



Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

**Clasificación y Explicabilidad de Imágenes
Tumorales Cerebrales con Aprendizaje
Profundo (Deep Learning)**

Autor:

Jorge Arranz Simón

Tutor:

D. Teodoro Calonge Cano

A los que me veis superarme paso a paso, gracias.

Agradecimientos

En primer lugar, me gustaría dar las gracias a la Universidad de Valladolid y a la Escuela de Ingeniería Informática, por proporcionarnos un magnífico lugar donde aprender y desarrollarnos como personas.

Me gustaría agradecer también a Teodoro, mi tutor, su tiempo y dedicación en este proyecto.

A mi familia, por hacerme crecer feliz y exigir siempre lo mejor de mí.

A mis amigos, por estar siempre dispuestos a todo y no cerrar nunca una conversación.

A todos los que me habéis acompañado durante estos años, si hoy en día soy quién soy, es por vosotros.

Gracias.

Resumen

Hoy en día, la Inteligencia Artificial (IA) se entrena para gran variedad de tareas, como el reconocimiento de patrones, la toma de decisiones y la resolución de problemas. De entre todo el abanico de posibilidades que ofrecen estas técnicas, en este trabajo se busca aplicar modelos basados en IA para resolver problemas relacionados con la Medicina, y más concretamente, con los tumores cerebrales, una de las causas de muerte principales entre hombres y mujeres. La detección prematura de los mismos es la mejor ayuda para lograr un tratamiento curativo, y es por esto, que el objetivo fundamental que se plantea es el desarrollo de un modelo, que utilice Redes Neuronales Convolucionales (CNN), y permita caracterizar las imágenes de resonancia magnética entre tres tipos de tumores: Meningioma, Glioma y tumor de Pituitaria. A mayores, se incluirá una capa de explicabilidad Grad-CAM, que permitirá visualizar las regiones de la imagen utilizadas para realizar la predicción.

El modelo final obtenido logra una tasa de clasificación superior al 90 %, pudiendo entender a través de la capa de interpretabilidad cuáles son las zonas de más influencia para cada predicción.

Palabras clave

Inteligencia Artificial, Aprendizaje Profundo, Redes Neuronales Convolucionales,
Grad-CAM, Tumores cerebrales, Clasificación de imágenes.

Abstract

Nowadays, Artificial Inteligente (AI) is used for a wide variety of tasks, such as pattern recognition, decision making and problem solving. From the full range of possibilities offered by these techniques, this project seeks to apply AI-based models to solve a Medical issue, and more specifical, with brain tumours, one of the main causes of death among men and women. The early detection of these tumours is the best way to achieve a sucessful treatment. For this reason, the main objective is to develop a model based on Convolutionales Neuronal Networks (CNN) which allows the characterisation of magnetic resonance images between thre types: Meningioma, Glioma and Pituitary tumour. In addition, a Grad-CAM explanatory layer will be included, which would allow visualisation of the image regions used to make the prediction.

The final model proposed reaches more than a 90% of correct answers, making possible to understand through the interpretability layer which are the most influential areas for each prediction.

Key words

Artificial Inteligence, Deep Learning, Convolutional Neuronal Networks, Grad-CAM, Brain tumour, Image Classification.

Índice general

1 Introducción	13
1.1 Motivación	14
1.2 Planteamiento inicial	16
2 Gestión del proyecto	19
2.1 Metodología de trabajo	19
2.2 Entregables del proyecto	20
2.2.1 Evaluación	21
2.3 Planificación	21
2.3.1 Planificación inicial	22
2.3.2 Variaciones en la planificación inicial	23
2.4 Entorno de trabajo	24
2.4.1 Hardware	24
2.4.2 Software	24
Sistema operativo	24
Lenguajes de programación y herramientas utilizadas	25
Python	25
Anaconda	25
Desarrollo Aplicación Web	25
Flask	26
Gunicorn	26
Astah	26
Overleaf	27
Otras Herramientas	27
2.5 Alternativas de la plataforma de trabajo	27
2.5.1 Lenguaje de programación	27
2.5.2 Biblioteca Python	28
3 Fundamento teórico	31
3.1 Teoría básica de Redes Neuronales	31
3.1.1 La neurona	33
3.1.2 La función de activación	34
3.1.3 La función de pérdida	35
Error cuadrático medio	35
Error absoluto medio	36
Binary Cross-Entropy	36
Entropía Cruzada Categórica	36

3.2	Redes Neuronales Convolucionales	37
3.2.1	La convolución	37
	Filtro	38
3.2.2	Tamaño de paso y relleno	39
	<i>Stride</i>	40
	<i>Padding</i>	40
3.2.3	Pooling	41
3.2.4	Aplanado o <i>flatten</i>	42
3.3	Explicabilidad	42
3.3.1	Explicabilidad en CNN	43
3.3.2	Grad-CAM	44
4	Tratamiento de los datos	47
4.1	Obtención de los datos	47
4.2	Transformaciones	48
	Generación de diccionarios	49
	Implementación de la clase abstracta <i>Dataset</i>	50
	Transformaciones	50
5	Modelado	53
5.1	Planteamiento de la red	53
5.2	Pasos para implementar una CNN	54
	Data Loaders	54
	Definición del modelo	54
	Entrenamiento del modelo	55
5.3	Comparación entre los diferentes modelos implementados	56
5.3.1	Modelos incluidos en la comparación	57
	Versión 1	57
	Versión 2	58
	Versión 3	58
	Versión 4	59
	Versión 5	59
5.3.2	Comparación de los resultados obtenidos	60
6	Explicabilidad	65
6.1	Implementación	65
7	Aplicación	67
7.1	Análisis	68
7.1.1	Requisitos funcionales	68
7.1.2	Requisitos no funcionales	68
7.1.3	Requisitos de información	68
7.1.4	Casos de uso	69
	CU-001. Subir Imagen	69
	CU-002. Diagnosticar	70
	CU-003. Obtener diagnóstico	70

CU-004. Obtener interpretabilidad	71
7.2 Diseño	71
7.2.1 Tecnologías utilizadas	71
7.2.2 Patrones de Diseño	72
Patrón MVC	72
Patrón Singleton	73
Patrón Fachada	74
7.2.3 Diagrama de clases	74
7.2.4 Diagramas de secuencia	75
CU-001 - Subir imagen	75
CU-002 - Diagnosticar	76
CU-003 - Obtener Diagnóstico	76
CU-004 - Obtener interpretabilidad	77
8 Conclusiones y trabajo futuro	79
8.1 Trabajo futuro	80
A Salida de las capas de la CNN	85
A.1 Filtros antes del entrenamiento	86
A.2 Representación 3D de los filtros	88
A.3 Paso de una imagen por la red	90
B Manual de instalación	95
B.1 Requisitos	95
B.2 Estructura de directorios	95
B.3 Instalación y despliegue	96
C Manual de Usuario	97
C.1 Acceso a la página	97
C.2 Subir una imagen	98
A través del botón <i>Examinar</i>	98
A través del <i>drag and drop</i>	98
C.3 Diagnóstico	99

Índice de figuras

1.1	Flujo de trabajo en la radiología [2]	14
1.2	Resultado de una resonancia magnética cerebral [4]	15
1.3	Ubicación de los tumores cerebrales. <i>Fuente: https://gammaknife.com.ec/</i> .	16
1.4	Logos Python y PyTorch	17
1.5	Esquema aplicación Web	18
2.1	Ciclo de vida de un proyecto basado en CRISP-DM [14]	20
2.2	Diagrama de Gantt 1	23
2.3	Diagrama de Gantt 2	23
2.4	Logo de GNU y Linux	24
2.5	Logo PyTorch	25
2.6	Logo de Anaconda	25
2.7	Logos de HTML, CSS y Javascript	26
2.8	Logo de Flask	26
2.9	Logo de Gunicorn	26
2.10	Logo de Astah	26
2.11	Logo de Overleaf	27
2.12	Logo de Draw.io	27
2.13	Logos de Python, R y Julia	28
2.14	Logos de Keras, TensorFlow y PyTorch	29
3.1	Ejemplo de MLP. <i>Fuente: bootcampai.medium.com</i>	32
3.2	Ejemplo de RNN. <i>Fuente: torres.ai/</i>	33
3.3	Ejemplo de CNN. <i>Fuente: kdnuggets.com</i>	33
3.4	Operación básica de una neurona	34
3.5	Algunas funciones de activación [26]	35
3.6	Operación de convolución [27]	37
3.7	Efecto de unos filtros concretos. <i>Fuente: herevego.com</i>	38
3.8	Operación de convolución para múltiples canales de entrada [26]	39
3.9	Paso de un filtro sobre una matriz en una convolución	40
3.10	Ejemplo de tamaño de paso. <i>Fuente: Computer.org</i>	40
3.11	Ejemplo de padding. <i>Fuente: analyticsindiamag.com</i>	41
3.12	Ejemplo de Max-pooling y Average-Pooling. <i>Fuente: Springer.com</i>	41
3.13	Flujo de una Red Neuronal Convolucional	42
3.14	Proceso de construcción de un sistema de IA robusto	43
3.15	Flujo de Grad-CAM	44
4.1	Distribución de clases	48

4.2	Ejemplo de observaciones en los subconjuntos de entrenamiento (superior) y test (inferior)	49
4.3	A la izquierda, la imagen transformada para entrenamiento, a la derecha, la imagen para test	51
5.1	Evolución de la precisión de la precisión durante la fase de entrenamiento del modelo	56
5.2	Arquitectura de Red - Versión 1	57
5.3	Arquitectura de Red - Versión 2	58
5.4	Arquitectura de Red - Versión 3	59
5.5	Arquitectura de Red - Versión 4	59
5.6	Arquitectura de Red - Versión 5	60
5.7	Resumen de la arquitectura proporcionado por PyTorch	63
5.8	Arquitectura definitiva de la red convolucional. Estilo LeNet	63
6.1	Grad-CAM sobre lo distintos tipos de tumores incluidos en el estudio	66
7.1	Diagrama de casos de uso	69
7.2	Patrón MVC. Fuente: https://developer.mozilla.org	73
7.3	Patrón Singleton	74
7.4	Patrón Fachada. [34]	74
7.5	Diagrama de clases	75
7.6	Diagrama de secuencia CU-001	75
7.7	Diagrama de secuencia CU-002	76
7.8	Diagrama de secuencia CU-003	76
7.9	Diagrama de secuencia CU-004	77
A.1	Primera capa de filtros	86
A.2	Segunda capa de filtros	86
A.3	Tercera capa de filtros	87
A.4	Cuarta capa de filtros	87
A.5	Representación tridimensional de filtros en la primera capa	88
A.6	Representación tridimensional de filtros en la segunda capa	88
A.7	Representación tridimensional de filtros en la tercera capa	89
A.8	Representación tridimensional de filtros en la cuarta capa	89
A.9	Primera capa convolucional	90
A.10	Primer Max-Pooling	90
A.11	Segunda capa convolucional	91
A.12	Segundo Max-Pooling	91
A.13	Tercera capa convolucional	92
A.14	Tercer Max-Pooling	92
A.15	Cuarta capa convolucional	93
A.16	Cuarto Max-Pooling	93
B.1	Estructura de las carpetas	95
C.1	Vista inicial	97

C.2	Ejemplo de subir una imagen a través del <i>drag and drop</i>	98
C.3	Vista cuando la imagen se ha subido correctamente	99
C.4	Vista mientras se ejecuta el modelo	99
C.5	Vista del resultado final	100

Índice de cuadros

2.1 Planificación del proyecto	22
5.1 Comparación entre las distintas versiones de CNN implementadas	61
7.1 Requisitos Funcionales	68
7.2 Requisitos No Funcionales	68
7.3 Requisitos de Información	68
7.4 Descripción del CU-001: Subir imagen	69
7.5 Descripción del CU-002: Diagnosticar	70
7.6 Descripción del CU-003: Obtener diagnóstico	70
7.7 Descripción del CU-004: Obtener explicabilidad	71

Siglas

CNN Red Neuronal Convolucional. 3, 16, 27, 33, 42, 43, 45, 54, 65

CRISP-DM Cross Industry Standard Process for Data Mining. 19

CV Visión Artificial. 14, 15

IA Inteligencia Artificial. 1, 5, 13, 14, 17, 43, 79

IRM Imágenes de Resonancia Magnética. 15

MLP Perceptrón Multicapa. 32, 42

MV Máquina Virtual. 80

MVC Modelo-Vista-Controlador. 72

NN Red Neuronal. 20, 21, 31

RNN Red Neuronal Recurrente. 32

SVM Support Vector Machine. 17

TFG Trabajo Fin de Grado. 14

UVa Universidad de Valladolid. 21, 24, 26

Capítulo 1

Introducción

Hoy en día, la Inteligencia Artificial (IA) es uno de los campos en desarrollo que más curiosidad e intriga suscitan entre toda la población, debido al potencial y la versatilidad de sus aplicaciones, de las que aún no se conoce ni el límite, ni el impacto que tendrán en la sociedad. De entre su infinidad de aplicaciones, destacan principalmente aquellas que, bien por su trascendencia en la vida cotidiana de las personas, o por los sorprendentes resultados que proporcionan, hacen que la Inteligencia Artificial no pase indiferente ante nadie.

En este marco, podemos encajar los avances desarrollados en procesamiento del lenguaje natural (NLP), que han supuesto el desarrollo de asistentes virtuales inteligentes con altas capacidades, o la generación de imágenes a través de texto, cuyo desarrollo llega a niveles en los que es difícil reconocer, a simple vista, si una imagen es real o ha sido creada por una máquina. Hay un sinfín de posibilidades que, según la tendencia actual, nunca dejarán de sorprendernos.

Si bien estos servicios son de cierta utilidad, la fuerza del avance en este campo no está limitada a un entretenimiento tan básico, sino que, tras todas estas aplicaciones más llamativas o mediáticas, se ocultan gran cantidad de funcionalidades de gran versatilidad, basadas en técnicas de IA, que mejoran verdaderamente la calidad de vida de las personas y proporcionan herramientas con una potencia inimaginable tiempo atrás.

Aunque hoy en día no sería correcto, en términos de ética [1], delegar ninguna decisión en modelos artificiales inteligentes, sí es necesario reconocer que proporcionan un argumento científico, aportando una mayor seguridad en la toma de decisiones y ofreciendo un argumento cuantificable en su defensa. La utilización de información masiva, tal que un humano jamás sería capaz de interpretar, y la rapidez de las soluciones, son algunas de las ventajas fundamentales que pueden ofrecernos estos sistemas.

Uno de los campos en los que este tipo de aplicaciones está en auge es la Medicina y, en particular, en la Radiología [2]. Esta especialidad médica se ocupa de generar imágenes del interior del cuerpo y utilizarlas para el diagnóstico. Por lo tanto, resulta esencial en gran parte de los procedimientos terapéuticos. Entre estos procedimientos, donde la aportación de la radiología es determinante, se enmarcan, entre tantos, los tratamientos oncológicos.

Cabe destacar que las muertes causadas por el cáncer ocupan y ocuparán los puestos

más altos de los rankings de las causas de muerte entre la población mundial [3], por lo que el avance hacia la mejora en el tratamiento de esta patología es una prioridad. Una de las líneas de actuación en la mejora de este tratamiento, en la que la Inteligencia Artificial ha irrumpido con gran impacto, es, precisamente, en la detección precoz de los tumores, considerada una de las claves para poder superar la enfermedad.

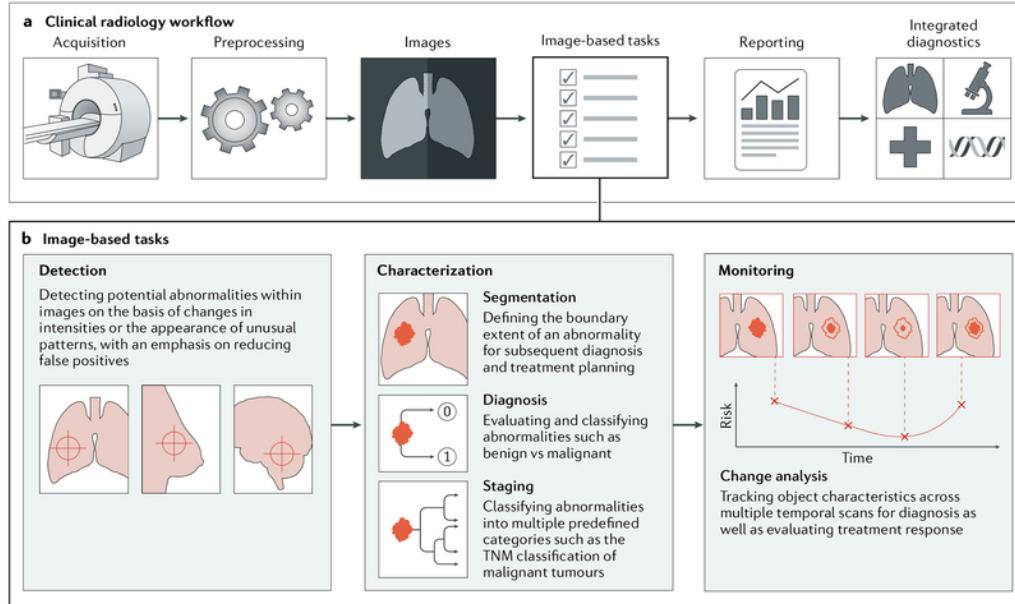


Figura 1.1: Flujo de trabajo en la radiología [2]

Estos modelos artificiales se introducen en todas aquellas tareas relacionadas con las imágenes, siendo posible aplicarlos, tanto para la detección, como para la caracterización y la monitorización. En todas estas fases, cumplen un papel fundamental, ya que supone un argumento científico cuantificable para apoyar la decisión que tome el profesional en cuestión.

1.1. Motivación

El objetivo de este Trabajo Fin de Grado es investigar las técnicas de Inteligencia Artificial relacionadas con el análisis de imágenes, conocidas como Visión Artificial (CV), que se centran en el desarrollo de algoritmos y modelos de IA, capaces de analizar, interpretar y comprender imágenes visuales de manera similar a como lo hacemos los seres humanos. Esto implica tareas como el reconocimiento de objetos, la segmentación de imágenes o la clasificación de contenido, entre otras tantas posibilidades.

Estas son las técnicas en las que nos debemos centrar para entender los avances en esta línea, investigando, tanto las de clasificación, en las que se basará la caracterización del tumor, según su tipo y localización, como en los procedimientos encargados de hacer que la decisión tomada a través del cómputo de millones de parámetros, sea fácilmente interpretable para el ojo (y cerebro) humano, acercando así el conocimiento generado por las redes neuronales a todas aquellas personas no profesionales en este ámbito de una forma visual.

El desconocimiento es muchas veces una causa de miedo en la sociedad, por lo que tratándose de temas tan sensibles como el que nos ocupa, cualquier información que contribuya a entender el porqué de la decisión tomada, ayudará a digerir mejor una situación en la que no se debe permitir un diagnóstico erróneo.

Con este objetivo en mente, se analizarán las técnicas de interpretabilidad más utilizadas hoy en día, para tratar de encontrar el método que aporte más información al problema concreto que se quiere tratar: la clasificación de tumores cerebrales.

Siguiendo los pasos del flujo de la figura 1.1, este proyecto se enmarca en la fase posterior a la obtención de las imágenes: la caracterización de los tumores. En el caso de ser cerebrales, la práctica más común en su detección es a través de las Imágenes de Resonancia Magnética (IRM). Con este fin, utilizan técnicas no invasivas a través de campos magnéticos y ondas de radio para obtener imágenes detalladas del interior del cuerpo humano.

En nuestro caso concreto, se utilizarán diferentes tomas obtenidas del cerebro, como las mostradas en la Figura 1.2, con las que se consigue crear una representación tridimensional fácilmente interpretable para el cerebro humano, en las que se puede analizar, desde cada uno de los perfiles, el estado de la masa cerebral.

El objetivo será tratar de aplicar las técnicas de Visión Artificial (CV) necesarias para realizar la traducción de las imágenes a la computadora, de tal manera, que no sólo sea capaz de codificarlas, sino que sea posible interpretarlas y seccionar aquellos aspectos, o zonas de la imagen en este supuesto, que son determinantes a la hora de detectar cada uno de los tumores.

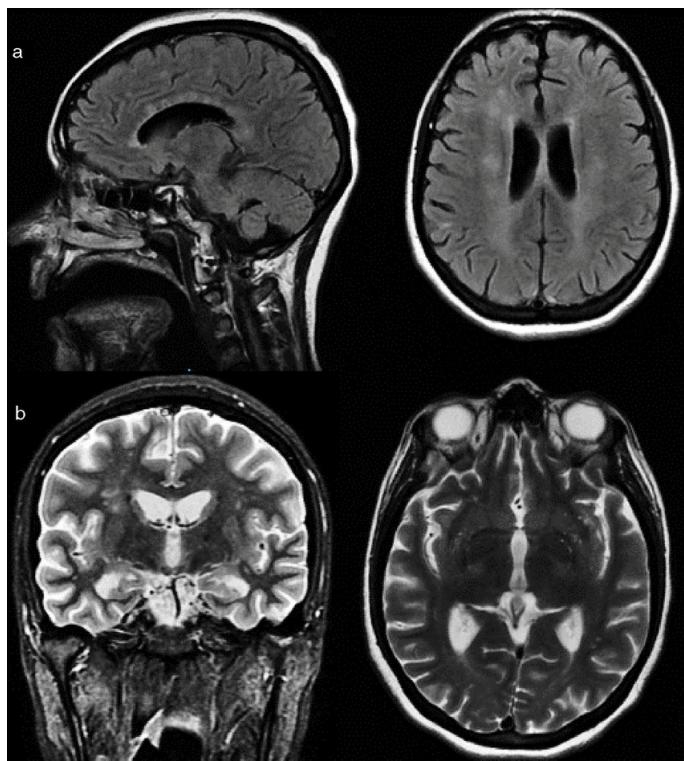


Figura 1.2: Resultado de una resonancia magnética cerebral [4]

Estas imágenes serán utilizadas como *inputs* para su caracterización en uno de los siguientes grupos [5]:

- **Gliomas.** Se originan en las médulas gliales, que son las células de soporte del sistema nervioso central.
- **Meningiomas.** Son tumores que se originan en las meninges, las membranas que recubren el cerebro y la médula espinal. A diferencia de los gliomas, son tumores benignos en la mayoría de los casos, por lo que no se diseminan a otras partes del cuerpo. Sin embargo, dependiendo de su tamaño y ubicación, pueden llegar a causar síntomas y requerir tratamiento.
- **Tumores de la hipófisis o pituitaria.** Este tipo de tumores se desarrollan en la glándula pituitaria, que se encuentra en la base del cerebro. Pueden ser benignos, pero en algunos casos pueden llegar a resultar malignos.
- **Sin tumor.**



Figura 1.3: Ubicación de los tumores cerebrales. Fuente: <https://gammaknife.com.ec/>

1.2. Planteamiento inicial

Con el objetivo en mente de realizar una clasificación a través de las imágenes, existen múltiples alternativas que pueden ser de utilidad, entre las que destacan los métodos basados en el Aprendizaje Profundo o *Deep Learning*. Las Redes Neuronales Convolucionales (CNN) obtienen los mejores resultados debido, principalmente, a su capacidad para extraer características relevantes de las imágenes de manera automática, y poder realizar así una clasificación precisa.

Las alternativas no basadas en Aprendizaje Profundo, como puede ser la extracción de características manuales, o los modelos estadísticos, presentan una gran cantidad de inconvenientes.

En el caso de las manuales, se requiere de un diseño no automático de algoritmos de extracción de características para identificar patrones específicos, como pueden ser los bordes, texturas o formas geométricas, entre otros. Sobre estos resultados se emplean algoritmos de clasificación más tradicionales como SVM o los vecinos más próximos. Se puede ver un ejemplo de clasificación con estas técnicas en [6].

Es por esto que se toma la decisión de realizar el estudio a través de la aplicación de técnicas de Aprendizaje Profundo, y en particular, a través de las Redes Neuronales Convolucionales.

A través de estas redes se tratará de extraer el máximo de información posible de las imágenes de las resonancias magnéticas, para poder clasificarlas posteriormente en uno de los grupos. Una vez que se ha realizado la clasificación, se obtendrá de las convoluciones una capa que nos informe del peso que ha tenido cada uno de los píxeles en la categorización, de forma que podamos juzgar el resultado obtenido a posteriori.

Con el objetivo de buscar la máxima efectividad y libertad a la hora de desarrollar el modelo, se ha tomado la decisión de realizar el proyecto en Python [7], lenguaje de programación de alto nivel, que ofrece grandes ventajas a la hora del desarrollo de proyectos basados en Aprendizaje Automático.



Figura 1.4: Logos Python y PyTorch

Estas ventajas radican en la claridad y legibilidad de su sintaxis, la flexibilidad de uso, su amplia comunidad y la integración de bibliotecas científicas. Concretamente, se ha decidido utilizar Pytorch [8], una biblioteca de código abierto especializada en el Aprendizaje Automático y la IA, ya que proporciona un entorno flexible y eficiente para la construcción y entrenamiento de los modelos de aprendizaje profundo. Asimismo, alcanza un alto grado de optimización para crear código paralelo, lo que permite aprovechar al máximo los recursos del sistema.

La decisión de escoger PyTorch frente a otros *frameworks* de Python para el desarrollo de modelos convolucionales, objetivo específico de este proyecto, se debe a dos motivos principales:

- La gestión de recursos del sistema que utiliza PyTorch. Debido a los cálculos basados en tensores, base de los *framework* de Deep Learning, se pueden utilizar todos los recursos hardware disponibles en la fase de entrenamiento, lo que supone una gran

ventaja en términos de rendimiento si se dispone de máquina multi-core.

- La personalización del código. Gracias a que esta herramienta de código abierto busca adaptarse al máximo a las necesidades del usuario, proporciona un ecosistema muy similar a Python, destacando sobre sus competidores en la facilidad que ofrece para desarrollar una red neuronal, desde su definición, hasta su entrenamiento y uso.

El desarrollo de la aplicación web se llevará a cabo en HTML, Javascript [9] y Python. Se implementará en Docker [10] una aplicación Flask [11], un microframework de Python para crear aplicaciones web ligeras y flexibles, sobre un servidor HTTP Gunicorn [12], que proporciona un entorno de ejecución fiable y escalable para las aplicaciones web. Esto permite manejar múltiples solicitudes concurrentes de manera eficiente.

Para la carga de datos se utiliza una pequeña base de datos en memoria. Redis [13] proporciona un almacenamiento rápido en memoria, lo que lo hace adecuado para aplicaciones como esta, que requiere un acceso rápido a los datos y no precisa de gran volumen de almacenamiento para funcionar.

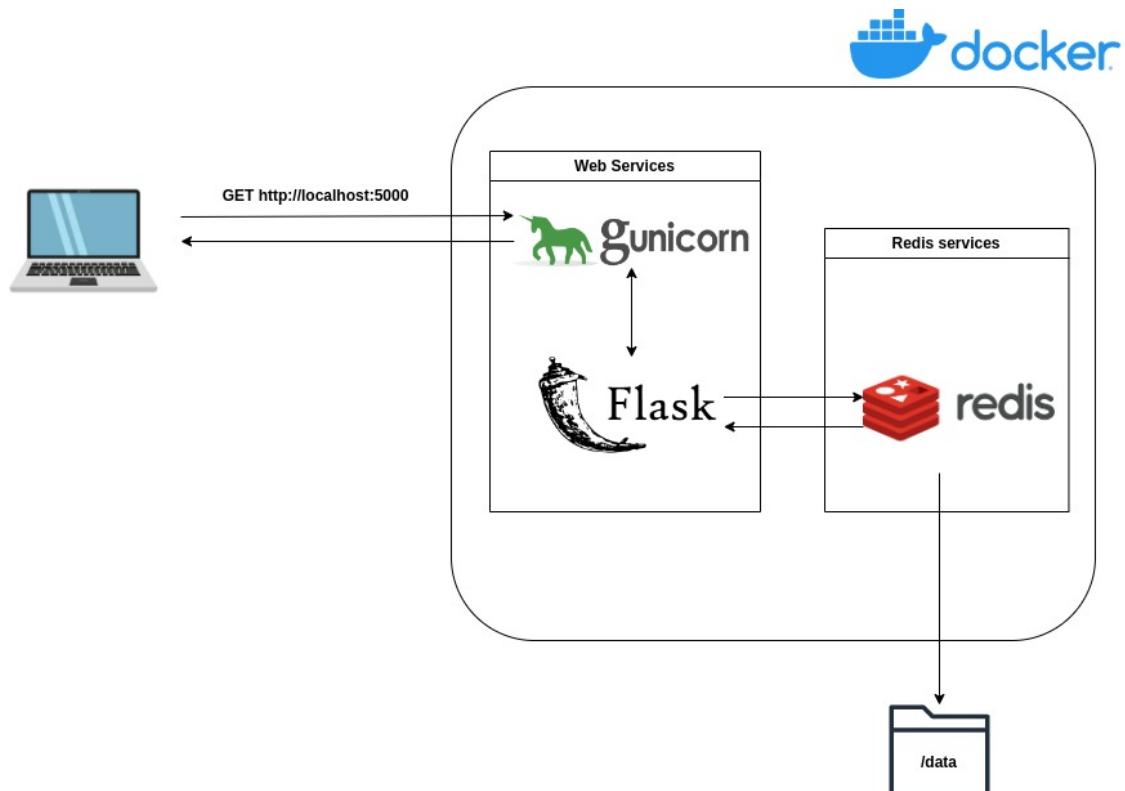


Figura 1.5: Esquema aplicación Web

Capítulo 2

Gestión del proyecto

En este capítulo se detallará la metodología de trabajo utilizada. Se describirán los dos enfoques planteados, así como la planificación temporal del proyecto.

2.1. Metodología de trabajo

Debido a que las necesidades del proyecto no encajaban con ninguna metodología de las comúnmente utilizadas, se decidió emplear de forma combinada dos planteamientos: uno tradicional, en el que se desarrolla un enfoque de cascada, y Cross Industry Standard Process for Data Mining (CRISP-DM), que está enfocado específicamente a planificar proyectos de minería de datos.

Es por esto que, de forma inicial, se realizó una planificación en la que se hizo un reparto del tiempo disponible, analizando todas las tareas y definiendo cuáles eran los objetivos y entregables del proyecto. Durante este proceso se tuvo en cuenta que, durante las fases relacionadas con la minería de datos, se seguirían los pasos que se plantean en CRISP-DM, descritos más adelante.

Como parte del enfoque más tradicional, se presentan una planificación del proyecto en cascada y la definición de los entregables. Todo ello sin perder de vista la diferencia en el campo de trabajo de cada fase, esencial a la hora de estimar el esfuerzo. En las tareas relacionadas con el desarrollo de modelos y técnicas de interpretabilidad, se trabaja a través de iteraciones. Esto lleva a un enfoque híbrido en el que no será necesario desarrollar todas las fases en cada iteración.

Para este bloque de trabajo, cuyo objetivo es la producción de los modelos, se opta por utilizar, como ya se ha avanzado, una metodología basada en CRISP-DM [14]: un modelo estándar abierto concebido principalmente para procesos de minería de datos. Fue desarrollado por un consorcio de más de 200 organizaciones con financiación de la Unión Europea [15] y, aunque el objetivo fuera modelar el desarrollo de proyectos de minería de datos, aporta la suficiente flexibilidad como para adaptarse a diversos estilos analíticos.

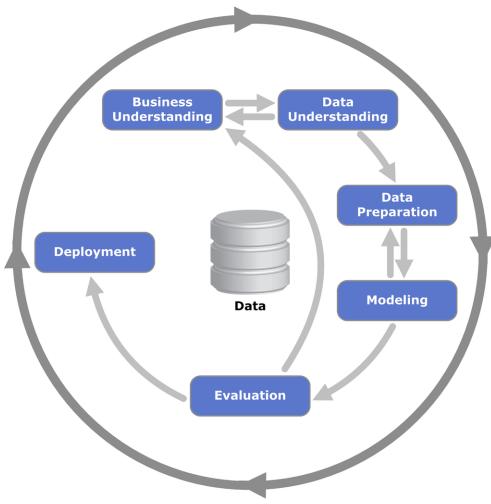


Figura 2.1: Ciclo de vida de un proyecto basado en CRISP-DM [14]

Se describen seis fases principales:

- **Comprensión empresarial:** se debe obtener una exposición clara del problema que se quiere resolver y sus objetivos.
- **Comprensión de los datos:** inspeccionar, describir y evaluar los datos disponibles. En esta etapa se realiza una variación, ya que se incluye también una fase de obtención de los datos y comparación entre las distintas fuentes.
- **Preparación de los datos:** se realiza una transición del estado inicial de los datos, hasta la forma necesaria para su análisis y procesamiento.
- **Modelado:** usa técnicas matemáticas para desarrollar modelos que se puedan usar para respaldar las decisiones comerciales. Debido a la naturaleza del proyecto que se va a realizar, en esta sección se enmarca el grueso del trabajo, donde se definirán los modelos y técnicas de interpretabilidad que mejor se comporten para el problema planteado.
- **Evaluación:** bajo una métrica prefijada y un objetivo claro se comprobará cuál de los modelos será el definitivo. Durante esta fase se realizará un proceso cíclico entre modelado y evaluación, hasta lograr un modelo que cumpla con los requisitos buscados.
- **Implementación:** integración de los modelos en la fase de negocio. Se realizará una pequeña aplicación web/script, donde se podrá utilizar el resultado obtenido, acompañado de un informe donde se incluirá todo el fundamento teórico del proceso.

2.2. Entregables del proyecto

Se consideran entregables aquellos hitos que supongan un cambio sustancial en el tipo de trabajo a realizar. Se comenzará por el diseño e implementación de la Red Neuronal

y, tras conseguir una optimizada con un rendimiento que cumpla las expectativas del proyecto, se añadirán las capas de interpretabilidad a la misma.

Finalmente, se utilizará una aplicación web *dockerizada* [10], en la que se pueda cargar una imagen y obtener tanto el resultado de la predicción, como el mapa de zonas calientes de la imagen usadas para la predicción.

- Entregable 1. Red Neuronal (NN) funcional que cumpla los requisitos fijados de comportamiento.
- Entregable 2. Red Neuronal optimizada tanto en rendimiento, como en eficiencia.
- Entregable 3. Visualización de la interpretabilidad de la clasificación de la NN.
- Entregable 4. Aplicación web funcional donde se puedan utilizar y mostrar los resultados obtenidos.

2.2.1. Evaluación

Para esta tarea, se tendrá en cuenta el resultado de cada uno de los entregables, así el cumplimiento de los plazos programados.

- Entregable 1. Se medirá la precisión de clasificación con imágenes que no hayan pertenecido al entrenamiento. Se fija el umbral en un 75 % de precisión para considerar la red como funcional.
- Entregable 2. Se considerará satisfactorio si se consigue aumentar la precisión de la red, hasta un 85 %, en la clasificación para el grupo de test.
- Entregable 3. Se evaluará y juzgará la superficie tumoral visible que logre destacar la red. El objetivo es utilizar aquellas imágenes en las que se pueda percibir un tumor de forma notable y comprobar si la red emplea la zona de la imagen en la que se encuentra el tumor para su clasificación.
- Entregable 4. Se evaluará su diseño visual, así como la eficacia y sencillez de uso.

2.3. Planificación

En esta sección se comentará la planificación que encaja con el enfoque más tradicional, realizada al comenzar el trabajo.

El proyecto tiene un reconocimiento de 12 créditos ECTS por la Universidad de Valladolid (UVa). Esto quiere decir que la duración completa del proyecto se estima en 300 horas. Debido a que las primeras tareas comenzaron el día 20 de marzo y debe finalizar antes del 23 de junio, se planificó un proyecto de 14 semanas, con una media de dedicación de 22 horas semanales.

Se detectó el riesgo de no poder cumplir con la carga habitual durante la última semana de mayo y, para no comprometer la entrega del proyecto, se reservaron 5 días para trabajar

en asuntos externos. En caso de que finalmente no fuera necesario, se dedicará este tiempo a la redacción del documento con vistas a finalizar antes del plazo marcado.

2.3.1. Planificación inicial

En la tabla 2.1, se definen todos los bloques de tareas con sus respectivas actividades desglosadas. Los hitos, marcados con (*), aparecen al final del bloque en el que deben ser alcanzados. En el diagrama de Gantt (Figuras 2.2 y 2.3), se aprecia de una forma más visual la duración del proyecto y de cada una de las fases, así como las dependencias entre las mismas.

Metodología	Fase	Tareas	Duración de la tarea	Duración Fase
Tradicional	Inicio del proyecto	Estudio del problema clínico	5	50
		Repaso del estado del arte	20	
		Lectura de estudios previos	25	
	Enfoque	Obtención del conjunto de datos	5	15
		Definición de los objetivos	5	
		Alcance del proyecto	5	
		Definición de alcance y objetivos	*	
CRISP	Tratamiento de datos	Preprocesamiento de Imágenes	5	5
		Estudio del entorno de desarrollo	15	
	Modelado	Elaboración de modelos iniciales	60	90
		Comparación de diferentes modelos	10	
		Elaboración del modelo final	5	
		Arquitectura definitiva	*	
	Explicabilidad	Inclusión de técnicas explicables	35	35
		Red definitiva	*	
	Evaluación	Análisis e interpretación	5	5
	Despliegue	Desarrollo de la aplicación Web	20	20
		Dockerizar	5	
		Aplicación en Docker definitiva	*	
	Documentación	Redacción del documento	70	80
		Revisiones	10	
		Documento completo	*	
		Total	300	

Cuadro 2.1: Planificación del proyecto

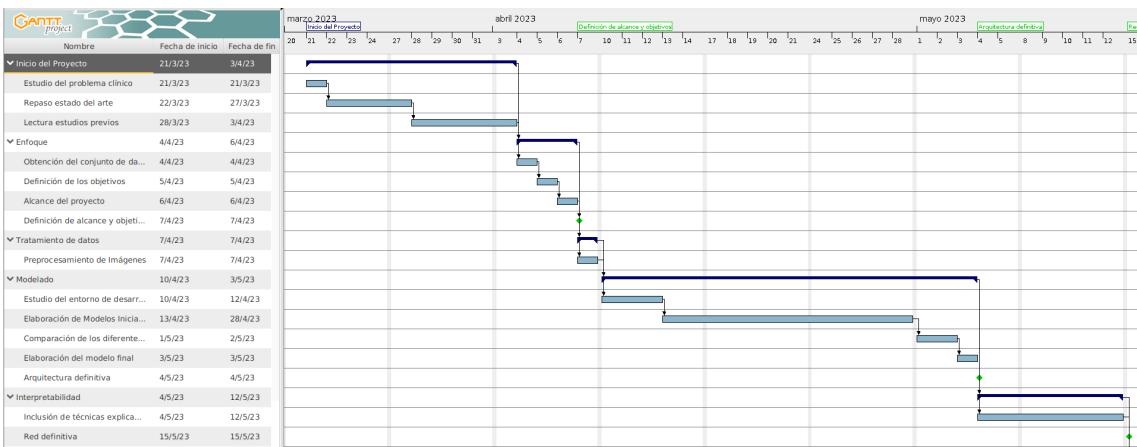


Figura 2.2: Diagrama de Gantt 1

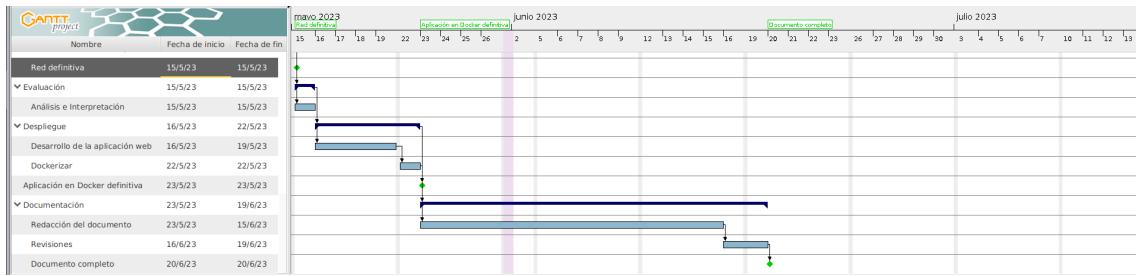


Figura 2.3: Diagrama de Gantt 2

Debido a que el Diagrama de Gantt no es específico para la metodología empleada, no se debe considerar como algo estricto, sino como un plazo orientativo para las tareas. Durante el proceso real, se espera que las tareas se realicen en un proceso cíclico, no lineal, como refleja el diagrama, siendo el tiempo de cada una de ellas la estimación con respecto al total, no teniendo por qué dedicarse todo el tiempo de forma continuada. La duración de los bloques será el tiempo total en que se realizarán todas las iteraciones. Si no hay retrasos en la planificación, el proyecto deberá finalizar el 20 de junio de 2023, dejando tres días de margen antes del cierre de plazo de la Convocatoria Ordinaria.

2.3.2. Variaciones en la planificación inicial

Finalmente, el proyecto se ha podido desarrollar acorde a la planificación propuesta, salvo pequeñas variaciones en la estimación del tiempo en algunas tareas. Estas variaciones se han equilibrado de forma natural de forma que se ha cumplido la meta de finalizar el proyecto antes del 23 de Junio.

La compensación se ha realizado de tal forma, que el tiempo sobrante en tareas como el estudio del entorno de desarrollo, se ha podido emplear en el desarrollo de la aplicación web o la redacción y revisiones del documento.

2.4. Entorno de trabajo

Se detallan a continuación los recursos hardware y software empleados para la realización de este trabajo.

2.4.1. Hardware

La computación y desarrollo del proyecto se ha llevado a cabo en una máquina virtual (MV) montada en Linux, cedida por el Departamento de Informática de la UVa para este fin. Fue configurada con 8 núcleos virtuales, 16GB de memoria RAM y 50GB de almacenamiento.

2.4.2. Software

En esta sección se describe todo el software utilizado en el proyecto, tanto el sistema operativo (SO), como todas las herramientas, programas y lenguajes utilizados, así como la justificación de cada elección en aquellos aspectos que sea necesario.

Sistema operativo

Todo el proyecto se ha realizado bajo el SO Ubuntu, una distribución de Linux basada en Debian GNU/Linux. Se descartó el uso de Windows, porque se ha demostrado que la eficiencia de las aplicaciones en Ciencia de Datos obtiene un rendimiento menor en este tipo de sistemas. La otra alternativa, Mac OS, ni se consideró, ya que se carece de licencias para máquinas virtuales (MV).

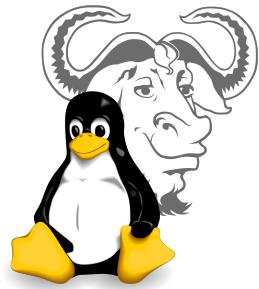


Figura 2.4: Logo de GNU y Linux

A las ventajas de rendimiento, se suma la sencillez que ofrece este SO para monitorizar la capacidad y ocupación en tiempo real de cada uno de los núcleos de la máquina, así como la sencillez para configurar el entorno de trabajo que, como se verá más tarde, incluirá el uso de Anaconda y Docker. Para la conexión con la máquina se ha utilizado SPICE [16], un sistema de visualización remota creado para espacios virtuales que permite ver un entorno de escritorio gráfico, que funciona correctamente con MV basadas en Linux.

Lenguajes de programación y herramientas utilizadas

Python

El proyecto se ha realizado en su gran mayoría en Python. Su elección se toma, fundamentalmente, debido a que dispone de bibliotecas optimizadas y especializadas en Aprendizaje Profundo, entre ellas PyTorch, que es la elegida para el desarrollo de este trabajo.



Figura 2.5: Logo PyTorch

Otros de los motivos por los que se escoge es por su sintaxis clara y sencilla, la gran cantidad de documentación disponible en línea de forma gratuita y la eficiencia computacional. Se comentan de forma más extensa los motivos que sustentan esta decisión en la sección 2.5.

Anaconda

Es una distribución de Python especializada en el ámbito del Análisis y la Ciencia de Datos. Es una plataforma que facilita la instalación de paquetes y entornos virtuales de Python, con los que la gestión de diferentes configuraciones de paquetes y versiones para proyectos específicos es más sencilla.

Dispone, además, de una amplia selección de paquetes, como NumPy, Pandas, Matplotlib, SciPy o Scikit-learn, empleados en el desarrollo del proyecto.



Figura 2.6: Logo de Anaconda

Desarrollo Aplicación Web

Se llevará a cabo en HTML, Javascript, CSS y Python. Se dockerizará de forma que no sea necesario tener en cuenta las dependencias a la hora de ejecutar la aplicación, aumentando así su portabilidad.



Figura 2.7: Logos de HTML, CSS y Javascript

Flask

Para el desarrollo de la aplicación web, se ha requerido del framework Flask, escrito en Python, lo que permite crear aplicaciones de forma rápida y sencilla. Facilita el desarrollo y ayuda a que la aplicación sea ligera y flexible.



Figura 2.8: Logo de Flask

Gunicorn

Es un servidor HTTP que sirve como interfaz de puerta de enlace entre el servidor web y Python. Permite comunicarse de forma ágil con el servidor y proporciona los servicios necesarios para una aplicación de tan poco peso como la realizada.



Figura 2.9: Logo de Gunicorn

Astah

Se aprovecha la licencia de Astah Professional disponible a través de la Escuela de Ingeniería Informática de la UVa. Esta herramienta permite desarrollar todos los diagramas necesarios para diseñar de forma integral la aplicación web. Se realizan tanto los diagramas de clase [17] como los diagramas de secuencia [18].



Figura 2.10: Logo de Astah

Overleaf

El documento se generará en Overleaf [19], un editor de L^AT_EX que permite el desarrollo de forma gratuita y en la nube, asegurando así el control de versiones y la posibilidad de trabajar desde cualquier equipo.



Figura 2.11: Logo de Overleaf

Otras Herramientas

Para el desarrollo de imágenes específicas, que requerían de un diseño manual, como es el caso de los esquemas de las Red Neuronal Convolucional (CNN), se ha utilizado una herramienta en línea subida en GitHub.

<https://alexlenail.me/NN-SVG/LeNet.html>

Para los esquemas especializados, se ha utilizado la herramienta *Draw.io*, un software gratuito y multiplataforma, disponible tanto en línea, como en una versión ejecutable en Linux.



Figura 2.12: Logo de Draw.io

2.5. Alternativas de la plataforma de trabajo

Se comentan y justifican, en este apartado, los diferentes motivos que han llevado a la elección del lenguaje de programación, así como de la biblioteca enfocada al Aprendizaje Automático que más se adaptara a las necesidades del proyecto.

2.5.1. Lenguaje de programación

El proyecto se realizará en un entorno de Python, pero esta no es la única herramienta que se ha evaluado a la hora de tomar la decisión. Lenguajes como Julia o R fueron barajados, ya que también se utilizan en el ámbito del Aprendizaje Profundo, aunque con enfoques y características distintas.

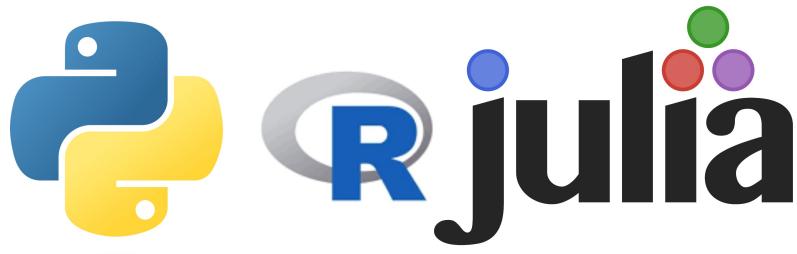


Figura 2.13: Logos de Python, R y Julia

Julia es un lenguaje de programación de alto nivel concebido específicamente para el Cómputo Científico y el Análisis Numérico. Está diseñado para ser rápido y eficiente. Su compilador de tiempo de ejecución permite obtener un rendimiento cercano al de lenguajes de bajo nivel como C++. Esto, unido al paralelismo y computación distribuida que ofrece, posibilita aprovechar al máximo los recursos de hardware disponibles para acelerar el entrenamiento y la inferencia en modelos de Aprendizaje Profundo.

Por su parte, R es un lenguaje de programación y un entorno de software dedicado al Análisis Estadístico y la Visualización de Datos. Si bien no fue inicialmente diseñado para el Aprendizaje Profundo, cuenta con diversos paquetes que permiten su implementación. R proporciona un conjunto completo de herramientas estadísticas y gráficas que pueden ser utilizadas en combinación con técnicas de Aprendizaje Profundo. Algunas bibliotecas de Python como “keras” y “tensorflow” brindan interfaces para implementar y entrenar modelos utilizando las bibliotecas subyacentes de Python.

El motivo por el que finalmente se ha tomado la decisión de realizar el proyecto en Python, es porque este lenguaje de programación ofrece una combinación de las ventajas comentadas, permitiendo aprovechar al máximo los recursos disponibles a través de bibliotecas especializadas en el Aprendizaje Profundo.

Su facilidad de uso, unida a la amplia comunidad y su gran ecosistema de bibliotecas, hacen de Python el lenguaje idóneo para este tipo de proyecto. Cabe destacar que la mayor parte de la bibliografía disponible para este tipo de propósitos está desarrollada en Python, lo que facilitará la comprensión e implementación de los modelos propios.

2.5.2. Biblioteca Python

Otra decisión fundamental tomada ha sido realizar la implementación en PyTorch, frente a otras bibliotecas disponibles y ampliamente utilizadas como Keras y TensorFlow.

Keras es una biblioteca de alto nivel que proporciona una API (Application Programming Interfaces) amigable para la construcción de modelos de redes neuronales. Su razón de ser es facilitar y agilizar el proceso de desarrollo de estos modelos, ofreciendo una experiencia más basada en la sencillez de uso que en la flexibilidad de diseño. Este es el aspecto fundamental por el que se ha desestimado el uso de esta biblioteca, ya que no ofrece una libertad completa a la hora del diseño y entrenamiento que, en contraposición, sí ofrece PyTorch. Este hecho resulta ser determinante en la decisión, ya que se busca poder controlar todos los aspectos del proyecto, pudiendo crear modelos claros y fácilmente legibles,

cosa que no es posible con Keras.

Respecto a TensorFlow, desarrollada por Google, es una de las bibliotecas más populares en este campo. Su principal desventaja es su compleja sintaxis, que siendo desconocida al inicio del proyecto, supone una dificultad añadida que no aporta ningún valor. Es por esto que también se desestima su uso inclinando la balanza en favor de PyTorch.



Figura 2.14: Logos de Keras, TensorFlow y PyTorch

Capítulo 3

Fundamento teórico

Una Red Neuronal (NN) es un algoritmo de aprendizaje vagamente inspirado en el funcionamiento del cerebro. De una manera similar a como se conectan y activan las neuronas en el cerebro, una Red Neuronal recibe unas entradas que, tras ser procesadas, generarán como resultado una serie de activaciones sobre las neuronas y, como consecuencia, producirán una salida.

Hay muchas arquitecturas y configuraciones posibles para las Redes Neuronales. El Teorema de Aproximación Universal [20] dice que siempre se puede encontrar una arquitectura para una NN lo suficientemente grande que, con un conjunto de pesos determinado, sea capaz de predecir cualquier salida para cualquier entrada. Esto hace ver la importancia que tienen este tipo de métodos, ya que significa que, para cualquier tarea o conjunto de datos dado, se puede obtener una arquitectura que, con sus correspondientes pesos ajustados, prediga exactamente el resultado que esperamos. La obtención de estos parámetros de la Red Neuronal se llama aprendizaje.

Las Redes Neuronales se caracterizan principalmente por:

- Adquirir el conocimiento a través de la experiencia, el cual es almacenado en los pesos de las conexiones interneuronales.
- Tener una altísima plasticidad y adaptabilidad, siendo capaces de cambiar dinámicamente junto con el medio.

3.1. Teoría básica de Redes Neuronales

Una NN está formada por cientos de pequeñas unidades, conocidas como neuronas artificiales, conectadas a través de unos coeficientes o pesos, que constituyen la estructura neuronal. Cada neurona tiene unas entradas ponderadas, una función de activación y una salida. La memoria del sistema se representa a través de los pesos de las conexiones entre las neuronas.

Aunque cada una de las neuronas puede procesar cierta información, el potencial de las computaciones neuronales viene dado por la interconexión entre las neuronas, que es,

precisamente, la que permite a las redes procesar grandes volúmenes de información y hacer predicciones con una sorprendente precisión.

Hay muchos tipos de redes diseñadas hoy en día, y se inventan nuevas a cada momento que pasa. Lo que las une es la transferencia de información entre las neuronas y conceptos como la función de activación y la optimización de los pesos. Se muestran en las figuras 3.1, 3.2 y 3.3 diferentes arquitecturas de red, diseñadas con diferentes fines pero con muchos conceptos en común.

El Perceptrón Multicapa (MLP) [21], mostrado en la fig. 3.1 es un tipo de arquitectura de red compuesto por múltiples capas de neuronas interconectadas, en las que las salidas de una capa se convierten en las entradas para la siguiente.

La primera capa se llama de entrada, la última, de salida, y las intermedias se denominan ocultas. Sus principales usos son la clasificación y la regresión, pudiendo ser utilizados para problemas en los que la relación de las entradas y las salidas no sea lineal.

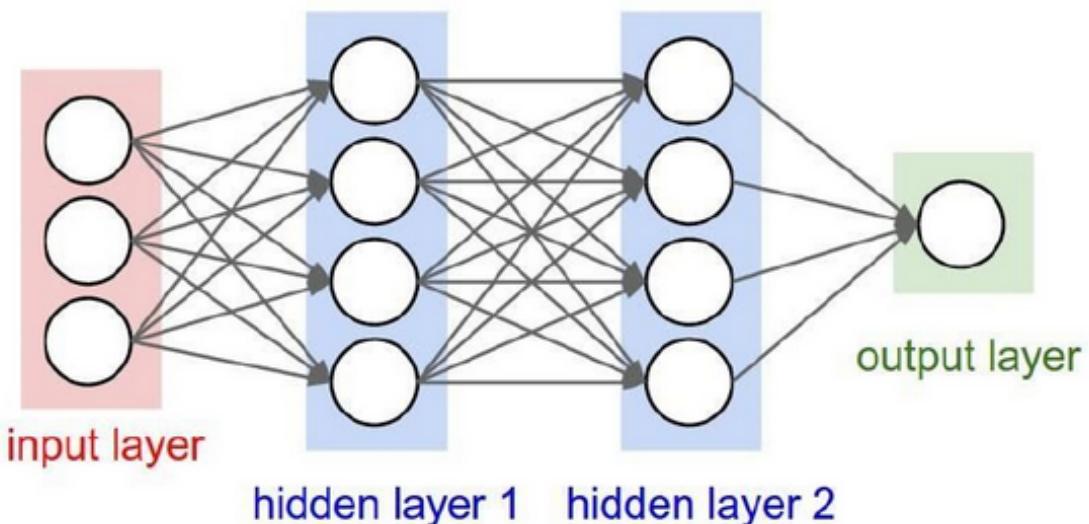


Figura 3.1: Ejemplo de MLP. Fuente: bootcampai.medium.com

Otro ejemplo de arquitectura de red son las Redes Neuronales Recurrentes RNN [22]. Su principal característica es que, al contrario que en las MLP, existen interconexiones que apuntan ‘hacia atrás’, como una especie de retroalimentación entre las neuronas dentro de las capas. En cada instante de tiempo, una neurona de una red recurrente puede recibir una entrada de la capa anterior, así como su propia salida o la salida de una capa superior.

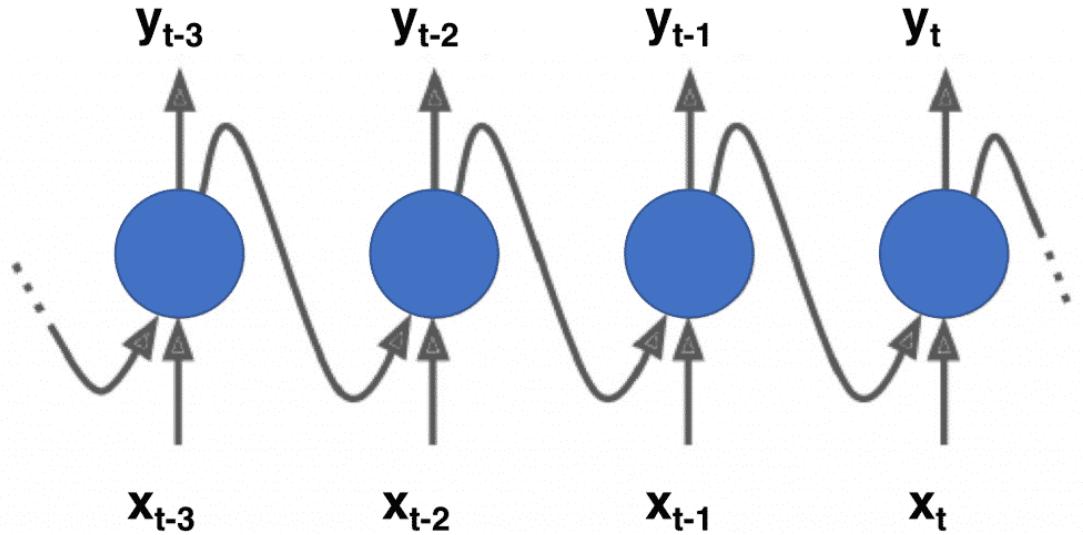


Figura 3.2: Ejemplo de RNN. *Fuente: torres.ai/*

Una última arquitectura ampliamente utilizada son las Redes Neuronales Convolucionales CNN [23], sobre las que se fundamenta este proyecto. Tiene como objetivo desarrollar la Visión por Computadora (CV), que trata de transmitir a los computadores la información que hay en las imágenes. Sus principales usos están relacionados con la clasificación de imágenes y la detección de objetos. Se desarrollará más en profundidad a lo largo del presente documento

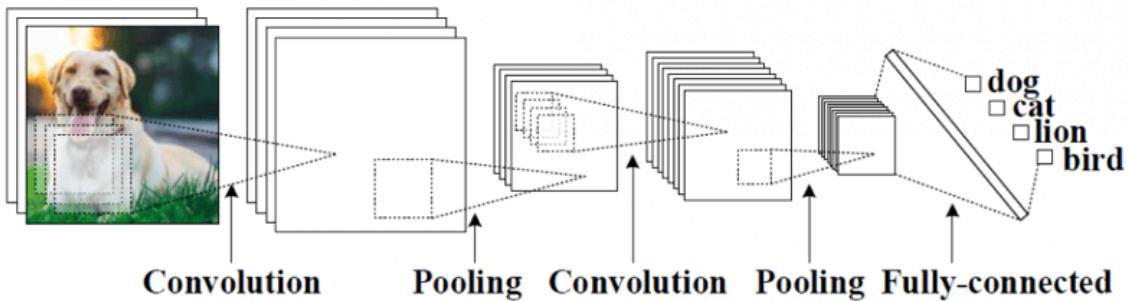


Figura 3.3: Ejemplo de CNN. *Fuente: kdnuggets.com*

3.1.1. La neurona

El modelo neuronal de McCulloch-Pitts [24] fue realizado en 1943 y fue el primer modelo neuronal artificial. Ha servido de base para el desarrollo de otros modelos computacionales, ya que permite realizar funciones lógicas y aritméticas simples. La neurona artificial será una unidad de cálculo utilizada para tratar de modelar el comportamiento de las neuronas del cerebro humano. El cálculo básico que realizan es la suma ponderada de las entradas, a la que se le aplica una función no lineal, como se muestra la figura 3.4.

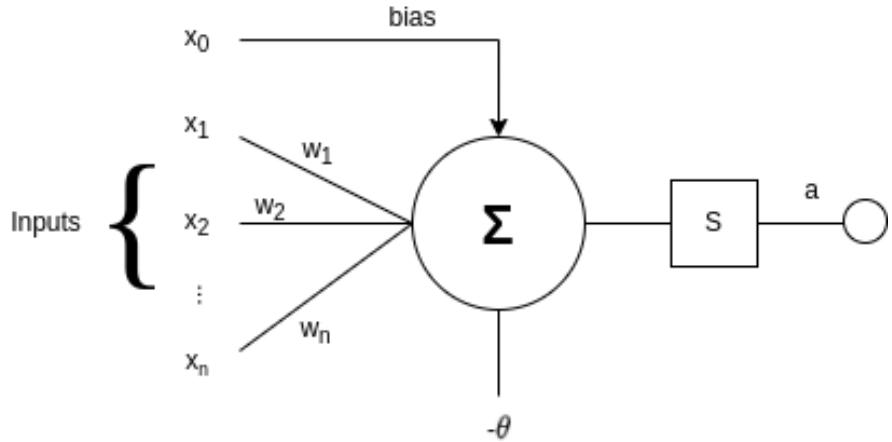


Figura 3.4: Operación básica de una neurona

En la figura 3.4, se puede ver que el valor de x_0 hace referencia al sesgo. Los valores de x_1, x_2, \dots, x_n son las entradas o *inputs*. w_i será el peso de la conexión entre la i-ésima entrada y la neurona. θ es el umbral o *threshold*. S es la función no lineal que, finalmente, se utiliza como activación, de la que se hablará más adelante. De esta forma, obtenemos que la salida de cada una de las neuronas queda representada por:

$$a = f(x_0 + \sum_{i=1}^n w_i x_i) \quad (3.1)$$

Esta suma ponderada de las entradas es el valor que se introduce a la función de activación, permitiendo que la red pueda aprender, así como modelar relaciones y patrones complejos de los datos.

3.1.2. La función de activación

La función de activación o de transferencia define la salida de cada una de las neuronas. Influencia la capacidad de las redes para aproximar la función objetivo ya que, si se tomara una función de activación lineal, solo se podrían aproximar a funciones lineales. Es por esto que una de sus funciones principales es la de introducir la no-linealidad en el modelo, aportando, así, la capacidad para aprender y representar funciones con un mayor grado de complejidad.

$$\text{Sigmoid activation}(x) = \frac{1}{1 + e^{-x}} \quad (3.2)$$

$$\text{ReLU activation}(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases} \quad (3.3)$$

$$\text{Tanh activation}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.4)$$

$$\text{Linear activation}(x) = x \quad (3.5)$$

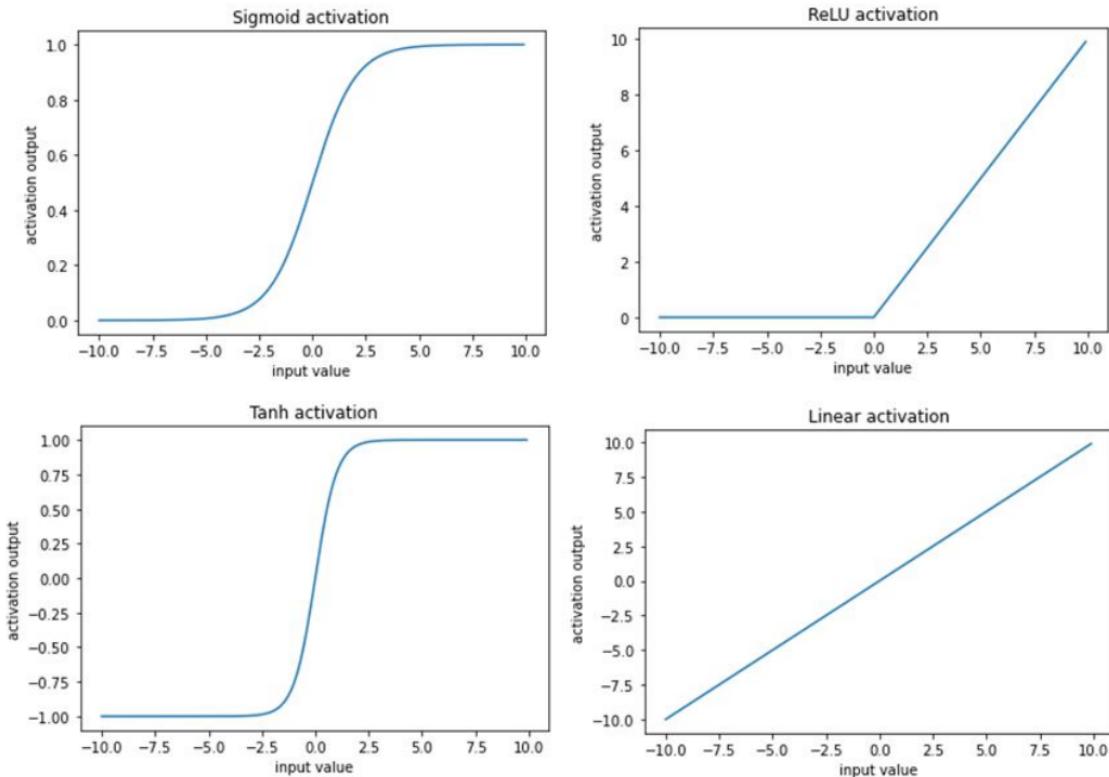


Figura 3.5: Algunas funciones de activación [26]

3.1.3. La función de pérdida

La función de pérdida de una Red Neuronal compara la salida obtenida con la real, midiendo, por tanto, cómo se está adaptando el modelo neuronal a la función objetivo.

Durante el proceso de entrenamiento, se trata de minimizar este valor, suponiendo esto una mejora en la calidad de la precisión del modelo. Su elección y diseño son fundamentales para obtener unos resultados consistentes en el entrenamiento y uso de las Redes Neuronales. Algunas de las funciones de pérdidas usadas con más frecuencia son las siguientes:

Error cuadrático medio

El error cuadrático medio o MSE (Mean Squared Error) se construye como el promedio de los errores. Se eleva al cuadrado para evitar compensar valores de diferente signo. Finalmente, se toma la media para obtener una medida comparable entre conjuntos de datos de distinto tamaño.

Esta métrica, descrita en la expresión 3.6, se suele utilizar para la predicción de valores de naturaleza continua. Si bien el valor de las neuronas en la capa de salida siempre es continuo, la diferencia entre regresión y clasificación depende de la interpretación que se le da a estas respuestas. Este tipo de métricas es más habitual para la predicción de valores continuos.

$$\text{Mean Squared Error (MSE) Loss} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (3.6)$$

Error absoluto medio

El error absoluto medio o MAE (Mean Absolute Error) se construye de forma análoga al MSE, sustituyendo el cuadrado de las diferencias por el valor absoluto del promedio de las diferencias entre las predicciones y los valores reales.

$$\text{Mean Absolute Error (MAE) Loss} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (3.7)$$

Esta métrica también se utiliza de forma típica en la predicción de variables continuas, teniendo en cuenta la aclaración que se realizó anteriormente.

Binary Cross-Entropy

La Entropía Cruzada Binaria (BCE) es una medida de las diferencias entre dos distribuciones, la real y la predicha, que se aplica en la clasificación de variables binarias.

En este contexto, la entropía hace referencia a una medida de incertidumbre o desorden en una distribución de probabilidad, por lo que el valor de la función de pérdida se calcula a partir de las probabilidades predichas por la red y la distribución de probabilidad objetivo. Cuanto mayor sea la incertidumbre o la variabilidad en la distribución de la probabilidad predicha, mayor será la entropía cruzada y, por lo tanto, cuanto más acertadas sean las predicciones de la red, menor será la entropía cruzada.

$$\text{Binary Cross-Entropy Loss} = -\frac{1}{N} \sum_{i=1}^N y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \quad (3.8)$$

Entropía Cruzada Categórica

La Entropía Cruzada Categórica (Categorical Cross-Entropy), es la extensión de la BCE para más de dos clases. Se basa también en este enfoque probabilístico para encaminar las predicciones hacia la función real. La aplicación de esta función de pérdidas requiere que las etiquetas utilicen una codificación ‘one-hot’, en la que cada clase se codifica con un vector binario de longitud igual al número de categorías, donde sólo una posición tiene el valor 1, representando la clase a la que pertenece, y el resto de componentes serán cero.

$$\text{Categorical Cross-Entropy Loss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(p_{ij}) \quad (3.9)$$

3.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN) son las más comunes a la hora de trabajar con imágenes, ya que sus características favorecen el entendimiento de las mismas, funcionalidad que no se puede conseguir a través de otras como, por ejemplo, un MLP.

No solo son útiles en la clasificación, sino que también son utilizadas para la detección de objetos, la segmentación de imágenes y mucho más.

3.2.1. La convolución

Es una operación matemática que sienta la base para este tipo de redes. En ella se mantiene la estructura espacial de la entrada, lo que es fundamental para no perder información. Básicamente, es la multiplicación de dos matrices, que como se ha visto en apartados anteriores, es la base del entrenamiento de cualquier red neuronal.

Una de ellas será la propia imagen, y la otra la conocida como núcleo o *kernel*, con un tamaño muy inferior. La convolución se llevará a cabo deslizando el filtro a través de la imagen, componiendo la matriz resultado como se muestra en la imagen 3.6.

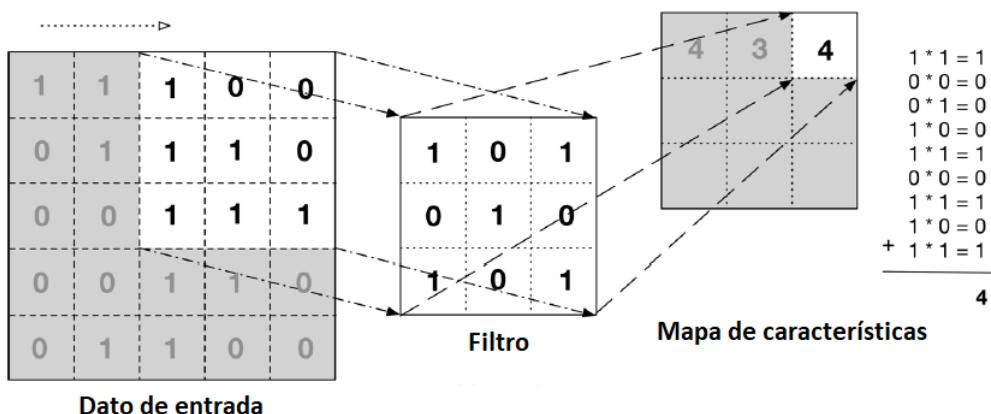


Figura 3.6: Operación de convolución [27]

Esta operación de suma ponderada es esencial para poder reconocer las formas simples de la imagen de entrada, segmentando las características para poder realizar la agrupación con el avance de las capas y llegando a poder realizar una clasificación en caso de que sea necesario.

Para el caso de imágenes de entrada con más de un canal, típicamente codificadas como RGB, la operación de convolución se lleva a cabo de forma que no se recorre cada uno de los filtros por separado, sino que el *kernel* repasa todas las capas de entrada unificando en una matriz toda la información disponible. En consecuencia, se debe pensar en el filtro como un bloque, en lugar de un plano.

Se muestra en el Anexo A una representación de los filtros empleados en el modelo final, el número de ellos en cada capa y una representación tridimensional de un ejemplo en cada una de las capas. Se puede apreciar cómo aumenta el volumen con el avance sobre el modelo, de forma que la combinación de características es cada vez mayor y más compleja.

Filtro

Es una matriz de pesos que se inicializa de forma aleatoria al empezar. El modelo trata de aprender sus valores óptimos con el aumento de las épocas de entrenamiento para tratar de extraer el máximo de características. El concepto de filtro presenta dos aspectos diferentes: qué aprenden y cómo se representan.

En general, el número de características que el modelo puede aprender es proporcional al número de filtros de la red convolucional, ya que cada uno de ellos se “especializará” en una característica concreta.

Por ejemplo, un filtro podría aprender solo a reconocer las líneas orientadas con una inclinación de 45 grados, y otro, por ejemplo, reconocería aquellas con orientación horizontal. Una vez que se dispone de un gran número de características, se combinan de forma que los dos filtros comentados comiencen a reconocer formas más complejas, como triángulos, o más elaboradas aún, como las orejas de un perro, proporcionando una alta activación cuando se encuentren este tipo de formas en una sección de la imagen de entrada.

$$\begin{array}{ccc}
 \text{Input Image} & * & \text{Filter (Kernel)} \\
 \begin{matrix} \text{A colorful portrait of a woman wearing a hat.} \end{matrix} & * & \begin{matrix} G_x = \\ \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \\ \\ G_y = \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \end{matrix} & = & \text{Output Image} \\
 \begin{matrix} \text{A black and white image showing vertical edges and highlights from the original portrait.} \end{matrix} & & \begin{matrix} \text{A black and white image showing horizontal edges and highlights from the original portrait.} \end{matrix}
 \end{array}$$

Figura 3.7: Efecto de unos filtros concretos. *Fuente: herevego.com*

El paso de los filtros por la imagen genera, como se puede ver en la Fig 3.8, un mayor número de canales pero de menor tamaño. Típicamente obtendremos una salida con tantos canales como filtros se empleen, ampliando su profundidad y disminuyendo su dimensión.

El número de filtros aplicados depende a su vez de los valores de paso o zancada (*stride*) y de relleno (*padding*), de los que se hablará más adelante. En el caso de imágenes con color, representadas con tres canales iniciales, se tratarán de forma independiente cada una de ellas.

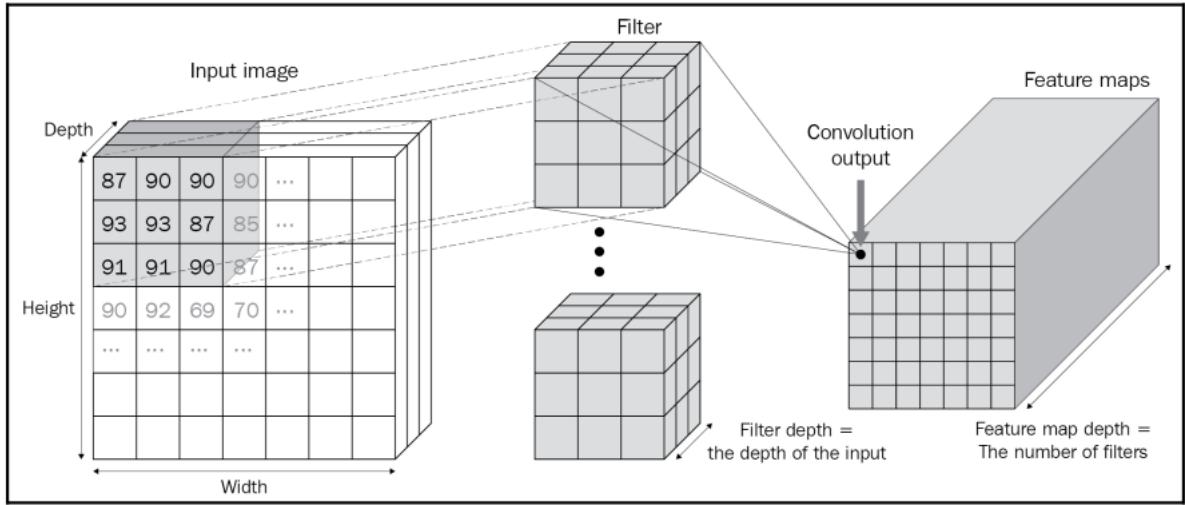


Figura 3.8: Operación de convolución para múltiples canales de entrada [26]

Se muestra en el diagrama procedente de [26] cómo la imagen de entrada es multiplicada por filtros con la misma profundidad, siendo el número de canales de la salida de la convolución proporcional al número de filtros. Dado un procesamiento típico en la primera capa convolucional sobre una imagen codificada en tres canales, como pueden ser los tres canales RGB, el proceso de convolución consistirá en varios pasos. En primer lugar, se inicializarán en la primera época, de forma aleatoria, los valores de, pongamos como ejemplo, 8 filtros o *kernels* de tamaño 3x3. Para cada uno de ellos, se realizará la operación de suma ponderada, de forma separada sobre cada canal de entrada. A continuación, se desliza el kernel sobre la capa correspondiente y, después de obtener el resultado de cada canal, se suman las matrices resultantes para obtener la salida final de la convolución, que en nuestro caso de ejemplo, tendrá una profundidad de 8 canales.

Este concepto es importante, ya que es precisamente lo que permitirá que la red pueda detectar las formas simples en las primeras convoluciones y convertirlas en formas complejas, con el avance de las capas a través de la combinación de las mismas. De esta forma, la primera capa convolucional podría especializarse en colores, una segunda capa podría especializarse en bordes, y las capas sucesivas a través de la combinación de bordes en esquinas, contornos, para posteriormente detectar partes de objetos. Al realizar la combinación entre estas partes de los objetos, la red será capaz de diferenciar entre los grupos de las entradas proporcionadas.

A continuación, se presentarán algunos conceptos y operaciones esenciales para las Redes Neuronales Convolucionales.

3.2.2. Tamaño de paso y relleno

En la convolución de la fig. 3.9, la operación se realiza superponiendo el filtro a través de la imagen, avanzando solo en una fila o columna, hasta recorrerla entera. Esto se traduce en una reducción de dimensionalidad en la salida, ya que el tamaño obtenido en términos de píxeles ha sido menor. El resultado puede suponer una pérdida de información, por lo que se introducen dos nuevos parámetros para controlar el paso de los filtros sobre la

matriz original.

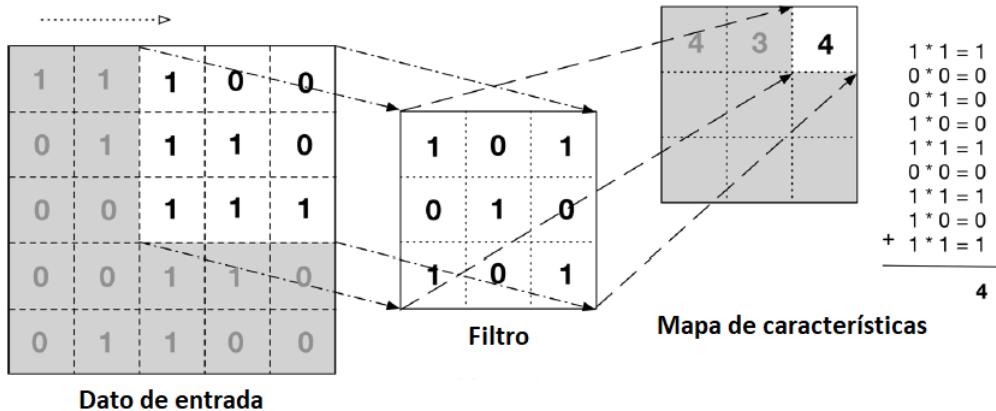


Figura 3.9: Paso de un filtro sobre una matriz en una convolución

Stride

Es el tamaño de paso, se explica como el número de píxeles que avanza el kernel durante la convolución. Esto define, por lo tanto, la cantidad de veces que se aplica el filtro, por lo que condiciona la cantidad de información que se pierde durante el proceso de convolución. Si escogemos un *stride* muy grande, se puede perder excesiva información pero, si por el contrario, el *stride* es demasiado pequeño, se puede aumentar de forma innecesaria el peso y el tiempo del proceso de convolución.

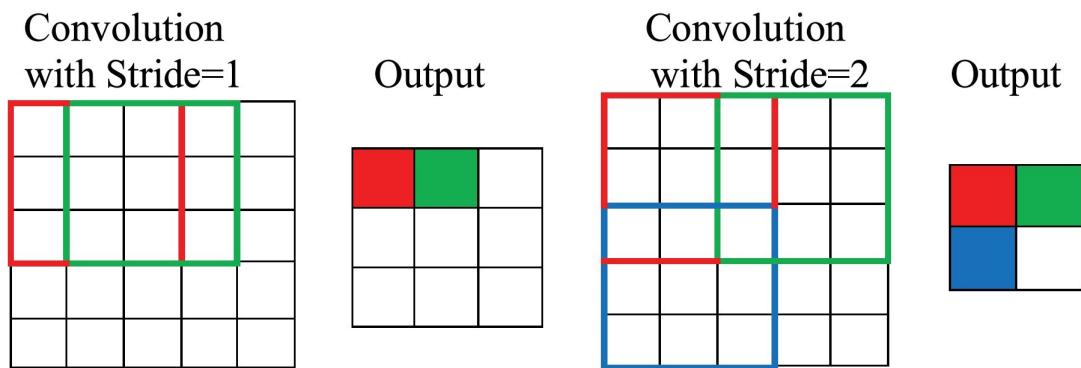


Figura 3.10: Ejemplo de tamaño de paso. Fuente: Computer.org

Padding

El relleno o *padding* es una técnica utilizada para mantener el tamaño de la imagen al paso por una capa de convolución. Define el número de píxeles (inexistentes), que el filtro puede utilizar al borde de la imagen. Añadir un *padding* de una dimensión supondría introducir la matriz de dimensiones $[n, m]$ en el centro de una matriz de ceros $[n+1, m+1]$. De esta forma, el resultado será la matriz original rodeada de ceros, de forma que el filtro pueda pasar a través de estos valores sin afectar al resultado.

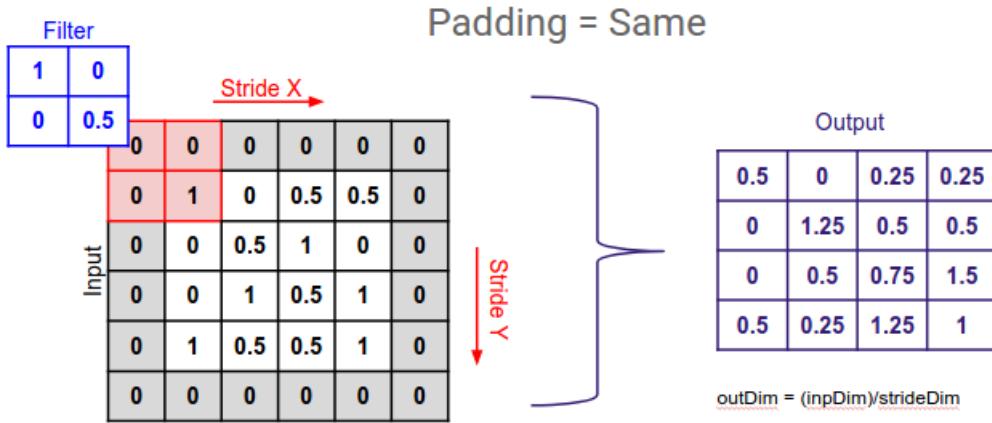


Figura 3.11: Ejemplo de padding. Fuente: analyticsindiamag.com

3.2.3. Pooling

Es una técnica que se aplica para reducir la dimensionalidad, extrayendo las características más relevantes. Divide la imagen en regiones y realiza una operación determinada dentro de cada región para producir un único valor de salida. Las dos operaciones más comunes, cuyo efecto se puede ver en la figura 3.12, son:

- **Max pooling:** Se toma el valor máximo de cada región y se descartan los demás, ayudando a resaltar características dominantes en la imagen o en los mapas de características de la red neuronal.
- **Average pooling:** Se calcula el promedio de los valores dentro de cada región. Esto favorece a suavizar las características y reducir el ruido en los datos.

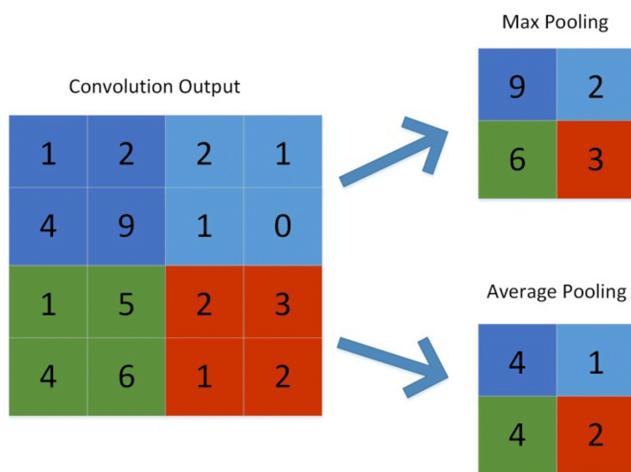


Figura 3.12: Ejemplo de Max-pooling y Average-Pooling. Fuente: Springer.com

Ambas operaciones tienen como objetivo disminuir la resolución espacial de los datos. Esto es beneficioso, ya que reduce la cantidad de parámetros y cálculos necesarios en las siguientes capas, lo que acelera el procesamiento y evita el sobreajuste del modelo.

3.2.4. Aplanado o *flatten*

Consiste, básicamente, en convertir la salida multidimensional de la capa de *pooling*, en un vector unidimensional que pueda servir de entrada a un MLP, que es la última fase de la clasificación.

En el caso de otras arquitecturas, en las que se emplean capas convolucionales, pero no tienen como objetivo realizar una clasificación en categorías, esta última capa puede no aparecer.

En conjunto, el flujo de una CNN sería el siguiente:

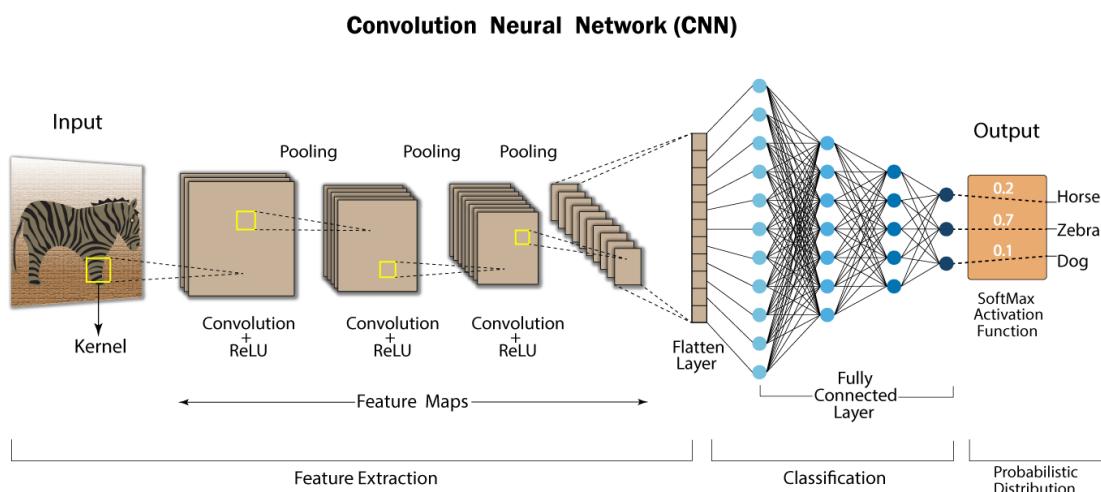


Figura 3.13: Flujo de una Red Neuronal Convolucional

3.3. Explicabilidad

Ha quedado reflejado que las Redes Neuronales Convolucionales tienen una alto potencial de predicción. Sin embargo, a medida que estos modelos se vuelven más complejos y se aplican en áreas críticas, como la Medicina, surge la necesidad de comprender y explicar cómo estos modelos llegan a sus predicciones.

La inclusión de técnicas de explicabilidad en las CNN se vuelve crucial para mejorar la transparencia y la confianza en los resultados, permitiendo que los usuarios y expertos comprendan y validen las decisiones tomadas por el modelo. Uno de los objetivos planteados para este trabajo es ahondar en este tipo de técnicas, seleccionado una que nos ayude a comprender el porqué de la clasificación.

Su inclusión se ve muy bien reflejada en el diagrama de proceso para la construcción de sistemas de IA robustos comentados en [23].

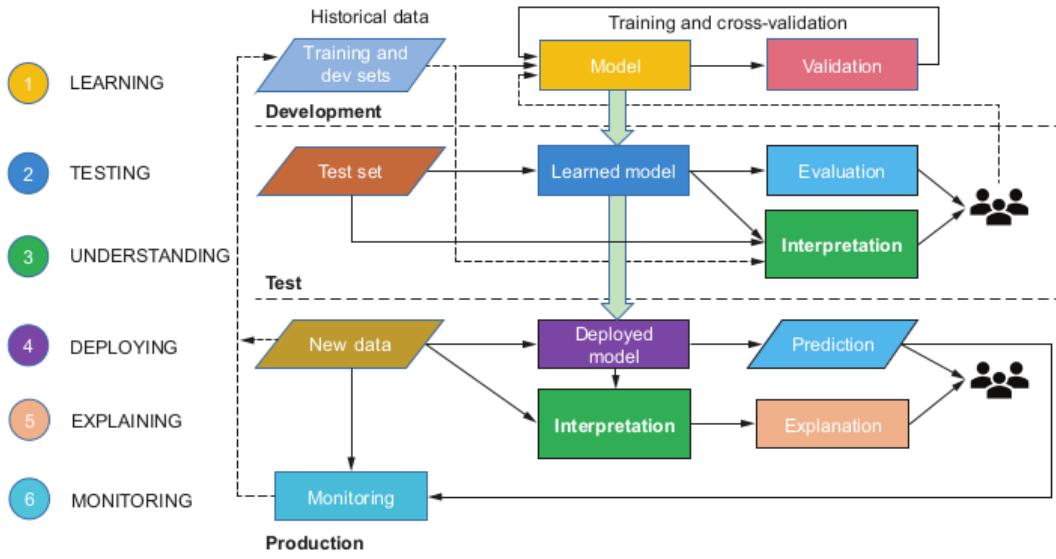


Figura 3.14: Proceso de construcción de un sistema de IA robusto

La fase de explicabilidad se incluye tras el despliegue, y su objetivo es interpretar cómo ha llegado el modelo a la predicción de los nuevos datos de entrada con un entorno de producción determinado. Esto nos permitirá disponer de un argumento válido para mostrárselo a los expertos, o a los usuarios del sistema, como respaldo de la predicción, proporcionando una explicación entendible para los humanos.

Las técnicas de explicabilidad dependen del tipo de sistema que utilizamos, ya que existen sistemas conocidos como *modelos de caja blanca*, en los que esta explicación es inherentemente transparente, e interpretar las profundidades del modelo no es un reto. Para este tipo de soluciones se utilizan técnicas *post hoc*, en las que se localizan aquellas entradas con una cierta importancia para la predicción del modelo. Si bien es cierto que para algunos *modelos de caja negra* también se pueden emplear este tipo de técnicas, habrá otros casos en los que no sea tan sencillo.

Estas técnicas pueden ser dependientes o independientes del modelo, siendo las dependientes aquellas que solo se pueden aplicar a ciertos tipos de modelos. También se puede realizar una clasificación dependiendo del alcance de la explicación, existiendo técnicas locales, cuyo objetivo es el de justificar la predicción de un ejemplo concreto, y globales, que se centran en el efecto global de la predicción del modelo.

3.3.1. Explicabilidad en CNN

Como ya se ha visto, para realizar una predicción a través de CNN, la imagen recorre múltiples capas convoluciones y *poolings* para la extracción de características. Para su explicación se pueden utilizar diferentes técnicas basadas en la visualización de los *kernels* y los gradientes o en la importancia de cada uno de los píxeles en la clasificación.

De entre todos los métodos posibles, se ha decidido emplear Grad-CAM para este estudio, con el objetivo de proporcionar una explicación para cada imagen en específico, donde se pueda ver de una forma clara qué partes de la imagen han sido determinantes a

la hora de realizar la predicción.

3.3.2. Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) es una técnica de explicabilidad utilizada para visualizar y comprender qué regiones de una imagen son importantes para la predicción. Fue propuesto por Selvaraju et al. en 2017 [28].

Se basa en la utilización de los gradientes de un objetivo cualquiera, a través del flujo de las convoluciones, hasta la última capa, para producir un mapa de localización donde se resaltan aquellas regiones de la imagen determinantes en la predicción. Apoyaron la validez de sus resultados en un estudio humano, en el que midieron a través de los resultados de Grad-CAM si los usuarios sin experiencia eran capaces de distinguir una red más potente de otra más ‘simple’, aún proporcionando ambas la misma predicción.

La diferencia de Grad-CAM frente a otras técnicas de explicabilidad, como los mapas de calor basados en el gradiente, es que este método se centra en las capas convolucionales, lo que permite entender qué características específicas activan a una clase en particular.

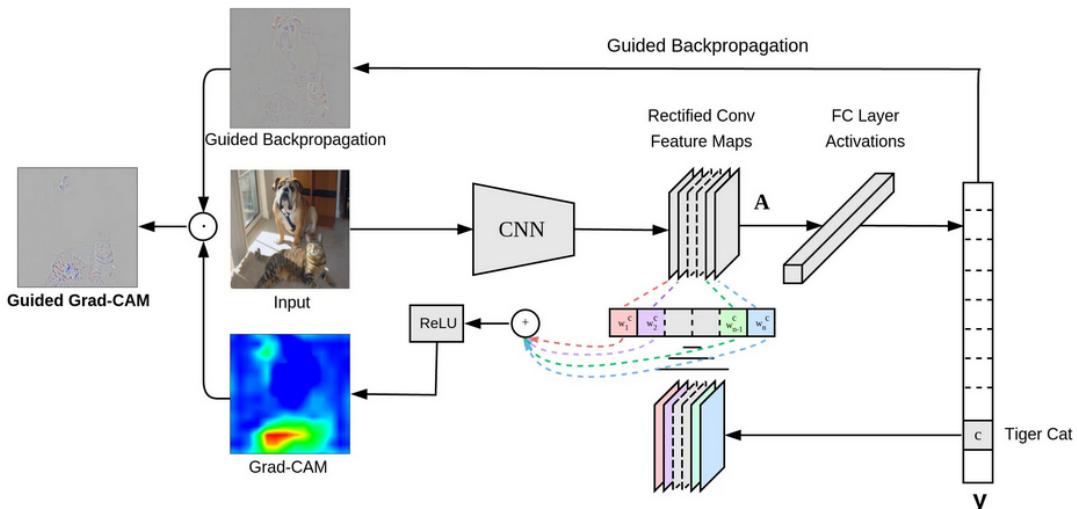


Figura 3.15: Flujo de Grad-CAM

Los pasos básicos que sigue Grad-CAM son los siguientes:

- Propagación hacia delante para obtener las activaciones en la última capa convolucional.
- Cálculo de gradientes de las clases objetivo con respecto a las activaciones de la última capa convolucional. Esto proporciona información sobre cómo los cambios en las activaciones afectan a la salida de la clase objetivo.
- Cálculo de los pesos: se realiza un promedio ponderado de los gradientes en cada canal de la capa convolucional. Los pesos se obtienen tomando el promedio de los

gradientes en cada canal y luego se ponderan según su importancia.

- Mapa de ponderación: los pesos resultantes se utilizan para ponderar las activaciones en cada canal de la última capa convolucional, generando un mapa de ponderación que resalta las regiones importantes para la clase objetivo.
- Generación del mapa de activación: el mapa de ponderación se combina con las activaciones de la última capa convolucional para obtener un mapa de activación final. En él se muestran las regiones más relevantes para la predicción realizada por la CNN.

Grad-CAM ha demostrado ser una técnica efectiva para la explicabilidad en CNN, ya que es fácil de implementar y proporciona información visualmente interpretable sobre las características que el modelo considera importantes para realizar una predicción específica.

Capítulo 4

Tratamiento de los datos

En este capítulo se detalla todo lo referente al proceso de obtención y preprocesamiento de los datos. Se debe tener en cuenta que este tratamiento previo de los datos tiene una parte genérica, por lo que no dependerá de las decisiones tomadas en el modelado. También se tratarán aquellos aspectos relacionados con las transformaciones de las imágenes, serán dependientes de los resultados comentados en el siguiente capítulo.

Se adelantarán algunos resultados con objeto de reflejar el resultado final, pero cabe destacar que el tratamiento de los datos forma parte de un proceso iterativo incremental, en el que se realizarán constantes modificaciones sobre algunas decisiones tomadas.

4.1. Obtención de los datos

Para la toma de los datos se recurrió a Kaggle [29], una plataforma en línea que se centra en la Ciencia de Datos y el Aprendizaje Automático, donde se puede acceder de forma gratuita a una amplia colección de conjuntos de datos abiertos.

Debido al gran repertorio disponible en esta plataforma, e incidiendo en el hecho de que todos los datos disponibles son abiertos, no se plantean otras plataformas para la obtención de los datos. Que los datos sean abiertos quiere decir que se pueden utilizar de forma gratuita, mientras sea con un objetivo académico o de investigación, y en ningún caso de manera lucrativa.

De entre los conjuntos de imágenes de resonancias magnéticas disponibles, relacionadas con tumores cerebrales, se escogió el *dataset* que disponía de una mayor cantidad de imágenes. Este *dataset* escogido tiene una cantidad total de 3264 imágenes, con la siguiente distribución de clases:

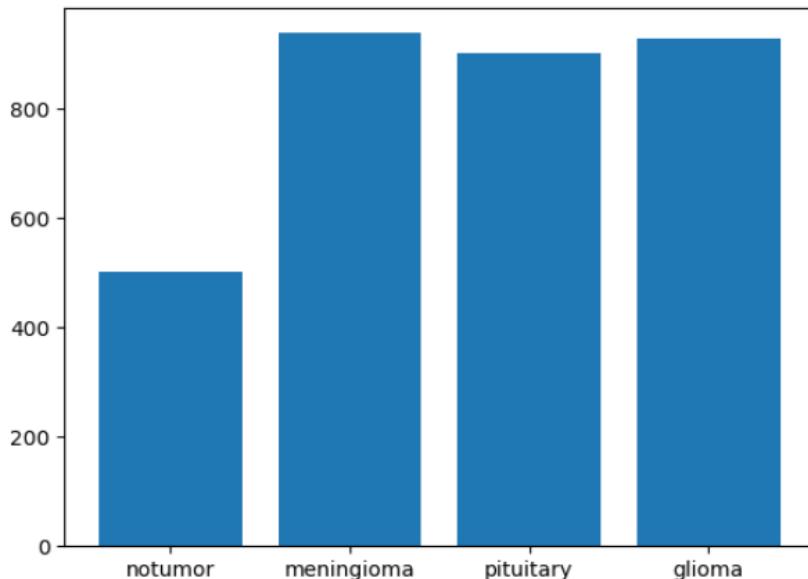


Figura 4.1: Distribución de clases

Para los tres grupos que presentan tumores cerebrales se dispone de más de 800 observaciones, siendo menor el número de imágenes disponibles para los cerebros sin presencia de tumores.

Se planteó la posibilidad de unir varios *datasets* de los disponibles, pero al no encontrar referencias de la obtención de los mismos, no se pudo asegurar que no se fuera a encontrar muestras repetidas, por lo que no se pudo llevar a cabo este proceso. La presencia de observaciones duplicadas implicaría que éstas tendrían un mayor peso o, en un caso peor, se realizarían las pruebas en la fase de test con imágenes que hayan participado en el proceso de aprendizaje, lo cual no se puede permitir.

Aún con este tamaño de datos, según los resultados obtenidos en el reto Kaggle propuesto a través de redes pre-entrenadas, se considera un volumen suficiente para lograr el objetivo mediante el entrenamiento de una red desde cero.

4.2. Transformaciones

Inicialmente, los datos obtenidos se dividían en el conjunto de entrenamiento y test, pero para evitar la pérdida de información debida a esta agrupación, durante uno de los incrementos se toma la decisión de juntar todos los datos, de forma que la separación en datos de entrenamiento y test se haga de forma dinámica. Esto evita la posibilidad de infraentrenar el modelo al no disponer de toda la información. Este cambio se realiza durante la fase de adaptación del modelo definitivo, entrenando las versiones iniciales con los grupos obtenidos directamente del reto Kaggle.

Se muestra como ejemplo una observación de cada clase en cada uno de los dos conjuntos formados:

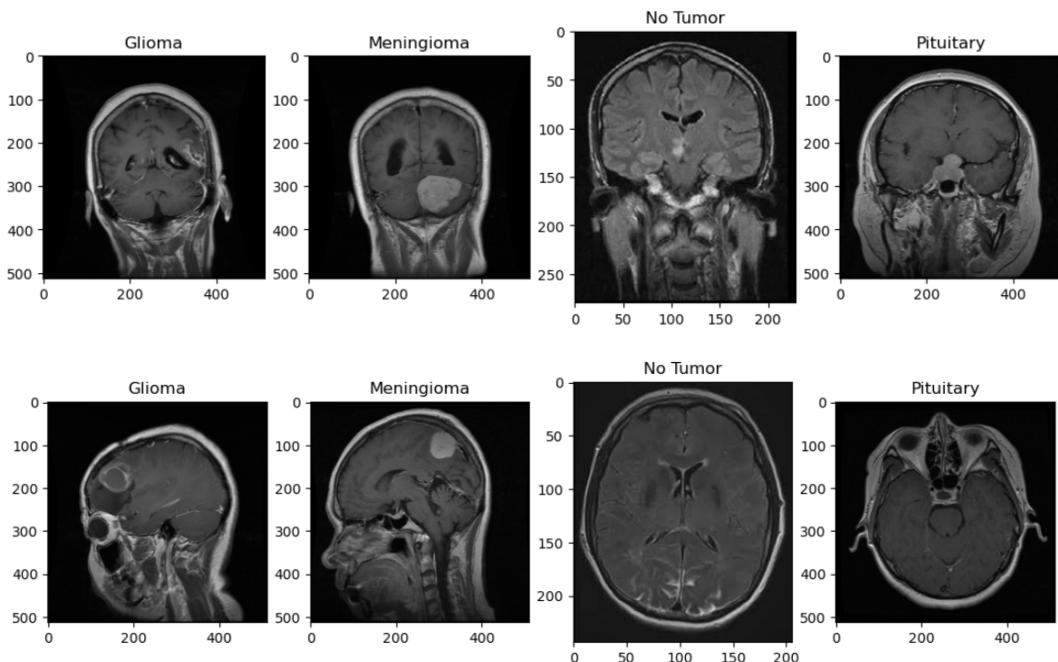


Figura 4.2: Ejemplo de observaciones en los subconjuntos de entrenamiento (superior) y test (inferior)

Se puede observar en esta primera visualización que no todas las imágenes están tomadas desde el mismo plano ni con los mismos tamaños.

En lo referente al tipo de imágenes, se considera que la red deberá ser capaz de llevar a cabo las predicciones para cualquiera de los posibles tipos de imágenes, por lo que no se realiza ningún cambio.

Respecto a su tamaño, se definirán las transformaciones necesarias para que se emplee un mismo tamaño, requisito indispensable para el correcto funcionamiento de la red.

Con este objetivo y siguiendo la metodología habitual a la hora de trabajar con PyTorch, se definen las características del conjunto de datos a partir de la clase *Dataset*, lo que permite cargar y procesar datos de manera eficiente durante el entrenamiento de un modelo de Aprendizaje Automático. Para la obtención del *dataset* final, compuesto por los datos transformados, se realizan las siguientes operaciones:

Generación de diccionarios

Para la lectura de los datos a partir de un directorio, en el que los datos ya están clasificados, se recorre de forma iterativa a partir de su raíz, asignando como valor de la llave del diccionario el nombre del subdirectorio que contiene todas las imágenes de la clase, y como valores, la ruta de las imágenes.

Se generarán dos diccionarios, uno para los datos de entrenamiento, que contendrá el 70 % de los datos, y otro para test, con el 30 % restante.

Implementación de la clase abstracta *Dataset*

Esta abstracción sirve como interfaz para crear conjuntos de datos personalizados, que se utilizarán para el entrenamiento y evaluación del modelo. De esta forma, se permite redefinir el acceso y procesamiento de los datos de forma que se adapte a las necesidades del modelo.

Se redefinen los métodos encargados de obtener el tamaño del *dataset*, que sumará el contenido de cada una de las claves del diccionario y devolverá el número total de imágenes que recoge, y el método encargado de la obtención de un elemento. Es necesario concretar estos procesos para que el modelo sea eficiente en su totalidad.

Debido a que el formato de entrada necesario para la red está compuesto por la representación tensorial de una imagen y su clase asociada, al proporcionar el índice del elemento que se quiere, se deben realizar varias operaciones:

En primer lugar, se obtiene su clase a partir de la posición en que se encuentre dentro del dataset, y a continuación, se carga la imagen a través del módulo proporcionado por la librería especializada en el procesamiento de imágenes ‘Pillow’.

Una vez que se dispone de la imagen en formato RGB, se realizan las transformaciones necesarias en función del grupo al que pertenezca: entrenamiento, o test. El método redefinido de obtención de un elemento devolverá en consecuencia la tupla tensor-clase, que servirá de entrada para el modelo convolucional.

Transformaciones

Aunque el tamaño del conjunto de datos se considere suficiente para poder entrenar el modelo, se toma la decisión final de aplicar técnicas de aumento de datos (*Data augmentation*) para mejorar la calidad de las predicciones. Esta decisión se aplica al modelo final pero, tal y como se explica en el siguiente capítulo, esto no ocurre en todas las versiones diseñadas. La única transformación que se aplicará a todos los datos será la redimensión, aunque variando el tamaño entre las versiones hasta considerar uno definitivo.

Se definen dos tipos de transformaciones:

- Para el caso de las imágenes de test, se aplicará la transformación a tensor, el tipo de datos utilizado por pytorch, y una redimensión de las imágenes a un tamaño común, finalmente fijado en 128x128 píxeles.
- Para el conjunto de entrenamiento se aplican, a mayores, unas transformaciones vertical y horizontal, aleatorias, para evitar el sobreajuste del modelo a los datos.

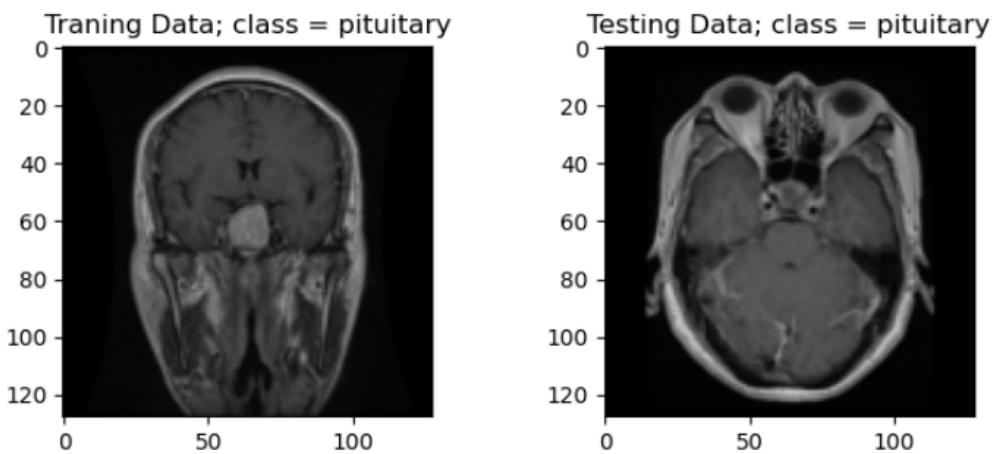


Figura 4.3: A la izquierda, la imagen transformada para entrenamiento, a la derecha, la imagen para test

En la figura 4.3, se puede observar cómo la imagen del entrenamiento ha sufrido una pequeña transformación, que ayudará a aumentar la variabilidad en los datos de entrada, mientras que la imagen del test ha sido tan solo redimensionada.

Capítulo 5

Modelado

En este capítulo se enmarca el grueso del trabajo realizado, ya que para alcanzar uno de los objetivos propuestos se debía probar un gran número de arquitecturas de red distintas, para su posterior comparación en términos de eficiencia y complejidad.

Se presenta el planteamiento sobre el que se fundamentan las diferentes versiones planteadas, los pasos necesarios para su diseño, sus diferencias y una comparación final. El propósito es obtener como resultado una arquitectura formada por aquellos parámetros que hayan mostrado un mejor comportamiento. Esta configuración será la que se tome como punto de partida de cara a la optimización, de forma que se cumpla con los requisitos fijados en el proyecto.

5.1. Planteamiento de la red

El objetivo de la red es, como se lleva tratando a lo largo de todo el documento, la clasificación de imágenes de resonancia magnética cerebrales en diferentes grupos, en función de si presentan algún tipo de tumor y, en caso de presentarlo, en qué grupo de los tres incluidos en el análisis encaja.

Para tratar de maximizar la precisión de las predicciones, se plantea una arquitectura básica sobre la que se realizarán múltiples versiones y pruebas, para tratar de encontrar qué parámetros se adaptan mejor a las propiedades del *dataset* considerado.

En lo que a la parte de extracción de características se refiere, esta estructura mínima sobre la que se sustentan todas las versiones, define más concretamente el diseño de los filtros de las capas convolucionales. Durante el paso por estas capas se tratará de mantener siempre una estructura en la que el número de filtros aumente gradualmente a lo largo de las capas convolucionales, consiguiendo extraer información a diferentes niveles de abstracción.

Los filtros iniciales, formados por un menor número de canales, serán los enfocados a obtener las características más simples y locales, como bordes y texturas, mientras que, a medida que adquieren un mayor número de canales, se ganará una mayor complejidad en las propiedades capturadas. El objetivo final, es obtener una serie de patrones abstractos

y de alto nivel. Esta estructura piramidal en la que se aumenta el número de filtros, permite una representación jerárquica y progresiva de los datos, facilitando la detección y la clasificación de objetos en imágenes.

Respecto a la sección de la red encargada de la caracterización, se fijan menos limitaciones, ya que el número de neuronas de la capa de entrada dependerá de las características extraídas tras el aplanado de los filtros obtenidos en la última capa, y el número de neuronas de la capa de salida será igual al de clases. Se busca una clasificación sencilla y eficaz, por lo que se tratará de incluir una o dos capas ocultas, como máximo, en las que se haga una gran disminución de neuronas, pero debido a que el número de las mismas será muy distinto en función de la arquitectura, no se fija una norma para este proceso.

5.2. Pasos para implementar una CNN

En primer lugar, hay que definir todos aquellos aspectos necesarios para generar un modelo en PyTorch, como son:

Data Loaders

En PyTorch, se define una estructura específica que facilita la carga y preparación de los datos para el entrenamiento y evaluación de los modelos, conocidos como *Data Loaders*.

Son parte de la biblioteca *torch.utils.data* y proporciona una interfaz conveniente para trabajar con conjuntos de datos. Simplifica operaciones como el muestreo o la generación de lotes. A estos *Data Loaders* se les proporciona como entrada un subconjunto de la muestra, tanto de entrenamiento como de test, y el tamaño de los lotes que se quiere, de forma que permite procesarlos de manera eficiente y en paralelo en las GPUs.

Otras ventajas que ofrecen son la mezcla automática, eliminando así el sesgo introducido por la ordenación inicial, y una iteración sencilla, proporcionando un recorrido simple de las instancias en cada época de entrenamiento y evaluación del modelo.

Definición del modelo

En PyTorch se debe definir una clase que herede de *nn.Module*. Esta clase proporciona una forma conveniente de encapsular parámetros entrenables, como pesos y sesgos, junto con métodos para realizar los cálculos en los datos de entrada.

Un objeto de esta clase puede contener instancias de su mismo tipo, lo que permite construir modelos complejos compuestos por diferentes capas y componentes unidos.

Como métodos esenciales se deben definir *__init__*, donde se definirán cada una de las capas del modelo y *forward*, método en que se especifica como se propagan los datos a través de las capas. Se especifica el paso entre las capas y las funciones de activación que actuarán en el proceso.

Para la definición de la red se diferencian dos secciones:

Una primera encargada de la extracción de características, que será la parte convolucional propiamente dicha. Es aquí donde se aplicarán los filtros correspondientes, con el objetivo de obtener el mayor número posible de características.

Una segunda de clasificación, que consistirá en una pequeña red densa. Para esta sección de la red se tomará como entrada el conjunto aplanado de características, extraídas por los filtros, y se obtendrá como salida el valor de la activación de las 4 neuronas de la capa de salida. Cada una de ellas representa una de las categorías y, por consiguiente, se tomará como valor predicho la clase representada por la neurona que obtenga el valor más alto.

Entrenamiento del modelo

Una de las ventajas que ofrece PyTorch, es la posibilidad de implementar de forma manual el entrenamiento del modelo.

Esta fase se incluye como un método en la clase definida para la red convolucional y se compone de varias etapas. En cada una de las épocas, se realizará una ejecución por lotes de todas las fases.

Para cada uno de estos lotes del conjunto de datos de entrenamiento, se extrae la representación matricial de las imágenes, x , y su clase, y . Antes de realizar el cálculo hacia atrás o *backpropagation*, se requiere reiniciar los gradientes acumulados en el procesamiento del lote anterior, lo que se consigue de forma sencilla con el método *zero_grad* del optimizador.

A continuación, se predice la clase de todas las imágenes del lote y se calcula la pérdida comparando las salidas del modelo con las etiquetas de los datos de entrada. Para esto se utilizará la función de pérdidas definida en el modelo. Una vez calculada, se lleva a cabo la retropropagación, para calcular y actualizar los gradientes de los parámetros, utilizando el método *backward*. Una vez realizados todos estos pasos, el optimizador actualiza los nuevos pesos basándose en los gradientes recién calculados.

Después de completar estos pasos para cada uno de los lotes, se entra en una fase en la que se testeará la precisión con las instancias que se han utilizado para el entrenamiento. En este paso no se requiere el cálculo de gradientes, ya que no se busca la actualización de los pesos. Se procede a la comparación de la clase predicha con la clase real para todos los datos y se computa la precisión porcentual, repitiendo este proceso para los datos de test, que no han intervenido en el entrenamiento.

Una vez que finaliza este proceso, se necesita evaluar cómo ha mejorado la red. Una de las posibilidades es visualizar de forma gráfica el valor de la la precisión de la predicción, tanto en el conjunto de datos de entrenamiento como en el de test. La evolución del modelo definitivo, del que se hablará más adelante, se muestra en la figura 5.1.

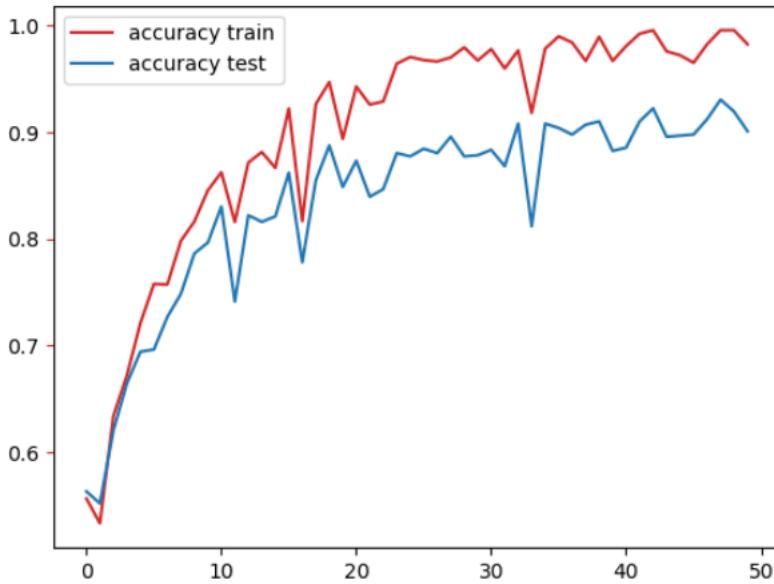


Figura 5.1: Evolución de la precisión de la precisión durante la fase de entrenamiento del modelo

5.3. Comparación entre los diferentes modelos implementados

Para tratar de maximizar las prestaciones del modelo se debe hacer un pequeño estudio acerca de los parámetros de la red, tratando de acercarse hacia una configuración cada vez más óptima, en la que se encuentre un equilibrio de entrenamiento que permita asegurar un buen rendimiento.

Las posibles variaciones se podrán realizar en:

- **Arquitectura de la red convolucional:** Se podrá variar tanto en el número de filtros como en el tamaño de los mismos, tratando de buscar un equilibrio que permita entrenar la red sin caer en un sobreajuste.
- **Capas ocultas de la red densa:** Se podrá variar el número y tamaño. La entrada se define por el número de características extraídas de la red convolucional y la salida por el número de clases, por lo que no se podrán realizar variaciones en estas capas.
- **Parámetros de la red:** Considerando que parámetros como el *stride* o el *padding* de los filtros no suponen una gran ventaja, no se estudiarán como variables y serán fijos para todos los filtros. Sí se incluyen en el estudio el número de instancias de los paquetes (*batch size*) y el ratio de aprendizaje del optimizador.
- **El tamaño de la imagen:** Se probarán diferentes dimensiones, teniendo en cuenta que imágenes pequeñas conseguirán un entrenamiento más rápido a costa de pérdida de información y tamaños grandes requieren un procesamiento más costoso, ampliando las posibilidades en la elección de los filtros.

Para cada una de las versiones se comentan cuáles son sus características más diferenciales y el porqué de cada una de las decisiones. Se puede ver el resumen comparativo de

todos los modelos en la Tabla 5.1. Se incluirá una representación de las capas convolucionales de cada modelo diseñadas con la herramienta.

5.3.1. Modelos incluidos en la comparación

Versión 1

En el diseño del primer modelo se busca obtener una pequeña red que procese imágenes con un tamaño reducido, ya que, en caso de ser efectivo, se reduciría ampliamente el coste y el tiempo de cómputo en el entrenamiento, pudiendo llegar a un buen modelo sin la necesidad de largos entrenamientos.

Es por ello que se decide fijar el número de píxeles en 56, el menor de los tamaños de las imágenes incluidas en el conjunto de datos, y no realizar transformaciones. No se emplean técnicas de Data Augmentation, ya que se desconoce cuál será el rendimiento inicial de la red. Respecto a los canales, solo se incluyen dos capas convolucionales y, combinando la premisa de crear capas gradualmente mayores y la sencillez que se busca para esta primera versión, se fija el número de filtros en 32 para la primera capa y 64 para la segunda.

La tasa de aprendizaje del algoritmo de optimización, que determina la magnitud de los ajustes que se realizan a los pesos en cada paso del proceso, se fija en 0.005. Un valor bajo de este parámetro forzará a que los ajustes realizados a los pesos sean pequeños en cada iteración, forzando un aprendizaje más lento y gradual. Supone una convergencia más lenta, pero aporta una mayor estabilidad y menor riesgo de sobreajuste.

La clasificación se realiza con una sola capa oculta que reduzca la entrada de 10.816 ($13 \times 13 \times 64$) características a 4096 neuronas.

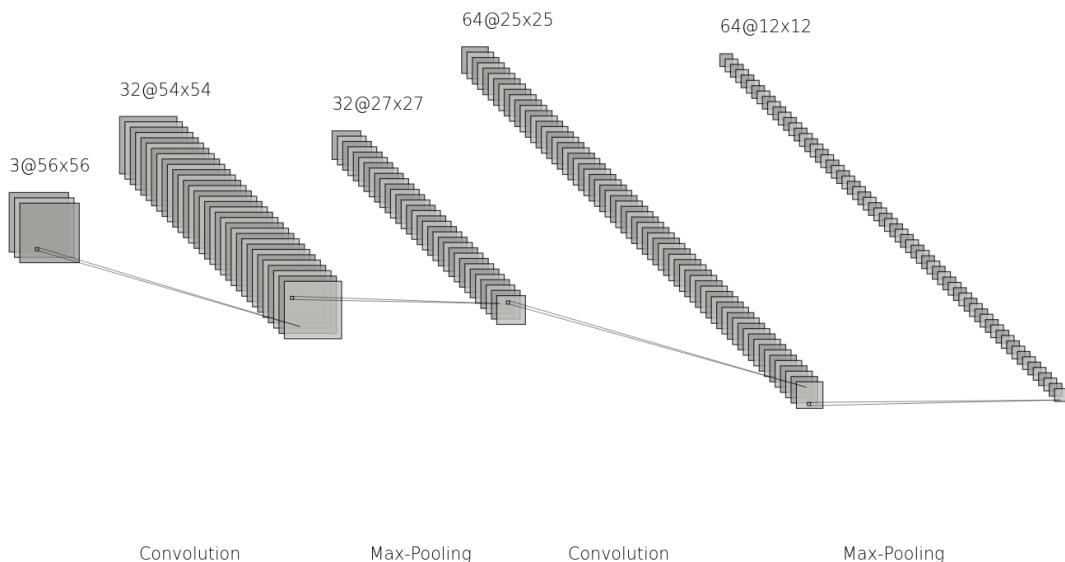


Figura 5.2: Arquitectura de Red - Versión 1

Versión 2

Para la segunda versión de diseña un modelo más gradual, en el que se aumenta el tamaño de las imágenes a 128x128 píxeles y se aplica Data Augmentation. Estas técnicas se basan en aplicar transformaciones aleatorias en los ejes vertical y horizontal, utilizando las funciones implementadas en PyTorch *RandomVerticalFlip* y *RandomHorizontalFlip*.

Debido a la gran diferencia en la precisión de clasificación entre los grupos de entrenamiento y test obtenidos en la primera versión, se incluye el *dropout* en la clasificación, una técnica de regularización cuyo propósito principal es evitar el sobreajuste. Consiste en apagar aleatoriamente un conjunto de neuronas en la fase de entrenamiento, para que se olvide de forma temporal algunas conexiones, forzando a la red a aprender de manera más robusta y redundante. Se comienza ignorando la mitad de las neuronas de la capa oculta para medir su rendimiento.

Respecto a la arquitectura de las capas convolucionales, se incluyen más capas para conseguir una extracción de características más gradual, comenzando con la extracción de 8 filtros y aumentando a 16, 32 y 64 finales en 4 capas. Se muestra de forma análoga a la versión anterior la arquitectura de la parte convolucional.

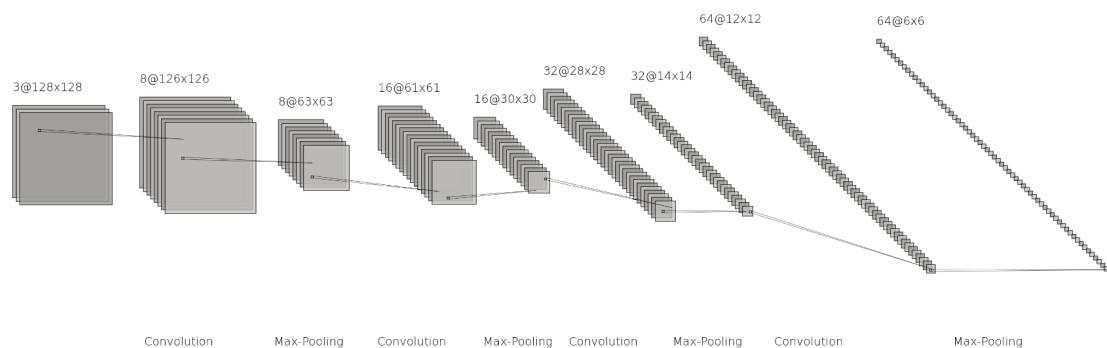


Figura 5.3: Arquitectura de Red - Versión 2

Versión 3

Como prueba, debido a la mejora en el rendimiento al incrementar el tamaño de la imagen en la segunda versión, se fija el tamaño para esta versión en 256x256 píxeles, aumentando así el peso del cómputo pero pudiendo encontrar una mejora al detectar de una forma más precisa las características de las imágenes.

Respecto a la versión anterior, se realiza una pequeña simplificación para buscar el equilibrio en el coste de la computación y se elimina una de las capas convolucionales, obteniendo directamente 16 filtros a través de la imagen inicial, de forma que se elimina una de las convoluciones.

También se disminuye el valor del *dropout rate* a un 0.25 para mantener más neuronas activas en la clasificación.

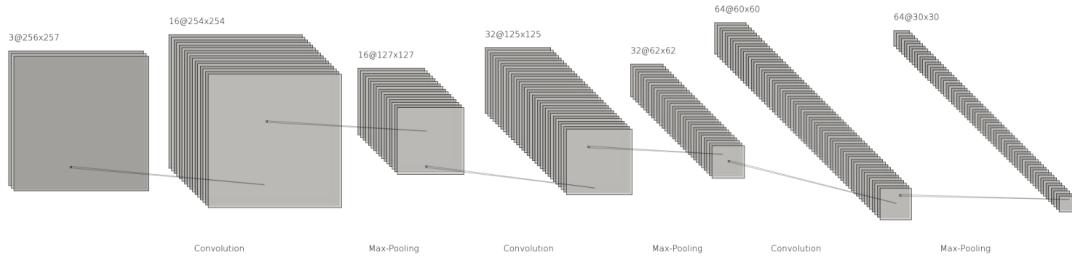


Figura 5.4: Arquitectura de Red - Versión 3

Versión 4

Para tratar de conseguir más argumentos que consigan dar luz a aquellos parámetros que favorecen al aprendizaje, se decide modificar algunos aspectos de la primera versión que parecen haber mejorado el entrenamiento. Es por ello que se vuelve a las dos mismas capas convolucionales, se escoge un tamaño de 128x128 y se incluyen las técnicas de *Data Augmentation* comentadas.

Estas técnicas incluyen un ruido aleatorio en los dos ejes principales de la imagen y, a mayores, se aplica otra que se centra en la nitidez de la imagen, modificando de forma aleatoria la claridad y los bordes de la imagen. Se emplea de forma análoga la implementación de PyTorch: *RandomAdjustSharpness*.

Se incluye también el *dropout*, que ya se venía utilizando en las últimas versiones, pero no fue empleada en la primera.

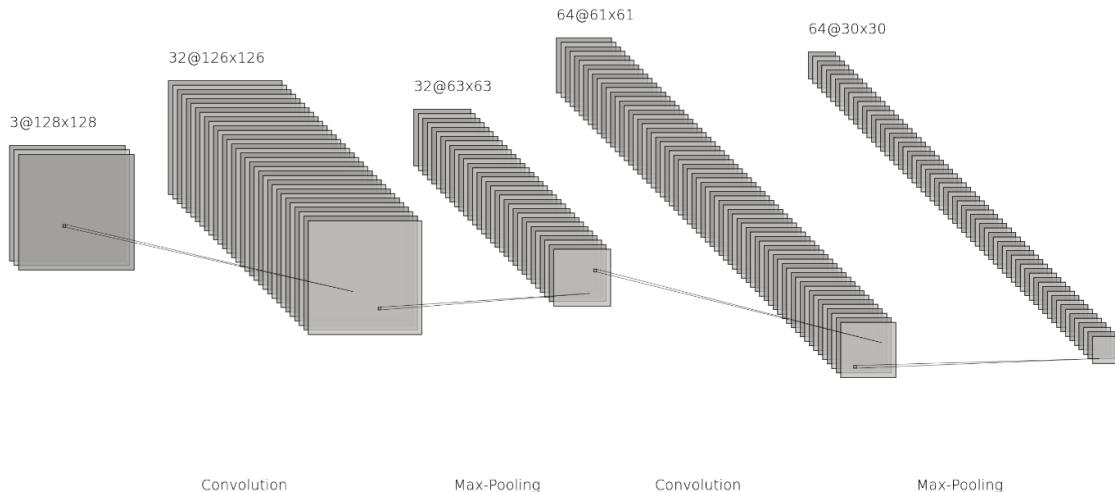


Figura 5.5: Arquitectura de Red - Versión 4

Versión 5

Como última versión antes de realizar una comparación más profunda, se implementa un modelo en el que se utilizan imágenes con un menor número de píxeles, escogiendo un tamaño de 64x64. No se incluyen técnicas de *Data Augmentation* y se aplican solo dos capas, en las que se obtienen 16 filtros a partir de los tres canales de entrada y finalmente,

32. Se entrenará solo con dos capas convoluciones, extrayendo un menor número de características complejas y bajando la tasa de aprendizaje a un valor muy pequeño, de 0.0001, para comprobar el efecto al clasificar manteniendo solo una serie de características más simples pero de forma más rigurosa.

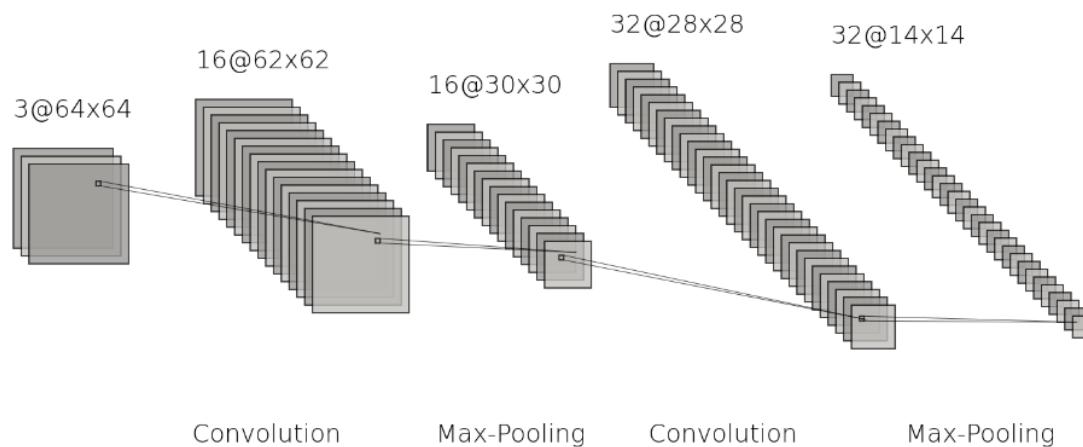


Figura 5.6: Arquitectura de Red - Versión 5

5.3.2. Comparación de los resultados obtenidos

Se muestra en la tabla 5.1 la comparación de las 5 versiones con el objetivo de analizar qué características son las que han llevado a obtener los mejores resultados. Tras esta comparación se diseñó una arquitectura final, sobre la que se refinaron algunos aspectos esenciales para obtener una mejora en los resultados.

Versión	Parámetros	Capas	AccTrain	AccTest
v1	img_size : 56, transform: NO channels_ini : 3, Initial_filters : 32, batch_size : 64, f_perdidas : nn.CrossEntropyLoss(), Learning_rate: 0.005	[3, 32] [32, 64]	0.8087	0.4036
v2	img_size : 128, trasform: RandomVertical + RandomHorizontal Channels.ini : 3, initial_filters : 8, batch_size : 64, dropout_rate : 0.5, f_perdidas : nn.CrossEntropyLoss(), Learning_rate: 0.005	[3, 8] [8, 16] [16, 32] [32, 64]	0.8902	0.5964
v3	img_size : 256, transform: NO channels_ini : 3, initial_filters : 8, batch_size": 64, dropout_rate : 0.25, f_perdidas: nn.CrossEntropyLoss(), learning_rate: 0.005	[3, 16] [16, 32] [32, 64]	0.9129	0.5812
v4	img_size: 128, transform: RandomVertical + AdjustSharpness + RandomHorizontal channels_ini : 3, initial_filters : 32, batch_size : 64, dropout_rate : 0.25, f_perdidas : nn.CrossEntropyLoss(), learning_rate: 0.005	[3, 32] [32, 64]	0.7063	0.4289
v5	img_size : 64, transform: NO channels_ini : 3, initial_filters : 16, batch_size : 64, dropout_rat : 0.25, f_perdidas: nn.CrossEntropyLoss(), learning_rate: 0.0001	[3, 16] [16, 32]	0.4648	0.3173

Cuadro 5.1: Comparación entre las distintas versiones de CNN implementadas

Se observa que los mejores resultados se obtienen en las versiones 2 y 3, donde se alcanzan valores cercanos al 0.9 en la precisión de predicciones sobre el conjunto de datos de entrenamiento, pero no se logra sobrepasar el 60 % de predicciones correctas sobre el conjunto de datos seleccionados para el test.

Al no cumplir ninguno de los modelos con las expectativas planteadas, a través del análisis combinado de las características de cada uno de los modelos, se tomarán las decisiones que se crean mejores para modelar una versión definitiva, que deberá tener un rendimiento muy superior al obtenido.

Esta comparación se realiza sabiendo que el análisis de una característica sin tener en cuenta el resto del modelo, no tiene por qué ser crucial, ni se tomarán decisiones fundamentadas en la mayor parte de los casos, ya que no sería correcto al haber varias modificaciones entre todos los incrementos.

En todos los casos se harán suposiciones que se puedan justificar a través de los resultados, pero siempre bajo esta máxima, y ninguna de las decisiones será fija o inamovible. Con esto en mente, se busca la arquitectura que proporcione los mejores resultados para el modelo.

La primera decisión que se toma es el tamaño de las entradas, que se decide fijar en 128x128 píxeles, ya que la versión 2 logra unos buenos resultados con este tamaño y la diferencia con la versión 3 no es suficiente como para aumentar la complejidad con tamaños más grandes. Se incluyen también las técnicas de *Data Augmentation*, concretamente las que incluyen pequeñas variaciones en los ejes vertical y horizontal, pero no la técnica que modifica la nitidez, ya que no se considera tan interesante.

Se elimina el *dropout*, ya que tras implementar todos estos modelos sin obtener un buen rendimiento en la precisión de predicción en el grupo de test, se plantea la posibilidad de que la red no esté sobreajustando, sino que sea justo al revés, y se esté infraentrenando, por lo que se necesitará extraer un mayor número de características para ajustar bien los pesos.

Con este objetivo, se fija la tasa de aprendizaje en 0.001, valor sobre el que no se ha podido entrar mucho en estudio debido a la gran variedad que se deberían incluir en las pruebas.

Si bien durante algunas pruebas se incluyó una técnica de un ajuste adaptativo de la tasa de aprendizaje, vistas en [30], se decidió no incluir este aspecto en el estudio para no aumentar la complejidad, y se plantea como trabajo futuro para mejorar los resultados obtenidos.

Con objeto también de obtener un mayor número de características y realizar un entrenamiento más profundo, se toma la decisión de disminuir el tamaño de los lotes de entrenamiento, que hasta el momento se habían mantenido fijos en un tamaño de 64 observaciones por lote, a un tamaño de 16. Con este cambio se consigue un ajuste más preciso y un mayor número de propagaciones, lo que se espera que se traduzca en una mejora en la predicción.

Se listan las características finales obtenidas y se muestra su resumen en PyTorch en la figura 5.7:

- Tamaño de imagen: 128x18 píxeles
- Transformaciones a las imágenes de entrada: RandomVertical + RandomHorizontal
- Tamaño de lotes: 16
- Dropout rate: 0
- Función de pérdidas: Cross Entropy Loss
- Optimizador: Root Mean Square Propagation
- Tasa de aprendizaje: 0.001

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 8, 126, 126]	224
ReLU-2	[-1, 8, 126, 126]	0
MaxPool2d-3	[-1, 8, 63, 63]	0
Conv2d-4	[-1, 16, 61, 61]	1,168
ReLU-5	[-1, 16, 61, 61]	0
MaxPool2d-6	[-1, 16, 30, 30]	0
Conv2d-7	[-1, 32, 28, 28]	4,640
ReLU-8	[-1, 32, 28, 28]	0
MaxPool2d-9	[-1, 32, 14, 14]	0
Conv2d-10	[-1, 64, 12, 12]	18,496
ReLU-11	[-1, 64, 12, 12]	0
MaxPool2d-12	[-1, 64, 6, 6]	0
Linear-13	[-1, 512]	1,180,160
ReLU-14	[-1, 512]	0
Linear-15	[-1, 64]	32,832
ReLU-16	[-1, 64]	0
Linear-17	[-1, 4]	260

Total params:	1,237,780
Trainable params:	1,237,780
Non-trainable params:	0

Input size (MB):	0.19
Forward/backward pass size (MB):	3.80
Params size (MB):	4.72
Estimated Total Size (MB):	8.71

Figura 5.7: Resumen de la arquitectura proporcionado por PyTorch

Se muestra también, de la misma forma que se hizo con las distintas versiones comentadas, la arquitectura final del modelo. Se incluye en la representación la red densa que realizará la función de clasificación. Esta red densa toma como entrada el aplanado de la última capa convolucional para llegar a la capa final con cuatro neuronas.

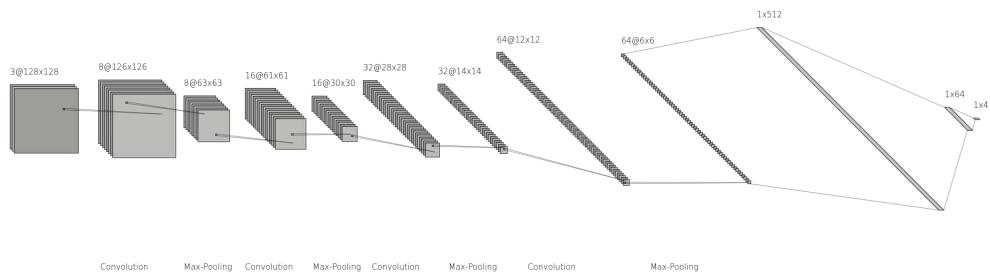


Figura 5.8: Arquitectura definitiva de la red convolucional. Estilo LeNet

Cabe destacar que no se incluye en la representación la capa de entrada del MLP, ya que esta capa tiene 2.304 (64x6x6) neuronas, resultado del aplanado de los filtros, y no permitiría ver la estructura importante.

Capítulo 6

Explicabilidad

El método implementado para la explicabilidad de la predicción es, como ya se ha adelantado, Grad-CAM. Como se explicó con anterioridad, esta técnica de interpretación permite visualizar las regiones importantes en una imagen de cara a la toma de decisiones en una Red Neuronal Convolutacional (CNN). A partir de los gradientes, se generan mapas de activación de clase, que resaltan las áreas relevantes para la clasificación, brindando una explicación visual del proceso seguido.

6.1. Implementación

No existe ningún módulo conocido en PyTorch para aplicar Grad-CAM, por lo que la implementación se realiza íntegra de forma manual. Para ello, tomando como referencia Kulkarni, A. at al. (2022) [31], se realiza el siguiente proceso:

En primer lugar se escoge una imagen, sobre la que se va a realizar el análisis de su predicción. De cara a generar el mapa de calor, se vuelve a evaluar a través de todas las capas convolucionales, de forma que se almacenen las activaciones de la última capa convolucional.

Una vez que se han obtenido estos valores, se realiza la retropropagación de los valores para calcular los gradientes de la predicción, y se calcula el promedio de los gradientes de la última capa convolucional. En último lugar, se multiplican las activaciones por los gradientes de esta última capa, para obtener la relevancia de las activaciones en cada canal, lo que ya permitirá calcular el mapa de calor promediando las activaciones.

Con el mapa de calor que recoge la relevancia de las activaciones en la predicción ya calculado, se muestra como una capa sobre la imagen original, de forma que se pueda ver la importancia de cada uno de los puntos sobre la estampa original. Se muestra un ejemplo del resultado sobre cada clase en la figura 6.1.

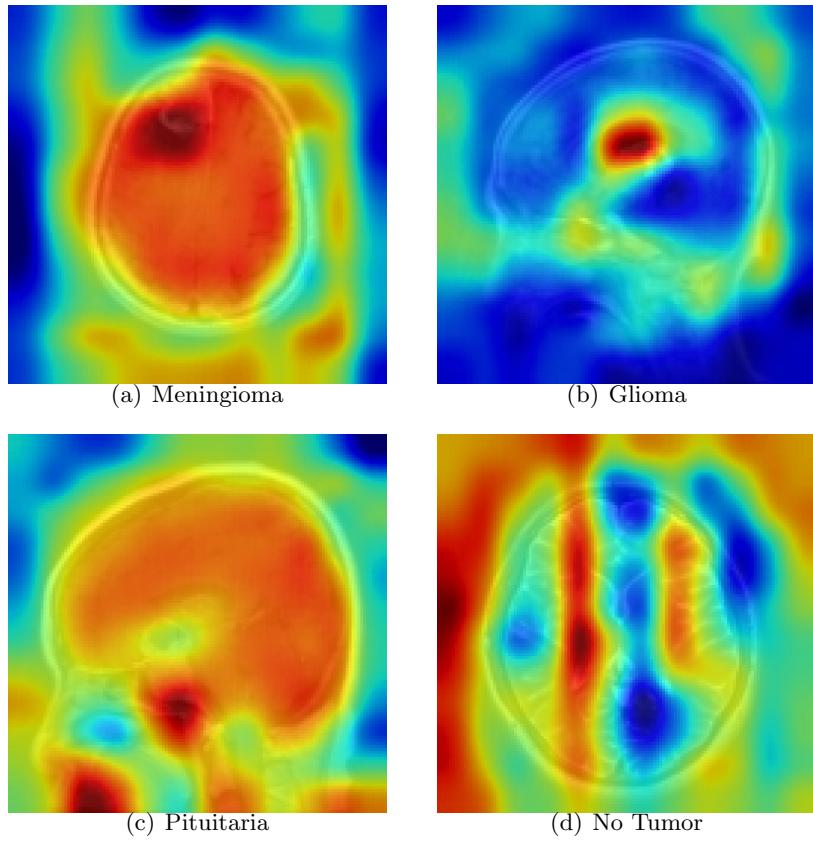


Figura 6.1: Grad-CAM sobre lo distintos tipos de tumores incluidos en el estudio

Ahora, la explicación de la toma de decisiones resulta más coherente, ya que se puede visualizar cómo en cada uno de los casos el tumor ha sido detectado y, por ende, asignado a su clase real.

En el caso del meningioma, figura 6.1(a), tumor que se forma en la capa más externa del tejido que recubre al cerebro, y a la médula espinal, vemos cómo el modelo toma exactamente esto como referencia, viéndose resaltada en el mapa toda la superficie del cerebro y con una influencia aún mayor en la zona más cercana a la médula espinal.

De forma similar, a la hora de detectar el glioma, figura 6.1(b), se puede observar cómo las zonas más determinantes se centran en el interior del cerebro, concretamente en el soporte viscoso, que es la zona en la que se originan este tipo de tumores.

Como se puede observar en la Fig. 6.1(d), las activaciones más influyentes resultan no tener ningún patrón, ya que durante su procesamiento, todos los filtros especializados en la búsqueda de las características representativas de cada una de las clases que contienen tumores, han tratado de localizarlos, pero sin poder encontrarlos. Es por este motivo por el que la imagen ha sido analizada por completo y, al no haber sido encontrado ningún patrón relacionado con ningún tumor, se asignan al grupo que no contienen tumores cerebrales.

En el tumor de pituitaria, Fig 6.1(c), también se puede apreciar, gracias a los tonos granates del mapa de calor, el peso de las activaciones. Éstas se localizan en la glándula pituitaria, que se encuentra detrás de la nariz, en la base del cerebro.

Capítulo 7

Aplicación

Debido a que el uso del modelo en un cuaderno de Jupyter restringe su uso a unos conocimientos de Python, se planteó como objetivo el desarrollo de una pequeña aplicación web sobre la que se pudieran utilizar los resultados obtenidos. Como se recoge en las tareas asociadas a la fase de despliegue de la planificación, se implementa esta pequeña aplicación web con pocos requisitos funcionales, ya que su único objetivo es acercar el uso del modelo a aquellas personas que no tienen unos conocimientos técnicos suficientes para utilizarla de otra manera.

Para ello, se crea un script de Python a partir del cuaderno de Jupyter, en el que se incluyen aquellas partes del código necesarias para poder realizar las predicciones y la implementación de la capa de explicabilidad. Para no tener que ejecutar la fase de entrenamiento se exportan los valores de los parámetros, de forma que se cargan en el modelo al iniciar la aplicación.

En este capítulo se presenta la documentación asociada al diseño de la aplicación.

7.1. Análisis

7.1.1. Requisitos funcionales

Identificador	Nombre	Descripción
RF-1	Cargar una imagen	El sistema permitirá la carga de imágenes.
RF-2	Diagnosticar	El sistema permitirá diagnosticar el tipo de tumor, a partir de una IMR, entre uno de los considerados en el proyecto.
RF-3	Visualizar diagnóstico	El sistema permitirá que los resultados, tanto del diagnóstico, como de la capa de explicabilidad, sean visibles.
RF-4	Visualizar interpretación	El sistema permitirá que la capa de explicabilidad sea visible.
RF-5	Diagnóstico múltiple	El sistema permitirá que se realice más de un diagnóstico.
RF-6	Conexión múltiple	El sistema debe soportar más de una conexión simultánea.

Cuadro 7.1: Requisitos Funcionales

7.1.2. Requisitos no funcionales

Identificador	Nombre	Descripción
RNF-1	Interfaz simple	El sistema permitirá que se realice la tarea de una forma sencilla.
RNF-2	Formato de imagen	El sistema permitirá que se utilicen los formatos .jpg, .jpeg y .png.
RNF-3	Lenguaje de programación	El cómputo interno debe realizarse en una versión de Python 3.8.
RNF-4	Tiempo	El sistema debe ser capaz de realizar el diagnóstico en menos de un minuto.

Cuadro 7.2: Requisitos No Funcionales

7.1.3. Requisitos de información

Identificador	Nombre	Descripción
RI-1	Modelos	El sistema permitirá almacenar el modelo que utilice para el diagnóstico.
RI-2	Diagnóstico	El sistema debe almacenar la imagen durante el proceso de diagnóstico.
RI-3	Resultados	El sistema debe almacenar el resultado, compuesto por la predicción de clase y una imagen hasta, al menos, el fin de la visualización.

Cuadro 7.3: Requisitos de Información

7.1.4. Casos de uso

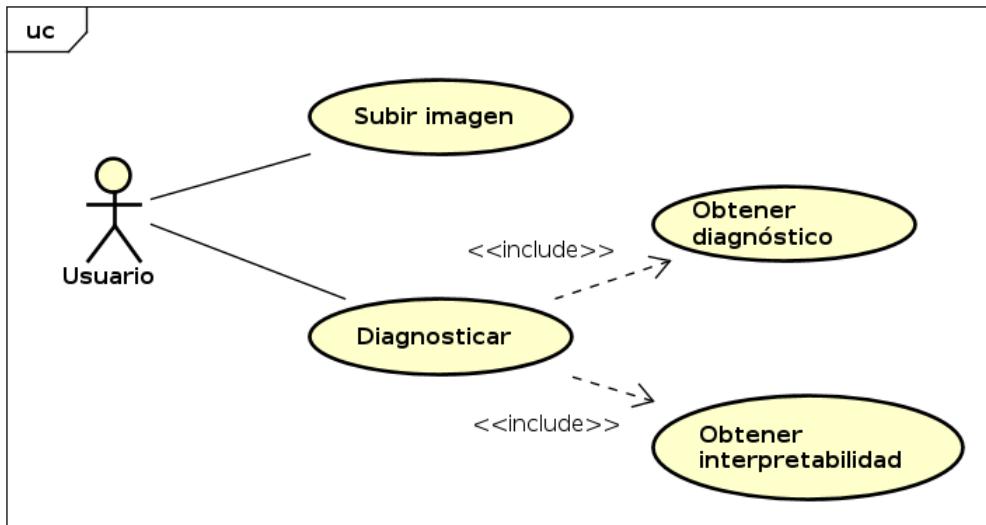


Figura 7.1: Diagrama de casos de uso

CU-001. Subir Imagen

CU-001	Subir Imagen
Descripción	El sistema permitirá al usuario subir una imagen en uno de los formatos aceptados desde su máquina local.
Actores	Usuario.
Precondiciones	1. El sistema, formado por el servidor y la aplicación, debe estar en funcionamiento y escuchando peticiones. 2. El usuario debe haber accedido a la dirección web donde está desplegada la aplicación.
Secuencia Normal	1. El usuario selecciona la opción que permite subir una imagen. 2. El usuario carga el archivo según el procedimiento de su SO. 3. El sistema verifica que el formato de la imagen sea uno de los aceptados. 4. El sistema almacena la imagen en el servidor. 5. El sistema muestra el nombre de la imagen.
Postcondiciones	1. La imagen se encuentra almacenada en el servidor
Secuencia alternativa	1a. El usuario utiliza la versión alternativa que permite subir la imagen mediante el método de 'drag-and-drop'. 2b. El CU continua en 3. 3a. El formato de la imagen no es uno de los aceptados. 3b. El sistema notifica al usuario que el formato de la imagen no es válido y recupera el estado inicial.

Cuadro 7.4: Descripción del CU-001: Subir imagen

CU-002. Diagnosticar

CU-002	Diagnosticar
Descripción	El usuario solicita al sistema el diagnóstico de una imagen previamente cargada en el sistema.
Actores	Usuario.
Precondiciones	<ol style="list-style-type: none"> 1. La imagen se encuentra almacenada en el servidor.
Secuencia Normal	<ol style="list-style-type: none"> 1. El usuario solicita efectuar un diagnóstico. 2. El sistema realiza el CU-003 ‘Obtener diagnóstico’. 3. El sistema realiza el CU-004 ‘Obtener interpretabilidad’. 4. El sistema muestra los resultados del diagnóstico y de la interpretabilidad.
Postcondiciones	<ol style="list-style-type: none"> 1. Los resultados del diagnóstico y la interpretabilidad son visibles.
Secuencia alternativa	<ol style="list-style-type: none"> 2a. 1- El CU-003 no se ejecuta correctamente. 2a. 2- El sistema muestra un mensaje de error y muestra la pantalla inicial. 3a. 1- El CU-004 no se ejecuta correctamente. 3a. 2- El sistema muestra un mensaje de error y vuelve a la pantalla inicial.

Cuadro 7.5: Descripción del CU-002: Diagnosticar

CU-003. Obtener diagnóstico

CU-003	Obtener diagnóstico
Descripción	El sistema calcula la predicción de la imagen seleccionada.
Actores	Sistema.
Precondiciones	<ol style="list-style-type: none"> 1. La imagen se encuentra almacenada en el servidor. 2. El modelo se encuentra almacenado en el servidor.
Secuencia Normal	<ol style="list-style-type: none"> 1. El sistema procesa la imagen. 2. El sistema carga el modelo. 3. El sistema calcula la predicción. 4. El sistema almacena la predicción
Postcondiciones	<ol style="list-style-type: none"> 1. El resultado de la predicción se encuentra almacenado en el sistema
Secuencia alternativa	<ol style="list-style-type: none"> 1a. 1- La imagen no se procesa correctamente. 1a. 2- El sistema muestra un mensaje de error y muestra la pantalla inicial. 2a. 1- El modelo no se carga correctamente. 2a. 2- El sistema muestra un mensaje de error y vuelve a la pantalla inicial. 3a. 1- El modelo lanza una excepción durante la predicción. 3a. 2- El sistema muestra un mensaje de error y muestra la pantalla inicial.

Cuadro 7.6: Descripción del CU-003: Obtener diagnóstico

CU-004. Obtener interpretabilidad

CU-004	Obtener interpretabilidad
Descripción	El sistema calcula la capa de interpretabilidad de la imagen seleccionada.
Actores	Sistema.
Precondiciones	1. La imagen se encuentra almacenada en el servidor. 2. El modelo se encuentra almacenado en el servidor.
Secuencia Normal	1. El sistema procesa la imagen. 2. El sistema carga el modelo. 3. El sistema calcula la interpretabilidad. 4. El sistema almacena la imagen resultado.
Postcondiciones	1. La imagen de interpretabilidad se encuentra almacenada en el sistema.
Secuencia alternativa	1a. 1- La imagen no se procesa correctamente. 1a. 2- El sistema muestra un mensaje de error y muestra la pantalla inicial. 2a. 1- El modelo no se carga correctamente. 2a. 2- El sistema muestra un mensaje de error y muestra la pantalla inicial. 3a. 1- El modelo lanza una excepción durante el cálculo de la interpretabilidad. 3a. 2- El sistema muestra un mensaje de error y muestra la pantalla inicial.

Cuadro 7.7: Descripción del CU-004: Obtener explicabilidad

7.2. Diseño

En relación al análisis realizado en la sección anterior, se presenta el diseño final de la solución adoptada, en el que se especifican las tecnologías utilizadas así como la arquitectura empleada. Se muestran también, en relación a esta fase de diseño, el diagrama de clases y la secuencia de cada caso de uso.

La primera decisión que condiciona el diseño es la relacionada con el RNF-1, donde se insta a que la aplicación debe ser sencilla de usar. Es por esto que se decide implementar la aplicación en Docker [10], ya que proporciona un entorno seguro en el que ejecutar la aplicación y evita la necesidad de un proceso de instalación costoso.

Una vez puesto en marcha, el usuario accederá al sistema a través de un navegador, dirigiéndose a la dirección web en la que se configure, evitando la necesidad de instalar software específico para el funcionamiento de esta app.

7.2.1. Tecnologías utilizadas

Como se especifica en el RNF-3, el servidor deberá ser montado en Python, ya que este fue el lenguaje escogido para la implementación de los modelos y de todo el funcionamiento interno de la aplicación.

El framework de Python escogido para la implementación es Flask [11], ya que es el más usado y recomendado para este tipo de tareas. Otro motivo que apoya la decisión de utilizar Flask es que dispone de una amplia documentación en línea que permite un fácil aprendizaje.

Flask incorpora directamente una gestión de sesiones, necesaria para cumplir con el RF-6, almacenando la información relativa a cada una de las peticiones que se realizan. Para el almacenamiento de estas sesiones se utiliza Redis [13], un motor de base de datos en memoria, ya que cumple con los requisitos planteados y presenta un soporte para su integración con Flask.

Si bien Flask suministra un servidor para el despliegue, se necesita de un servidor web HTTP que gestione las peticiones. La herramienta escogida para esta tarea es Gunicorn [12], que actúa como un servidor de aplicaciones y se encarga de recibir las peticiones HTTP, dirigiéndolas a la aplicación Flask correspondiente para su procesamiento. También gestiona la concurrencia y los diferentes hilos para manejar múltiples solicitudes de forma simultánea, lo que mejora la calidad de la respuesta y el rendimiento de la aplicación web.

Como resumen, se podría entender Flask como el framework que desarrolla la lógica de la aplicación web, y Gunicorn la encargada de ejecutar la aplicación y gestionar la concurrencia de las solicitudes entrantes.

Para la implementación de la apariencia de la web se utilizarán de forma combinada HTML, JavaScript y CSS, permitiendo el manejo tanto de la apariencia como de la funcionalidad de la web propiamente dicha.

7.2.2. Patrones de Diseño

Patrón MVC

El patrón de diseño Modelo-Vista-Controlador (MVC) es una arquitectura de software que se utiliza comúnmente para desarrollar aplicaciones web. El objetivo principal de este patrón es separar la lógica de la aplicación en los tres componentes principales que componen su nombre: el Modelo, la Vista y el Controlador. Cada uno de ellos tiene una responsabilidad [32].

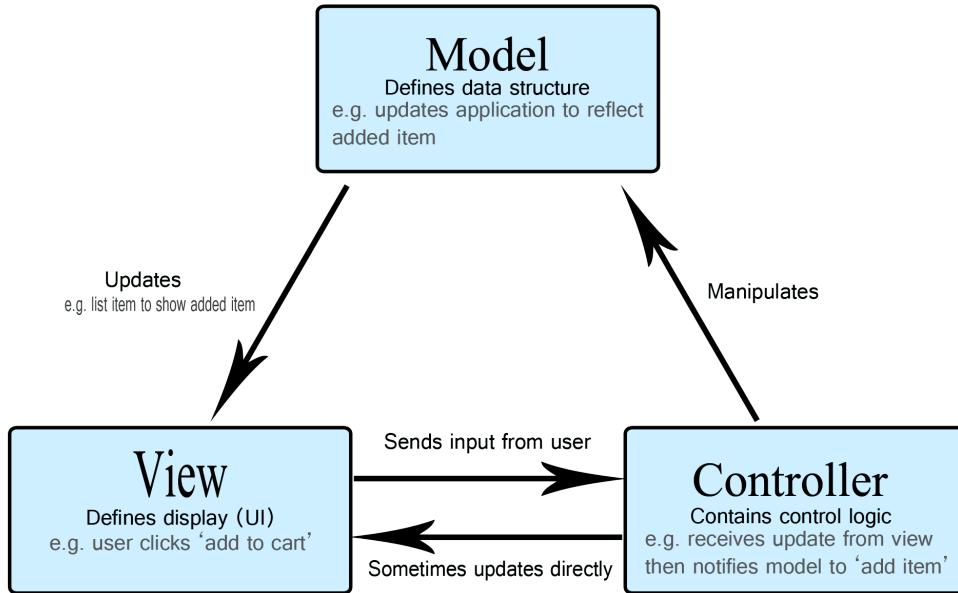


Figura 7.2: Patrón MVC. Fuente: <https://developer.mozilla.org>

El Modelo se encarga de representar la capa de datos de la aplicación, gestionando los datos, las reglas de negocio y la lógica de la aplicación. Es el encargado de interactuar con la base de datos, realizar cálculos y almacenar información relevante. Proporciona los métodos e interfaces necesarias para acceder y modificar los datos de la aplicación.

En esta aplicación, el modelo será desarrollado a través de los scripts python, en los que se desarrollan todo el procesamiento necesario para la predicción.

La Vista es la capa de presentación de la aplicación, que se ocupa de presentar la información al usuario de forma visual. Se desarrollará en HTML y se guiará el estilo a través de CSS.

El Controlador, encargado de la comunicación entre el Modelo y la Vista, es el que asume la recepción de las solicitudes del usuario a través de la interfaz y de su procesamiento. Esta funcionalidad será implementada en JavaScript.

Patrón Singleton

Conocido como patrón de única instancia, este patrón de diseño permite restringir la creación de objetos pertenecientes a una clase, o el valor de un tipo a un único objeto. Con esto, se garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella [33]. Se muestra su diseño en la figura 7.3.

En el caso concreto de esta aplicación, el patrón Singleton se aplicará a la clase relativa al modelo, de forma que será el controlador quien gestione la concurrencia de peticiones al modelo. Esto evitará que se ejecute el modelo de forma simultánea en distintas sesiones.

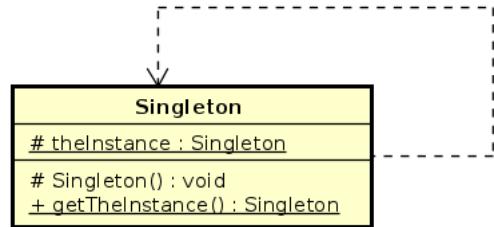


Figura 7.3: Patrón Singleton

Patrón Fachada

El patrón fachada define una interfaz global de acceso para que la aplicación no sea dependiente de los métodos del modelo. De esta forma, el sistema recibirá las dependencias y las transmitirá al modelo a través de la fachada, desacoplando las dependencias y permitiendo que la aplicación sea fácilmente extensible. En el caso de querer realizar otro tipo de predicciones, se deberá cumplir con las especificaciones de la fachada. [34]

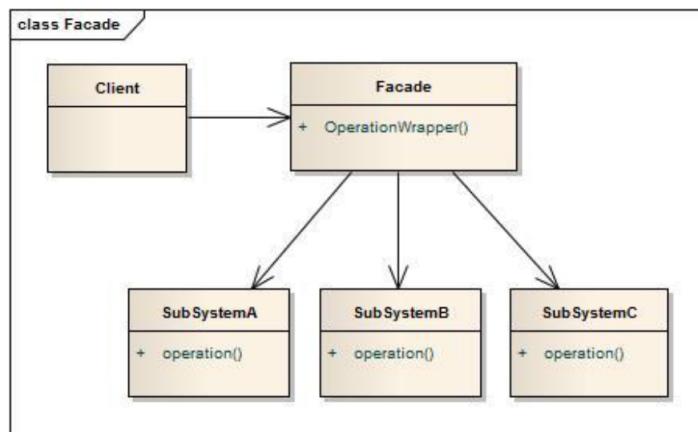


Figura 7.4: Patrón Fachada. [34]

7.2.3. Diagrama de clases

Se presenta el diagrama de clases [17], una representación visual de los objetos de clases en el sistema, donde se representan las relaciones entre los distintos elementos del sistema, con el fin de que el observador vea qué componentes necesita el sistema y cómo influyen los unos en los otros.

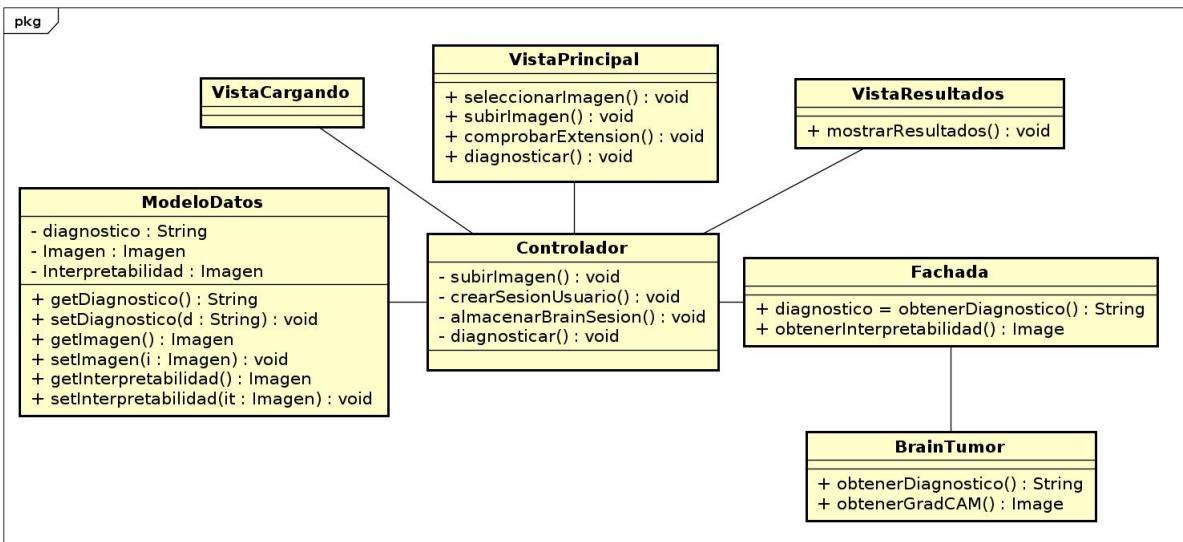


Figura 7.5: Diagrama de clases

7.2.4. Diagramas de secuencia

Se presentan en este apartado los diagramas de secuencias para cada uno de los casos de uso, con el objetivo de mostrar de forma clara el funcionamiento de la aplicación a través del diseño. Los diagramas de secuencia son un tipo de diagrama usado para modelar interacción entre objetos en un sistema según UML [35].

CU-001 - Subir imagen

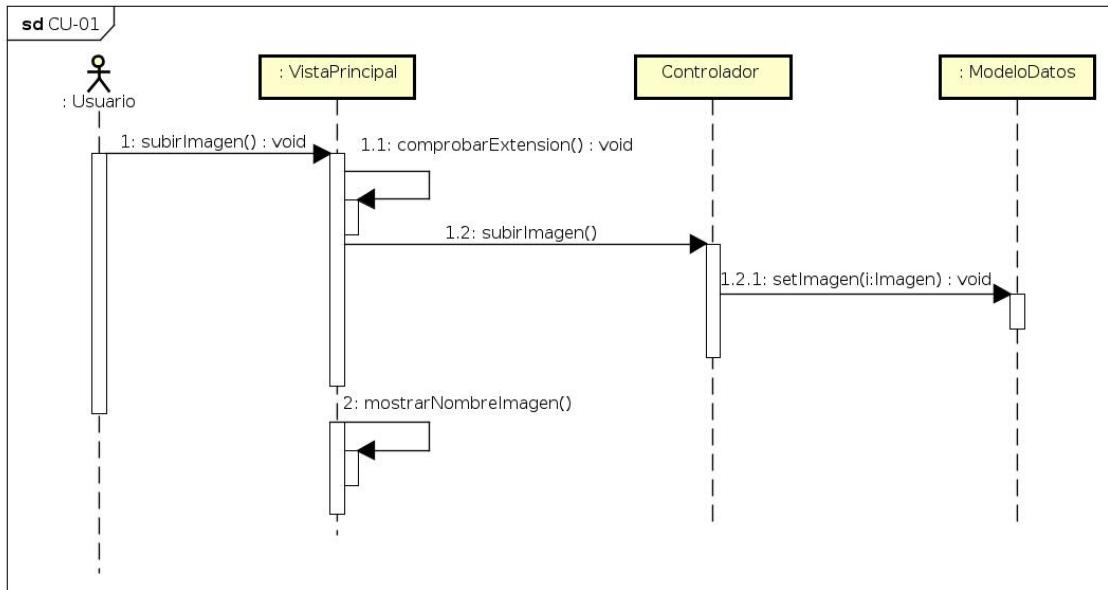


Figura 7.6: Diagrama de secuencia CU-001

CU-002 - Diagnosticar

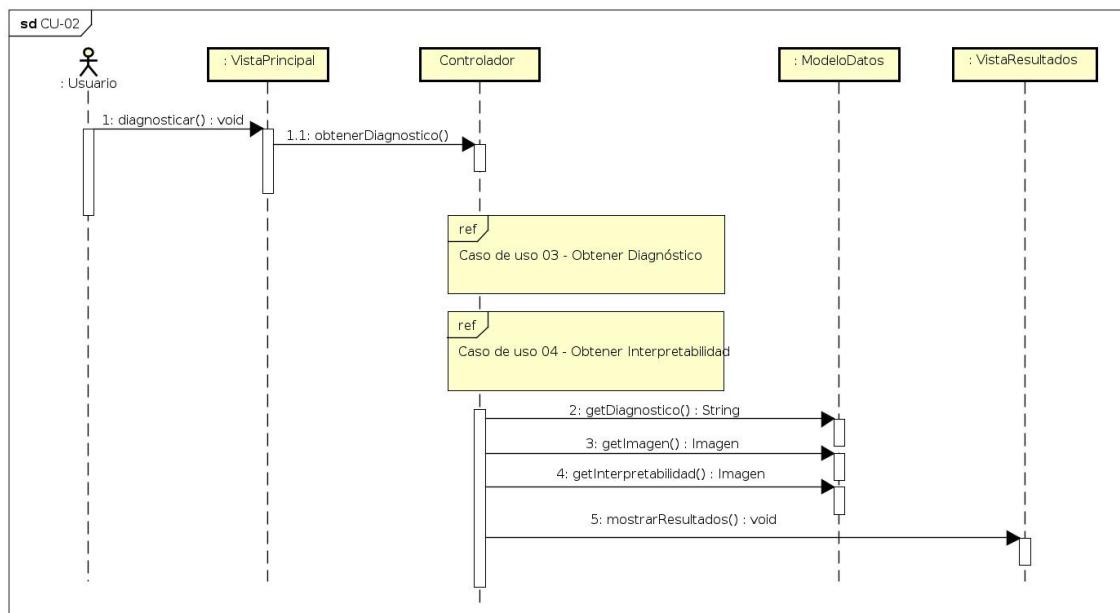


Figura 7.7: Diagrama de secuencia CU-002

CU-003 - Obtener Diagnóstico

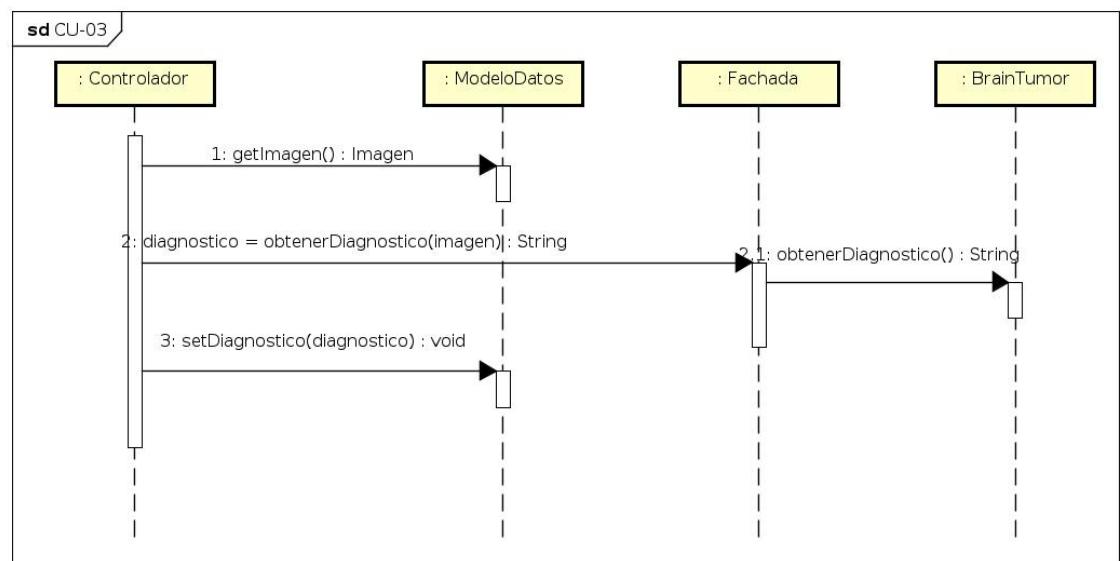


Figura 7.8: Diagrama de secuencia CU-003

CU-004 - Obtener interpretabilidad

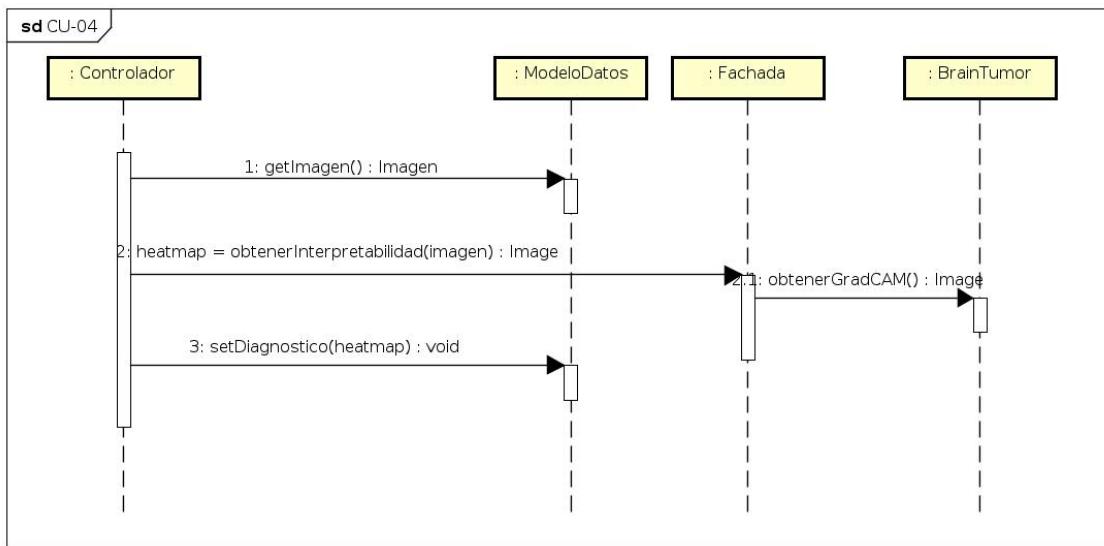


Figura 7.9: Diagrama de secuencia CU-004

Capítulo 8

Conclusiones y trabajo futuro

Durante la realización de este proyecto, de forma transversal, se han aplicado diferentes herramientas y conocimientos adquiridos durante la formación en el grado de Ingeniería Informática. Cabe destacar, entre las principales, la planificación y diseño el proyecto, la aplicación de técnicas de Inteligencia Artificial para el procesamiento de imágenes, y la Ingeniería del Software.

Gracias a estos conocimientos, se ha podido desarrollar el proyecto sin que sufra ningún retraso significativo respecto de la planificación inicial, lo que se considera satisfactorio. Debido a que la asunción del esfuerzo planteado era alta, haber sufrido pequeños retrasos en algunas tareas no eclipsa el buen trabajo realizado, que ha permitido dividir de una forma muy eficiente el esfuerzo. De este hecho se aprende que se debe realizar una estimación al alza del tiempo, ya que cualquier imprevisto hará que, de otra manera, no se cumpla la planificación.

Respecto a la sección principal y más técnica del proyecto, el desarrollo de un modelo eficiente y funcional para la clasificación de tumores cerebrales, se han debido ampliar los conocimientos adquiridos. Esta parte de formación ha sido enriquecedora, ya que se ha hecho a través de una gran bibliografía que ha ayudado a asentar aquellos conceptos que, aún pareciendo que ya eran sólidos, no era así.

Entre las técnicas aprendidas en este aspecto, el grueso de los esfuerzos han ido enfocados a la familiarización de la sintaxis de PyTorch y la implementación de Grad-CAM, dos aspectos que eran completamente desconocidos y de los que se puede decir que se ha cumplido con las expectativas.

Respecto a la evaluación del proyecto en general, las conclusión es muy positiva, ya que se han satisfecho todos los objetivos, logrado una alta tasa de precisión en la clasificación, superando el 90 %, y se ha implementado una capa de explicabilidad que logra detectar en un gran número de casos la localización del tumor, logrando justificar el porqué de la decisión tomada.

La aplicación web cumple con los requisitos que se marcaron: acercar el funcionamiento del modelo a aquellas personas que no tienen un conocimiento suficiente de Python. Se ha cumplido con el objetivo, ya que siguiendo la breve guía de instalación, disponible en el apéndice B, se puede desplegar la aplicación web. Una vez puesta en marcha, el diseño

cumple con los requisitos mínimos, ya que permite realizar cuantas predicciones se quiera de una forma rápida e intuitiva.

Como conclusión final, se quiere destacar el enorme potencial de este tipo de aplicaciones ya que, una vez conseguidos estos buenos resultados, solo se puede ir a mejor. La aplicación de estas técnicas no tiene fin, y obtener resultados tan positivos como el presentado a lo largo de este trabajo es un claro indicador de que hay mucho por descubrir.

8.1. Trabajo futuro

Durante el desarrollo del proyecto se han visto ciertos aspectos que, principalmente por falta de tiempo, no se han podido incluir como ampliaciones al trabajo. Entre estos aspectos, los que se han considerado más interesantes y se plantean como avance futuro sobre este trabajo son:

- **Desarrollo de una técnica para la optimización de parámetros en PyTorch.** El método empleado para la comparación de las distintas versiones es, como se ha podido ver, muy manual. Se ha realizado una comparación de forma aproximada y basándose en suposiciones, que aún habiendo dado resultado, se podría haber mejorado. Es por esto que se abre la puerta a desarrollar técnicas que traten de optimizar los parámetros de una forma más empírica y objetiva.
- **Desarrollo de otras técnicas de explicabilidad.** Si bien se ha implementado la técnica que se ha considerado la más adecuada, puede resultar de interés la inclusión de otras de las técnicas descritas en la bibliografía [23], [31] o [26].
- **Despliegue de la aplicación web en un servidor.** De esta forma la aplicación estaría disponible para su acceso desde cualquier equipo conectado a la red sin necesidad de ninguna instalación. Se planteó implementarla en la Máquina Virtual de la escuela, pero debido a que su fin exclusivo es el desarrollo del proyecto y, consecuentemente, será reubicada después de su finalización, no se llevó a cabo. Se plantea como trabajo futuro la búsqueda de un servidor gratuito que permita desplegar una aplicación de tan poco peso como la realizada.
- También enfocado a la mejora de la aplicación web, se plantea como trabajo futuro una **aplicación completa** en la que se integren distintos tipos de diagnósticos, de forma que se pueda considerar útil con un carácter general.

Bibliografía

- [1] European Commission. *Boosting European digital industry and society: Commission sets out vision for Europe's digital transformation by 2030*. European Commission, mayo de 2023. URL: https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=60419.
- [2] Ahmed Hosny et al. «Artificial intelligence in radiology». En: *Nat Rev Cancer* 18.8 (ago. de 2018), págs. 500-510. DOI: [10.1038/s41568-018-0016-5](https://doi.org/10.1038/s41568-018-0016-5).
- [3] Centers for Disease Control and Prevention (CDC). *National Center for Health Statistics. Health insurance coverage: Early release of estimates from the National Health Interview Survey, January-September 2021*. Centers for Disease Control y Prevention (CDC), ene. de 2022. URL: <https://www.cdc.gov/nchs/data/databriefs/db456.pdf>.
- [4] Ramiro Gilberto Ruiz-García et al. «Neuropsiquiatría del síndrome de Susac: a propósito de un caso». En: *Neuropsiquiatría* (2019).
- [5] Juan Carlos Gómez-Vega, María Isabel Ocampo Navia y Oscar Feo Lee. «Epidemiología y caracterización general de los tumores cerebrales primarios en el adulto». En: *Universitas Medica* 60 (mar. de 2019), págs. 47-60. ISSN: 2011-0839. URL: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S2011-08392019000100047&nrm=iso.
- [6] Vijay Wasule y Poonam Sonar. «Classification of brain MRI using SVM and KNN classifier». En: *2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS)*. 2017, págs. 218-223. DOI: [10.1109/SSPS.2017.8071594](https://doi.org/10.1109/SSPS.2017.8071594).
- [7] *Python*. Python Software Foundation. URL: <https://www.python.org/>.
- [8] *PyTorch*. PyTorch. URL: <https://pytorch.org/>.
- [9] *Javascript*. Javascript. URL: <https://js.org/>.
- [10] *Docker*. Docker. URL: <https://www.docker.com/>.
- [11] *Flask*. Flask. URL: <https://flask.palletsprojects.com/>.
- [12] *Gunicorn*. Gunicorn. URL: <https://gunicorn.org/>.
- [13] *Redis*. Redis. URL: <https://redis.io/>.
- [14] *CRISP*. IBM. URL: <https://www.ibm.com/docs/es/spss-modeler/saas?topic=guide-introduction-crisp-dm>.
- [15] *DMP*. Forbes. URL: <https://www.forbes.com/sites/metabrown/2015/07/29/what-it-needs-to-know-about-the-data-mining-process/#2065f3a3515f>.
- [16] *SPICE*. SPICE. URL: <https://www.spice-space.org/>.

- [17] Donald Bell. «UML Basics: The class diagram». En: *IBM.[Online] IBM* 15.09 (2004).
- [18] Donald Bell. «UML basics: The sequence diagram». En: *Retrieved July 17* (2004), pág. 2015.
- [19] Overleaf. Overleaf. URL: <https://overleaf.com/>.
- [20] Laura Vanessa Sosa Jerez y Luis Carlos Zamora Alvarado. «Estructura de redes neuronales (MLP) y su aplicación como aproximador universal». Tesis de pregrado. Universidad Distrital Francisco José de Caldas, 2022. URL: <http://hdl.handle.net/11349/30489>.
- [21] Hassan Ramchoun et al. «Multilayer Perceptron: Architecture Optimization and Training». En: *International Journal of Interactive Multimedia and Artificial Intelligence* 4.3 (2016), págs. 13-17. DOI: [10.9781/ijimai.2016.415](https://doi.org/10.9781/ijimai.2016.415). URL: <https://ijimai.org/journal/bibcite/reference/2523>.
- [22] Wojciech Zaremba, Ilya Sutskever y Oriol Vinyals. *Recurrent Neural Network Regularization*. 2015. arXiv: [1409.2329 \[cs.NE\]](https://arxiv.org/abs/1409.2329).
- [23] Ajay Thampi. *Interpretable AI: Building Explainable Machine Learning Systems*. Manning Publications, 2022.
- [24] D. Michie, D.J. Spiegelhalter y C.C. Taylor, eds. *Machine Learning, Neural and Statistical Classification*. Chichester: Ellis Horwood, 1994. ISBN: 0-13-106360-2.
- [25] Johannes Lederer. *Activation Functions in Artificial Neural Networks: A Systematic Overview*. 2021. arXiv: [2101.09957 \[cs.LG\]](https://arxiv.org/abs/2101.09957).
- [26] V Kishore Ayyadevara y Yeshwanth Reddy. *Modern Computer Vision with PyTorch*. Packt, 2020.
- [27] Rpubs. Rpubs. URL: https://rpubs.com/dsotilccama/CNN_Markdown.
- [28] Ramprasaath R. Selvaraju et al. «Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization». En: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. de 2017.
- [29] Kaggle. Kaggle. URL: <https://www.kaggle.com/datasets>.
- [30] Michael Avendi. *PyTorch Computer Vision Cookbook: Over 70 recipes to solve computer vision and image processing problems using PyTorch 1.x*. Packt Publishing, 2020. URL: <https://www.packtpub.com/product/pytorch-computer-vision-cookbook/9781838644833>.
- [31] Akshay Kulkarni, Adarsha Shivananda y Nitin Ranjan Sharma. *Computer Vision Projects with PyTorch: Design and Develop Production-Grade Models*. Apress, 2022. DOI: [10.1007/978-1-4842-8539-5](https://doi.org/10.1007/978-1-4842-8539-5). URL: <https://www.apress.com/gp/book/9781484285388>.
- [32] Yanette Díaz González y Yenisleidy Fernández Romero. «Patrón Modelo-Vista-Controlador.» En: *Telemática* 11.1 (jun. de 2012), págs. 47-57. URL: <https://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15>.
- [33] Krzysztof Stencel y Patrycja Wegrzynowicz. «Implementation Variants of the Singleton Design Pattern». En: *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*. Ed. por Robert Meersman, Zahir Tari y Pilar Herrero. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, págs. 396-406. ISBN: 978-3-540-88875-8.

- [34] Shuai Jiang y Huaxin Mu. «Design patterns in object oriented analysis and design». En: *2011 IEEE 2nd International Conference on Software Engineering and Service Science*. 2011, págs. 326-329. DOI: [10.1109/ICSESS.2011.5982229](https://doi.org/10.1109/ICSESS.2011.5982229).
- [35] Xiaoshan Li, Zhiming Liu y H. Jifeng. «A formal semantics of UML sequence diagram». En: *2004 Australian Software Engineering Conference. Proceedings*. 2004, págs. 168-177. DOI: [10.1109/ASWEC.2004.1290469](https://doi.org/10.1109/ASWEC.2004.1290469).

Apéndice A

Salida de las capas de la CNN

Se exponen en este anexo, con ánimo de clarificar algunas de las explicaciones, el estado de los *kernels* antes de realizar el entrenamiento, así como su tamaño. En el tamaño se puede observar esta concepción de cubo que se reflejó en la explicación.

Para hacer énfasis en el concepto de volumen, se incluye también la representación de un filtro de cada una de las capas, donde se aprecia su forma real, capaz de tomar información de todas las dimensiones.

Se muestra como ejemplo el paso de una imagen por la red, de forma sucesiva se extraen los filtros resultado tras el paso por las diferentes capas convolucionales y de Pooling.

A.1. Filtros antes del entrenamiento

Primera capa de filtro antes del entrenamiento
Tamaño de los filtros: `torch.Size([8, 3, 3])`

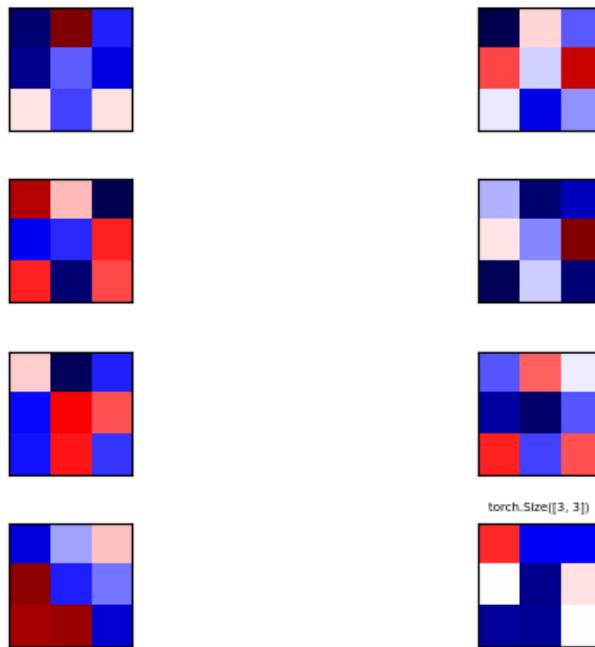


Figura A.1: Primera capa de filtros

Segunda capa de filtro antes del entrenamiento
Tamaño de los filtros: `torch.Size([16, 3, 3])`

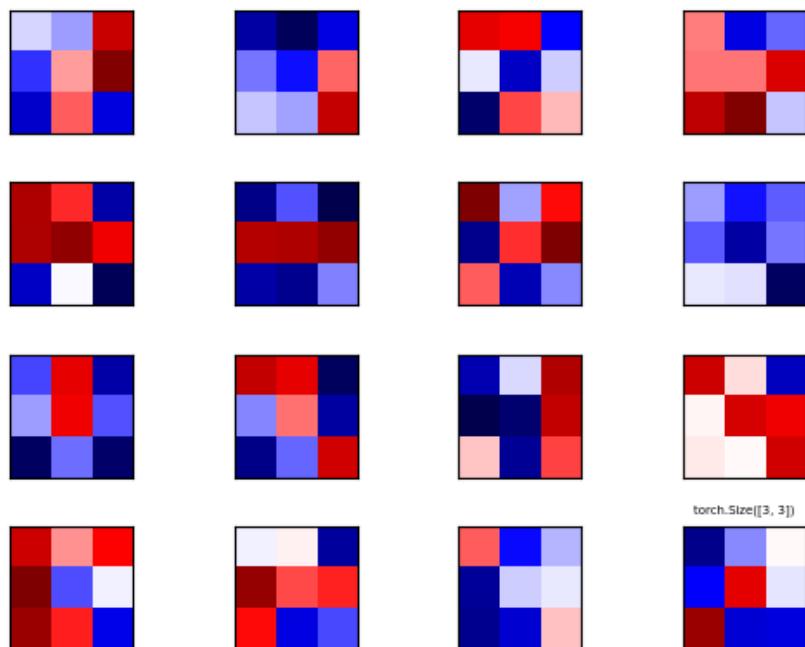


Figura A.2: Segunda capa de filtros

Tercera capa de filtro antes del entrenamiento
Tamaño de los filtros: `torch.Size([32, 3, 3])`

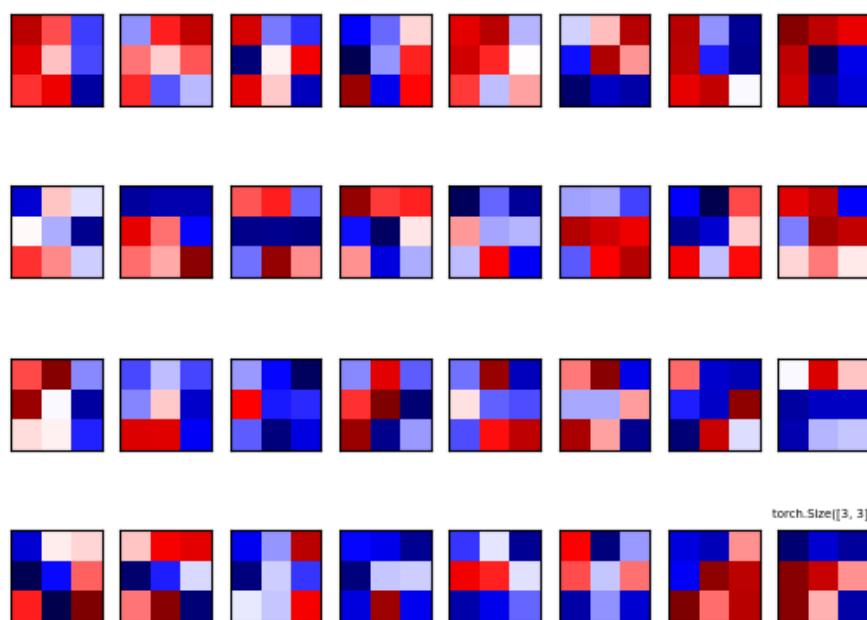


Figura A.3: Tercera capa de filtros

Cuarta capa de filtro antes del entrenamiento
Tamaño de los filtros: `torch.Size([64, 3, 3])`

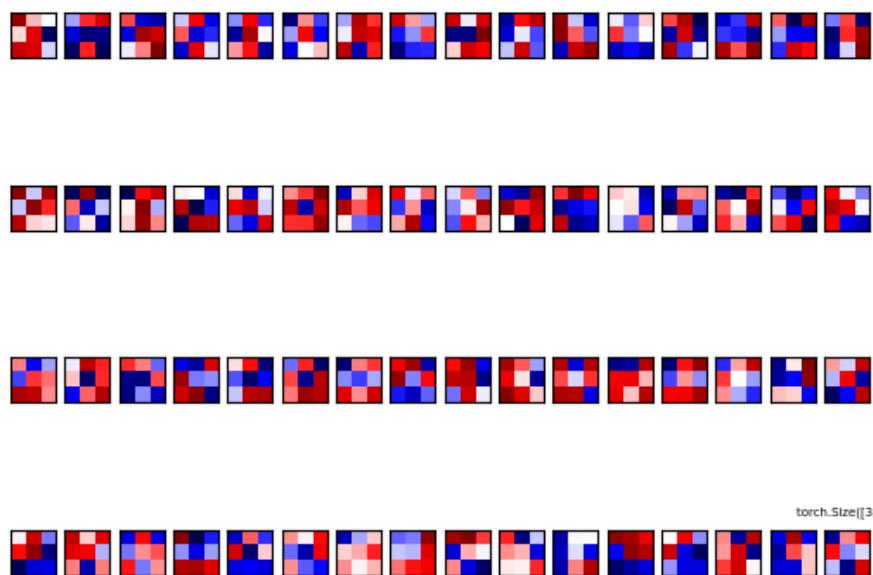


Figura A.4: Cuarta capa de filtros

A.2. Representación 3D de los filtros

Representación tridimensional
de un filtro en la capa 1

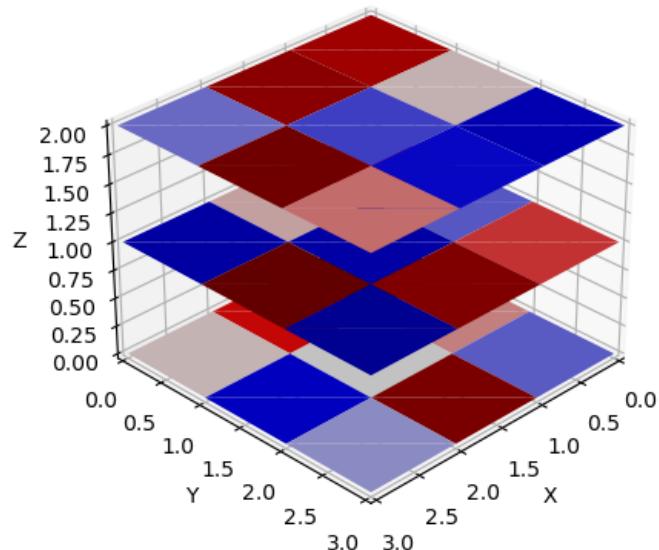


Figura A.5: Representación tridimensional de filtros en la primera capa

Representación tridimensional
de un filtro en la capa 2

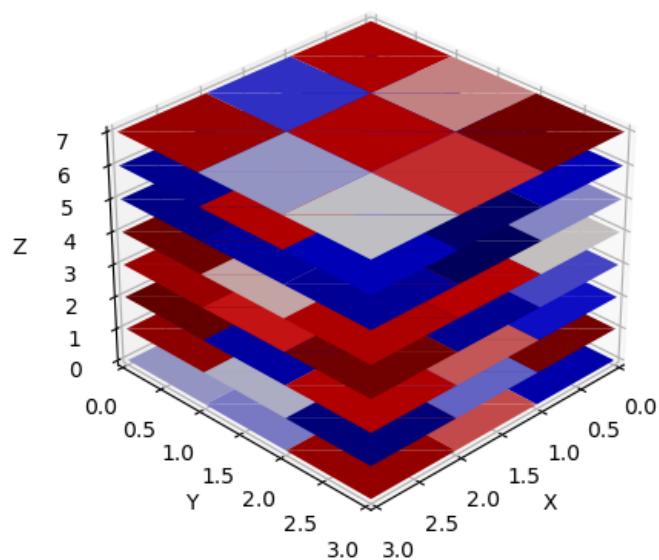


Figura A.6: Representación tridimensional de filtros en la segunda capa

Representación tridimensional de un filtro en la capa 3

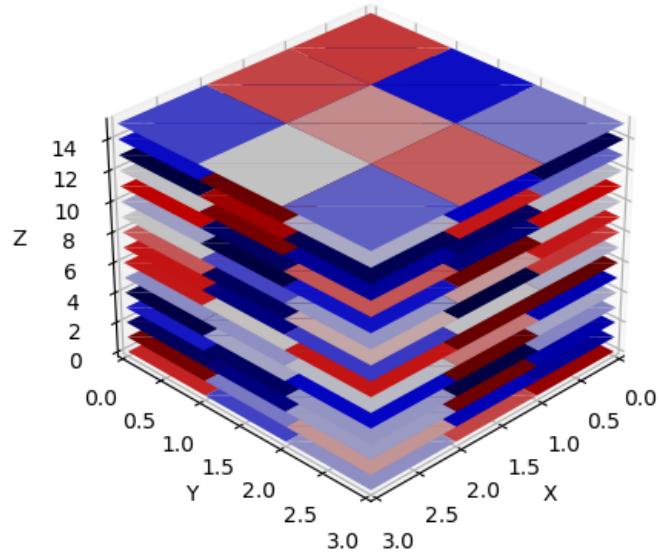


Figura A.7: Representación tridimensional de filtros en la tercera capa

Representación tridimensional de un filtro en la capa 4

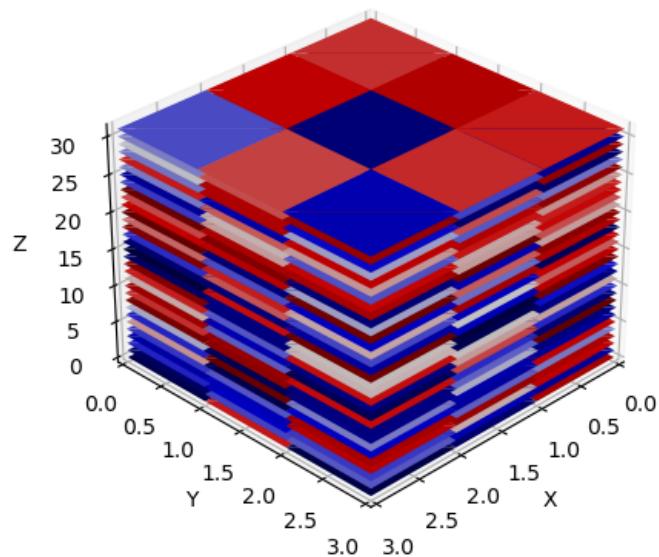


Figura A.8: Representación tridimensional de filtros en la cuarta capa

A.3. Paso de una imagen por la red

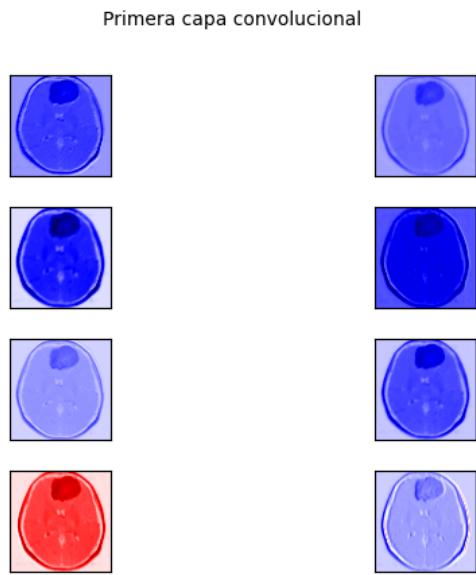


Figura A.9: Primera capa convolucional

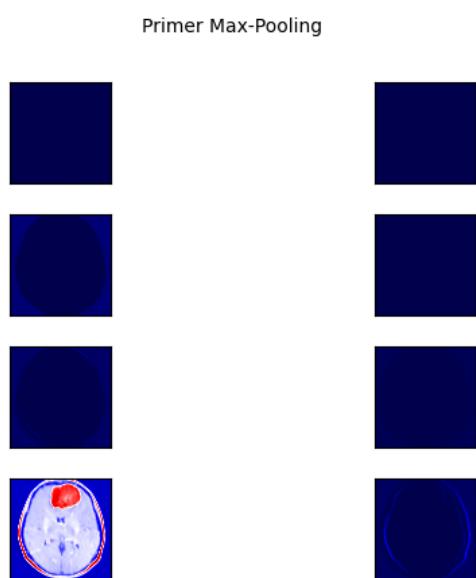


Figura A.10: Primer Max-Pooling

Segunda capa convolucional

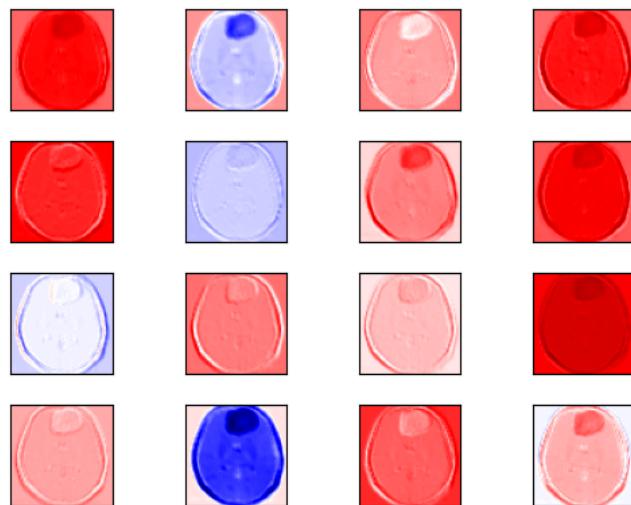


Figura A.11: Segunda capa convolucional

Segundo Max-Pooling

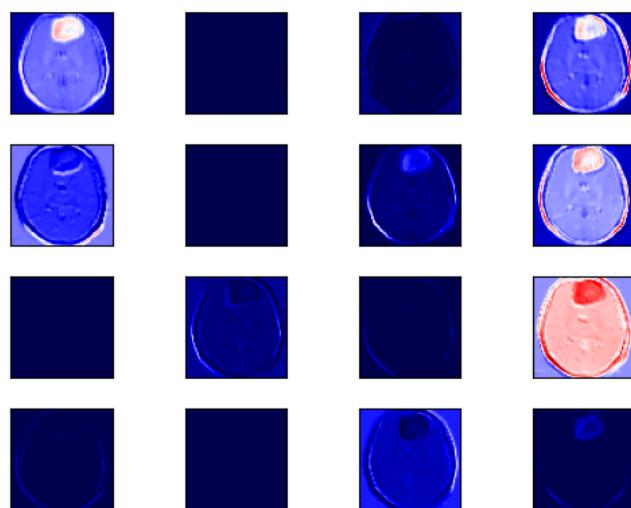


Figura A.12: Segundo Max-Pooling

Tercera capa convolucional

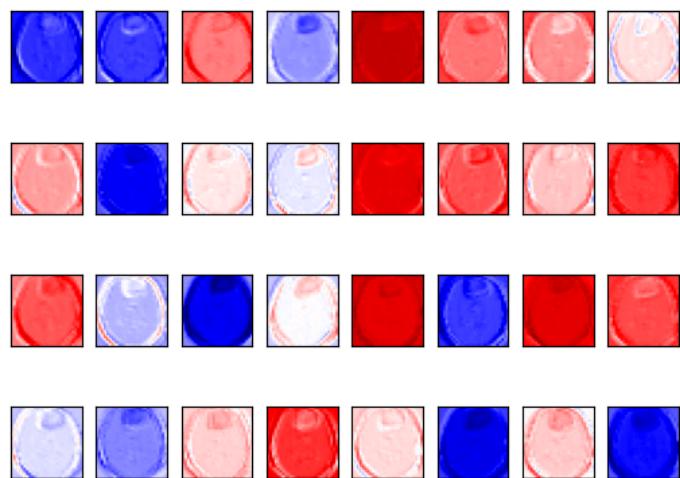


Figura A.13: Tercera capa convolucional

Tercer Max-Pooling

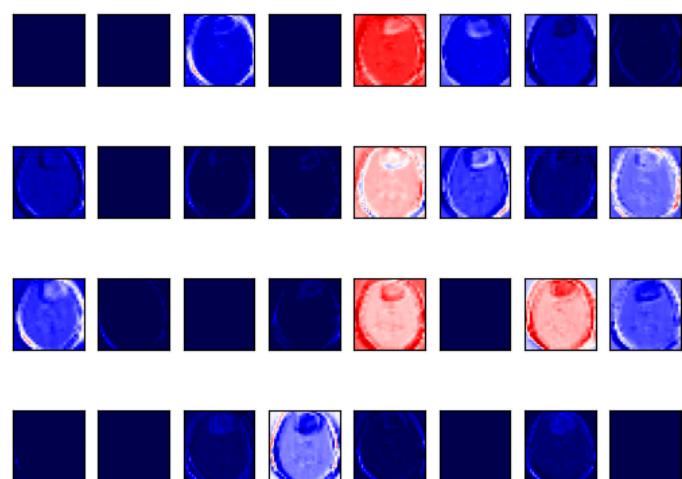


Figura A.14: Tercer Max-Pooling

Cuarta capa convolucional

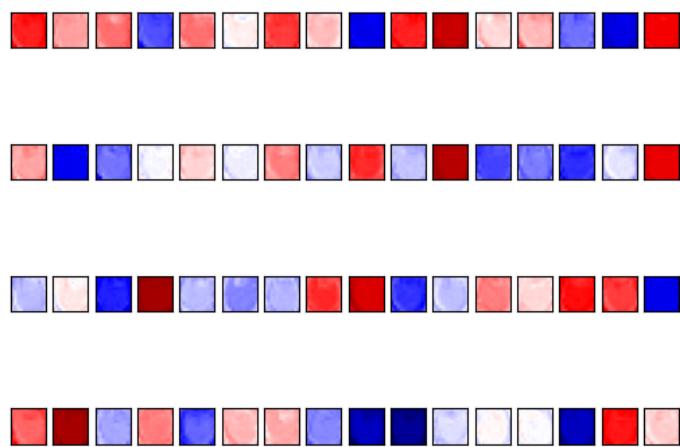


Figura A.15: Cuarta capa convolucional

Cuarto Max-Pooling

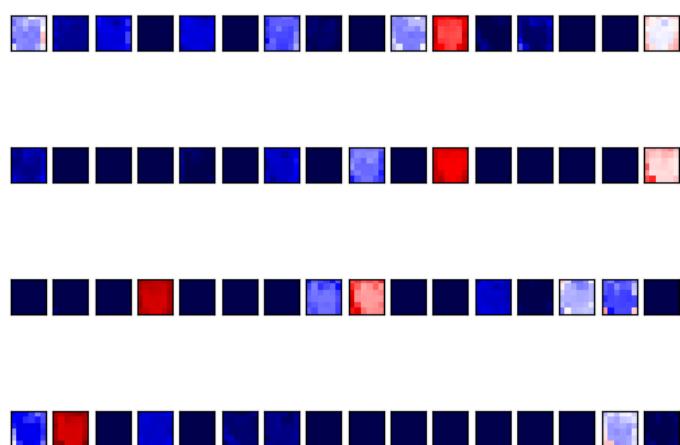


Figura A.16: Cuarto Max-Pooling

Apéndice B

Manual de instalación

Se presentan los pasos a seguir para instalar y configurar la aplicación web a nivel de servidor. Una vez realizado, se podrá acceder desde la máquina a través de cualquier navegador.

B.1. Requisitos

Las condiciones sobre las que ha probado la aplicación y que garantizan su correcto funcionamiento son:

- Ubuntu 22.04. (Kernel 5.19.0-42-generic)
- Docker, para el despliegue de la aplicación de forma segura
- Docker Compose, para crear e iniciar el servidor

Se debe seguir la instalación de Docker a través de la guía oficial disponible en:

<https://docs.docker.com/engine/install/ubuntu/>

De la misma manera, se debe realizar la instalación de Docker Compose a través de la documentación oficial:

<https://docs.docker.com/compose/install/>

B.2. Estructura de directorios

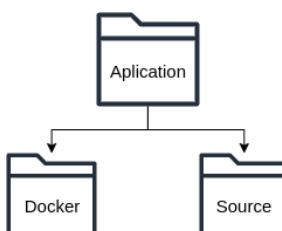


Figura B.1: Estructura de las carpetas

B.3. Instalación y despliegue

El proceso de despliegue de la aplicación web contenida en Docker se realiza de la siguiente manera:

1. Abrir la carpeta 'Docker' en una terminal
2. Ejecutar el comando:

```
docker-compose up
```

3. Acceder a la dirección *http://localhost:5000*.

Es posible que Docker Compose solicite permisos de administrador, ya que necesita abrir el puerto especificado en la configuración.

En el momento que se quiera abortar la ejecución del sistema, se recomiendo hacerlo a través del comando:

```
docker-compose down
```

De la misma manera, si el sistema lo solicita se concederán permisos de administrador para ejecutar esta operación.

Apéndice C

Manual de Usuario

Aún habiendo diseñado la aplicación para que su uso sea simple e intuitivo, se considera de interés incluir en el presente documento un manual de usuario, que aclare las posibles dudas que puedan surgir durante su uso.

C.1. Acceso a la página

Tras realizar la instalación de la aplicación web B, se debe acceder a un navegador desde el mismo equipo y especificar la dirección por defecto:

<http://localhost:5000>

De esta manera se accederá a la vista principal C.1, donde se podrá proceder a subir una imagen.

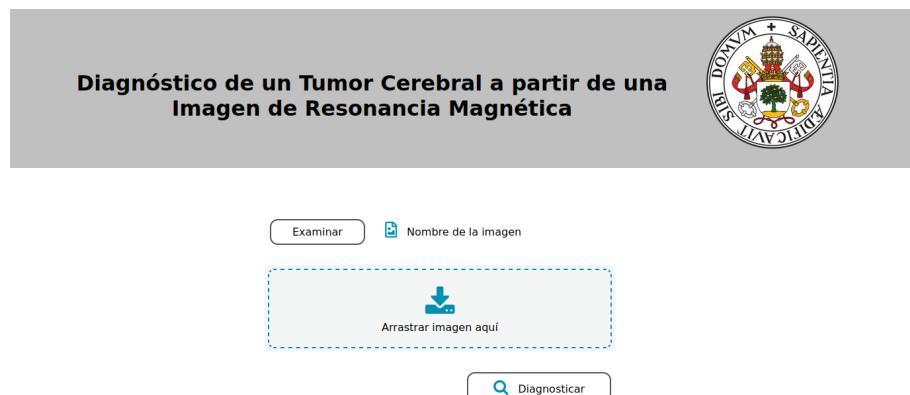


Figura C.1: Vista inicial

C.2. Subir una imagen

Desde la página principal se permite realizar dos acciones: subir una imagen o diagnosticar. Lo primero que debemos hacer es subir una imagen por cualquiera de las dos vías disponibles:

A través del botón *Examinar*

A través de este botón se accederá al gestor de archivos desde el que se podrá buscar y seleccionar la imagen sobre la que queremos realizar el diagnóstico. Solo se pueden seleccionar imágenes con extensión *.jpg*, *.jpeg* o *.png*.

No está permitido seleccionar varias imágenes de forma simultánea. Si se hiciera, el sistema mostrará un mensaje de error.

A través del *drag and drop*

La otra posibilidad disponible es el uso del área de *drag and drop*, sobre el que se puede arrastrar la imagen directamente desde el gestor de archivos. Se muestra un ejemplo en la fig. C.2.

Al igual que para la vía anterior, en caso de seleccionar varias imágenes o un formato no permitido, se mostrará un mensaje de error y se deberá subir una única imagen de los tipos permitidos.

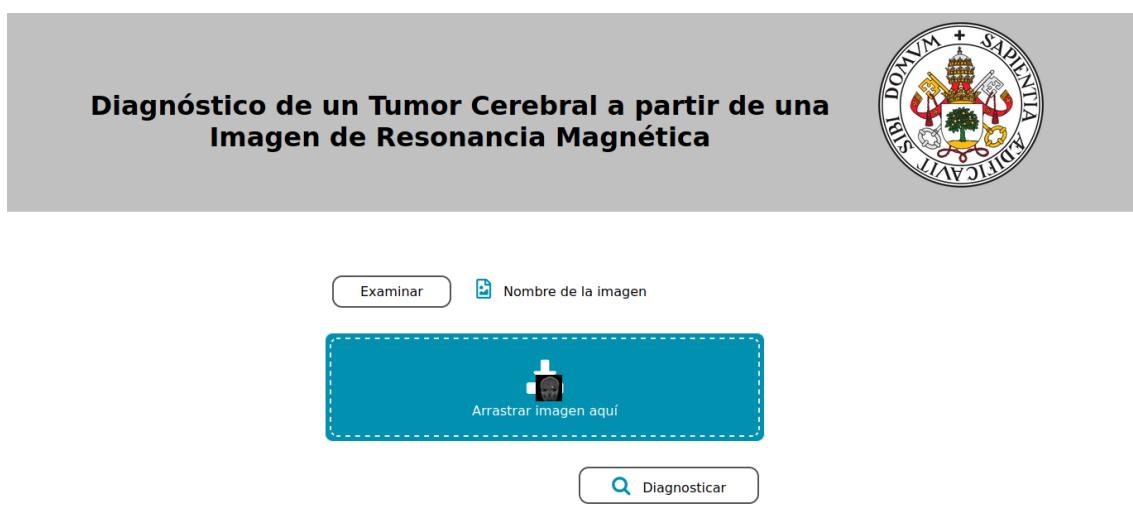


Figura C.2: Ejemplo de subir una imagen a través del *drag and drop*

C.3. Diagnóstico

Una vez cargada la imagen en el sistema se podrá leer su nombre como se muestra en la figura C.3.

Pulsando el botón 'Diagnosticar' el sistema comienza con la ejecución del modelo.

Durante unos instantes se mostrará la ventana de la figura C.4 para indicar que se está realizando la predicción, y tras este breve tiempo se mostrarán los resultados.

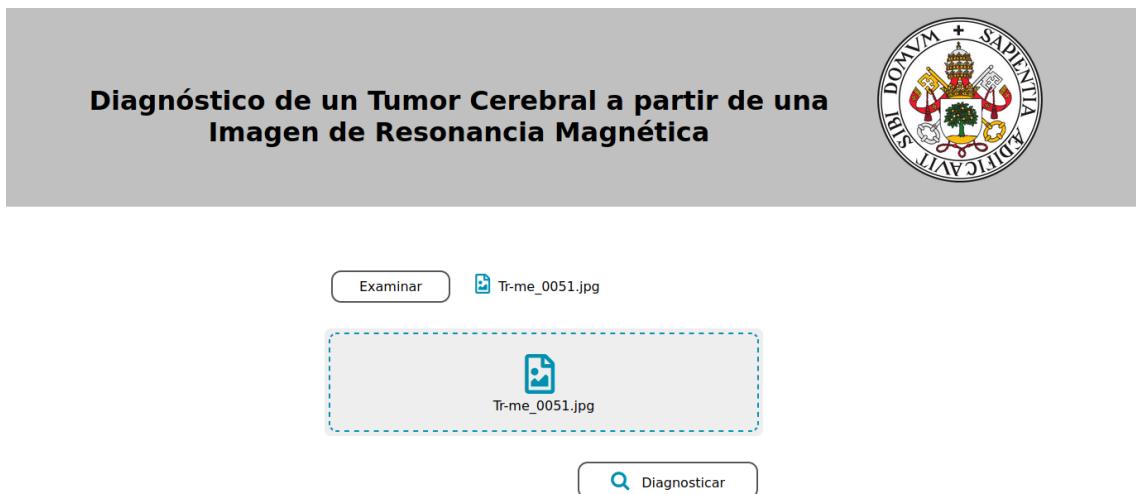


Figura C.3: Vista cuando la imagen se ha subido correctamente



Figura C.4: Vista mientras se ejecuta el modelo

En la ventana de resultados se puede ver tanto la predicción del tipo de tumor que ha realizado el modelo, como la comparación de la imagen original con la imagen en la que

se superpone el mapa de calor generado por Grad-CAM. Se muestra un ejemplo en la Fig. C.5.

Si se quisiera realizar un nuevo diagnóstico a partir de otra imagen, se debe pulsar el botón 'Realizar un nuevo diagnóstico' para acceder de nuevo a la vista principal.

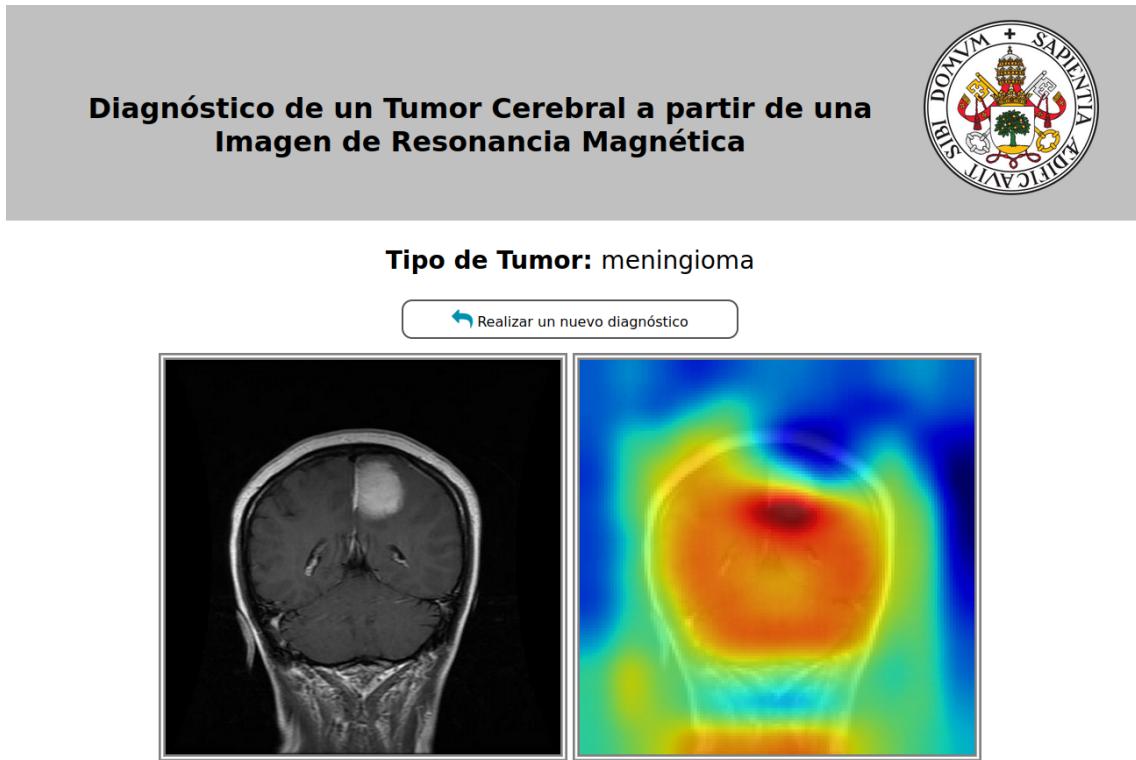


Figura C.5: Vista del resultado final