ha utilizado GitLab, que posee tanto herramientas para control de versiones, como organización con hitos y tableros que permiten realizar *sprints*, planificar tareas y supervisar el progreso; así como compartirlo con otras personas.

2.1. Metodología de trabajo

Para organizar el desarrollo del proyecto se optó por seguir una metodología similar a *Scrum*. Esta forma de planificación estructura el trabajo en cinco fases que orientan el proceso [10]:

1. **Inicio**: En esta fase se define el enfoque que se quiere dar al proyecto y los objetivos del mismo.

2. **Planificación y estimación**: Aquí se seleccionan las tareas prioritarias del *backlog* para incluirlas en el siguiente *sprint* y se define un objetivo claro para el mismo.

3. **Implementación**: En esta fase se desarrolla el trabajo planificado. Cada *sprint* suele durar entre una y cuatro semanas (en el caso de este proyecto, fueron de dos), y durante su transcurso se pretende completar las tareas establecidas.

4. **Revisión y retrospectiva**: Esta fase tiene lugar al final de cada *sprint*, donde se realiza una revisión y se presentan los avances alcanzados. Después, se lleva a cabo una retrospectiva, en la que se evalúan los resultados y posibles mejoras.

5. **Lanzamiento**: En esta fase se entregan los resultados finales del proyecto. En nuestro caso incluye la aplicación, junto a los modelos de IA desarrollados y la documentación por escrito.

Como se ha mencionado antes, la metodología de trabajo empleada se ha basado en *sprints* de dos semanas de duración, al final de los cuales se realizaban reuniones con los tutores para presentar los avances, evaluar el trabajo y resolver dudas.

Para la organización de tareas se han utilizado tres tableros:

- *Backlog*, donde se añadían las tareas según se iban definiendo.
- *Open*, donde se arrastraban las tareas a realizar durante el *sprint* en cuestión.
- *Closed*, donde se colocaban las tareas una vez finalizadas.

Al inicio de cada *sprint* se revisaban las tareas del *Backlog* y se añadían nuevas si era necesario. Después, se seleccionaban las que se abordarían en el *sprint* y se movían a *Open*, las no finalizadas de *sprints* anteriores se trasladaban al nuevo.

En total, se han llevado a cabo ocho *sprints*; del *Sprint* 0, que comenzó el 29/01/2025, hasta el *Sprint* 8 que marca la finalización del proyecto. Esto ha supuesto unas 350 horas de trabajo entre recogida de imágenes, desarrollo y realización de la memoria. Además, se redactó un *Abstract*, recogido en el anexo D, y una presentación junto al profesional sanitario, que fueron presentado en el congreso anual del ACYLHE [11], la asociación castellano-leonesa de hepatología.

2.2. Control de versiones

Para la gestión, almacenamiento y control de versiones del código desarrollado durante el proyecto, se ha hecho uso de la herramienta GitLab.

El desarrollo se ha realizado principalmente a través de una rama de desarrollo (dev), donde se desarrollaba el código. Una vez obtenida la versión final, todos los cambios se unificaban en la rama principal, de esta manera resulta más sencillo mantener esta sección limpia.

Este sistema de organización permite gestionar más eficazmente el avance de todo el proyecto, además de realizar cambios sin arriesgar la versión final y; por último, tener un seguimiento claro de los cambios realizados a lo largo del desarrollo del trabajo.

Capítulo 3

Fundamento Teórico

En este capítulo se abordan los conceptos teóricos en los que se basa el desarrollo del sistema propuesto, de cara a contextualizar el uso de técnicas de Inteligencia Artificial en la rama del análisis de imágenes médicas.

3.1. Visión por computador en salud

La Inteligencia Artificial (IA) en el ámbito de la Medicina se basa en el desarrollo de algoritmos capaces de analizar grandes volúmenes de datos clínicos con el objetivo de detectar patologías, predecir riesgos o asistir a los profesionales a realizar sus diagnósticos. Su aplicación permite mejorar la precisión en la toma de decisiones y acelerar diversos procesos, convirtiéndose en una herramienta complementaria de gran valor para los profesionales de la salud [12].

Uno de los ámbitos donde el avance ha resultado más significativo es en el campo de la radiología, donde la digitalización de la toma de imágenes ha abierto la puerta a la utilización de modelos de aprendizaje automático que, junto a grandes conjuntos de datos, acompañados de su diagnóstico, permite entrenar IAs capaces de sugerir automáticamente un diagnóstico a partir de una imagen que no habían visto previamente [13]. Aunque esta práctica resulta de gran ayuda al trabajo de los radiólogos, aún no es capaz sustituir el diagnóstico del profesional en cuestión, pero sí apoyarlo para realizar una labor más eficiente.

Sin embargo, actualmente esto solo se aplica a diagnósticos específicos como la detección de fracturas en radiografías o de nódulos en mamografías, ya que aún presentan limitaciones a la hora de generalizar para diagnósticos más complejos.

En el campo de la visión por computador, las Redes Neuronales profundas han demostrado unos resultados comparables al de expertos humanos en determinadas áreas, como la detección del cáncer de mama, lesiones pulmonares o cerebrales [14]. Además de su precisión,

otra ventaja clave de la IA es su capacidad para procesar volúmenes masivos de imágenes médicas de forma rápida, ayudando a reducir la carga de trabajo del personal sanitario y destacando automáticamente aquellas regiones de interés clínico.

3.2. Redes Neuronales Artificiales

Son modelos computacionales de Inteligencia Artificial, encuadrados dentro del Aprendizaje Automático y que se basan en la estructura y el funcionamiento del cerebro.

Están compuestas por unidades elementales llamadas neuronas artificiales, que imitan, de forma simplificada, el comportamiento de las neuronas biológicas. Estas neuronas se interconectan entre sí, lo que permite que, al trabajar en conjunto, puedan modelar comportamientos complejos a partir de unidades individuales simples.

Por lo general, las neuronas se organizan en capas: la de entrada, que recibe los datos iniciales; la de salida, que genera el resultado final; y entre ellas, las ocultas, donde se realizan múltiples transformaciones intermedias. Esta estructura puede observarse en la figura 3.1.

Las neuronas de una capa están conectadas con las de las capas contiguas, por lo que se influencian unas a otras. El grado de influencia viene dado por los pesos de la red.

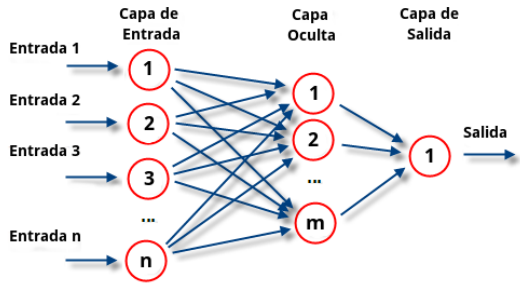

Figura 3.1: Capas de una red neuronal artificial
Fuente: `https://atriainnovation.com/blog/que-son-las-redes-neuronales-y-sus-funciones/`

Neurona Artificial

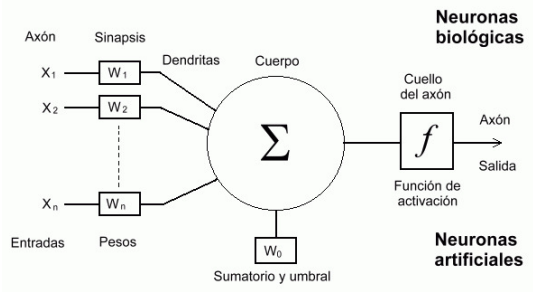

Figura 3.2: Comparación neurona biológica con artificial
Fuente: `https://www.cs.us.es/~fsancho/Blog/posts/Redes_Neuronales.md`

Las Neuronas Artificiales son modelos matemáticos que, como se muestra en la imagen 3.2, reciben una serie de entradas $x_1, ..., x_n$, cada una de las cuales se multiplica por un parámetro asociado $w_1, ..., w_n$, denominados pesos. Estos pesos son los valores que la red ajusta durante el entrenamiento. La suma ponderada de estas entradas se combina con un término adicional llamado sesgo (w_0). Finalmente, antes de generar la salida de la neurona, el resultado pasa por la llamada función de activación.

Función de Activación

Una función de activación es una función matemática que se aplica antes de la salida de cada neurona. Estas funciones permiten que la red aprenda y pueda asimilar representaciones no lineales. Existen múltiples y diferentes funciones de activación, entre ellas cabe destacar a modo de ejemplo:

- Funcion Sigmoide, utilizada para clasificación binaria:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Rectificador Lineal (ReLu):

$$Relu(z) = max(0, z)$$

- Función SoftMax, utilizada para clasificación muticlase:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

Función de pérdida

Como ya se ha mencionado las redes neuronales aprenden ajustando sus pesos. Para ello, una vez que la red produce una salida, esta se compara con el resultado real mediante una función de pérdida, que mide cuánto se ha acercado el modelo al resultado deseado.

En base a esto, se aplica la retropropagación, que actualiza los pesos de la red desde la capa de salida hasta la capa de entrada, pasando por actualizar las neuronas de la capa oculta.

Cabe destacar que, durante este proceso, las neuronas de las capas ocultas no reciben directamente el error total, sino una proporción. Esta fracción se calcula en relación a la contribución que cada neurona ha tenido originalmente, lo que permite ajustar sus pesos según su aportación en la salida de la red, corrigiéndolos e intentando reducir el valor de la función de pérdida.

Entre las funciones de pérdida más habituales encontramos la Entropía Cruzada, que puede variar en relación al tipo de clasificación que se realice, binaria o multiclase.

Siendo:

- **M** el número de clases.
- **log** el logaritmo natural.
- **o** la observación o muestra de entrada.
- **c** la clase real a la que pertenece la muestra.
- **y** una variable binaria, que indica si la clasificación es correcta o no para la observación (**o**).
- **p** la probabilidad predicha de que la observación (**o**) sea de la clase (**c**)

Cuando M = 2 se usa la Entropía Binaria Cruzada, (BCE) que se define como:

$$BCE = (y \log(p) + (1 - y) \log(1 - p)) \tag{3.1}$$

Cuando M > 2 se usa la Entropía Categoría Cruzada (CCE), que se define como:

$$CCE = \sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{3.2}$$

Calcula una función de pérdida independiente para cada clase por cada observación y suma el resultado.

3.3. Redes convolucionales

Las Redes Neuronales Convolucionales (CNN) son una especialización de las redes neuronales artificiales diseñadas para procesar datos estructurados en forma de matriz, como imágenes. Su principal atributo es la capacidad de extraer automáticamente patrones y características espaciales de la imagen que se le aporta [15].

Está compuesta, habitualmente, por varias capas convolucionales, que se encargan de extraer un mapa de características; estas suelen ir seguidas por una o más capas lineales completamente conectadas, que procesan la información extraída para realizar la clasificación. Imagen: 3.3.

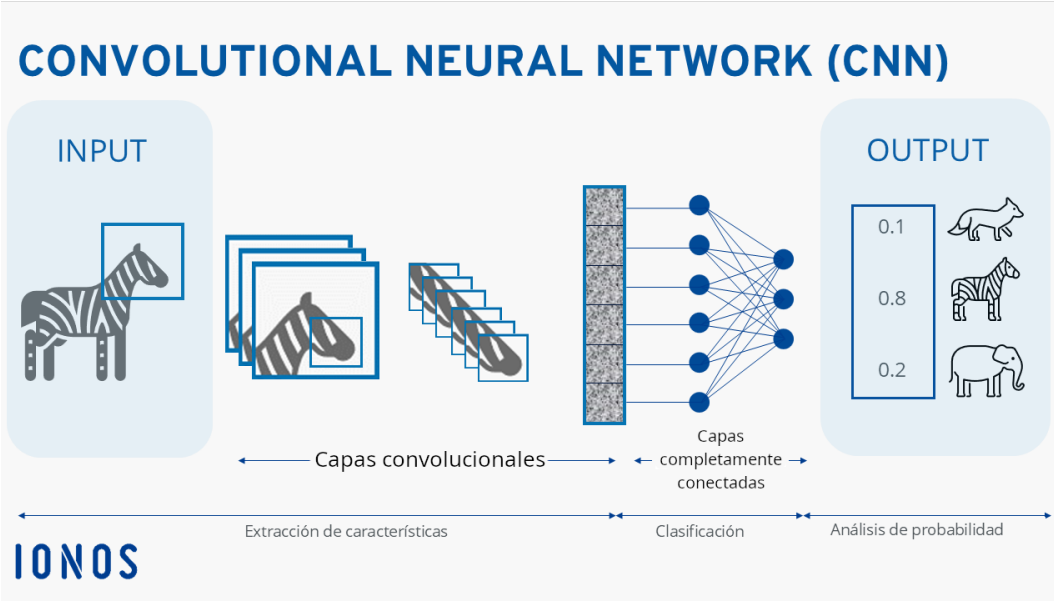

Figura 3.3: Representación de una CNN
Fuente: `https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/convolutional-neural-network/`

3.3.1. Convolución

La convolución es una operación matemática que se puede definir como el producto de dos matrices: una imagen o un mapa de características (como en nuestro caso) y un filtro. La imagen, que suele representarse como un tensor, puede tener uno o varios canales, en función de si está en escala de grises o en color. Estos canales o dimensiones, van aumentando según realizan convoluciones. Se definen tal que (H, W, C) siendo:

- H la altura.

- W el ancho
- C el número de canales.

Filtro: es una pequeña matriz rectangular, cuyas dimensiones son más pequeñas que la imagen sobre la que se aplica. Su función es desplazarse sobre la imagen de izquierda a derecha y de arriba a abajo mediante pasos, multiplicando sus valores por los de la región que corresponda y sumando todos los resultados.

El trabajo de estos filtros es resaltar ciertas características espaciales. La característica a destacar dependerá del tipo de filtro utilizado. La idea es utilizar múltiples filtros que capturen diferentes particularidades de cada imagen. Lo que nos dará múltiples matrices de salida, tantas como filtros hayamos usado. Es decir, iremos teniendo más canales.

Paso: corresponde a la cantidad unidades que el filtro se desplaza en cada dirección cada vez que se va a aplicar.

Rellenado: es la técnica utilizada para mantener el tamaño de la matriz por cada capa de convolución, o bien para que el filtro tenga espacio de operar dentro de una imagen sin rebasar el borde, su utilización consiste en rodear la matriz normalmente de ceros. El rellenado define el numero de pixeles con los que se rodeará la imagen.

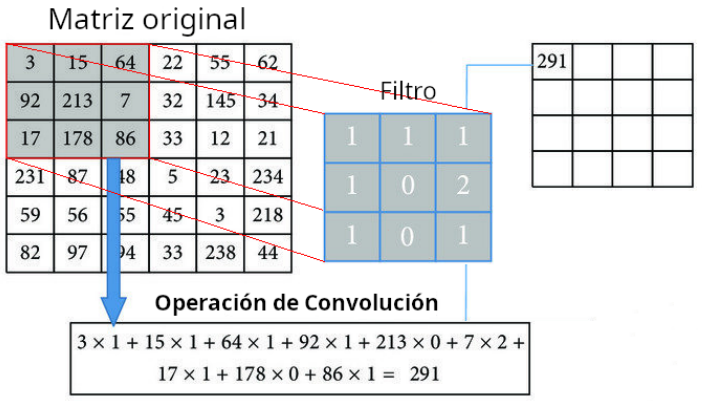

Figura 3.4: Ejemplo de calculo de una convolución.
Fuente :`https://www.researchgate.net/figure/An-example-of-convolution-calculation_fig5_313848047`

Como se puede observar en la figura 3.4, la matriz original posee unas dimensiones de 6x6 píxeles, mientras que la de salida es de 4x4 píxeles; es decir, el resultado final de la convolución es una matriz cuyas dimensiones son menores que la matriz de entrada. Por cada filtro que apliquemos obtendremos una matriz de salida, por lo que aunque se va reduciendo la resolución (H,W) se va aumentando el numero de canales (C). Es decir, si tenemos una imagen en escala de grises, de un solo canal (H,W,1) y le aplicamos un filtro de tres canales (h,w,3), obtendremos tres nuevas matrices más pequeñas [15].

La reducción del tamaño de una matriz al aplicarle un filtro viene determinada por la fórmula 3.3, donde, por cada dimensión de la matriz:

- N corresponde al tamaño de la matriz.
- F corresponde al tamaño del filtro.
- S corresponde al paso.
- P corresponde al relleno.

$$\left[\frac{N - F + 2P}{S}\right] + 1 \tag{3.3}$$

3.3.2. *Pooling*

Dentro de las CNN, las capas de *pooling* tienen como objetivo principal reducir la resolución espacial de los datos obtenidos, disminuyendo el número de parámetros y, por lo tanto, reduciendo la cantidad de cálculos a realizar. De igual manera, son utilizadas para hacer que la red se mantenga invariante ante pequeños desplazamientos y ayudan a reducir el sobreentranmiento[16].

Este proceso divide la matriz en subconjuntos de tamaño NxN y aplica la misma función a cada uno de ellos, como se puede ver más adelante en la figura 3.5. Las funciones de *pooling* más utilizadas son [17]:

Max Pooling: devuelve el valor máximo del subconjunto que captura el filtro. Esto ayuda a resaltar las características dominantes de la matriz.

Average Pooling: devuelve el promedio de los valores del subconjunto que captura el filtro. Esto suaviza las características y reduce el ruido de los datos.

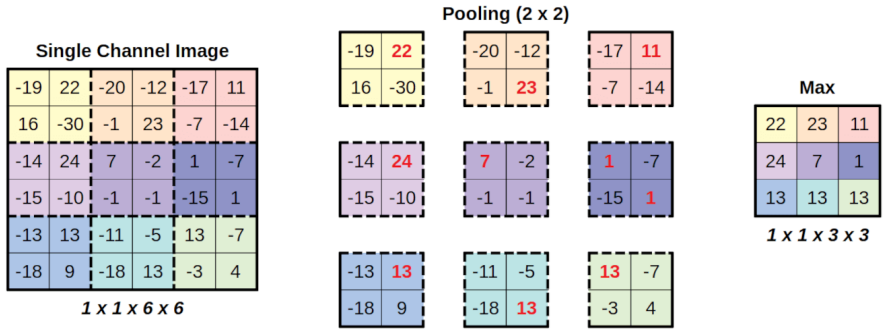

Figura 3.5: Ejemplo de aplicación de *Max Pooling*
Fuente: `https://commons.wikimedia.org/w/index.php?curid=150823502`

3.3.3. Paso de capas convolucionales a capas lineales

Uno de los pasos fundamentales en una CNN es la transición espacial de los mapas de características a un vector que pueda ser trasferido a las capas completamente conectadas. Existen varías estrategias para realizar esta operación, entre las que destacan:

Aplanado

El aplanado, tal y como se puede ver en la imagen 3.6, consiste en transformar la salida de una capa convolucional; es decir, el mapa de características, de dimensiones (H, W, C) a un vector unidimensional de tamaño (HxWxC) que sirva de entrada a las capas completamente conectadas. Esta técnica es ventajosa puesto que preserva toda la información espacial de canales de la capa convolucional; sin embargo, genera un gran número de parámetros, lo que incrementa el uso de memoria y el sobreajuste, especialmente en conjuntos de datos pequeños.

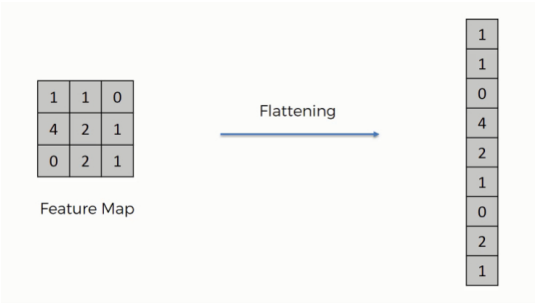

Figura 3.6: Ejemplo de aplanado
Fuente: `https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening`

Global Average Pooling

Otra forma de conectar las capas convolucionales con el clasificador es utilizar Gloval Average Pooling (GAP). En lugar de aplanar, promedia cada canal del mapa de características (H, W, C), de esta forma reduce las dimensiones espaciales (H,W) a un único valor, lo que resulta en un vector de tamaño C, considerablemente más pequeño que en el aplanado; asimismo, reduce el número de parámetros, minimiza el sobreajuste y actúa como regularizador estructural, forzando al modelo a detectar qué hay en la imagen, más que dónde está [18].

3.4. Sobreajuste

El sobreajuste es un problema de los modelos de Aprendizaje Automático, este se produce cuando los modelos se ajustan demasiado bien a los datos de entrenamiento, hasta el punto de aprendérselos y memorizar el ruido; es decir, información irrelevante, lo que impide que generalicen a nuevas muestras no conocidas [19]. Esto puede suceder por múltiples factores:

- Un modelo con demasiada complejidad.
- Un conjunto de datos demasiado pequeño.
- Entrenamientos demasiado prolongados en el tiempo.
- Una combinación de los tres anteriores.

Existen múltiples técnicas para combatir este sobreajuste detalladas a continuación.

Parada Temprana

Esta técnica consiste en detener el entrenamiento del modelo de manera temprana. No obstante, esto puede provocar que el modelo quede infraajustado y no aprenda correctamente.

Normalización por lotes

La normalización por lotes (*Bacth Normalization*) es una técnica que, mediante el ajuste de las entradas de cada capa, centrándolas alrededor de cero y reescalándolas a un tamaño estándar, tiene el efecto de agilizar y aumentar la estabilidad del entrenamiento de las redes neuronales[20]. Esto se logra gracias a que:

- Ayuda a reducir el desplazamiento de las covariables internas; es decir, los cambios en las distribuciones de las activaciones de las capas de la red.
- Actúa como regularizador al introducir ruido en el entrenamiento.
- Mitiga el desvanecimiento y explosión de los gradientes de la red.

Dropout

El *dropout* es una técnica de regularización que consiste en apagar las conexiones, dándoles el valor cero, de algunas de las neuronas de forma aleatoria durante el entrenamiento, afectando tanto a su cálculo hacia adelante, como a su retropropagación. Esto contribuye principalmente a combatir el sobreajuste de la red; asimismo, dado que tiene que calcular menos parámetros, también permite a la misma entrenar más rápido [21].

A continuación, en la figura 3.7 se puede ver un esquema representativo de la técnica previamente citada.

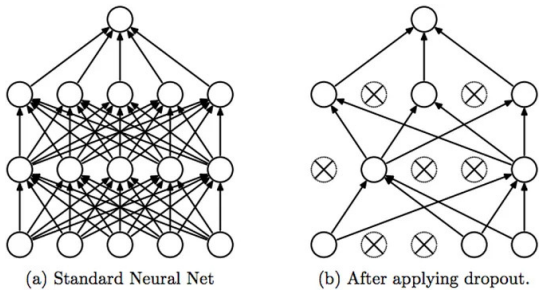

Figura 3.7: Ejemplo de aplicación de *Dropout* a una red.
Fuente: `http://jmlr.org/papers/v15/srivastava14a.html`

3.4.1. Grad-CAM

El principal problema de las redes neuronales es su escasa interpretabilidad, lo cual se acentúa según los modelos crecen y se hacen mas profundos. Las decisiones que toma una red vienen dadas por los pesos de la red. Sin embargo, para los humanos estos números no ofrecen una explicación comprensible de cómo ha llegado la red a determinadas clasificaciones. Por ello, a menudo se suele decir que las redes neuronales son cajas negras, ya que producen una salida a partir de una entrada, pero nos es demasiado difícil interpretar el razonamiento que condujo a esa salida[22].

No obstante, las CNN tienen una ventaja, ya que es posible visualizar las activaciones de sus capas convolucionales, y esto nos dará una idea de a qué partes de la imagen está prestando mas atención la red, como en la imagen 3.8. Para ello existen múltiples técnicas, entre las que cabe destacar Grad-CAM (*Gradient-weighted Class Activation Mapping*), utilizada en este proyecto.

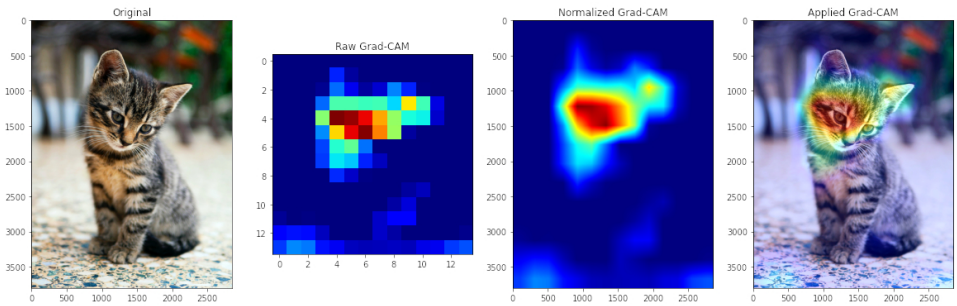

Figura 3.8: Ejemplo de Grad-CAM aplicado a una red neuronal convolucional de clasificación de animales.
Fuente: `https://dlhr.de/assets/8-0.jpg`

La técnica de aplicación de activación de clase ponderada por gradiente (Grad-CAM), genera un mapa de calor de los gradientes de las activaciones de las neuronas de una capa convolucional, normalmente la última, para una clase objetivo (por ejemplo, "enfermo"). Tal y como se puede ver en la imagen 3.9, los gradientes indican en qué dirección van a cambiar los pesos de una red respecto a la salida que ha producido, indicándonos si contribuyen positiva o negativamente; a pesar de que, normalmente, solo se utilizan las contribuciones positivas [23]. De esta manera, proporciona una herramienta visual que mejora notablemente la interpretabilidad.

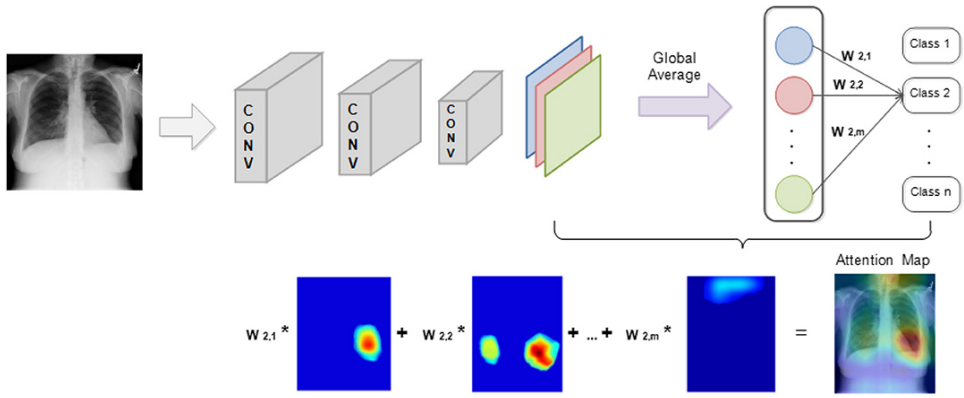

Figura 3.9: Esquema del procesamiento seguido por de Grad-CAM
Fuente: `https://www.frontiersin.org/files/Articles/583427/frai-03-583427-HTML-r1/image_m/frai-03-583427-g004.jpg`

Capítulo 4

Conjunto de datos

En este capítulo se describe el conjunto de datos utilizado para entrenar y evaluar el modelo. Se detallan las transformaciones aplicadas durante el preprocesamiento y la división en los subconjuntos de entrenamiento y test.

4.1. Descripción de los datos

El conjunto de datos está formado por archivos de imágenes JPG de ecografías de diferentes órganos de la región abdominal, principalmente hígados. Estas imágenes proceden de dos fuentes de datos distintas, con lo cual siguen dos procesamientos ligeramente diferentes.

Estas ecografías han sido obtenidas con un ecógrafo Cannon Aplio i700, imagen incluida en el anexo C.1, este produce unas imágenes con resolución de 1280x960 píxeles, en formato JPG y con una interfaz como la que se muestra en la imagen 1.2.

4.1.1. Origen

Hospital Universitario Río Hortega

La primera fuente de imágenes ha sido proporcionada por el Hospital Universitario Rio Hortega (HURH), recogidas de forma manual del ecógrafo previamente mencionado. En total, se han recogido 12156 imágenes, inicialmente sin etiquetar ni filtrar, que posteriormente han sido categorizadas por un radiólogo.

La recogida y etiquetado de esta imágenes no se realizó en una sola sesión, sino que se llevó a cabo mediante visitas periódicas al hospital. Las categorías creadas por el profesional fueron las siguientes: bazo, cálculos y pólipos en la vesícula, páncreas, riñón, hígados sanos,

hígados con esteatosis, hígados con cirrosis, hígados con hepatocarcinoma y lesiones hepáticas benignas.

OneDrive

La segunda fuente de imágenes procede de un repositorio de OneDrive compartido por los médicos con los que se ha colaborado, este contiene una selección de 954 imágenes de hígados divididas en tres categorías. Esta distribución se puede ver en la tabla 4.1.

Categorías	Imágenes	Proporción
Hígados sano	274	28.72 %
Hígados con cirrosis	476	49.90 %
Hígados con hepatocarcinoma	204	21.38 %
Total:	954	100 %

Cuadro 4.1: Distribución del conjunto de imágenes pertenecientes al OneDrive.

Estás imágenes han sido extraídas de su sistema informático y pertenecen a diferentes hospitales de Castilla y León, detallados en el anexo C.

4.2. Preprocesamiento

Todo el preprocesamiento está gobernado por unos *scripts* en Python que se encargan de cada paso descrito. Finalmente, hay un *script* encargado de controlar toda la tubería de procesamiento y lanzar el resto en orden. Esto se hace así para tener modularidad y para poder, en caso de desearlo, lanzarlos por separado o sólo hasta solo cierto punto; de esta manera se pueden llevar a cabo más pruebas, subcategorías o un refinamiento más específico de las imágenes.

Desde el inicio del proceso, antes de clasificar las imágenes manualmente, y durante todos los pasos, se han ido construyendo archivos CSV con la información original de las imágenes como: nombres, rutas de procedencia y finales, categoría a la que pertenecen y método ecográfico utilizado. De este modo, a la hora de cargarlas al modelo, se pueden seleccionar aquellas que se quieren y realizar diferentes pruebas con ellas.

Esto se realiza, además, para poder trazar los cambios que han ido sufriendo las imágenes y para, en caso de necesitarlo, relacionar aquellas que originalmente venían juntas, pues podría resultar interesante para comparar zonas diferentes de un mismo hígado, o para unir varias imágenes de un mismo paciente y utilizarlas como una única. Sin embargo, esto finalmente no se llevó a cabo.

Por último, estos CSV también son útiles por si se produce algún error en las imágenes o se pierden, ya que facilitan su identificación sin necesidad de iniciar el procesado completo

de ellas y se pueden usar como archivos de anotaciones para cargar los datos y entrenar los modelos.

4.2.1. Hospital Universitario Río Hortega

Las imágenes se extraen directamente del ecógrafo, que se encarga de anonimizarlas, a un disco duro externo, siguiendo una estructura de carpetas como la que se muestra en la figura 4.1. Estas carpetas toman como nombre la marca temporal correspondiente al momento en que se ponen en cola para ser exportadas al disco duro. En su interior, se encuentra otra carpeta similar que, finalmente, contiene las imágenes; estas, al estar anonimizadas, no contienen el nombre de los pacientes, pero sí incluyen la marca temporal de cuando fueron tomadas, junto a un identificador numérico diferente por cada imagen en la carpeta. Por lo que, aunque fueran separadas, se podrían volver a relacionar entre sí.

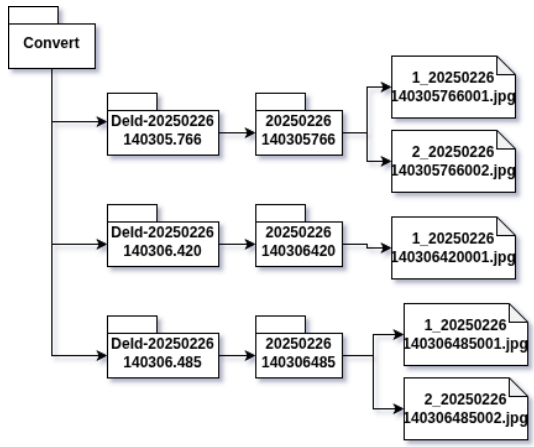

Figura 4.1: Estructura de carpetas generadas por el ecógrafo.

Esta estructura hace difícil y tedioso su etiquetado manual. Por lo que, una vez en el disco duro, se hace un primer preprocesamiento, con un *script* en BASH, que mueve todas las imágenes a la carpeta raíz *Convert*, borra los vídeos, elimina toda la estructura de carpetas y crea las carpetas de las categorías creadas por el profesional para que el etiquetado suponga unicamente arrastrar la imagen a la carpeta de la categoría correspondiente o se descarte.

Una vez extraídas, un radiólogo especializado en el sistema digestivo clasifica las imágenes en las categorías mencionadas. Asimismo, descarta las que no se ven correctamente, las que pertenecen a órganos que no tienen interés, tienen menús del ecógrafo o; que por cualquier otra característica o defecto, sean consideradas no aptas por el profesional.

Durante este proceso, se realizó un primer análisis exploratorio de las imágenes y se descubrió que, aunque la primera muestra data del 24/08/2024, no hay imágenes de forma consistente hasta el 29/01/2025, pues el ecógrafo no las almacenaba correctamente. Por

ello, se partió de menos imágenes de la esperadas, esto obligó a realizar distintas visitas adicionales al hospital a lo largo del tiempo y motivó a la búsqueda de otras fuentes de datos complementarias.

Una vez completado el proceso de etiquetado y recolección, se realizó un análisis de las imágenes para conocer la cantidad definitiva en cada categoría. Más adelante, en la tabla 4.2, se presenta ya distribuido el resultado final tras todas las visitas al hospital. De las 12156 imágenes extraídas originalmente, se conservaron 5314, es decir, el 43.72 %, lo cual nos deja con no demasiadas imágenes.

Durante este análisis, también se observó que algunas de las imágenes presentan colores sobre la zona de la ecografía, debido a la técnica Dópler. Esto se tratará más adelante en la sección de preprocesamiento común.

Categoría	Imágenes	Proporción
Bazo	330	6.21 %
Cálculos y pólipos en la vesícula	191	3.59 %
Páncreas	314	5.91 %
Riñón	356	6.7 %
Hígados sanos	2600	48.93 %
Hígados con esteatosis	684	12.87 %
Hígados con cirrosis	540	10.16 %
Hígados con hepatocarcinoma	20	0.38 %
Lesiones hepáticas benignas	279	5.25 %
Total	5314	100 %

Cuadro 4.2: Distribución del conjunto de imágenes del HURH tras su etiquetado.

4.2.2. OneDrive

Las imágenes del OneDrive compartido por el equipo médico también se encontraban anonimizadas. Cada una cuenta con un número identificador aleatorio por paciente, seguido de un guion y un subidentificador por cada ecografía del paciente. Tal que el formato sería: '295166-1' junto a la extensión del archivo.

Estos archivos se encuentran divididos en tres carpetas, cada una correspondiente a las categorías previamente descritas, y Éstas, a su vez, organizadas en subcarpetas corrrespondientes a cada subida de imágenes realizada por el profesional. Al agrupar todas las imágenes de las subcarpetas de una misma categoría, se observa que puede haber imágenes de un mismo paciente, con subidentificadores de ecografía iguales en distintas subcarpetas, lo cual produce colisiones en los nombres de los archivos.

Esto puede deberse a dos motivos:

- Que sean el mismo archivo duplicado y que haya sido etiquetado varias veces, en ocasiones distintas, con el mismo identificador.

- Que sean archivos diferentes del mismo paciente, pero al ser subidas por separado tengan el mismo subindentificador y, por lo tanto, el mismo nombre.

La agrupación de las imágenes se realiza mediante el cálculo del *hash*, esto permite determinar si son o no iguales a una ya existente en la categoría. En el caso de que el duplicado sea exacto, una de las dos imágenes se descarta. Si son diferentes, se va aumentando en uno el subidentificador hasta que no produzca choques. Durante este proceso, también se detectaron algunas imágenes corruptas, para esta tarea se usó del paquete Pillow de Python, y fueron descartadas.

Una vez solucionado esto, se abordó el siguiente problema, y es que cada ecógrafo produce imágenes muy desiguales entre sí, con diferentes resoluciones, interfaces, escalas de grises, texturas de ecografía, técnicas de ecografía, anotaciones sobre la imagen y sectores de ecografía dentro de la imagen.

Estas variaciones suponen un problema muy grave para el entrenamiento del modelo, ya que si las ecografías recogidas del Aplio poseen muchos menos hepatocarcinomas que otras, las CNN no aprenderán a centrarse en las características más relevantes, si no que lo harán en las diferencias de las imágenes. Además, esto imposibilitaría llevar a cabo la solución que propuesta para dividir las ecografías Dópler de las de Modo B.

Tras un análisis, se concluyó que la manera más conveniente de abordar el problema es categorizar las imágenes en sus diferentes resoluciones, de forma que, una vez separadas, se pueda elegir las que fueran más similares a las que ya se poseía. Finalmente se decidió utilizar solo las imágenes que procedían de ecógrafos Aplio, como las recogidas en el HURH. Por lo que, de todo el conjunto, únicamente se seleccionaron las imágenes que poseían una resolución 1280x960 píxeles.

Tras todo este proceso de filtrado y clasificación, se conservaron 368 imágenes, entre Modo B y Dópler, lo que representa el 38.57 % de las originales, estas se pueden ver en la tabla 4.3.

Categorias	Imagenes	Proporción
Hígados sanos	243	66.03 %
Hígados con cirrosis:	90	24.46 %
Hígados con hepatocarcinoma	35	9.51 %
Total	368	100 %

Cuadro 4.3: Distribución del conjunto de imágenes del OneDrive tras su preprocesado.

4.2.3. Preprocesamiento común

Como se ha mencionado anteriormente, las imágenes con técnica Dópler presentan colores sobre la zona de interés de la ecografía; ya que no hay manera de evitar esto, y además provocan diferencias que pueden perjudicar el entrenamiento de los modelos, se decide separar estas imágenes de las de Modo B.

Para ello, se utiliza una zona concreta de la imagen: un medidor que normalmente está en escala de grises si la ecografía es Modo B, pero que tiene colores si se está usando la técnica Dópler, véase figura 4.2. Con lo cual, se hace un recorte de la imagen a ese medidor y se analizan las diferencias entre las medias de los tres canales RGB. Esto no se aplica a la imagen completa, puesto que al tener muchos más píxeles, las diferencias entre las medias se diluyen y varían notablemente entre las imágenes, lo que no permite una división precisa de ellas.

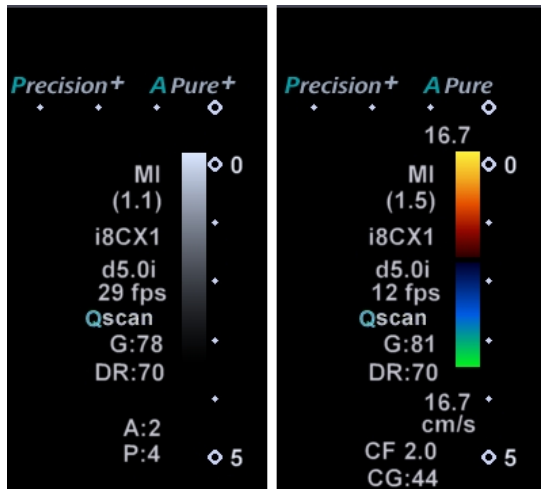

Figura 4.2: Diferencias entre modo B (izquierda) y Dópler (derecha)

Llegados a este punto, como se puede ver en la tabla 4.4, entre ambas fuentes de datos, se tienen 5682 imágenes fusionando ambos modos.

Modo Ecografía	Imágenes	Proporción
Modo B	4607	81.08 %
Dópler	1075	18.92 %
Total	5682	100 %

Cuadro 4.4: Distribución de los modos de imagen en el conjunto de datos.

Una vez realizada esta parte, y antes de entrenar los modelos con las imágenes, se recortan al sector de la ecografía; es decir, la zona de interés. Este paso se realiza por dos razones:

- Las imágenes poseen una resolución considerablemente grande, en términos de entrenar modelos. Esto incrementa el número de parámetros; y, a su vez, ralentiza el entrenamiento y aumenta el uso de memoria, lo que dificulta entrenar modelos mas complejos.
- Existe mucha información de la interfaz que no aporta información relevante o que incluso puede perjudicar para el aprendizaje del modelo.

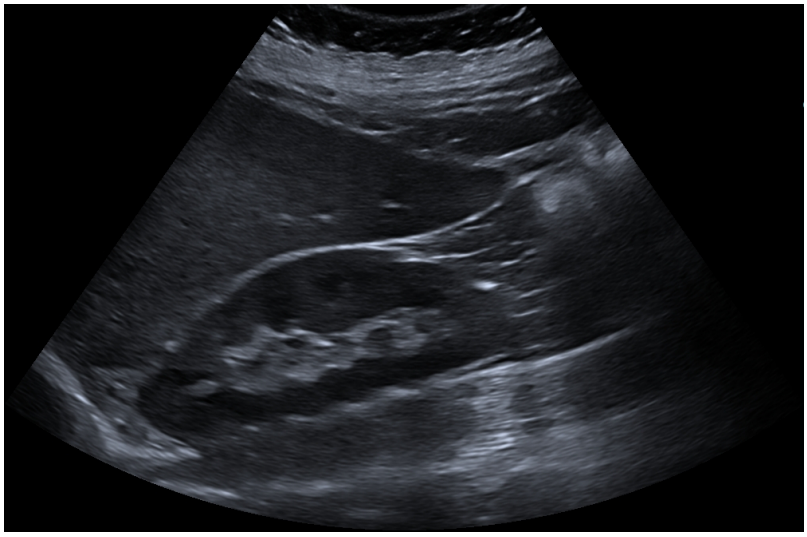

Figura 4.3: Ejemplo de ecografía recortada.

A la hora de realizar el recorte de las imágenes se presenta una dificultad: dependiendo de los ajustes del ecógrafo en el momento de tomar la ecografía, la zona de la ecografía no es igual para todas las muestras, produciendo sectores ecográficos que varían en tamaño y en desplazamiento. Por ello, no hay una única medida de recorte que se adecúe a todas las ecografías.

Primero, se opta por buscar el mínimo cuadrado que contenga a la mayoría de sector de las ecografías. Ya que, ser agresivos con el recorte no es una opción idónea debido a que nada asegura que la información clínica relevante de la ecografía este centrada.

Por ello, se concluyó que la mejor opción era hacer una distribución de la posición y el tamaño de cada recuadro. Estos datos no siguen una distribución normal, por lo que usar la media y la desviación para captar la mayoría de ellos no es la mejor opción. En su lugar, se opta por usar los cuartiles 10 y 90 para las cotas inferior y superior respectivamente.

Esto produce imágenes con un tamaño de 1004x661 píxeles, como se muestra en la figura 4.3. Este recorte se produce en el momento de cargar los datos al modelo, de manera que las imágenes se conservan intactas hasta entonces, lo cual permite realizar múltiples pruebas.

Los resultados de este primer intento no fueron favorables y, tras aplicar Grad-Cam, se detectó que el modelo estaba trampeando los resultados, fijándose en anotaciones que se cuelan al recorte, véase figura 4.4. En principio las anotaciones pueden parecer inocuas; sin embargo, son más habituales en las ecografías en las que hay una dolencia, por lo que son contraproducentes a la hora de que el modelo aprenda, ya que tiende a prestar atención en la presencia o ausencia de ellas en vez de en la zona de interés de la ecografía.

Por este motivo, es necesario eliminar algunas anotaciones visibles en las imágenes, producidas por la interfaz del ecógrafo a la hora de realizarlas. Se pueden identificar dos tipos

de anotaciones:

- Las mediciones, que generan uno varios marcos en la parte inferior izquierda de la ecografía y unos puntos dentro de la zona de interés. Estos marcos, que también pueden variar de posición y tamaño, pueden intentar ocultarse. Sin embargo, respecto a los puntos no se ha encontrado solución, ya que taparlos también revelaría su presencia.
- Esquemas anatómicos, usados para identificar la zona que es representada, y que aparecen con mas frecuencia en las imágenes que muestran patologías.

Descartar estas imágenes no parece la mejor opción, pues se cuenta con un número limitado de muestras. Por ello, se opta por eliminar las anotaciones cuando sea posible, y finalmente, se consiguió ocultar un gran número de etiquetas.

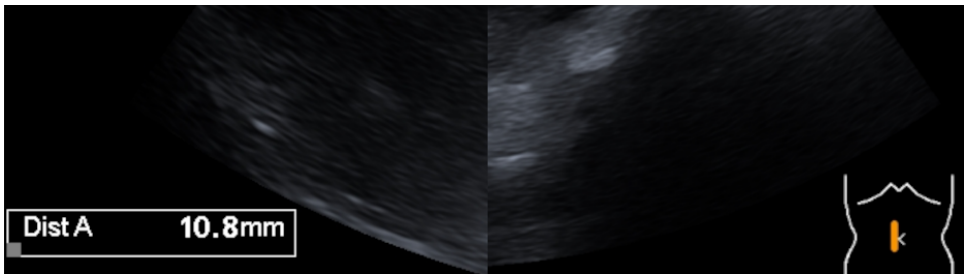

Figura 4.4: Recorte de las anotaciones de las ecografías.

En ambos casos se utiliza *OpenCV* para encontrar contornos en las zonas donde potencialmente puede haberlos, y cubrirlos con el color de fondo de la ecografía. Sin embargo, realizar todo este proceso para las diferentes ecografías del OneDrive hubiese sido complejo y hubiera requerido mucho tiempo, lo cual excede completamente del alcance del proyecto.

4.2.4. Transformaciones a los datos

Hasta este punto se ha descrito el preprocesamiento que se realiza a las imágenes antes de entrenar los modelos con ellas. A la hora de hacerlo, se ha creado una tubería de transformaciones que permite:

- Pasar las imágenes a escala de grises o dejarlas en RGB.
- Recortar las imágenes a los tamaños descritos, asegurando que se conserva la región de interés.
- Espejar horizontalmente un porcentaje de las imágenes de entrenamiento de forma aleatoria (normalmente el 50 %) con el objetivo de aumentar la variabilidad de los datos.

- Reducir el tamaño de las imágenes, normalmente a la mitad, resultando en una resolución de 502x331 píxeles, o distintas variaciones dependiendo del modelo que se quiera entrenar.
- Normalizar las imágenes, tanto si están en RGB o escala de grises, de alguna de las siguientes formas:
  - Aplicando los valores de media y desviación estándar que se deseen.
  - Calculando la media y la desviación estándar para el conjunto de datos de entrenamiento deseado.
  - Normalizando los valores de los pixeles al rango [-1,1] utilizando media 0.5 y desviación 0.5.
  - Usando los valores de *ImageNet*[24] ampliamente recomendados. Estos valores se muestran en la siguiente tabla: 4.5

	RGB	Escala de grises
Media	0.485, 0.456, 0.406	0.0840
Desviación	0.229, 0.224, 0.225	0.1069

Cuadro 4.5: Valores de normalización de *ImageNet*.

4.3. División del conjunto de datos

Para entrenar los modelos de clasificación, se divide el conjunto de datos en dos subconjuntos: entrenamiento y prueba. Esta división otorga 2/3 de los datos para el entrenamiento y 1/3 para el conjunto de pruebas, esto permite disponer de numerosos ejemplos para entrenar al modelo asegurando que los resultados sean válidos.

Una parte importante a la hora de realizar esta distribución es tener en cuenta que las clases no están equilibradas; es decir, en las muestras predominan los hígados sanos frente a otros, especialmente los hepatocarcinomas. Por ello, se tiene en cuenta la distribución inicial de clases del conjunto y se mantiene esa proporción en ambos subconjuntos; de esta manera, se asegura que tanto el conjunto de entrenamiento como el de prueba sigan siendo representativo y que no favorezca a una clase por encima de otra durante el entrenamiento y la evaluación.

De esta división se encarga una clase creada *LiverImg*, que forma parte de la estructura de carga de datos y que se desarrollará más adelante. Esta clase se vale de la función *train_test_split* del paquete *scikit-learn* a la hora de realizar las divisiones.

Capítulo 5

Diseño y construcción del sistema

En este capítulo se describe cómo se ha desarrollado el sistema de clasificación de ecografías: desde el entorno de trabajo del proyecto y la carga de datos, pasando por el diseño de la red hasta el uso de modelos preentrenados y el entrenamiento.

5.1. Entorno de trabajo

Para el desarrollo y entrenamiento de los modelos se han utilizado las siguientes herramientas:

- **Sistema Operativo:** Ubuntu 24.04.2 LTS.
- **GPU:** GeForce GTX 1060 3GB VRAM. La que se disponía en el momento de realización del trabajo.
- **Lenguaje de programación:** Python 3.12.3.
- **Librería de aprendizaje profundo:** Pytorch 2.6.0, seleccionado por su sencillez, grado de precisión y personalización a la hora de crear los modelos, frente a otros como Keras o Tensorflow.
- **Monitorización de entrenamiento:** Tensorboard, para la visualización de métricas y gráficos en tiempo real.
- **Optimización de parámetros:** Optuna, para la exploración y búsqueda de los hiperparámetros más eficientes.

5.2. Carga de datos

El código para procesar y cargar las muestras de datos puede resultar confuso y difícil de mantener. Lo ideal es que el código de nuestro conjunto de datos sea independiente del modelo; de esta manera aseguramos una mejor legibilidad, modularidad y reutilización. PyTorch ofrece dos primitivas de datos para ayudar con esta tarea: *torch.utils.data.Dataset* y *torch.utils.data.DataLoader*, que permiten tanto el uso de conjuntos de datos propios como conjuntos de datos precargados.

Dataset se encarga de gestionar las muestras y sus etiquetas correspondientes; así como de las características y etiquetas de nuestro conjunto de datos, devolviendo una muestra cada vez. Al entrenar un modelo, se busca pasar muestras en pequeños lotes para acelerar la carga de datos y reducir el sobreajuste del modelo. Esta es la tarea de *DataLoader* que se encarga de envolver un iterable alrededor de *Dataset* que abstrae esta complejidad para facilitar el acceso a las muestras[25].

5.3. *Dataset LiverImg*

Ya que se dispone de ecografías de múltiples órganos de la región abdominal, se ha desarrollado un conjunto de datos con tres modos de funcionamiento (*ORGAN_CLASSIFICATION HEALTHY_LIVERS* y *CIRRHOTIC_STATE*), en los que se profundizará más adelante.

Para este proyecto se ha creado una clase llamada *LiverImg* que hereda de *torchvision.dataset.Dataset*, que puede ser usada con *DataLoaders*, y que se usa para entrenar los modelos. Esta clase encapsula toda la lógica correspondiente a la carga de datos y se encarga de:

- Carga de imágenes según el modo de funcionamiento.
- Búsqueda del ultimo CSV de anotaciones disponible (o uno en específico).
- Carga del conjunto de datos completo, de entrenamiento o de prueba.
- Selección de las imágenes correctas correspondientes al conjunto y al modo de funcionamiento deseados.
- Filtrado según las especificaciones de las ecografías (Modo B o Dópler).
- Aplicación de transformaciones a las imágenes y sus correspondientes etiquetas.

La lista de modos disponibles que se detallan a continuación está modelizada por la clase *DatasetMode*.

ORGAN_CLASSIFICATION

El modo *ORGAN_CLASSIFICATION* tiene como propósito englobar todo el conjunto de imágenes disponibles agrupadas en cinco clases diferentes:

- Hígado (sanos, con cirrosis, con hepatocarcinoma, con esteatosis y lesiones benignas).
- Riñón.
- Bazo.
- Páncreas.
- Vesícula.

La intención de este modo es poder entrenar modelos que diferencien entre diferentes órganos, lo cual, aunque no alcanza una gran relevancia clínica, es útil para aprendizaje sobre el desarrollo de modelos de aprendizaje automático.

HEALTHY_LIVERS

Dado que el proyecto está centrado en hígados, se ha implementado también el modo *HEALTHY_LIVERS*, que carga únicamente imágenes hepáticas. Esta modalidad surge como respuesta a la insuficiencia de muestras de hepatocarcinomas, lo cual perjudica una división más detallada; por ello, el objetivo de este modo es establecer una clasificación binaria entre hígados. Agrupándolos, según su diagnóstico, en:

- Hígado sano.
- Hígado enfermo, que engloba: hígado con cirrosis, hígado con hepatocarcinoma, hígado con esteatosis y lesiones hepáticas benignas.

La decisión de introducir las lesiones hepáticas benignas dentro de la categoría "enfermo" viene justificada por que, aunque no se consideren malignas, siguen siendo, principalmente, tumores, y aunque no supongan un riesgo deberían ser detectados. Asimismo, colocarlos dentro de la categoría "sano" podría hacer que, al asemejarse a un hepatocarcinoma, aumentaran las probabilidades de que este se clasificase, equívocamente, como sano y es preferible clasificar como "enfermo" una lesión benigna, que dejar pasar un cáncer.

CIRRHOTIC_STATE

Finalmente, dado que el objetivo del proyecto es la detección de cirrosis y hepatocarcinomas, se creó el modo *CIRRHOTIC_STATE*, englobando las categorías de:

- Hígado sano.
- Hígado con cirrosis.
- Hígado con hepatocarcinoma.

Puesto que la finalidad perseguida por este proyecto es diferenciar entre estos tres casos, el resto de categorías de hígado no han sido incluidas.

5.4. Creación de modelos

Para la realización de este trabajo se han desarrollado dos clases diferentes que se encargan de la creación, entrenamiento, validación, guardado y carga de sus respectivos modelos de IA. Estas clases son: *CustomCNN*, una red neuronal personalizable, y *PretrainedModels*, que engloba múltiples modelos preentrenados.

5.4.1. *CustomCNN*

El objetivo, al desarrollar este modelo de red convolucional, es diseñar una arquitectura adaptable que facilite la experimentación y se ajuste adecuadamente a los requerimientos de la tarea de clasificación. Por ello, se ha implementado una red neuronal convolucional, *CustomCNN*, heredando de la clase *torch.nn.Module* de Pytorch, diseñada para ser modular, configurable y compatible tanto con clasificación binaria como multiclase.

Inicialización

La clase CustomCNN permite en sus parámetros de entrada una configuración flexible de:

- El tamaño de la entrada.
- El número de capas convolucionales junto a sus correspondientes parámetros: canales, tamaños de filtro, paso y rellenado.
- Capas de *pooling* con sus respectivos parámetros de filtro, paso y rellenado.
- Capas, opcionales, de *Batch Normalization*.
- Una capa, opcional, de *GAP* entre las capas convolucionales y lineales, si no, se usa un aplanado normal.
- Número configurable de capas lineales completamente conectadas, con sus respectivos tamaños.

- Número de capas, configurables de *Dropout*.

Para lograr este objetivo, se tomó la decisión de implementar dos funciones privadas, que son llamadas en el constructor de la clase. Estas son:

- *_make_conv()*, responsable de la construcción de los bloques convolucionales. Cada uno de ellos se compone de una capa convolucional 2D, opcionalmente seguida de una capa de *Batch Normalization*, una activación ReLU y una capa de *MaxPooling*.
- *_make_fcl()*, encargada de construir los bloques lineales completamente conectados. Cada uno de ellos se compone de una capa lineal, seguida opcionalmente por una capa de *Dropout*, excepto en la última capa antes de la salida, donde no se aplica para evitar posibles efectos negativos en el rendimiento de la clasificación.

Los bloques convolucionales y los bloques lineales se añaden por separado en listas, que se encapsulan mediante *nn.Sequential*. Esta arquitectura modular permite que PyTorch gestione automáticamente la conexión secuencial entre los bloques, reduciendo la complejidad y simplificando la implementación de la función *forward()*, encargada de la propagación hacia adelante.

Entrenamiento

La función *fit()* se implementó con el objetivo de que se responsabilizara de toda la lógica del entrenamiento el modelo, incluyendo parada temprana ante la falta de mejora, validación opcional del modelo, registro opcional en TensorBoard y *callback* para Optuna.

Esta función acepta como entradas los *DataLoaders* de entrenamiento y validación, una función de pérdida, un optimizador, un *scheduler* (ajustador automático de la tasa de aprendizaje) e hiperparametros como *max_epochs*, *patience* y *batch_size*. Para esto se vale de un bucle de entrenamiento donde se llama a *train_loop* y, finalmente, devuelve tres listas con los valores de:

- La pérdida sobre el conjunto de entrenamiento.
- La pérdida sobre el conjunto de validación
- La exactitud sobre el conjunto de validación.

Época de entrenamiento

Para encapsular el código a una época completa del bucle de entrenamiento, se ha implementado la función *train_loop*. Esta se encarga de:

- El envío de datos al dispositivo de computo correspondiente (CPU o GPU).

- Realizar la propagación hacia adelante del modelo.
- Calcular el valor de la función de pérdida.
- Propagar el error hacia atrás y actualizar los pesos mediante el optimizador.

La función acepta como argumentos un *DataLoader*, la función de pérdida y el optimizador, y devuelve el valor promedio de la pérdida obtenida durante la epoca.

Evaluación

Para evaluar el rendimiento del modelo se ha implementado la función *evaluation_loop*, que activa el modo evaluación del modelo. Esto fuerza que las capas *Dropout* y *Batch Normalization* se comporten de forma determinista.

A continuación, se inicializan las métricas necesarias para la evaluación y, con el cálculo de gradientes desactivado, se realizan los cálculos de correspondientes. La función contempla tanto el caso de clasificación binaria como multiclase, ajustando las métricas según corresponda. Al finalizar, devuelve un diccionario con las métricas ya calculadas.

Puntos de guardado

Con el fin de permitir la interrupción y reanudación del entrenamiento, se han desarrollado puntos de guardado en archivos pth (una extensión de archivos de Pytorch). Cada modelo se guarda en una ruta específica que refleja tanto el modo de uso del conjunto de datos (*DatasetMode*) de *LiverImg* como la arquitectura de capas utilizada.

Para gestionar los puntos de guardado, se han implementado dos funciones:

- *save_checkpoint* que guarda en un archivo pth los hiperparámetros, la arquitectura del modelo, los pesos, el optimizador y su estado, el *scheduler* y su estado, la pérdida y las épocas entrenadas. En caso de que existiera un archivo de guardado del mismo modelo con peores resultados, este es eliminado y sustituido por el nuevo.
- *save_checkpoint* se encarga de cargar únicamente los pesos del modelo. Se usa principalmente para recuperar la mejor versión del modelo (que no necesariamente corresponde con la última época) para su posterior evaluación.

Carga de modelos

Para realizar la carga completa de modelos, incluyendo toda la información guardada con *save_checkpoint*, se ha implementado el método de clase *load_model*, que se encarga de leer los datos del archivo de guardado y, a partir de ellos:

- Inicializar una instancia del modelo y cargar sus pesos.
- Inicializar el optimizador con el que se entrenó y cargar su estado.
- Inicializar el *scheduler* y recuperar su estado correspondiente.

Finalmente, devuelve estos componentes junto a toda la información necesaria para continuar con el entrenamiento: el número de época, el valor de la pérdida, así como los parámetros relacionados con la carga, las transformaciones y el modo de los datos utilizados.

Registro

Con el objetivo de facilitar la depuración de los modelos, analizar su evolución y trazar el proceso de entrenamiento, se ha implementado un sistema de registro. En cada época, el método *report_csv*, registra la arquitectura del modelo, las métricas y los hiperparámetros relevantes en un archivo CSV, generando una fila por época.

Adicionalmente, se integra el uso de TensorBoard, lo que permite visualizar en tiempo real la evolución de las métricas y facilita el análisis comparativo entre distintas configuraciones de entrenamiento.

5.4.2. *PretrainedModel*

La clase *PetrainedModel* proporciona un envoltorio para un acceso unificado del entrenamiento y evaluación con modelos preentrenados. Esta ha sido desarrollada para facilitar la comparación con y entre este tipo de modelos.

Los modelos, listados en la tabla 5.1, se han utilizado con sus pesos por defecto; y, posteriormente, se han continuado entrenando con nuestras imágenes. La elección de estos modelos y no otros reside en las diferencias de sus distintas arquitecturas CNN y ViT, y en que son los que mejor encajaban con la capacidad de cómputo disponible.

Modelo	Número de parámetros	GFLOPS
ConvNeXt_Tiny	28.6M	4.46
Densenet-121	8.0M	2.83
ResNet-18	11.7M	1.81
EfficientNet B0	5.3M	0.39
ViT_B_16	86.6M	17.56

Cuadro 5.1: Número de parámetros, en millones, y coste computacional de los modelos preentrenados utilizados, en Giga-FLOPS.

Ademas, *PetrainedModel* se encarga de adaptar el número de clases según el modo del conjunto de datos utilizado, definido en *DatasetMode*. Para ello modifica la capa de salida del modelo seleccionado, asegurando su compatibilidad con la tarea de clasificación correspondiente

Inicialización

Para instanciar un objeto de esta clase basta con indicar tres parámetros:

- El nombre del modelo deseado de la lista de modelos disponibles en *ModelNames*.
- El modo de conjunto de datos deseado de la lista de modos definida en *DatasetMode*.
- De manera opcional, descargar los pesos preentrenados por defecto, lo cual se puede omitir si posteriormente se va a cargar una versión de ese modelo desde un archivo.

Esto simplifica la creación de modelos ya que:

- Evita la definición manual del número de capas.
- Abstrae de la carga de pesos o modelos.
- Centraliza la inicialización de múltiples arquitecturas en una única interfaz.
- Gestiona la ruta de guardado y carga del modelo según su configuración, facilitando la trazabilidad y reproducibilidad.

Transformaciones

Otra de las funciones que realiza esta clase es la adaptación de la tubería de transformaciones (que incluye el recorte de imágenes, la normalización y la conversión a tensor) según el modelo seleccionado, dado que no todos los modelos aceptan cualquier tamaño de entrada.

En el caso de ViT, solo acepta entradas de tamaño 224x224 píxeles y; para el resto, utiliza el tamaño de las ecografías recortadas y reducidas aproximadamente 4.5 veces.

Entrenamiento

Al igual que con *CustomCNN*, se ha implementado un método *fit* encargado de gestionar todo el proceso de entrenamiento de la red. Esto incluye validación, parada temprana y guardado automático del modelo con mejor puntuación. No se desarrolla en detalle en el proyecto puesto que su implementación presenta muchas similitudes con la de *CustomCNN*

Época de entrenamiento

Durante el entrenamiento, *fit* utiliza *train_loop*, que ejecuta la propagación hacia adelante, el cálculo de la pérdida, la retropropagación y la actualización de pesos para cada lote de imágenes en cada época. Dado que esta implementación también es muy similar a

la de *CustomCNN*, se destacarán exclusivamente las mejoras específicas para el uso de modelos con altas exigencias computacionales y de memoria. Para lo cual se ha integrado el uso de*torch.amp*, que realiza el entrenamiento con precisión mixta y ayuda a reducir significativamente el consumo de memoria y acelera los cálculos sin sacrificar la precisión del modelo.

Además, se añade una gestión manual de memoria, liberándola mediante llamadas a *gc.collect()* y *torch.cuda.empty_cache()*.

Evaluación

Se implementa una función que, a partir de un DataLoader, permite la evaluación del modelo elegido calculando las métricas de pérdida, exactitud, F1 y AUROC. Permite, además, guardar el modelo si se desea.

Carga y guardado de modelos

Se implementa el método *save*, que se encarga de guardar los pesos del modelo junto a sus transformaciones. Asimismo, se implementa el método *load*, que restaura los pesos y las transformaciones del modelo, a partir de un archivo pth.

De la misma manera que con *CustomCNN*, las rutas se generan a partir del modo de funcionamiento y la arquitectura del modelo.

Carga de Datos

Se crea la función *get_dataloaders* que se encarga de generar los *DataLoaders* para el entrenamiento y la validación. Esta función recibe una clase, como *LiverImg* y aplica las transformaciones necesarias según el modelo correspondiente.

Acceso a Capas Intermedias y Grad-CAM

Se define la propiedad *get_last_conv_layer* que proporciona acceso transparente a la última capa convolucional del modelo, facilitando la aplicación de la técnica Grad-CAM. Esta funcionalidad está disponible para todas las arquitecturas basadas en CNN; sin embargo, no se implementa para el modelo ViT debido a su arquitectura no convolucional, lo cual dificulta este proceso.

Capítulo 6

Evaluación y resultados

En este capítulo se recogen y analizan los resultados obtenidos durante la fase de evaluación del proyecto. Los resultados se organizan en función de las tres tareas de clasificación planteadas, abordando tanto el comportamiento sobre el conjunto de entrenamiento como su capacidad de generalización sobre el conjunto de prueba.

6.1. Evaluación

El proceso de evaluación permite comprobar si los modelos han aprendido patrones correctamente y si son capaces de generalizar a datos nunca vistos. Para ello, se detallan las técnicas utilizadas y se comparan los resultados.

6.1.1. Métricas

La evaluación de los modelos entrenados se ha llevado a cabo mediante el uso de diferentes métricas estándar en problemas de clasificación como la exactitud, precisión, sensibilidad, especificidad y la puntuación F1. A continuación, se definen distintos términos de los que se ha hecho uso:

- **TP**: Verdaderos positivos.
- **TN**: Verdaderos negativos.
- **FP**: Falsos positivos.
- **FN**: Falsos negativos.
- **P**: Número de muestras positivas en el conjunto. $P = TP + FN$.

- **N**: Número de muestras negativas en el conjunto. $N = FP + TN$.

Las métricas utilizadas han sido:

Exactitud

La exactitud aporta una idea general de la cantidad de aciertos de un modelo; no obstante, puede resultar engañosa para conjuntos de datos con clases desbalanceadas. Se calcula como la proporción de las muestras clasificadas correctamente frente al total [26].

$$\text{Exactitud} = \frac{TP + TN}{P + N}$$

Para conjuntos con varias clases desbalanceadas es conveniente utilizar la Macro Exactitud, que calcula la exactitud por cada clase de manera independiente; de esta manera se otorga la misma importancia a cada clase a pesar de contar con menos muestras. Una equivocación o acierto en una clase minoritaria tendrá el mismo peso que en una mayoritaria [27]. Su cálculo se lleva a cabo mediante la siguiente fórmula:

$$\text{Exactitud}_{\text{macro}} = \frac{1}{C} \sum_{i=1}^{C} \frac{\text{T}_i}{\text{N}_i}$$

donde:

- **C**: Número de clases.
- T_i: Número de predicciones correctas para cada clase i.
- N_i: Número de muestras totales para la clase i.

Esto asegura que en casos como el de este proyecto, donde la clase Hepatocarcinoma es minoritaria, esta reciba la misma atención que aquellas con un elevado número de muestras. Además, resulta de gran utilidad dada la relevancia de conocer la capacidad de nuestros modelos para clasificar la clase Hepatocarcinoma.

Precisión

La precisión (o valor predictivo positivo) indica la cantidad de los positivos predichos que son verdaderamente positivos y refleja cuán exactas y fiables son las predicciones positivas de nuestro modelo. Su cálculo se expresa mediante la siguiente fórmula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Una alta precisión se traducirá en que se produzcan menos falsos positivos [28]. No obstante, en nuestro caso esto no posee especial relevancia, ya que la ecografía no deja de ser una prueba de cribado. Por ello, no es de gravedad predecir que un paciente pueda tener un problema, ya que se descartará más adelante.

Sensibilidad

La sensibilidad (o tasa de verdaderos positivos) indica la cantidad de positivos reales que el modelo es capaz de detectar. Su cálculo se lleva a cabo mediante la siguiente fórmula:

$$\text{Sensibilidad} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Una alta sensibilidad implica la presencia de menos falsos negativos [28]. Esto es esencial en nuestro caso, ya que catalogar como sano a un paciente enfermo puede presentar un alto riesgo para su salud.

Especificidad

La especificidad (o tasa de verdaderos negativos) indica la proporción de negativos reales que nuestro modelo es capaz de detectar correctamente; esto resulta crucial para prevenir falsas alarmas innecesarias. Se calcula como:

$$\text{Especificidad} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

En nuestro caso, a pesar de la importancia que posee, resulta menos relevante que la sensibilidad, ya que la presencia de demasiados falsos positivos podría llevar a la realización de otras pruebas de forma necesaria y generar malestar en los pacientes.

Puntuación F1

La puntuación F representa la media armónica entre la precisión y la sensibilidad, de manera que permite representar ambas en una sola métrica. Su cálculo se realiza mediante la siguiente fórmula:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Especificidad}}{\text{Precision} + \text{Especificidad}} = \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}}$$

6.1.2. Comparativa de modelos

En esta sección se realiza una comparación de los resultados de los modelos entrenados. Se comparan las mejores versiones encontradas para cada modelo.

Primero mostraremos las métricas del conjunto de entrenamiento para comprobar si los modelos han conseguido aprender sobre los datos. Después, se presentan los resultados sobre el conjunto de prueba, donde se puede ver cuánto consiguen generalizar los modelos ante nuevas muestras.

Categorización CIRRHOTIC_STATE

El objetivo principal del proyecto es la detección del hepatocarcinoma, por lo que primero nos centraremos en estos resultados. En esta categoría tenemos tres clases.

	EFFICIENT	CONV	DENSE	RESNET	VIT	CustomCNN
Exactitud macro	33.13 %	83.98 %	41.89 %	33.01 %	33.33 %	39.78 %
Exactitud	69.43 %	92.53 %	50.8 %	25.82 %	71.6 %	73.41 %
Precision	31.59 %	92.89 %	40.24 %	18.85 %	23.87 %	70.20 %
Sensibilidad	33.13 %	83.98 %	41.89 %	33.01 %	33.33 %	39.78 %
Especificidad	66.45 %	91.87 %	74.99 %	66.18 %	66.67 %	70.47 %
F1	29.43 %	87.75 %	34.16 %	13.88 %	27.82 %	40.54 %

Cuadro 6.1: Resultados sobre el conjunto de entrenamiento de CIRRHOTIC_STATE.

Viendo los resultados de los modelos preentrenados sobre el conjunto de entrenamiento, Tabla: 6.1, se puede decir que el desequilibrio de clases afecta gravemente al redimiento. La mayoría de los modelos apenas alcanzan una macro exactitud del 33 %, lo que indica que tienden a predecir casi exclusivamente una única clase (Hígado Sano), haciendo que estas conclusiones sean esperables. CONV es el modelo que mejores resultados ha obtenido a la hora de aprender sobre los datos, con una Macro Exactitud del 83.98 % y F1 87.75 % lo que indica un buen balance entre Sensibilidad, de espacial importancia, y Precisión.

Por su parte, el modelo CustomCNN presenta un desempeño más modesto en entrenamiento, con una Exactitud Macro del 39.78 % y un F1 del 40.54 %. A pesar de que su rendimiento está por debajo de CONV, estos valores sugieren que también está siendo capaz de aprender patrones relevantes para distinguir las tres clases; sin embargo, su efectividad es menor. Su sensibilidad 39.78 % y precisión 70.20 % indican que posee un mejor control para evitar falsos positivos, aunque le cuesta mantener un equilibrio en la detección de las clases minoritarias.

	EFFICIENT	CONV	DENSE	RESNET	VIT	CustomCNN
Exactitud macro	33.74 %	45.67 %	38.21 %	32.97 %	33.33 %	36.33 %
Exactitud	71.05 %	78.6 %	46.43 %	25.59 %	72.17 %	73.15 %
Precision	34.82 %	62.63 %	37.36 %	8.57 %	24.06 %	43.11 %
Sensibilidad	33.74 %	45.67 %	38.21 %	32.97 %	33.33 %	36.33 %
Especificidad	67.11 %	76.7 %	71.73 %	66.23 %	66.67 %	69.84 %
F1	30.18 %	48.27 %	31.14 %	13.6 %	27.94 %	34.59 %

Cuadro 6.2: Resultados sobre el conjunto de prueba de CIRRHOTIC_STATE.

Respecto a los resultados sobre el conjunto de prueba, Tabla: 6.2, se puede observar que, como es esperable, el rendimiento disminuye. Sigue destacando CONV, habiendo conseguido generalizar lo aprendido de manera más eficiente, superando el 45 % en exactitud macro y el 48 % en F1. El resto de modelos preentreandos siguen mostrando resultados cercanos a predecir aleatoriamente, lo que confirma que no han aprendido a diferenciar adecuadamente las clases minoritarias (cirrosis y hepatocarcinoma).

En este caso CustomCNN presenta un rendimiento intermedio. No llega alcanzar los resultados de CONV pero supera al resto de modelos en métricas clave. Demuestra tener cierta capacidad de generalización y sugiere que, pese a no ser óptimo, tiene potencial para mejorar con ajustes adicionales en arquitectura o entrenamiento.

Categorización HEALTHY_LIVERS

	EFFICIENT	CONV	DENSE	RESNET	VIT	CustomCNN
Exactitud	50.13 %	78.86 %	47.86 %	52.14 %	64.82 %	75.49 %
Precision	48.48 %	72.22 %	47.86 %	0 %	87.06 %	76.61 %
Sensibilidad	67.19 %	90.75 %	100 %	0 %	31.13 %	70.24 %
Especificidad	34.46 %	67.95 %	0 %	100 %	95.75 %	80.31 %
F1	56.32 %	80.43 %	64.74 %	0 %	45.86 %	73.29 %

Cuadro 6.3: Resultados sobre el conjunto de entrenamiento de HEALTHY_LIVERS.

Al analizar los resultados de los modelos preentrenados en el conjunto de entrenamiento, Tabla: 6.3, se observa cómo la mayoría de los clasificadores obtienen malos resultados, estando cerca de la predicción aleatoria. Destaca, por encima de los demás, CONV que consigue una F1 de 80.43 %, lo que denota un aprendizaje equilibrado entre sano y enfermo, alcanzando además una alta sensibilidad 90.75 %, especialmente relevante en un contexto clínico.

El modelo CustomCNN, específicamente entrenado para esta tarea, obtiene buenos resultados con una Exactitud del 75.49 %, F1 del 73.29 % y una sensibilidad del 70.24 %. Aunque no alcanza la sensibilidad tan alta de CONV, mantiene un buen balance general, con una precisión del 76.61 % y una especificidad del 80.31 %, lo que sugiere que es efectivo para evitar falsos positivos, un aspecto también importante para no sobrediagnosticar a los pacientes.

Esto indica que CustomCNN logra un aprendizaje equilibrado, con capacidad para distinguir adecuadamente entre sanos y enfermos.

	EFFICIENT	CONV	DENSE	RESNET	VIT	CustomCNN
Exactitud	47.98 %	72.64 %	47.56 %	52.44 %	60.21 %	70.36 %
Precision	46.59 %	66.39 %	47.56 %	0 %	74.83 %	69.70 %
Sensibilidad	64.05 %	86.06 %	100 %	0 %	24.62 %	66.67 %
Especificidad	33.4 %	60.47 %	0 %	100 %	92.49 %	73.72 %
F1	53.94 %	74.95 %	64.47 %	0 %	37.05 %	68.15 %

Cuadro 6.4: Resultados sobre el conjunto de prueba de HEALTHY_LIVERS.

Respecto a los resultados de los modelos preentrenados sobre el conjunto de prueba, Tabla: 6.4, podemos ver que CONV continúa siendo modelo que mejor se comporta, habiendo conseguido generalizar parte de lo aprendido en el entrenamiento. El resto de modelos, EfficientNet, DenseNet, ResNet y ViT, muestran rendimientos muy limitados tanto en entrenamiento como en prueba, con valores bajos en métricas clave como precisión y sensibilidad, lo que evidencia su incapacidad para distinguir correctamente las clases en este problema.

Por su parte, CustomCNN también muestra una generalización adecuada, con una Exactitud del 70.36 %, F1 del 68.15 %, y sensibilidad del 66.67 %. Aunque estas métricas disminuyen respecto al entrenamiento, siguen siendo superiores a la mayoría de los modelos preentrenados restantes, lo que confirma que su arquitectura y entrenamiento personalizado contribuyen a una capacidad sólida de clasificación binaria.

Categorización ORGAN_CLASSIFICATION

	EFFICIENT	CONV	DENSE	RESNET	VIT	CustomCNN
Exactitud macro	20.46 %	79.37 %	20.1 %	20.08 %	38.21 %	91.28 %
Exactitud	73.02 %	92.58 %	73.05 %	72.98 %	78.35 %	95.33 %
Precision	23.2 %	93.39 %	24.61 %	17.48 %	68.25 %	60.52 %
Sensibilidad	20.46 %	79.37 %	20.1 %	20.08 %	38.21 %	80.36 %
Especificidad	80.07 %	95.58 %	80.02 %	80.06 %	85.01 %	95.33 %
F1	17.82 %	83.51 %	17.1 %	17.09 %	39.01 %	86.56 %

Cuadro 6.5: Resultados sobre el conjunto de entrenamiento de ORGAN_CLASSIFICATION.

En los resultados de los modelos preentrenados, Tabla: 6.5, se puede ver cómo la mayoría de modelos consiguen una Exactitud Macro cercana al 20 %; no obstante, su Exactitud normal es más alta, lo cual indica que tienden a predecir una única clase. CONV es, de nuevo, el modelo que mejor ha aprendido sobre los datos.

Respecto a nuestro modelo personalizado, CustomCNN obtiene los mejores resultados, con una Exactitud Macro de 91.28 %, una sensibilidad del 80.36 % y un F1 de 86.56 %. Esto

indica que, demás de aprender a diferenciar entre las cinco clases, tiene un buen balance de Precisión y Sensibilidad.

	EFFICIENT	CONV	DENSE	RESNET	VIT	CustomCNN
Exactitud macro	19.96 %	64.34 %	20 %	19.98 %	39.19 %	69.38 %
Exactitud	73.39 %	87.47 %	73.54 %	73.46 %	79.47 %	80.43 %
Precision	14.7 %	88.23 %	14.71 %	14.73 %	69.68 %	62.63 %
Sensibilidad	19.96 %	64.34 %	20 %	19.98 %	39.19 %	69.38 %
Especificidad	79.97 %	92.17 %	80 %	80.06 %	85.32 %	92.51 %
F1	16.93 %	69.08 %	16.95 %	16.95 %	41.32 %	65.60 %

Cuadro 6.6: Resultados sobre el conjunto de prueba de ORGAN_CLASSIFICATION.

En cuanto a los resultados obtenidos de los modelos preentrenados sobre el conjunto de prueba, Tabla: 6.6, CONV continua posicionado como el único modelo que consigue aprender y generalizar. Los demás mantienen sus malos resultados, siendo incapaces de reconocer más de una clase.

El modelo personalizado, CustomCNN, sigue mostrando un rendimiento mejor que los preentrenados, a pesar de que tampoco consiga generalizar eficazmente su conocimiento, lo cual podría indicar cierto sobreajuste de los datos del entrenamiento.

6.2. Resultados

De los resultados obtenidos se puede concluir que, los modelos preentrenados, a excepción de CONV, no han logrado aprender e identificar con eficacia las imágenes que ya han visto, y por lo tanto no generalizan su conocimiento a imágenes desconocidas. Sus resultados son bajos y cercanos a la probabilidad de elegir aleatoriamente, lo que indica que tienden a predecir la clase mayoritaria. El modelo CustomCNN creado de cero, no alcanza los resultados de CONV pero obtiene un rendimiento intermedio, consiguiendo superar al resto de modelos.

Esto puede atribuirse a varios factores:

- No haber sabido encontrar modelos con arquitecturas lo suficientemente adecuadas para esta tarea, ni una configuración óptima de hiperparámetros específica para este tipo de imágenes.
- No haber aplicado las transformaciones apropiadas a los datos respecto al tamaño de entrada, normalización o canales.
- El desequilibrio entre las categorías, predominando las imágenes de hígados sanos por encima de las patologías. Especialmente relevante la escasez de muestras de imágenes de hepatocarcinoma, limitando significativamente el proyecto.

- La variación entre la posición de los órganos y su orientación entre las ecografías. Esto sugiere que la ecografía no es la mejor prueba de imagen para estos modelos, ya que es un estudio muy dinámico y operador-dependiente, pudiendo la toma de estas imágenes estar sesgada a la preferencia personal del operador. Podrían ser más adecuadas pruebas que generan imágenes más estáticas como los TAC (Tomografía Axial Computarizada) o la RM (Resonancia Magnética), que no tienen estás dependencias, y sobre las cuales hay muchos más trabajos con aplicación de modelos de visión por ordenador.

Sin que esto haya afectado negativamente a los resultado, una consideración importante es que la gran mayoría de las ecografías han sido etiquetadas por un único profesional. Esto podría haber transferido, de forma no intencionada, sus propios sesgos a los modelos. Lo ideal hubiera sido contar con varios profesionales para la tarea de anotación, lo que aportaría más diversidad y objetividad a las clasificaciones.

Finalmente, aunque el conjunto de datos utilizado en este trabajo ha sido ampliado respecto al empleado en el *abstract* recogido en el anexo D,los resultados y las conclusiones extraídas coinciden en gran medida con los allí presentados.

Capítulo 7

Aplicación y despliegue

A pesar de que el objetivo del presente proyecto ha sido desarrollar los modelos, el preprocesamiento y la preparación de los datos, se ha considerado necesario el desarrollo de una aplicación web que integre los modelos de clasificación entrenados, facilitando su uso a los usuarios sin conocimientos previos. Este capítulo se centra en el proceso de elaboración de dicha web, así como en la fase de despliegue.

7.1. Análisis

7.1.1. Requisitos

Requisitos funcionales

ID	Nombre	Descripción
RF-01	Elegir modelo	El sistema debe permitir elegir entre múltiples modelos para clasificar.
RF-02	Elegir modalidad	El sistema debe permitir elegir entre tres modalidades de clasificación.
RF-03	Subir imagen	El sistema debe permitir la subida de imágenes para su clasificación.
RF-04	Clasificar	El sistema debe poder clasificar ecografías.
RF-05	Visualizar decisión	El sistema debe mostrar cómo ha tomado la decisión mediante el algoritmo Grad-CAM.
RF-06	Generar informe	El sistema debe permitir generar y descargar un informe PDF del diagnóstico.

Cuadro 7.1: Tabla de requisitos funcionales.

Requisitos no funcionales

ID	Nombre	Descripción
RNF-01	Facilidad de uso	La aplicación debe de poder ser usada por un usuario sin amplios conocimientos de informática.
RNF-02	Lenguaje de programación	El sistema debe desarrollarse en Python 3.12.
RNF-03	Imágenes	El sistema debe poder aceptar imágenes en formato PNG, JPG, JPEG y BMP.
RNF-04	Entorno	El sistema se deberá poder ejecutar en cualquier maquina que tenga Docker instalado.
RNF-05	Accesibilidad	La aplicación debe de poder ser accedida a través de un navegador web.
RNF-06	Rapidez	La aplicación debe de poder realizar la clasificación y visualización en menos de un minuto.

Cuadro 7.2: Tabla de requisitos no funcionales.

Requisitos de información

ID	Nombre	Descripción
RI-01	Modelos	El sistema debe almacenar los modelos utilizados para clasificar.

Cuadro 7.3: Tabla de requisitos de información.

7.1.2. Casos de uso

Diagrama de casos de uso

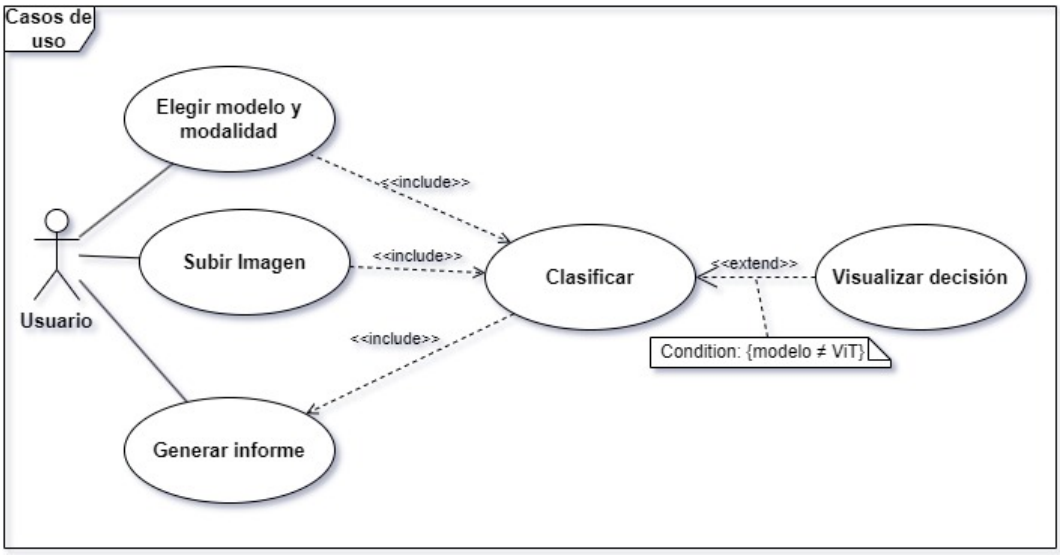

Figura 7.1: Diagrama de casos de uso.

Descripción de los casos de uso

CU-01	**Elegir modelo y modalidad**
Actor	Usuario.
Descripción	El sistema debe permitir al usuario elegir uno de los posibles modelos para una de las tres modalidades de clasificación.
Precondiciones	1. Tanto la aplicación como el servidor están funcionando. 2. El usuario ha accedido a la web.
Flujo principal	1. El sistema selecciona automáticamente una modalidad por defecto. 2. El sistema selecciona automáticamente un modelo por defecto. 3. El sistema busca la mejor versión del modelo seleccionado para la modalidad elegida. 4. El sistema carga el modelo para esa modalidad y se lo indica al usuario. 5. El usuario puede modificar alguna selección o mantenerlas.
Flujo alternativo	3a. El sistema no encuentra ninguna versión del modelo para esa modalidad. 3b. El sistema comunica al usuario que el modelo no está disponible para esa modalidad. 5a. El usuario cambia una de las opciones. Se vuelve al paso 3.

Cuadro 7.4: Descripción del caso de uso CU-01: Elegir modelo y modalidad.

CU-02	**Subir imagen**
Actor	Usuario.
Descripción	El sistema debe permitir al usuario subir imágenes desde su máquina local.
Precondiciones	1. Se ha ejecutado el CU-01: Elegir modelo y modalidad. 2. Hay un modelo cargado en el sistema.
Flujo principal	1. El usuario sube una imagen al sistema. 2. El sistema comprueba que la imagen está en un formato compatible. 3. El sistema muestra la imagen que se ha subido al usuario. 4. El sistema ejecuta automáticamente el CU-03: Clasificar.
Flujo alternativo	2a. El sistema detecta un formato de archivo no compatible. 2b. El sistema comunica al usuario que el formato no es valido. Se vuelve al paso 1.

Cuadro 7.5: Descripción del caso de uso CU-02: Subir imagen.

CU-03	**Clasificar**
Actor	Sistema.
Descripción	El sistema debe clasificar la imagen subida para la modalidad seleccionada con el modelo elegido.
Precondiciones	1. Hay una imagen subida al sistema. 2. Hay un modelo seleccionado y cargado.
Flujo principal	1. El sistema procesa la imagen. 2. El sistema pasa la imagen por el modelo. 3. El sistema obtiene una predicción y la muestra. 4. El sistema comprueba que se puede realizar visualización. 5. De ser posible, el sistema invoca el caso de uso CU-04: Visualizar decisión.
Flujo alternativo	4a. El sistema determina que no se puede realizar la visualización. 4a. El sistema termina el caso de uso sin invocar el CU-04: Visualizar decisión.

Cuadro 7.6: Descripción del caso de uso CU-03: Clasificar.

CU-04	**Visualizar Decisión**
Actor	Sistema.
Descripción	El sistema procesa la visualización de interpretabilidad sobre la imagen cargada.
Precondiciones	1. Hay una imagen cargada en el sistema. 2. Hay un modelo cargado en el sistema. 3. Se ha obtenido un diagnóstico. 4. El modelo cargado debe de poder visualizar decisión.
Flujo principal	1. El sistema carga los ajustes de visualización por defecto. 2. El sistema procesa la imagen. 3. El sistema genera visualización de interpretabilidad. 4. El sistema le muestra al usuario la visualización generada. 5. El sistema permite al usuario cambiar los ajustes de la visualización.
Flujo alternativo	5a. Si el usuario elige otra opción el sistema vuelve al paso 3.

Cuadro 7.7: Descripción del caso de uso CU-04: Visualizar decisión.

CU-05	Generar informe
Actor	Usuario.
Descripción	El sistema debe permitir al usuario generar un informe con los resultados de la clasificación.
Precondiciones	1. Se ha obtenido un diagnóstico.
Flujo principal	1. El usuario selecciona generar el informe. 2. El sistema añade al informe los resultados de la clasificación. 3. El sistema comprueba que haya una visualización hecha. 4. Si existe, el sistema añade la visualización al informe. 5. El sistema genera el PDF. 5. El sistema descarga el informe.
Flujo alternativo	3a Si sistema no encuentra la visualización, el sistema continua con el paso 5 sin incluirla.

Cuadro 7.8: Descripción del caso de uso CU-05: Generar Informe.

7.2. Diseño

7.2.1. Patrones de diseño

Patrón Modelo-Vista-Controlador (MVC)

El patrón Modelo-Vista-Controlador (MVC) es una arquitectura de software que separa una aplicación en tres componentes principales:

- **Modelo**: encargado de la gestión de los datos, la lógica de negocio y el estado de la aplicación.
- **Vista**: encargada de la presentación y la interfaz con el usuario.
- **Controlador**: encargado de orquestar el flujo entre el modelo y la vista, respondiendo a las acciones del usuario.

A pesar de que Streamlit no sigue un MVC clásico, ya que mezcla interacción y renderizado en un mismo flujo, se ha intentado estructurar la aplicación en tres módulos que lo adaptan.

- *main.py* actúa como Controlador, encargándose del flujo general. Hace de intermediario entre *app_logic.py* y *app_view.py*.
- *app_logic.py* actúa como Modelo, manejando la lógica de negocio. Contiene las funciones para la búsqueda y carga de modelos, el preprocesamiento de las imágenes y la generación de informes.
- *app_view.py* actúa como Vista, definiendo la interfaz visual.

Aunque esta división no cumple estrictamente con el patrón MVC, sí respeta el principio de separación de responsabilidades. Esto hace que el código sea modular, escalable y más fácil de mantener; asimismo, evita mezclar la lógica con la presentación, lo que permite modificar la interfaz sin afectar la lógica interna.

7.2.2. Arquitectura

7.2.3. Diagrama de clases

Por cuestiones de claridad, el diagrama de clases ha sido divido en dos partes:

- *LiverImg* y sus clases adyacentes. Figura: 7.2.

- Los modelos de clasificación y sus clases adyacentes. Figura: 7.3.

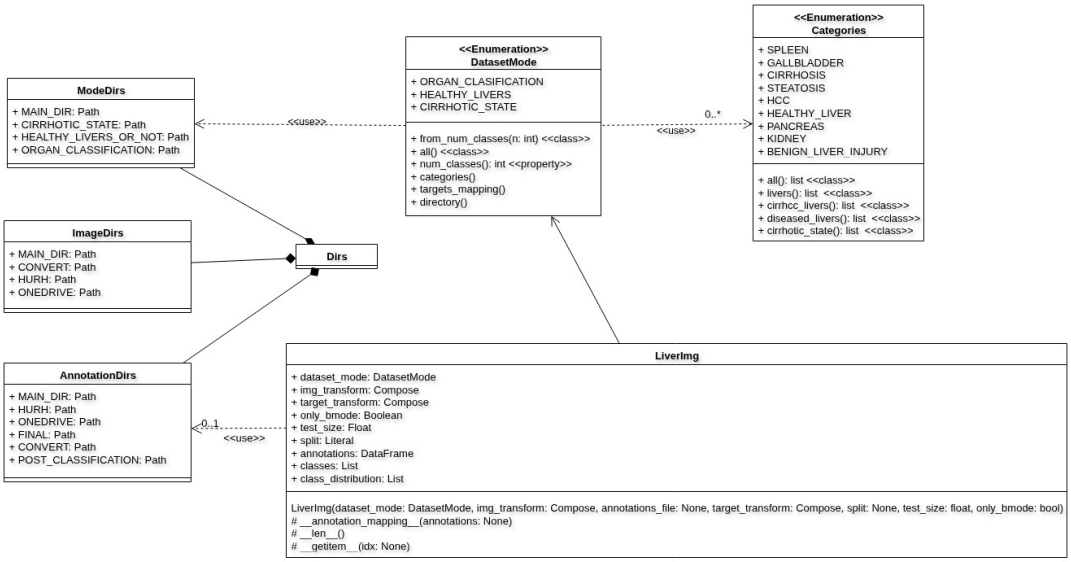

Figura 7.2: Diagrama de Clases de *LiverImg*.

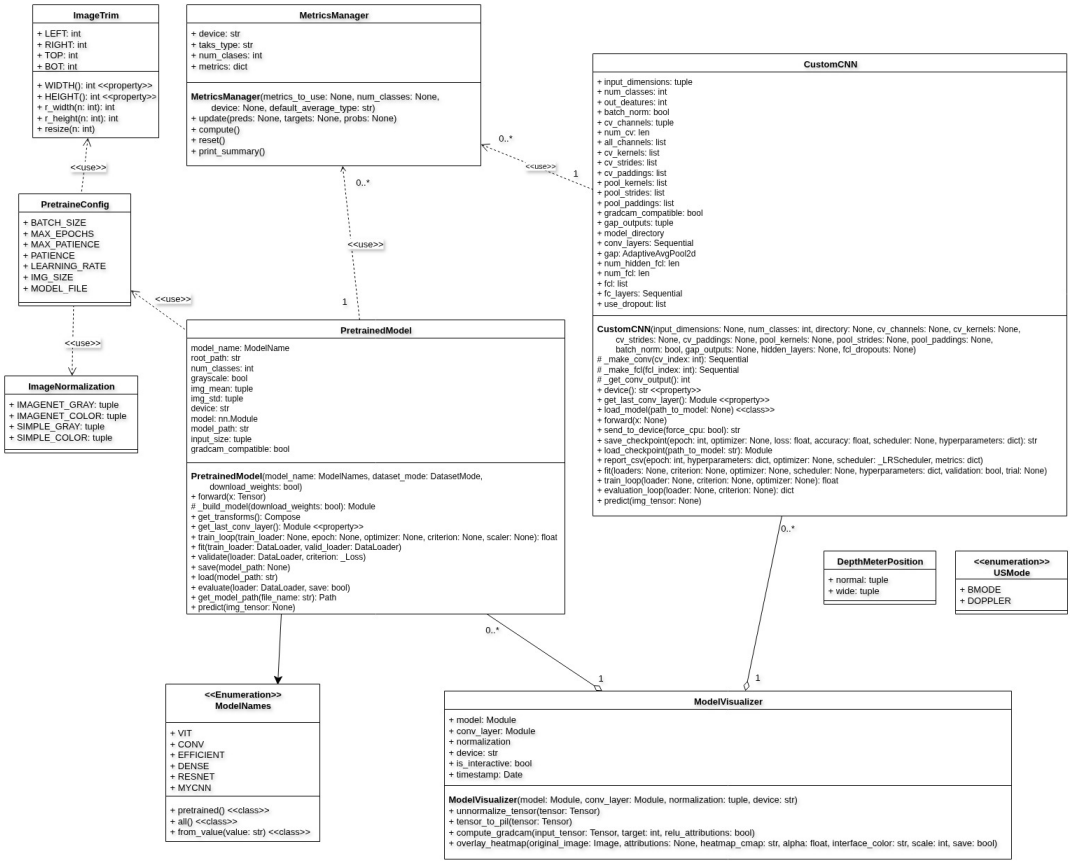

Figura 7.3: Diagrama de Clases de los modelos de clasificación.

7.2.4. Diagramas de secuencia

En esta sección se presentan los diagramas de secuencia asociados a los casos de uso y se describe el flujo principal de la aplicación.

CU-01: Elegir modelo y modalidad

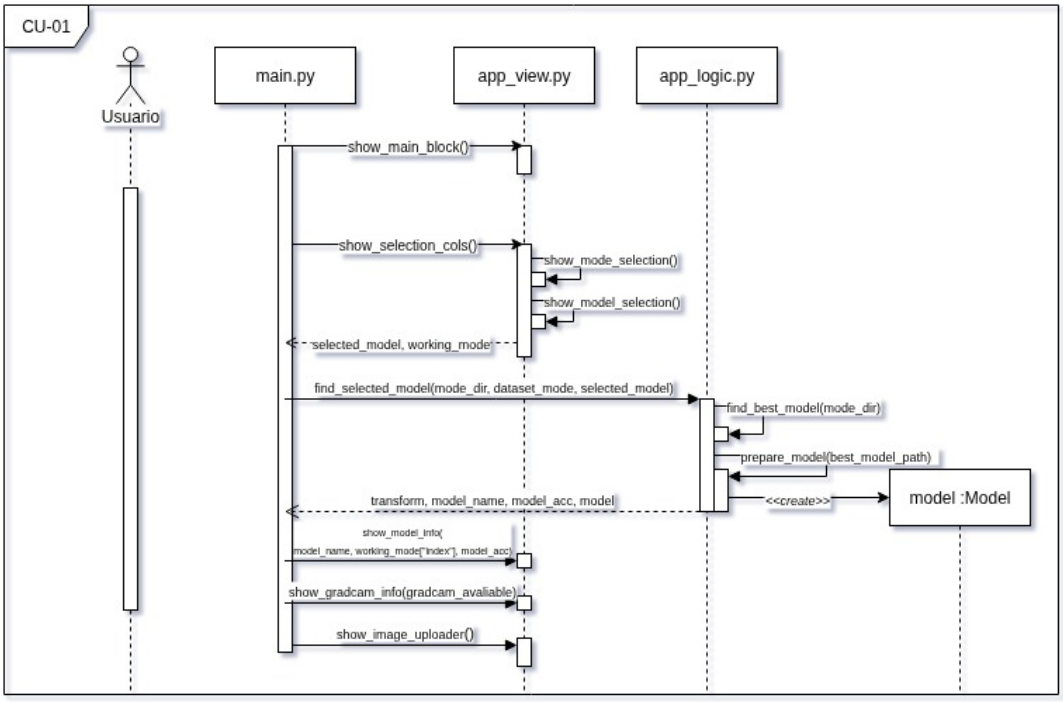

Figura 7.4: Diagrama de secuencia del flujo principal de CU-01.

CU-02: Subir imagen

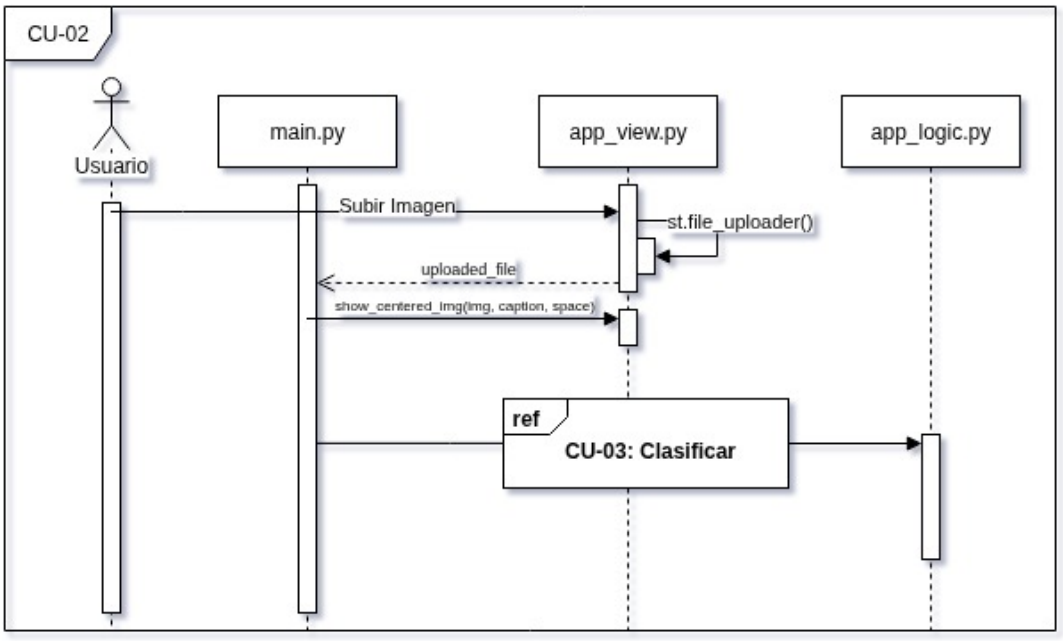

Figura 7.5: Diagrama de secuencia del flujo principal de CU-02.

CU-03: Clasificar

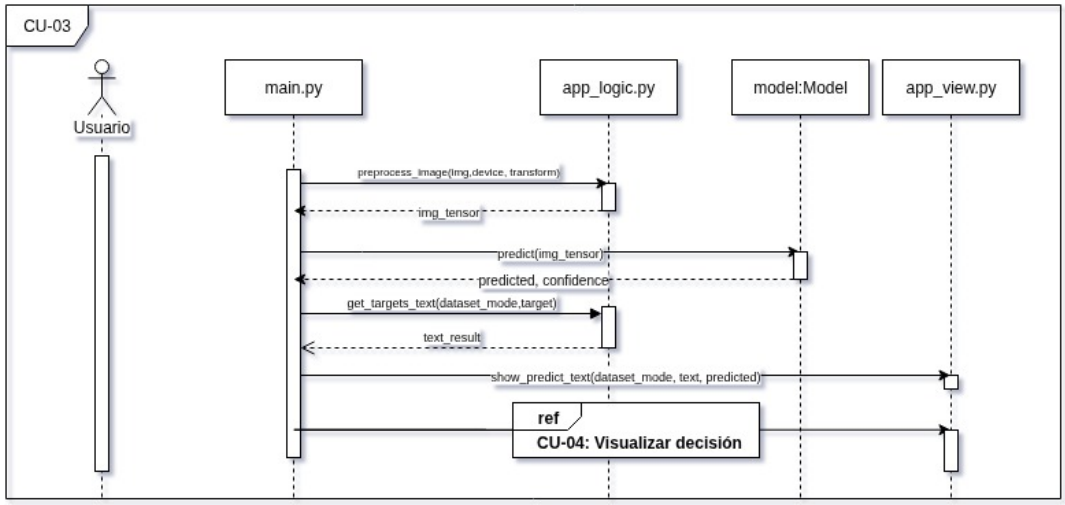

Figura 7.6: Diagrama de secuencia del flujo principal de CU-03.

CU-04: Visualizar decisión

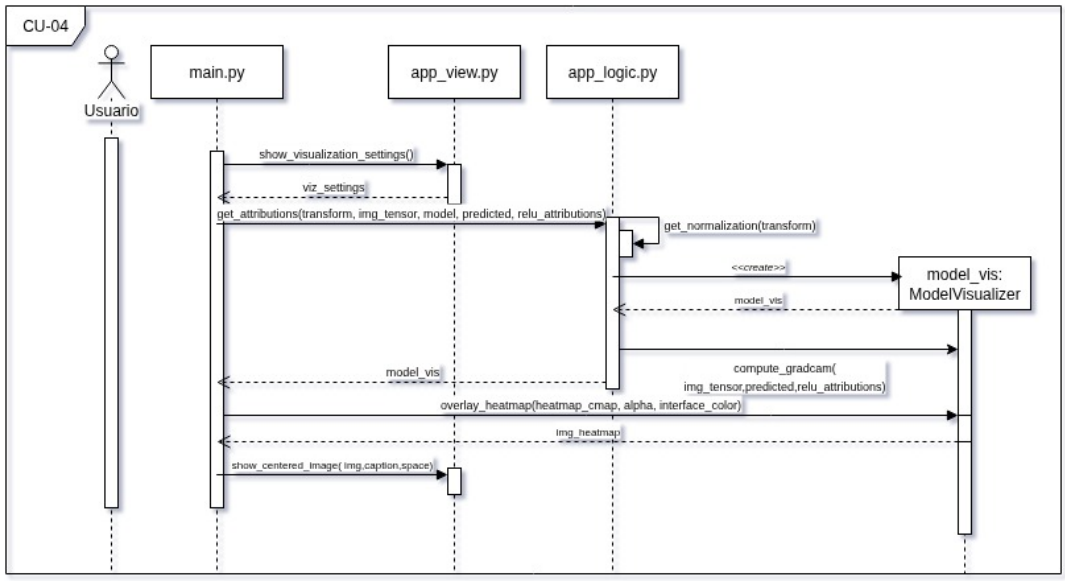

Figura 7.7: Diagrama de secuencia del flujo principal de CU-04.

CU-05: Generar Informe

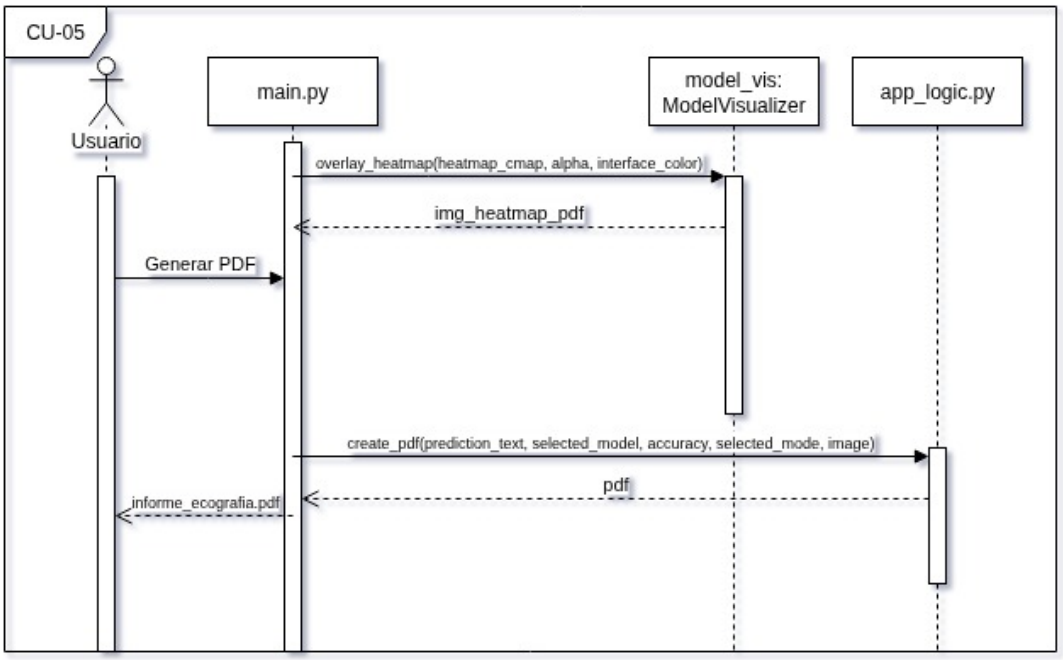

Figura 7.8: Diagrama de secuencia del flujo principal de CU-05.

7.3. Implementación

7.3.1. Tecnologías utilizadas

En el desarrollo de la aplicación web, se ha hecho uso de las siguientes tecnologías:

- Python: elegido por su coherencia con el resto del proyecto,por su rapidez a la hora de desarrollar código y por sus múltiples opciones de *frameworks* web, como Flask, Django, FastAPI o Streamlit.
- Streamlit: biblioteca de Python de código abierto, especialmente popular en proyectos de Aprendizaje Automático. Permite un desarrollo rápido de aplicaciones web ya que se puede hacer uso Python; además, tolera el uso de código HTML y CSS y el despliegue de la aplicación en plataformas como *Streamlit Community Cloud*, *Heroku*, o *AWS*[29].
- Nginx: utilizado como proxy inverso y para el manejo de las conexiones SSL.
- Docker: utilizado para el despliegue del proyecto. Permite ser desplegado en cualquier máquina que lo tenga instalado.

7.3.2. Configuración de Nginx

La configuración de Nginx ha sido diseñada para:

- Redirigir el tráfico del puerto 80, conexiones HTTP, al 443, HTTPS.
- Servir HTTPS utilizando certificados SSL que actualmente son autofirmados, por ser un proyecto de TFG.
- Actuar como proxy inverso de la aplicación de Streamlit, evitando exponer el puerto directamente.

7.3.3. Docker

Para el despliegue de la aplicación en Docker se han utilizado dos contenedores: uno para Streamlit y otro para Nginx. El levantamiento de ambos se hace a través de un *docker-compose* que, además de la configuración de cada contenedor, crea una red interna para que se puedan comunicar.

Contenedor Streamlit

Su función consiste en:

- Construir la imagen con un Dockerfile e indicar desde dónde montar la imagen.
- Levantar volúmenes para los modelos de IA, lo que hace que los modelos no se tengan que cargar en la imagen y así ocupe menos. De esta manera, se tarda menos en cargar y construir. Además, sirve para evitar reconstruir la imagen si se actualizan los modelos, por lo que su entrenamiento queda completamente independiente de la web.
- Exponer solo a la red interna el puerto 8501, que es el utilizado por defecto por Streamlit.
- Relanzar el contenedor en caso de fallo.

El Dockerfile se encarga de, partiendo de una imagen *python:3.12-slim*, crear la estructura de carpetas necesarias para el funcionamiento de la aplicación, instalar dependencias del sistema, instalar los paquetes Python necesarios, copiar los archivos de código y lanzar la aplicación.

Contenedor Nginx

En el caso de este contenedor, se utiliza la ultima imagen de Nginx almacenada en el servidor de Dockerhub. Asimismo, se montan volúmenes para los certificados y para el archivo

de configuración; de este modo los certificados o la configuración pueden ser cambiados o renovados sin necesidad de relanzar el contenedor. Además, se exponen al exterior los puertos, 80 y 443, necesarios para HTTP y HTTPS.

Capítulo 8

Conclusiones y líneas futuras

En este capítulo se recogen las conclusiones más relevantes derivadas del trabajo realizado. A partir de los resultados obtenidos, se reflexiona sobre los objetivos alcanzados y el aprendizaje recibido. Además, se plantean posibles líneas futuras de investigación y mejoras que podrían mejorar el proyecto.

8.1. Consecución de objetivos

Respecto a los objetivos propuestos al inicio del proyecto se consideran logrados satisfactoriamente:

- El haber conseguido un significativo número de imágenes de ecografías y que estás se hayan procesado de forma que puedan ser reutilizadas en trabajos futuros.
- El desarrollo de modelos personalizables de visión por ordenador, que pueden aplicarse a otros contextos, con otros conjuntos de datos y que aceptan tanto clasificación binaria como multiclase.
- El entrenado de modelos preentrenados y propios para clasificar imágenes de ecografías de la región abdominal.
- La implementación de una aplicación que, dada una ecografía, pueda realizar una clasificación incluyendo la visualización de la toma de decisiones del modelo.

8.2. Aprendizaje percibido

Durante el desarrollo de este proyecto se ha percibido un aprendizaje significativo en las siguientes áreas:

- Programación Orientada a Objetos.
- Desarrollo de modelos de Aprendizaje Automático, con especial énfasis en modelos de visión por ordenador y, en concreto, las Redes Neuronales Convolucionales.
- Uso de tecnologías de contenedores como es Docker.
- Aplicación de Patrones de Diseño.
- Desarrollo de aplicaciones web.
- Organización y planificación del trabajo, tanto de manera individual como en lo relativo a la recogida escalonada de los datos.

De todo este proceso cabe poner en valor lo enriquecedor que ha sido:

- Trabajar con datos reales, habiendo sido extraídos directamente de la fuente. Para ello fue necesario acudir al hospital, lo que permitió hablar con los médicos además de ver y aprender de primera mano los procesos que se realizan a las muestras.
- Desarrollar una tubería completa para el procesado de estos datos, desde la extracción en crudo hasta su preparación final para el entrenamiento de modelos.
- Colaborar junto a profesionales de la salud, lo cual desembocó en la posibilidad de colaborar en la escritura y publicación del *abstract* previamente mencionado.

8.3. Trabajo futuro

A pesar de que se considera satisfactoria la consecución de los objetivos establecidos, existen distintas líneas de trabajo sobre las que se podrían realizar avances en las siguientes áreas:

Respecto a las imágenes:

- Ampliar la diversidad de imágenes de ecografías. De forma que no exista un desequilibrio tan grande entre las clases.
- Igualar la cantidad y distribución de imágenes obtenidas de cada ecógrafo.
- Realizar un preprocesamiento más intensivo de las imágenes que permita aprovechar un mayor número de las que los expertos etiquetan o aplicar diferentes aproximaciones (como dividir en partes el sector ecográfico), de una forma más similar a la realizada en los trabajos mencionados en la sección del estado del arte.
- Implementar técnicas de aumento de datos aplicadas únicamente a las clases donde se tienen menos muestras, reduciendo el desbalance.

Respecto a los modelos de IA:

- Implementar conexiones residuales en *CustomCNN*, convirtiéndola una de las opciones de la arquitectura.
- Integrar en *CustomCNN* los mecanismos de optimización de memoria utilizados en *PretrainedModels*.
- Aplicar técnicas de entrenamiento más avanzadas, como puede ser la Validación Cruzada, lo que contribuiría a obtener estimaciones más fiables del rendimiento del modelo, aproximándolo mejor al error esperado.
- Entrenar los modelos en máquinas más potentes que admitieran arquitecturas más profundas y complejas, lotes de tamaños más grandes y reducir el tiempo de entrenamiento de los modelos. Asimismo, implementar mecanismos como la paralelización.

Respecto a la aplicación web:

- Mostrar más información sobre los modelos utilizados para clasificar, como sus métricas o gráficos de entrenamiento.
- Incorporar una opción que utilice un ensamblaje de modelos para la predicción, lo cual permitiría alcanzar un diagnóstico más eficaz.
- Mejorar la seguridad, ya que se realiza un tratamiento de datos médicos potencialmente sensibles.

Apéndice A

Manuales

A.1. Manual de instalación

Este anexo detalla los pasos a seguir para llevar a cabo la instalación de la aplicación web desarrollada en el proyecto, así como la información necesaria para el correcto uso de ella por los usuarios.

A.1.1. Aplicación web

El lanzamiento de la aplicación web se debe hacer desde la raíz del proyecto, ejecutando el siguiente comando:

```
docker compose up
```

Como requisito, es necesario tener instalado Docker[30] y Docker Compose[31]. Dependiendo del sistema operativo, puede que se requieran permisos de superusuario (`sudo`).

Para levantar los contedores también se puede utilizar el *script* `rebuild.sh`, que:

1. Detiene contenedores y volúmenes existentes.
2. Elimina los contenedores no utilizados.
3. Reconstruye y reinicia los contenedores desde cero.

Este *script* es útil para reiniciar el entorno en caso de errores o para aplicar cambios en el código.

Modelos

Para que la aplicación pueda clasificar imágenes, es necesario que los modelos estén disponibles en las rutas correspondientes, sobre las cuales Docker monta los volúmenes.

Se puede lanzar la aplicación sin modelos, pero en ese caso no podrá realizar tareas de clasificación. Más adelante, pueden entrenarse o reentrenarse sin necesidad de reiniciar la aplicación.

Los modelos se almacenan en:

`data/model_state/<modo_funcionamiento>/<tipo_modelo>/`

A.1.2. Entrenamiento de modelos

Para entrenar los modelos, primero deben instalarse las dependencias de Python mediante:

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

Existen dos tipos de modelos, cada uno con su propio script de entrenamiento. Ambos deben ejecutarse desde la raíz del proyecto:

- **Modelo personalizado:**

  ```
  python -m code.scripts.models.train_custom
  ```

- **Modelos preentrenados:**

  ```
  python -m code.scripts.models.train_pretrained
  ```

Ambos scripts solicitan argumentos que deben proporcionarse en su ejecución.

Si la ejecución de un modelo personalizado se detiene, puede retomarse con:

```
python -m code.scripts.models.retrain_custom
```

Alternativa con `PYTHONPATH`

También es posible definir la raíz del proyecto como `PYTHONPATH` para ejecutar los scripts directamente:

```
export PYTHONPATH=$(pwd)
```

En ese caso, será necesario una de las siguientes opciones:

- Dar permisos de ejecución:

```
chmod +x ruta_al_archivo.py
```

- O utilizar el intérprete de Python:

```
python ruta_al_archivo.py
```

Requisitos de datos

Para entrenar los modelos, se requieren imágenes en las siguientes carpetas:

```
data/images/HURH/<categoria>/<modo_ecografia>/
data/images/OneDrive/<categoria>/1280x960/<modo_ecografia>/
```

Así como los archivos de anotaciones correspondientes en:

```
data/csv/final/
```

Si no se dispone de ellos, es necesario ejecutar el proceso de preprocesamiento de imágenes para generarlos.

A.1.3. Preprocesamiento

Para ejecutar los scripts de preprocesamiento es necesario tener imágenes almacenadas en dos carpetas. Además de las imágenes, en cada carpeta se podrá encontrar:

- Ocho directorios que representan categorías de imágenes cuya ruta es:

  `data/images/HURH/`

- Tres directorios que representan categorías de imágenes cuya ruta es:

  `data/images/OneDrive/`

No es necesario disponer de imágenes de ambas fuentes para realizar el preprocesamiento.

Ejecución del preprocesamiento

Para preparar los datos, ejecute el siguiente script desde la carpeta raiz del proyecto:

```
python -m code.scripts.preprocessing.pipeline_classification
```

Una vez lanzado, este script realiza las siguientes tareas:

1. Eliminar artefactos y anotaciones visibles de la interfaz de las ecografías.
2. Separar las imágenes en dos carpetas según la técnica: `bmode` (modo B) y `doppler`.
3. En el caso de OneDrive, filtrar las imágenes compatibles con las del ecógrafo *Aplio i700* y aplicar los pasos anteriores.

Asimismo, genera archivos `CSV` con información de anotación para:

- Registrar las rutas de las imágenes.
- Indicar la técnica utilizada (modo B o Doppler).
- Permitir la reconstrucción del proceso en caso de error.

A.2. Manual de usuario

Tras haber lanzado los contenedores de `Streamlit` y `Nginx`, la web será accesible desde un navegador. Para acceder desde el mismo equipo, vaya a la siguiente dirección:

`https://localhost`

Esto lanzará la vista de la aplicación, tal y como se muestra en la figura: A.1.

Figura A.1: Vista al entrar en la aplicación.

A partir de aquí, el usuario tiene opción de:

- Cambiar las selecciones del modo de clasificación y de los modelos usados para clasificar.
- Subir una imagen, ya sea arrastrándola o seleccionando el botón de subir imagen.

Una vez cargada la imagen, se mostrará en pantalla y será clasificada automáticamente por el sistema, exponiendo un resultado por pantalla, como se observa en la figura A.2.

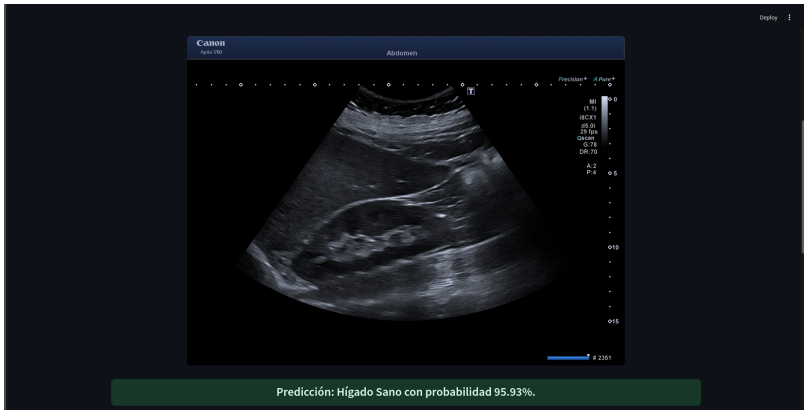

Figura A.2: Vista tras subir una imagen a la aplicación.

Debajo de la imagen, aparecerán nuevas opciones para ajustar la visualización de la toma de decisiones del modelo, junto con dicha visualización. Como se muestra en la figura: A.3.

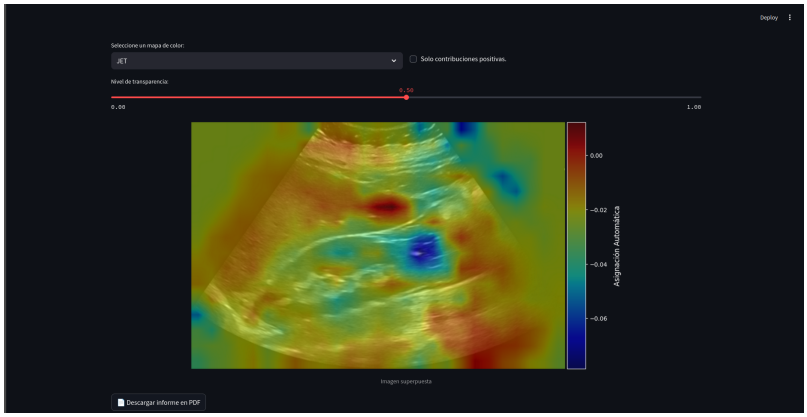

Figura A.3: Visualización de la toma de decisiones del modelo.

Tanto si se ha conseguido mostrar la visualización, como si no, en la parte inferior de la página aparecerá un botón para descargar un informe en formato PDF con los resultados obtenidos. Figura A.4.

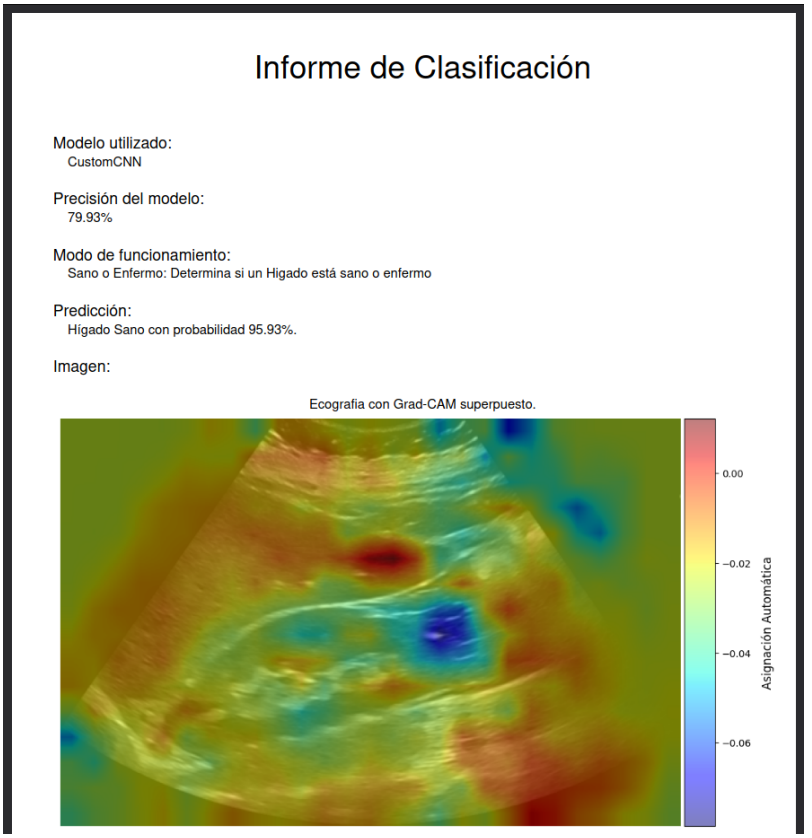

Figura A.4: Ejemplo de un informe PDF, una vez descargado y abierto.

Apéndice B

Contenidos del CD-ROM

Apéndice C

Aporte de imágenes

Lista de Hospitales que han proporcionado imágenes para el proyecto:

- Hospital Universitario Rio Hortega en Valladolid.
- Hospital Clínico Universitario de Valladolid.
- Hospital Virgen De La Concha en Zamora.
- Hospital Provincial de Zamora.
- Hospital Santa Bárbara en Soria.
- Hospital Nuestra Señora de Sonsonetes en Ávila.
- Hospital Comarcal de Medina del Campo.
- Hospital de León.
- Hospital del Bierzo en Ponferrada.
- Hospital Comarcal de Benavente.
- Hospital Santiago Apóstol Miranda de Ebro.
- Hospital General Río Carrión de Palencia.
- Hospital Clínico Universitario de Salamanca.

Ecógrafo del que se han obtenido las imágenes usadas en el proyecto:

Figura C.1: Ecógrafo Canon Aplio i700.

Apéndice D

Abstract

Título: APLICACIÓN DE MODELOS DE INTELIGENCIA ARTIFICIAL EN EL DIAGNÓSTICO ECOGRÁFICO DE CIRROSIS Y HEPATOCARCINOMA. ¿SOBRAREMOS LOS MÉDICOS EN EL FUTURO?

Autores: Diego Rodríguez Arroyo (1), Marina de Benito Sanz (2), Daniela Samantha Ortiz Chimbo (2), Elena Velasco Martínez (2), Jorge Ruiz Rodríguez (2), María Jordán de la Fuente (2), Laura Jiménez González (2), Irene Peñas Herrero (2), Félix García Pajares (2), Raúl García Pajares (3), Adrián Sánchez Zapico (3), Gloria Sánchez Antolín (2). (1) Estudiante de Ingeniería Informática de la UVA. (2) Servicio de Digestivo, HURH, Valladolid. (3) Ingeniero de HP SCDS.

Introducción: La ecografía es una prueba de imagen accesible, inocua y sencilla de aplicar en el cribado de los pacientes cirróticos; sin embargo, requiere de un operador entrenado y consume tiempo, por lo que sería interesante automatizar el diagnóstico.

Materiales y métodos: Se recogieron 1079 imágenes ecográficas de hígado clasificadas en 3 categorías: 755 hígados sanos, 313 cirróticos y 11 con hepatocarcinoma (CHC). Para el preprocesamiento de las imágenes y construcción de los modelos se utiliza Python 3.12 y Pytorch, y se ejecutan en una máquina Linux con una GPU 1060 3Gb. Las imágenes son recortadas a la zona de interés y se aplican CNN con diferentes arquitecturas, así como modelos ViT. Los datos se dividen en 2/3 para el entrenamiento y 1/3 para la evaluación, manteniendo la distribución de categorías del conjunto inicial.

Objetivo principal: Desarrollar modelos para la detección de CHC mediante ecografías de hígado.

Resultados: Cuando se trata de clasificar únicamente un hígado en sano o enfermo, los modelos CNN obtienen tasas de acierto en torno al 98 % en las imágenes que ya han visto y un 75 % en las que no. Cuando se trata de diferenciar entre sano, cirrosis o CHC; los CNN tienen una tasa de acierto de aproximadamente 75 % en las imágenes que han visto previamente y 56 % en las que no, con mejores resultados del modelo de desarrollo propio

frente a los preentrenados en las imágenes de evaluación.

Conclusiones: Los modelos logran aprender e identificar con eficacia las imágenes que ya han visto; sin embargo, no generalizan eficazmente su conocimiento a imágenes desconocidas. Esto puede deberse al desequilibrio entre las 3 categorías, habiendo solo 11 imágenes de CHC (lo cual es la gran limitación de nuestro estudio, estando actualmente aumentando el número de imágenes ecográficas con CHC) y a que la posición de los órganos y su orientación varía mucho entre las ecografías. Nuestros resultados sugieren que la ecografía no es la mejor prueba de imagen para estos modelos ya que es un estudio muy dinámico y operador-dependiente, pudiendo ser más adecuadas pruebas que generan imágenes más estáticas como el TAC o la RM.

Bibliografía

[1] Geert Litjens y col. "A survey on deep learning in medical image analysis". En: *Medical Image Analysis* 42 (2017), págs. 60-88. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1361841517301135.

[2] Dinggang Shen, Guorong Wu y Heung-Il Suk. "Deep Learning in Medical Image Analysis". En: *Annual Review of Biomedical Engineering* 19.Volume 19, 2017 (2017), págs. 221-248. ISSN: 1545-4274. DOI: https://doi.org/10.1146/annurev-bioeng-071516-044442. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-bioeng-071516-044442.

[3] Rafael Bañares Cañizares y D. Rincón García. "Cirrosis hepática". En: *Medicine: Programa de Formación Médica Continuada Acreditado* 12 (2016), págs. 597-605. ISSN: 0304-5412. URL: https://dialnet.unirioja.es/servlet/articulo?codigo=5508446.

[4] Alex S. Befeler y Adrian M. di Bisceglie. "Hepatocellular carcinoma: Diagnosis and treatment". En: *Gastroenterology* 122.6 (2002), págs. 1609-1619. ISSN: 0016-5085. DOI: https://doi.org/10.1053/gast.2002.33411. URL: https://www.sciencedirect.com/science/article/pii/S0016508502130071.

[5] *Ultrasonido*. Accedido: 10/06/2025. URL: https://medlineplus.gov/spanish/ency/article/003336.htm.

[6] *Ecografía Doppler*. Accedido: 10/06/2025. URL: https://medlineplus.gov/spanish/pruebas-de-laboratorio/ecografia-doppler/.

[7] Delia-Alexandrina Mitrea y col. "Hepatocellular Carcinoma Recognition from Ultrasound Images Using Combinations of Conventional and Deep Learning Techniques". En: *Sensors* 23.5 (2023), pág. 2520. ISSN: 1424-8220. DOI: 10.3390/s23052520. URL: https://www.mdpi.com/1424-8220/23/5/2520.

[8] Delia Mitrea. *HCC_MedicalImages_US*. https://www.kaggle.com/datasets/deliamitrea1234/hcc-medicalimages-us. 2023. URL: https://www.kaggle.com/datasets/deliamitrea1234/hcc-medicalimages-us.

[9] Michał Byra y col. "Transfer learning with deep convolutional neural network for liver steatosis assessment in ultrasound images". En: *International Journal of Computer Assisted Radiology and Surgery* (2018). DOI: https://doi.org/10.1007/s11548-018-1843-2.

[10] Gabriel Mancuzo. *5 Fases de la Metodología Scrum*. 2023. URL: https://blog.comparasoftware.com/fases-metodologia-scrum/.

[11] Asociación de Castilla y León de Hematología y Hemoterapia (ACYLHE). *Asociación de Castilla y León de Hematología y Hemoterapia*. https://acylhe.es/. Accedido el 30 de junio de 2025. 2025. URL: https://acylhe.es/.

[12] Eric J. Topol. "High-performance medicine: the convergence of human and artificial intelligence". En: *Nature Medicine* 25.1 (ene. de 2019), págs. 44-56. ISSN: 1546-170X. DOI: 10.1038/s41591-018-0300-7. URL: https://doi.org/10.1038/s41591-018-0300-7.

[13] Invox Medical. *Usos de la inteligencia artificial para el diagnóstico médico*. Accedido: 26/06/2025. Invox Medical. 2023. URL: https://www.invoxmedical.com/blog/usos-de-la-inteligencia-artificial-para-el-diagnostico-medico.

[14] IBM Corporation. *Inteligencia artificial en medicina*. Accedido: 26/06/2025. IBM. 2024. URL: https://www.ibm.com/es-es/topics/artificial-intelligence-medicine.

[15] Aditi Kothiya. *Understanding "convolution" operations in CNN*. Analytics Vidhya. Jun. de 2021. URL: https://medium.com/analytics-vidhya/convolution-operations-in-cnn-deep-learning-compter-vision-128906ece7d3.

[16] Dominik Scherer, Andreas Müller y Sven Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". En: *Artificial Neural Networks – ICANN 2010*. Ed. por Konstantinos Diamantaras, Wlodek Duch y Lazaros S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, págs. 92-101. ISBN: 978-3-642-15825-4.

[17] Sandra Navarro. *Capas de pooling en una red neuronal convolucional*. KeepCoding. Abr. de 2024. URL: https://keepcoding.io/blog/capas-pooling-red-neuronal-convolucional/.

[18] Min Lin, Qiang Chen y Shuicheng Yan. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE]. URL: https://arxiv.org/abs/1312.4400.

[19] *What is overfitting?* IBM. Oct. de 2021. URL: https://www.ibm.com/think/topics/overfitting.

[20] Sergey Ioffe y Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[21] Nitish Srivastava y col. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". En: *Journal of Machine Learning Research* 15.56 (2014), págs. 1929-1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[22] DataScientest. *¿Qué es el método Grad-CAM?* DataScientest. 2024. URL: https://datascientest.com/es/que-es-el-metodo-grad-cam.

[23] MathWorks. *Grad-CAM explains why deep learning networks make certain predictions*. https://es.mathworks.com/help/deeplearning/ug/gradcam-explains-why.html. Accedido: 2025-07-02. 2025.

[24] Olga Russakovsky y col. "ImageNet Large Scale Visual Recognition Challenge". En: *International Journal of Computer Vision* 115.3 (dic. de 2015), págs. 211-252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

[25] *Pytorch. Datasets and DataLoaders*. Accedido: 26/06/2025. URL: https://docs.pytorch.org/tutorials/beginner/basics/data_tutorial.html.

[26] *Classification: Accuracy, recall, precision, and related metrics*. Accedido: 27/06/2025. Google. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

[27] *Classification (ML training) Optimizing metrics*. Accedido: 27/06/2025. Prediktera. URL: https://help.prediktera.com/breeze/classification-ml-training.

[28] *Choosing the Right Metrics: Recall, Precision, PR Curve and ROC Curve Explained*. Accedido: 27/06/2025. EvidentlyAI. 2025. URL: https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall.

[29] Dora Gurova. *What is Streamlit: All Why's and How's Answered*. Ene. de 2025. URL: https://uibakery.io/blog/what-is-streamlit.

[30] Docker Inc. *Get Docker*. Último acceso: julio de 2025. 2024. URL: https://docs.docker.com/get-docker/.

[31] Docker Inc. *Install Docker Compose*. Último acceso: julio de 2025. 2024. URL: https://docs.docker.com/compose/install/.