UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

TESIS DOCTORAL

Preprocesamiento y posprocesamiento de datos para Deep Learning utilizando GPUs

Deep Learning data preprocessing and postprocessing using GPUs

---

Autor: Luis Felipe Romero Caparrós

Dpto. Arquitectura de Computadores (Universidad de Málaga)

Director 1: Dra. Pilar Martínez Ortigosa

Dpto. de Informática (Universidad de Almería)

Director 2: Dr. Gerardo Bandera Burgueño

Dpto. Arquitectura de Computadores (Universidad de Málaga)

Programa: Doctorado en Ingeniería Mecatrónica

Línea: Computación de Altas Prestaciones

MÁLAGA, 4 de mayo de 2025

AUTOR: Luis Felipe Romero Caparrós

http://orcid.org/0000-0001-8475-2725

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña LUIS FELIPE ROMERO CAPARRÓS
Estudiante del programa de doctorado INGENIERÍA MECATRÓNICA de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: PREPROCESAMIENTO Y POSPROCESAMIENTO DE DATOS PARA DEEP LEARNING UTILIZANDO GPUS

Realizada bajo la tutorización de NICOLÁS GUIL MATA y dirección de PILAR MARTÍNEZ ORTIGOSA Y GERARDO BANDERA BURGUEÑO (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 15 de MARZO de 2025

Fdo.: LUIS FELIPE ROMERO CAPARRÓS Doctorando/a	Fdo.: NICOLÁS GUIL MATA Tutor/a
Fdo.: PILAR MARTÍNEZ ORTIGOSA Y GERARDO BANDERA BURGUEÑO	

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

Director/es de tesis

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

Dra. Pilar Martínez Ortigosa

Catedrática del Departamento de Informática de la Universidad de Almería

Dr. Gerardo Bandera Burgueño

Profesor titular del Departamento de Arquitectura de Computadores de la Universidad de Málaga.

CERTIFICAN:

Que la memoria titulada "Preprocesamiento y posprocesamiento de datos para Deep Learning utilizando GPUs", ha sido realizada por D. Luis Felipe Romero Caparrós bajo nuestra dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Mecatrónica.

Dra. Pilar Martínez Ortigosa

Codirectora de la tesis.

Dr. Gerardo Bandera Burgueño

Codirector de la tesis.

Autorización para la lectura de la Tesis e Informe de la utilización de las publicaciones que la avalan

Los abajo firmantes declaran, bajo su responsabilidad, que autorizan la lectura de la tesis del doctorando D. Luis Felipe Romero Caparrós titulada "Preprocesamiento y posprocesamiento de datos para Deep Learning utilizando GPUs" y que ninguna de las publicaciones que avalan dicha tesis han sido utilizadas en tesis anteriores

Dra. Pilar Martínez Ortigosa

Codirectora de la tesis.

Dr. Gerardo Bandera Burgueño

Codirector de la tesis.

Resumen

Esta tesis presenta un enfoque innovador que extiende el uso de las Unidades de Procesamiento Gráfico (GPUs) al preprocesamiento y posprocesamiento de datos dentro del Ciclo de Vida de la ciencia de datos para aplicaciones de *Deep Learning*. Tradicionalmente, las GPUs se han utilizado principalmente en las etapas de entrenamiento y validación de modelos, pero esta tesis demuestra cómo su capacidad de procesamiento masivamente paralelo puede acelerar significativamente tareas como la generación, transformación y análisis de datos.

Los métodos propuestos permiten superar limitaciones clave, como la insuficiencia de datos y los altos tiempos de procesamiento, lo que amplía la aplicabilidad de las herramientas de inteligencia artificial a nuevos dominios. Los resultados obtenidos en los casos de estudio, evidencian mejoras significativas en precisión, escalabilidad y eficiencia computacional.

Esta investigación también introduce herramientas y algoritmos que fortalecen la integración de la inteligencia artificial en nuevas aplicaciones prácticas, marcando un avance hacia sistemas más accesibles, robustos y efectivos.

Palabras clave: Aprendizaje profundo, GPU, redes neuronales, ciencia de datos, moléculas, descubrimiento de fármacos, modelo digital de elevaciones, detección de epilepsia.

Abstract

This thesis presents an innovative approach that extends the use of Graphics Processing Units (GPUs) to data preprocessing and postprocessing within the data science lifecycle for deep learning applications.Traditionally, GPUs have been used primarily in the model training and validation stages. However, this work demonstrates how their massively parallel processing capability can significantly accelerate tasks such as data generation, transformation, and analysis.

The proposed methods overcome key limitations, such as insufficient data and high processing times, thereby extending the applicability of artificial intelligence tools to new domains.The results obtained in the case studies demonstrate significant improvements in accuracy, scalability, and computational efficiency.

Furthermore, the research introduces tools and algorithms that facilitate the integration of artificial intelligence into new practical applications, thereby signifying a progression towards more accessible, robust, and effective systems.

Keywords: Deep Learning, GPU, neural networks, training, data science, molecules, drug discovery, digital elevation model, epilepsy detection.

Agradecimientos

Detrás de cada proyecto, y especialmente aquellos que ocupan una parte importante y decisiva en la vida de uno, hay mucho esfuerzo y horas de trabajo detrás pero, sobre todo, hay personas. Llegado el final de un largo recorrido, quiero dar las gracias a todas y cada una de ellas que me ayudaron a levantarme en los momentos de dudas y desesperanza para poder alcanzar este día, donde, con vista atrás, puedo reconocer que el camino mereció la pena.

Plantearse una meta, un propósito que lograr o un objetivo de vida, nunca es fácil. Especialmente cuando el abanico de posibilidades es grande, cuando se anda en busca de un sentido vital y cuando esos compañeros de promoción empiezan a progresar profesionalmente. Quiero reconocer en primer lugar, por tanto, a aquellos que me animaron a dar el primer paso y me ayudaron a recordar quién era y cuáles eran mis fortalezas para poder lograrlo.

Gracias, Papá, a ti especialmente. Por la gran admiración que siento por ti desde pequeño, por la inspiración que me ha llevado siempre a querer ser como tú: inteligente, trabajador y humilde. Fuiste quien despejó todas mis dudas cuando más perdido estaba, quien me hizo ver que era un camino bonito y el premio grande, quien durante toda esta etapa ha estado incondicionalmente ahí conmigo, ayudándome a resolver todos los problemas y dificultades que surgían. Te debo en gran parte los resultados que en esta tesis se exponen y sé que sin ti no hubiera sido posible. Sin duda, el mejor maestro que se puede tener en la vida, y no me refiero únicamente a lo académico.

Ninguno de la familia podríamos ser quienes somos sin Mamá, mi ejemplo a seguir. Porque nada de lo que haga en la vida sirve, si no tengo amor. Gracias Mamá, por ese amor que demuestras en las pequeñas cosas cotidianas, por tu apoyo y ánimos cuando todo se me venía encima y porque siempre me has recordado que debía creer en mí y en lo que valgo. Has sido el verdadero motor, porque cada cosa que he hecho ha sido con este sentido que me enseñas.

Gracias a mi hermano Javi, porque te debo mucho de mi autoestima. Sentirme admirado por ti, con esas preguntas para que te ayudara a resolver tus dudas académicas y profesionales, me ha recordado siempre que esto que hacía, merecía la pena. Y debo reconocer que los papeles se han invertido en muchas ocasiones donde he sido yo quien he tenido que recurrir a ti. Verte crecer tanto en todos los aspectos me ha hecho sentir realmente orgulloso de ti, a la vez que descubro un potencial cada vez más grande por las cosas que puedes llegar a hacer en la vida.

Gracias a toda mi familia, por su confianza y por acompañarme en cada paso que doy. En especial a mi abuela Mari, mi alma gemela como siempre he dicho, quien con paciencia me ha animado siempre a seguir luchando; y a mis abuelos Lupi, Federico y Yolanda, quienes siempre me han preguntado por mis progresos y me han recordado lo orgullosos que se sentían de mí.

No puedo olvidarme de Lola, cuyas sesiones de *coaching* fueron decisivas para optar por esta última etapa universitaria, en un momento de grandes dudas y falta de identidad personal, cuando no sabía si optar por el mundo emprendedor, la empresa privada o seguir formándome con este programa de doctorado.

Por otra parte, no han sido pocos, precisamente, a los que le debo mi crecimiento académico, gran parte de los conocimientos adquiridos y los resultados logrados en esta tesis. Por ello, quiero agradecer:

A mi directora de tesis Pilar, por proponerme en todas las investigaciones del departamento que han dado pie a los casos de estudio que se exponen en esta tesis. Gracias también por tu dedicación y ayuda durante estos años pero, sobre todo, por reconocer y celebrar mi esfuerzo y resultados. Toda la motivación volcada en tus investigadores te ha convertido en una auténtica líder.

A mi "epilbro" Marcos, un auténtico amigo que me ha acompañado durante estos cuatro años. Un gran compañero con el que compartir todos los logros de esta investigación y un profesor que me ha guiado en mis primeros pasos con el *Deep Learning* y otras muchas tecnologías. Gracias también por todas esas bromas, risas, charlas intensas y cervezas que han hecho que disfrutara de todos los congresos y momentos compartidos. Gracias también al resto del equipo de la Universidad de Almería: Nico, Juanjo, Laura, Juani, Savins, Juan y todos los demás.

A mi director Gerardo, especialmente por su ánimo, su aportación en los trabajos publicados y su gran labor en esta última etapa.

A la Universidad de Almería y la Universidad de Málaga, por todos los recursos facilitados, los contratos de investigación y la tutorización de esta tesis.

A la unidad de Neurofisiología del Hospital Carlos Haya de Málaga y a aquellos pacientes que con paciencia y sacrificio colaboraron con las mediciones clínicas para la investigación.

Para concluir, me gustaría hacer una mención especial a todos los que de una manera más indirecta, pero con una profunda importancia, me han impulsado a crecer interiormente, ayudándome a encontrar un verdadero sentido a mis metas vitales y aportando un valor diferencial a cada esfuerzo.

El gran protagonista de mi crecimiento personal fuiste Tú, Señor, a quien debo las ganas de aportar los dones que me diste a la sociedad, amando y ofreciendo cada pequeña tarea para un futuro mejor. Viniste a salvarme cuando más oscuro lo veía todo, cuando parecía que se rompían todos mis esquemas, para recuperar el niño alegre que siempre había sido y esa ilusión que me caracterizaba.

Gracias también a mis padres espirituales: Guillermo, Manu, David y Miguel, quienes me ayudaron a discernir la voluntad de Dios y me enseñaron que el amor y el esfuerzo es el camino.

Por supuesto, también agradecer de corazón a mi prima Patri, pieza fundamental en mi proceso de redescubrimiento personal. Gracias por las risas, los llantos, tu paciencia

resolviendo mis dudas y perdonando mis errores y mil cosas más que ya conoces. Te busqué cuando más lo necesitaba y ahí estuviste.

Y como no, a Margot, mi gran casualidad y suerte. Aunque parezca absurdo, unos pocos meses te han bastado para sacar fortaleza, alegría y felicidad a esta última etapa de la tesis. La admiración que siento por ti y por las ganas que le pones a la vida me han inspirado para que yo hiciera lo mismo. Haces que todo sea muy fácil. Gracias también a sus padres, Margot y Ángel, por la gran acogida.

Gracias a todos mis amigos, que siempre estuvisteis dispuestos a un buen rato para despejarse de todo, a charlar de temas profundos, a reírnos hasta que dolieran las mejillas. Para poder mencionaros a todos, necesitaría añadir, quizás, un capítulo más a esta tesis. Todos sabéis lo importantes que sois para mí.

Por último, a toda la familia de Effetá y la Parroquia de san Miguel, donde siempre me he sentido tan a gusto y acogido, donde he podido desarrollarme como persona y darme a los demás.

Producción científica

Publicaciones

A Lightweight Execution Manager for Training TensorFlow Models under the Slurm Queuing System	
Línea de investigación	3
Revista	*Acta Polytechnica Hungarica*
Cuartil	Q2 (*Engineering, Multidisciplinary*)
Factor de impacto	1,4
Autores	M. Lupión, N. C. Cruz, F. Romero, J.F. Sanjuán & P. M. Ortigosa
Fecha	21-01-2025
DOI	10.12700/APH.22.3.2025.3.4 [1]

SkewEngine: enhancing performance of intensive calculations on regular meshes	
Línea de investigación	2
Revista	*Journal of Supercomputing*
Cuartil	Q2 (*Computer Science, Hardware & Architecture*)
Factor de impacto	3,3
Autores	F. Romero, P. M. Ortigosa, G. Bandera & L. F. Romero
Fecha	24/02/2024
DOI	10.1007/s11227-024-05923-2 [2]

Drugs discovery by shape similarity using Deep Learning	
Línea de investigación	1
Revista	*Journal of Optimization Theory and Applications*
Cuartil	Q2 (*Applied Mathematics*)
Factor de impacto	1,9
Autores	F. Romero, Luis F. Romero, P. M. Ortigosa, & J. L. Redondo
Fecha	15/07/2023
DOI	10.1007/s10957-024-02589-x [3]

Congresos internacionales

Euro-PAR 2024	
Trabajo	On the use of hybrid computing for accelerating EEG preprocessing [4]
Línea de investigación	3
Autores	F. Romero, M. Lupión, N. C. Cruz, L. F. Romero & P. M. Ortigosa
Ponente	F. Romero
Trabajo	A lightweight execution manager for training TensorFlow models...
Línea de investigación	3
Autores	M. Lupión, N. C. Cruz, F. Romero, J. F. Sanjuan & P. M. Ortigosa
Ponente	M. Lupión
Fecha	26/08/2024 - 30/08/2024
Lugar	Madrid, España

UCAmI 2023	
Trabajo	Edge IoT System for Wearable Devices: Real-Time Data Processing... [5]
Línea de investigación	3
Autores	M. Lupión, F. Romero, Luis F. Romero, J.F. Sanjuán & P. M. Ortigosa
Ponente	M. Lupión
Fecha	28/11/2023 - 30/11/2023
Lugar	Riviera Maya, México

CMMSE 2023	
Trabajo	Accelerating Epilepsy Diagnosis and Prediction on EEG Signals... [6]
Línea de investigación	3
Autores	F. Romero, M. Lupión, L. F. Romero, J. F. Sanjuan & P. M. Ortigosa
Ponente	F. Romero
Trabajo	SkewEngine: enhancing performance of intensive calculations... [7]
Línea de investigación	2
Autores	F. Romero, P. M. Ortigosa, G. Bandera & L. F. Romero
Ponente	L. F. Romero
Fecha	03/07/2023 - 08/07/2023
Lugar	Rota, España

EUROPT 2022	
Trabajo	Drugs discovery by shape similarity using Deep Learning [8]
Línea de investigación	1
Autores	F. Romero, L. F. Romero, J. L. Redondo & P. M. Ortigosa
Ponente	F. Romero
Fecha	29/07/2022 - 30/07/2022
Lugar	Lisboa, Portugal

Congresos nacionales

CEDI 2024	
Trabajo	Optimización de lectura de archivos EDF en C++: Avances para... [9]
Línea de investigación	3
Autores	F. Romero, M. Lupión, N. C. Cruz, L. F. Romero & P. M. Ortigosa
Ponente	F. Romero
Trabajo	A Herramienta de gestión de entrenamientos de TensorFlow... [10]
Línea de investigación	3
Autores	M. Lupión, N. C. Cruz, F. Romero, J. F. Sanjuan & P. M. Ortigosa
Ponente	M. Lupión
Fecha	17/06/2024 - 19/06/2024
Lugar	A Coruña, España

Jornadas Sarteco 2023	
Trabajo	Metodología de adquisición y procesamiento de datos para la... [11]
Línea de investigación	3
Autores	F. Romero, M. Lupión, L. F. Romero, J. F. Sanjuan & P. M. Ortigosa
Ponente	F. Romero
Trabajo	SkewEngine: Reorganización de mallas regulares para cálculos... [12]
Línea de investigación	2
Autores	F. Romero, G. Bandera & L. F. Romero
Ponente	L. F. Romero
Fecha	19/09/2023 - 22/09/2023
Lugar	Ciudad Real, España

Jornadas Sarteco 2022	
Trabajo	Reconocimiento de fármacos mediante Inteligencia Artificial [13]
Línea de investigación	1
Autores	F. Romero, L. F. Romero, J. L. Redondo & P. M. Ortigosa
Ponente	F. Romero
Fecha	21/09/2022 - 23/09/2022
Lugar	Alicante, España

Actas de conferencias

On the Use of GPU Computing for Accelerating EEG Preprocessing	
Línea de investigación	3
Congreso	*Euro-Par 2024: Parallel Processing*
Autores	F. Romero, M. Lupión, N. C. Cruz, Luis F. Romero & P. M. Ortigosa
Fecha	26-08-2024
DOI	10.1007/978-3-031-69583-4_19 [4]

Edge IoT System for Wearable Devices: Real-Time Data Processing, Inference, and Training for Activity Monitoring and Health Evaluation	
Línea de investigación	3
Congreso	*UCAmI 2023*
Autores	M. Lupión, F. Romero, Luis F. Romero, J.F. Sanjuán & P. M. Ortigosa
Fecha	26-11-2023
DOI	10.1007/978-3-031-48642-5_13 [5]

Otros trabajos

Nombre	**Tipo**	**Línea**	**Descripción**
GDAL Totalviewshed	Librería	2	Librería de cuencas visuales totales basadas en SkewEngine
Molecule Finder	App Web	1	Software comparativo de fármacos
AccelerEDFX	App web	3	Software de simplificación de ficheros EDF generados a partir de un EEG
AccelerEDFX Lib	Librería dinámica	3	Librería dinámica compilada en C++ para la generación de datasets para redes neuronales
Skew Engine	Librería dinámica	2	Librería dinámica compilada en C++ con la clase SkewEngine

Estancias de investigación nacional

Entidad	**Inicio**	**Fin**	**Motivo**	**Proyecto**
UAL	16/03/23	15/10/24	Contrato	Pulsera inteligente para la predicción, detección y notificación de ataques epilépticos (EPILSERA).
UAL	29/04/22	28/07/22	Contrato	Descubrimiento de fármacos con DL.

Formación adicional

Jornadas ≥ 5 **h** < 15 **h**

Id	Organizador	Título	Duración
1	NVIDIA DLI	Fundamentals of Accelerated Computing with CUDA C/C++	8h
2	NVIDIA DLI	Fundamentals of Deep Learning	8h

Cursos ≥ 15 **h** < 50 **h**

Id	Organizador	Título	Duración
1	FGUMA	Introducción al Internet de las Cosas con Arduino	25h
2	FGUMA	Deep Learning. Introducción	25h
3	IMFAHE	International Mentor Program (IMP)	20h

Expertos ≥ 100 **h** < 300 **h**

Id	Organizador	Título	Duración
1	UMA	Deep Learning y CUDA	125h
2	UGR	MOOC Machine Learning y Big Data para la Bioinformática. 2ª Edición	100h

Índice general

Índice de figuras

Índice de tablas

Acrónimos

ANN Red Neuronal *Autoencoder* (*Autoencoder Neural Network*). 28, 34, 118, 120, 122

CNN Red Neuronal Convolucional (*Convolutional Neural Network*). 20–22, 117, 118, 125

CPU Unidad Central de Procesamiento (*Central Processing Unit*). 5, 10, 12, 13, 15, 16, 41, 42, 54, 58, 59, 76, 81, 85, 88, 89, 93, 95, 106, 107, 113

DEM Modelo Digital de Elevaciones (*Digital Elevation Model*). 76, 79, 80, 88

DL *Deep Learning*. 4, 5, 8, 9, 16, 23, 30–32, 39, 56, 95, 121–123

DNN Red Neuronal Densamente Conectada (*Dense Neural Network*). 19–21

DWT Transformada Wavelet Discreta (*Discrete Wavelet Transform*). XIX, 40, 94, 95, 99, 101, 105, 108, 111–115, 124, 125

EDF Formato de Datos Europeo (*European Data Format*). 39, 40, 94–97, 102–104, 108–110, 112–115, 118, 120, 125

EEG Electroencefalograma. 7, 38–40, 47, 94–97, 99, 108–110, 114, 118, 124, 125

FFT Transformada Rápida de Fourier (*Fast Fourier Transform*). 77, 94, 95, 98, 99, 106–108

GAN Red Generativa Antagónica (*Generative Adversarial Network*). 19

GPU Unidad de Procesamiento Gráfico (*Graphics Processing Unit*). 2–5, 7–16, 37, 39–42, 45, 47, 54, 58, 59, 68, 75, 80, 81, 85, 88–90, 93, 95, 102, 105–108, 110–113, 121, 124, 125

IA Inteligencia Artificial. 1–3, 5, 9, 12, 16, 123

LRT Transformada Radon Local (*Local Radon Transformation*). 91, 92

ML *Machine Learning*. 3, 9, 10, 16, 25, 29, 47, 49, 94

RL Aprendizaje por Refuerzo (*Reinforcement Learning*). 29

sDEM Modelo Digital de Elevaciones sesgado (*skew-DEM*). 80, 87, 88, 90

SIMD *Single Instruction, Multiple Data*. 13, 59

SM *Streaming Multiprocessor*. 12

STFT Transformada de Fourier de Tiempo Corto (*Short Time Fourier Transform*). 40, 98, 99, 105, 106, 108–110, 113, 114, 124, 125

VAE *Variational Autoencoder*. 28, 29, 118

VS Cribado Virtual (*Visual Screening*). 5, 47, 48

WCS Servicio de Cobertura Web (*Web Coverage Service*). 39

WMS Servicio de Mapas Web (*Web Map Service*). 39

Capítulo 1

Introducción

1.1. Antecedentes y motivación

No cabe duda del impacto profundo y creciente que tiene la Inteligencia Artificial (IA) en la sociedad. Ha experimentado un crecimiento exponencial en los últimos años, quedando ya atrás aquellos tiempos donde era un tema abordado únicamente por expertos en el ámbito científico o incluso aquellas noticias que alertaban de los beneficios y retos que podía presentar esta tecnología. Hoy en día, no solo es conocida por toda la sociedad, sino que está plenamente integrada y lo inusual es encontrar a una persona que no haya usado nunca un modelo de procesamiento de lenguaje, que haya creado alguna imagen mediante modelos generativos o, al menos, que no se haya beneficiado indirectamente por alguna gestión automatizada realizada por un modelo entrenado.

El impacto de la IA ha sido ampliamente abordado y estudiado por S. Russell y P. Norving (2021) [14], quienes proporcionan una visión panorámica e integral del presente y futuro de la IA. También existen otros autores que, desde un punto de vista histórico, matemático, estadístico o filosófico han abordando tanto los aspectos técnicos como las implicaciones éticas y sociales [15–21]. En [14] se menciona, de manera general, la importancia de la IA para:

- la **automatización y productividad**, donde se está revolucionando toda la industria, la atención médica o los servicios financieros, mejorando la eficiencia y reduciendo costes, pero planteando desafíos sobre el empleo y la reestructuración laboral;
- la **toma de decisiones**, que van desde las inmediatas hasta otras más estratégicas en el mundo de los negocios así como diagnósticos más precisos y personalizados en medicina;

- la **vida cotidiana**, con sistemas de recomendación, vehículos autónomos o el procesamiento del lenguaje natural, entre otros, para una vida orientada al bienestar y la seguridad.

Sin embargo, también presentan unas **consideraciones éticas y sociales** que deben tenerse en cuenta a medida que la IA se extiende y que son analizadas en mayor profundidad por S. Russell (2019) [22]. Y es cierto que surgen preocupaciones por la privacidad, el sesgo de los algoritmos y la toma de decisiones autónomas que no se alineen con la dignidad humana que, como sociedad, debemos poner siempre como prioridad. Valores como la vida, la justicia, la igualdad, la libertad y la responsabilidad deben ser siempre la referencia para preservar este fin último.

Los autores también presentan una serie de tendencias futuras de la IA, marcando especialmente la idea de la colaboración hombre-máquina. Así, se desea una IA más generalizada que pueda abarcar una variedad de tareas más amplia, que colabore con el avance científico, que sea ética, explicable —es decir, que los algoritmos puedan justificar sus decisiones de forma comprensible para los humanos—, justa y que se desarrolle con las medidas de ciberseguridad adecuadas para que sea resistentes a fallos, ataques y malas decisiones.

En esta tesis doctoral, se han considerado algunas de estas tendencias ya que se proponen varios enfoques que avancen hacia una IA al servicio de la vida, esclareciendo la realidad mediante una visión más comprensiva de la naturaleza o la salud, para que el ser humano pueda discernir en última instancia con toda la información posible. Los modelos de IA que se presentarán resolverán problemas muy concretos, alejándose de la tendencia futura de una IA generalizada pero que permitirá mantenerla dentro de unos límites seguros y éticos para lograr avances científicos.

Los grandes pilares que justifican el gran desarrollo de la IA son el *Big-Data* [23], el uso de las Unidades de Procesamiento Gráfico (GPUs) [24, 25] y el desarrollo de *frameworks* y herramientas, como PyTorch [26] y Tensorflow [27], que facilitan la programación y la creación de modelos.

Las GPUs se destacan por su capacidad de procesamiento masivamente paralelo, lo que las hace ideales para el entrenamiento de redes neuronales profundas, una tarea intensiva en cálculos. Gracias a ello, las GPUs han permitido acelerar significativamente el entrenamiento y la implementación de modelos de inteligencia artificial, mejorando la eficiencia y eficacia de estos sistemas en diversas aplicaciones.

Sin embargo, a pesar de estos avances, uno de los problemas más recurrentes en la mayoría de proyectos donde se quiere aplicar la IA es la disponibilidad y calidad de los

datos.[1] A menudo, los conjuntos de datos no son lo suficientemente amplios o no están bien estructurados, lo que afecta al rendimiento y a la generalización de los modelos. En este contexto, la Ciencia de los Datos ha surgido como un enfoque integral que abarca todo el Ciclo de Vida del desarrollo de los modelos [28]:

1. Comprensión del problema, mediante cierto conocimiento experto e identificación de los objetivos a alcanzar.

2. Recopilación de toda la información posible, mediante medida de datos, sensorización, anotaciones, etc.

3. Exploración y análisis de los datos, mediante visualización y cálculo de estadísticos descriptivos.

4. Preparación de los datos, mediante normalización, aumento de datos o eliminación de aquellos no relevantes.

5. Aprendizaje de modelos usando técnicas de *Machine Learning* (ML).

6. Validación y prueba para contrastar la calidad del modelo obtenido.

7. Implementación e interpretación de informes que determinen si los modelos aportan información útil sobre el problema.

Se volverá a hacer referencia a este Ciclo de Vida en el Capítulo 3, ya que será el procedimiento que se seguirá en todos los casos de estudio.

Tradicionalmente, las GPUs se han empleado principalmente en las fases de entrenamiento y validación, donde la carga computacional es más intensa. Sin embargo, el potencial de las GPUs se extiende más allá de estas etapas, como se representa en la Figura 1.1.

En esta tesis, se propone extender el uso de las GPUs a las fases de preprocesamiento y posprocesamiento de los datos. El preprocesamiento incluye tareas como la normalización, el aumento de datos y la limpieza de los mismos, mientras que el posprocesamiento se enfoca en la evaluación y análisis de los resultados obtenidos. De esta manera, al aprovechar la capacidad de cálculo paralelo de las GPUs en estas fases, es posible acelerar todo el Ciclo de Vida de la ciencia de los datos, mejorando tanto la eficiencia como la escalabilidad de los proyectos de IA en nuevas aplicaciones.

[1] Se debe recordar que uno de los pilares del desarrollo de la IA es precisamente el *Big-Data*.

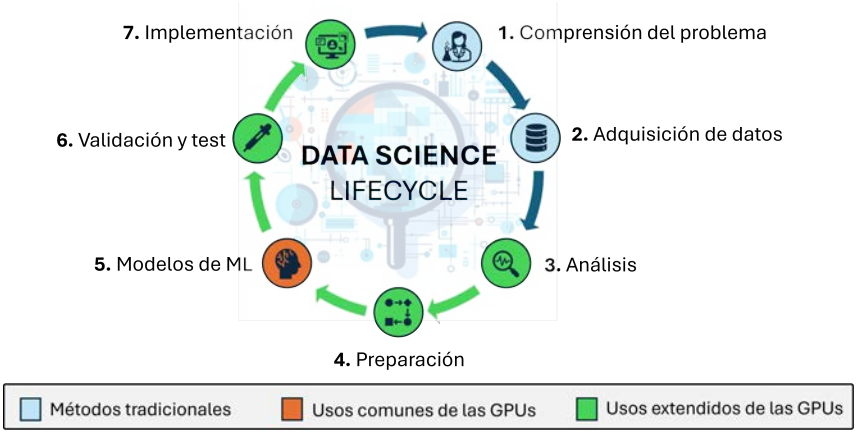


Figura 1.1: Ciclo de Vida de la ciencia de datos.

1.2. Objetivos

En esta tesis, independientemente de cada caso de estudio concreto, se proponen una serie de objetivos principales que servirán para englobar las distintas líneas de investigación en un mismo marco conjunto: el preprocesamiento y posprocesamiento de datos para el *Deep Learning* (DL) . Estos objetivos, marcados durante la planificación de la tesis, son:

Objetivo 01. Construir nuevos conjuntos de datos, más relevantes o útiles, para aprovechar los puntos fuertes del aprendizaje profundo, como las redes de convolución.

Objetivo 02. Mejorar la eficiencia de cálculo del preprocesamiento de los datos (mediante técnicas de filtrado, normalización, etc.) usando GPUs.

Objetivo 03. Reducir los tiempos de los algoritmos de optimización mediante predicciones por redes neuronales, entrenada con los datos preprocesados.

Objetivo 04. Demostrar la utilidad de los métodos para los distintos algoritmos de entrenamiento de las redes neuronales profundas.

Objetivo 05. Desarrollar las aplicaciones que integren la generación de los conjuntos de datos y los modelos de predicción.

Objetivo 06. Ampliar las aplicaciones de la IA en problemas que no se hayan podido utilizar porque no se tengan los datos suficientes.

1.3. Casos de estudio

La materialización de los objetivos planteados requiere su aplicación a una serie de casos de estudio concretos, de distinta naturaleza, que permitan demostrar la capacidad de generalización de la metodología propuesta en esta tesis. Se definen, a continuación, las tres líneas de investigación a las que se aplica, con intención de introducirlas brevemente, definir los objetivos específicos que se persiguen y resumir algunos de los avances logrados.

1.3.1. Descubrimiento de fármacos

La búsqueda de una o varias moléculas en una base de datos utilizando su geometría tiene un gran interés desde el punto de vista bioquímico, pero requiere un enorme esfuerzo computacional debido a la complejidad de los algoritmos de comparación entre objetos tridimensionales y el tamaño de las bases de datos que gestiona la industria farmacéutica. Una de las más desarrolladas y ampliamente utilizada es el Cribado Virtual (VS, *Virtual Screening*) [29,30]. Se trata de una técnica computacional utilizada para identificar aquellos compuestos que son más afines o tienen características similares a uno dado de referencia.

En el VS se utiliza mucho el DL, empleando diversos formatos de entrada, tales como el *footprint* [31], fórmulas químicas o la información de los archivos mol[2] [32]. Sin embargo, en esta línea de investigación, se ha realizado una gran aportación al generar nuevos datos de entrada. En este caso, el entrenamiento de las redes neuronales se ha realizado con cientos de imágenes de cada molécula. Estas han sido renderizadas (usando GPUs y CPUs) mediante proyecciones en planos cuyas normales están igualmente distribuidas en el espacio 3D, usando espirales de Fibonacci. Los resultados obtenidos, tanto en precisión como en tiempo de cálculo, superan las expectativas, abriendo un esperanzador camino de investigación, no sólo para la búsqueda de nuevos fármacos, sino también en el reconocimiento de objetos tridimensionales usando inteligencia artificial.

[2]Los archivos mol cuentan con una gran cantidad de información sobre los átomos y enlaces que la componen las moléculas. Incluyen las proyecciones 2D de las moléculas sobre los 3 planos cartesianos, la carga, el isótopo, etc.

1.3.2. Cómputo intensivo de datos con fuerte dependencia radial

En muchas aplicaciones, tales como la manipulación de datos hiperespectrales, la exploración de datos de resonancia magnética o la identificación de cuencas visuales en modelos digitales de elevación, es necesario realizar operaciones aritméticas sobre cada punto de una malla de datos que impliquen al resto de puntos de la misma, lo que puede resultar en un problema computacionalmente intratable. Se presenta en esta línea de investigación SkewEngine, una herramienta diseñada para mejorar el rendimiento de cálculos intensivos en mallas regulares de datos 2D o 3D, como imágenes, volúmenes de datos multiespectrales o modelos digitales de elevación. SkewEngine soluciona este problema mediante la reorganización de la malla en memoria según una dirección espacial preferente, lo que permite una mayor eficiencia en la realización de cálculos intensivos.

Se demuestra que SkewEngine ofrece una mejora significativa en la velocidad de los cálculos para una variedad de casos de prueba, lo que sugiere que puede ser una herramienta útil en una mucho más amplia gama de aplicaciones en las que se requiere procesamiento intensivo de datos en mallas regulares.

1.3.3. Detección de epilepsia

La epilepsia es una enfermedad caracterizada por la hiperexcitabilidad e hipersincronización de la actividad neuronal. Según la Organización Mundial de la Salud, alrededor de 50 millones de personas sufren de epilepsia [33] y aproximadamente el 30 % de los pacientes no responden correctamente a los tratamientos convencionales [34]. La detección precisa de las crisis epilépticas es crucial para evaluar la carga de la enfermedad, prevenir la muerte súbita inesperada en la epilepsia y mejorar la calidad de vida de los afectados.

Actualmente, la monitorización de las convulsiones se basa en las declaraciones de los pacientes y sus familias, pero esto puede ser poco confiable debido a la falta de informes, las convulsiones que se pasan por alto y las dificultades para recordarlas [35]. Si bien la videoelectroencefalografía a largo plazo es el estándar para diagnosticar y evaluar la epilepsia, las grabaciones son costosas de obtener y su análisis requiere mucho tiempo por parte del personal médico [36].

Otro reto importante en este contexto es la elevada frecuencia de muestreo de los equipos médicos. Estos dispositivos registran una gran cantidad de datos en intervalos de tiempo muy cortos, lo que aumenta considerablemente el volumen de información generada [37]. Esta sobrecarga de datos puede poner a prueba las capacidades de almacenamiento, transmisión y procesamiento de los mismos. La capacidad de registrar datos a altas frecuencias es esencial para obtener una visión precisa y detallada de la salud del paciente, pero también

requiere sistemas robustos capaces de manejar y procesar estos datos con rapidez.

Con este caso de estudio se pretende reducir esta ingente cantidad de datos mediante técnicas de filtrado y compresión. El objetivo es que puedan ser evaluados por una red neuronal que permita detectar las crisis epilépticas y clasificarlas en función de distintos parámetros biológicos. La aportación en este campo ha sido una herramienta de lectura y transformación de los datos en imágenes bidimensionales que representan las señales cerebrales de un Electroencefalograma (EEG) en ventanas de tiempo. Los resultados muestran mejoras en manipulación y el procesamiento de los datos, reduciendo el tiempo de lectura de los archivos en aproximadamente 400 veces, comprimiendo los datos hasta en un 99 % y acelerando cientos de veces los tiempos de preprocesamiento de archivos de gran tamaño cuando se utiliza la GPU.

1.4. Estructura del documento

El presente trabajo se organiza en los siguientes capítulos:

1. **Introducción**
   - Expone el contexto, la motivación y los objetivos de la investigación.
   - Describe los casos de estudio utilizados para validar las propuestas y su relevancia en la práctica.
   - Presenta la estructura general del documento para orientar al lector.
2. **Fundamento Teórico**
   - Introduce los conceptos clave relacionados con las GPUs, las redes neuronales y el aprendizaje profundo.
   - Revisa la literatura y los avances más recientes en las áreas de preprocesamiento y posprocesamiento de datos para aplicaciones de inteligencia artificial.
3. **Metodología**
   - Explica el enfoque metodológico adoptado para resolver los problemas planteados siguiendo las etapas del ciclo de vida de la ciencia de los datos.
   - Describe las fuentes de datos, las técnicas de procesamiento y los modelos propuestos.
   - Detalla el pipeline completo para las redes neuronales, desde la generación y transformación de datos hasta la validación de los modelos.
4. **Experimentación**

- Presenta los casos de estudio en detalle:
  - **Descubrimiento de fármacos**: Uso de GPUs para generar proyecciones moleculares y su uso en el DL .
  - **SkewEngine**: Herramienta para cálculos intensivos de datos con dependencia radial.
  - **Detección de epilepsia**: Implementación de técnicas avanzadas de preprocesamiento para la clasificación automática de crisis epilépticas.
- Discute los resultados obtenidos y su impacto práctico en cada caso.
- Contrasta los hallazgos con trabajos previos y discute sus implicaciones.
- Identifica las limitaciones del trabajo realizado.

5. **Conclusiones y Líneas Futuras**
   - Resume las principales aportaciones de la investigación en líneas generales.
   - Destaca el impacto de los hallazgos en la comunidad científica y en la práctica profesional de cada una de las líneas de investigación.
   - Propone posibles líneas de investigación futuras basadas en los resultados obtenidos para cada uno de los casos de estudio.
6. **Referencias** Lista completa de todas las fuentes citadas en la tesis.

Capítulo 2

Fundamento teórico

El presente capítulo proporciona los conceptos y bases necesarias para entender las tecnologías y metodologías empleadas a lo largo de esta tesis. Se abordan, por un lado, las características, evolución, arquitectura y programación de las Unidades de Procesamiento Gráfico (GPUs), cuya configuración masivamente paralela resulta esencial para optimizar tareas de alto coste computacional. Por otro lado, se describen los principios de las redes neuronales y el DL , destacando su rol central en la implementación de modelos avanzados de IA.

2.1. Unidades de procesamiento gráfico (GPUs)

2.1.1. Definición y características

Las Unidades de Procesamiento Gráfico (GPUs) son dispositivos especializados diseñados originalmente para acelerar el procesamiento de gráficos en aplicaciones como videojuegos y diseño 3D. Su arquitectura permite realizar un gran número de cálculos de manera simultánea, lo que las hace ideales para tareas altamente paralelizables como el ML, simulaciones físicas y análisis de datos masivos.

Entre sus características principales cabe mencionar:

- **Escalabilidad**: Gracias a su arquitectura masivamente paralela, compuesta por miles de núcleos capaces de ejecutar múltiples hilos en paralelo.
- **Alta eficiencia energética**: Logran un rendimiento excepcional en comparación

con las CPUs para tareas específicas.

- **Productividad:** Se habilitan multitud de mecanismos para que cuando un hilo, o *thread* en inglés, pase a realizar operaciones que no permitan su ejecución veloz, otro oculte su latencia tomando el procesador de forma inmediata.
- **Optimización para matrices**, especialmente diseñadas para operaciones como multiplicaciones matriciales, comunes en gráficos y ML.

Si se compara con la CPU, se observa que esta es más eficiente en tareas secuenciales o dependientes, mientras que la GPU es especialmente útil para ejecutar miles de cálculos de manera simultánea. En la Tabla 2.1 se recogen las principales diferencias que explican la diferencia de rendimiento en las distintas tareas.

Tabla 2.1: Diferencias principales entre CPU y GPU.

	CPU	**GPU**
Arquitectura	Menos núcleos, pero más potentes y optimizados para tareas secuenciales.	Miles de núcleos más pequeños, optimizados para tareas paralelas.
Finalidad	Generalista: ideal para un amplio rango de tareas.	Especialista: ideal para cálculos masivos paralelizables.
Latencia	Diseñada para baja latencia en operaciones secuenciales.	Diseñada para alta capacidad de *throughput* (ancho de banda computacional).
Uso principal	Ejecución de sistemas operativos, aplicaciones generales, gestión de recursos.	Procesamiento gráfico, ML, simulaciones científicas.

2.1.2. Historia y evolución de las GPUs

Las GPUs han experimentado una transformación significativa desde su concepción inicial, como procesadores especializados en gráficos, hasta convertirse en un componente clave en la computación moderna, particularmente en inteligencia artificial, tratamiento de datos y simulaciones. Su evolución puede dividirse en varias etapas clave.

Los inicios: Aceleración de gráficos

En las **décadas de 1980 y 1990**, las GPUs surgieron como tarjetas gráficas diseñadas para acelerar el procesamiento gráfico en aplicaciones multimedia y videojuegos [38]. Estas primeras generaciones estaban enfocadas en la renderización en 2D y 3D. Por ejemplo, dispositivos como la *S3 Graphics ViRGE* y la *ATI Rage* fueron pioneros en esta etapa.

En **1999**, NVIDIA introdujo la *GeForce 256*, considerada la primera GPU moderna. Supuso un gran avance al incorporar 23 millones de transistores, 32 MB de DRAM de 128 bits y un reloj central de 120 MHz. Este dispositivo integró capacidades como la transformación e iluminación ($T\&L$) y el sombreado acelerado por hardware, marcando el inicio del término "GPU" [39].

La GPU programable: Computación gráfica avanzada

Entre 2001 y 2006, las GPUs se hicieron más flexibles con la introducción de *shaders* programables, lo que permitió a los desarrolladores diseñar efectos gráficos personalizados. El lanzamiento de *DirectX 9* por parte de Microsoft introdujo nuevas funcionalidades de sombreado, consolidando el uso de lenguajes de programación como HLSL y GLSL [40]. Modelos como la *GeForce FX* de NVIDIA y la *Radeon 9700* de ATI son ejemplos de esta etapa.

GPU para computación general: Nacimiento del GPGPU

El año **2006** marcó un hito con la introducción de CUDA (*Compute Unified Device Architecture*) por parte de NVIDIA, permitiendo que las GPUs se usaran para tareas científicas y técnicas más allá de los gráficos [41]. Esto dio lugar al concepto de computación general en GPUs (*General-Purpose computing on Graphics Processing Units, GPGPU*) [42]. Posteriormente, en **2008**, surgió el estándar abierto **OpenCL**, desarrollado por Khronos Group, permitiendo la programación heterogénea en diferentes arquitecturas.

GPUs en la era del Deep Learning

En **2012**, las GPUs jugaron un papel crucial en el desarrollo del aprendizaje profundo. Un momento clave fue el uso de GPUs NVIDIA para entrenar el modelo AlexNet, que ganó la competencia ImageNet, demostrando cómo las GPUs reducían significativamente los tiempos de entrenamiento en redes neuronales profundas [43].

Desde entonces, arquitecturas como *NVIDIA Volta* y sus *Tensor Cores*, optimizadas para operaciones matriciales esenciales, han consolidado el papel de las GPUs en aplicaciones de inteligencia artificial [44].

GPUs modernas y su impacto actual

En los últimos años, GPUs avanzadas como la *NVIDIA A100* (**2020**) y la *AMD Instinct MI200* (**2021**) fueron diseñadas para satisfacer demandas crecientes en computación de alto rendimiento (HPC) e inteligencia artificial. Más recientemente, NVIDIA ha lanzado la serie *GeForce RTX 50*, destacando modelos como la *RTX 5090* y la *RTX 5080*, que incorporan la arquitectura *Blackwell* y memoria GDDR7, ofreciendo un rendimiento significativamente superior a los de generaciones anteriores [45]. Por su parte, AMD ha anunciado la serie *Radeon RX 9000*, basada en la arquitectura RDNA 4, con mejoras en trazado de rayos y capacidades de IA [46]. La aparición de modelos masivos como GPT y las aplicaciones generativas han reforzado la posición de las GPUs como tecnologías indispensables para el procesamiento masivo de datos.

Impacto y perspectivas futuras

Con el avance de la inteligencia artificial, la computación en la nube y las simulaciones científicas, el papel de las GPUs continúa expandiéndose. Además, áreas emergentes como la computación cuántica híbrida y las simulaciones biológicas están comenzando a aprovechar la capacidad de cómputo masivo de las GPUs .

2.1.3. Arquitectura de la GPU

Las GPUs están compuestas por múltiples ***Streaming Multiprocessors*** (SMs). Cada SM incluye:

- ***Cores***: Procesadores capaces de realizar operaciones básicas de números en punto flotante, números enteros o tensores (estructuras numéricas multidimensionales).
- **Unidades de memoria**: La memoria de una GPU se organiza en varios niveles, optimizados para diferentes velocidades de acceso y tamaños de almacenamiento:
  - Memoria global: Accesible por todos los SMs, pero con alta latencia. Suele ser tres veces más rápida que la memoria principal de la CPU.
  - Memoria compartida: Permiten el intercambio de datos entre los *cores* dentro del SM. Es unas 500 veces más rápida que la global, pero limitada a cada SM.
  - Registros: Memoria más rápida, pero accesible únicamente por el correspondiente *core*.

- **Unidades de control SIMD (*Single Instruction, Multiple Data)***: Gestionan grupos de hilos (*threads*) llamados *warps* (32 *threads* en GPUs NVIDIA).
- **Unidades especializadas**: Algunas GPUs modernas incluyen unidades para tareas específicas, como *cores* optimizados para operaciones matriciales (*Tensor Cores*) o para trazado de rayos en tiempo real (*Ray Tracing Cores*).

Los *threads* son las unidades más pequeñas de ejecución en una GPU. Cada *thread* se encarga de procesar un dato o un conjunto de datos. Todos los hilos de un *warp* deben ejecutar la misma instrucción en un momento dado. Existe una jerarquía de *threads* según la configuración del paralelismo:

- *Threads*: Unidades básicas de ejecución creadas en un tiempo muy efímero.
- *Warps*: Grupos de 32 *threads*.
- *Blocks*: Conjuntos de *warps* que comparten memoria.
- *Grids*: Colección de bloques que define la escala de paralelismo.

La Figura 2.1 muestra de manera esquemática cómo se compone la GPU a diferentes niveles (a) y cómo se agrupan los *threads* para un procesamiento SIMD (b).

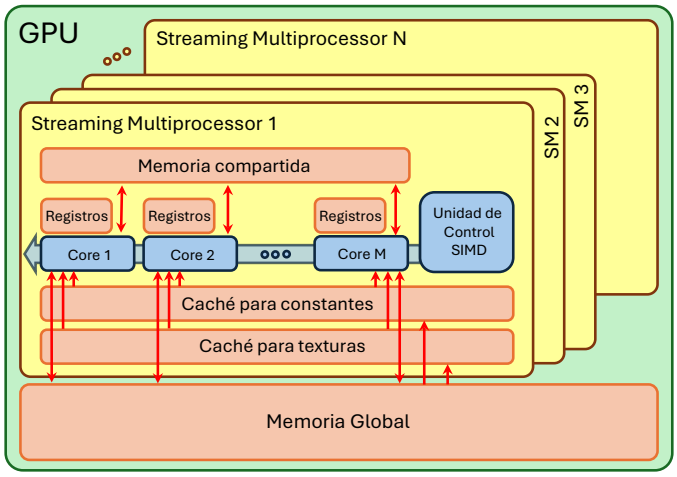
2.1.4. Modelos de programación

La programación en GPUs es clave para aprovechar al máximo su potencial de paralelismo y optimizar la transferencia de datos entre CPU y GPU para evitar cuellos de botella. Dos de los principales modelos son CUDA [47], desarrollado por NVIDIA , y OpenCL [48], un estándar abierto. Además, existen diversas librerías y *frameworks* diseñados para tareas específicas.

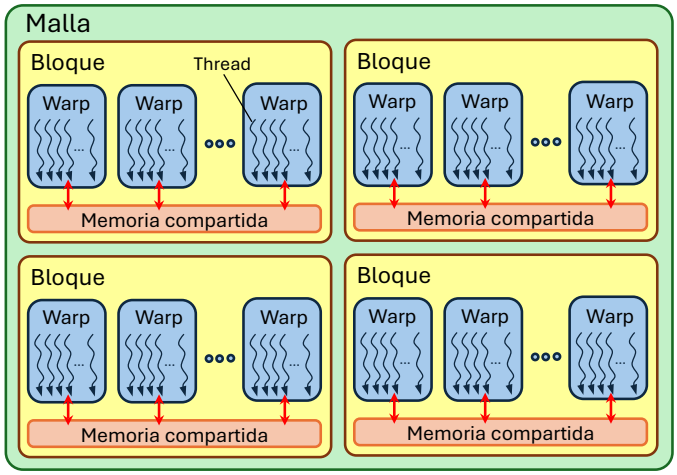
CUDA (*Compute Unified Device Architecture*)

CUDA [47] es una plataforma desarrollada por NVIDIA para aprovechar a tres niveles la potencia de una GPU en aplicaciones de propósito general:

- **Software**: Permite programar la GPU con mínimas pero potentes extensiones SIMD para lograr una ejecución eficiente y escalable.



(a) Arquitectura



(b) Jerarquía de *threads*

Figura 2.1: Esquema de la arquitectura de la GPU y jerarquía de *threads*.

- **Firmware**: Ofrece un driver para la programación GPGPU que es compatible con el que se utiliza para renderizar. Mediante el uso de sencillas APIs se manejan los dispositivos, la memoria, etc.
- **Hardware**: Habilita el paralelismo de la GPU para programación de propósito general a través de un número de multiprocesadores gemelos dotados de un conjunto de núcleos computacionales arropados por una jerarquía de memoria, como se ha definido en el Apartado 2.1.3.

Esta plataforma ofrece un rendimiento optimizado para hardware de NVIDIA , un soporte robusto con actualizaciones frecuentes, documentación amplia y es compatibles con varios lenguajes de programación como C, C++, Python (a través de PyCUDA), y Fortran.

OpenCL (*Open Computing Language*)

OpenCL [48] es un estándar abierto y multiplataforma para programación en dispositivos heterogéneos como CPUs, GPUs, FPGAs y otros aceleradores. Desarrollado por el grupo Khronos, OpenCL busca ofrecer flexibilidad y portabilidad. Entre sus características principales, cabe destacar:

- **Independencia de hardware**: OpenCL permite escribir código que puede ejecutarse en múltiples dispositivos, incluyendo GPUs de diferentes fabricantes (NVIDIA, AMD, Intel...).
- **Portabilidad**: Un único código puede adaptarse a varias plataformas con mínimas modificaciones.
- **Uso en entornos diversos**: Es ideal para aplicaciones que requieren compatibilidad con hardware variado, como supercomputación y sistemas embebidos.

Sin embargo, su programación es más compleja y menos intuitiva con respecto a CUDA.

Librerías y *frameworks* populares

Los desarrolladores pueden apoyarse en diversas herramientas para simplificar la programación en GPUs. Algunas de las que se han empleado en esta tesis son:

- **cuDNN** [49]: Librería desarrollada por NVIDIA para operaciones comunes en redes neuronales profundas, como convoluciones, normalización y funciones de activación. Mejora la velocidad de entrenamiento y optimiza la construcción de modelos.
- **TensorRT** [50]: Herramienta de NVIDIA para la construcción de los modelos con el fin reducir la latencia y el consumo energético. Es compatible con modelos entrenados en TensorFlow, PyTorch y ONNX.
- **PyCUDA y PyOpenCL** [51]: Extensiones de Python para interactuar con GPUs mediante CUDA y OpenCL, respectivamente. Permiten escribir código más accesible para aplicaciones científicas y prototipos rápidos.

2.1.5. Limitaciones y desafíos del uso

Se ha visto que el uso de GPUs en tareas de cálculo intensivo y en inteligencia artificial ofrece importantes ventajas gracias a su capacidad de paralelización masiva, pero también

presenta limitaciones y desafíos que deben tenerse en cuenta.

Entre las principales limitaciones del **hardware**, destaca la memoria limitada de las GPUs (de 8 a 48 GB en modelos comerciales) en comparación con la memoria principal de las CPUs, lo que puede dificultar el manejo de grandes volúmenes de datos. Además, la transferencia de datos entre la CPU y la GPU, generalmente a través de un bus PCIe, resulta ser un cuello de botella en el rendimiento [52]. Debido a que el coste de las transferencias es un punto crítico en las aplicaciones, solo es recomendable el uso de GPUs a partir de 10 millones de datos.[1] A esto se suman desafíos térmicos, que limitan el rendimiento prolongado en el tiempo, y el elevado coste tanto en hardware como en consumo energético.

Desde el punto de vista del **software**, programar para GPUs requiere conocimientos avanzados en tecnologías como CUDA o OpenCL, ya que, para sacar un máximo rendimiento, se deben aprovechar los recursos de la memoria compartida y los registros frente a la memoria global, bastante más lenta. Además, algunos algoritmos, debido a sus características secuenciales y por la dependencia de los datos, no son fácilmente paralelizables. Esto limita el uso de GPUs en ciertas tareas.

2.2. Redes neuronales y Deep Learning

En primer lugar es importante situar el *Deep Learning* dentro del marco del *Machine Learning*, que a su vez es solo una subconjunto de la Inteligencia Artificial [53]. En estos momentos, es quizás la rama más dinámica que está haciendo que la IA esté en pleno auge.

Con el objetivo de situar estos campos, se muestra a continuación una aproximación simple [54] de cada uno de estos conceptos.

La **Inteligencia Artificial (IA)** se refiere al esfuerzo de automatizar tareas intelectuales normalmente realizadas por humanos, representando la inteligencia que muestran las máquinas en contraste con la inteligencia natural de los humanos.

El ***Machine Learning*** **(ML)** es un subcampo de la Inteligencia Artificial, permite a las máquinas aprender sin programación explícita, encontrando patrones en los datos para construir algoritmos de predicción específicos para problemas concretos.

El ***Deep Learning*** **(DL)** es un caso especial de ML, donde se emplean algoritmos basados en las neuronas humanas (redes neuronales artificiales). Usan múltiples capas de

[1] Es bastante pedagógica la comparación entre la GPU para una pequeña aplicación y un vuelo comercial para ir de Málaga a Granada.

procesamiento para aprender representaciones de los datos, realizando transformaciones, tanto lineales como no lineales, a partir de los datos de entrada para obtener una salida cercana a la deseada. Dependiendo de la categoría, esta salida deseada consistirá en etiquetas, un determinado patrón, o una acción considerada como satisfactoria para lograr un objetivo. El aprendizaje consiste en ajustar los parámetros de estas transformaciones para que la salida predicha sea lo más próxima posible a la salida esperada.

2.2.1. Neurona artificial simple

Una neurona artificial simple [55] es el componente fundamental de las redes neuronales artificiales y actúa como una unidad de procesamiento que realiza operaciones matemáticas en los datos de entrada. También es conocida comúnmente por el nombre de Perceptrón [56]. Una neurona recibe una serie de datos de entrada $\bar{X}$ y realiza una combinación algebraica de los mismos para producir una salida $\hat{Y}$, pasando previamente dicha combinación por una función de activación f que se encargará de "apagar o encender" dicha neurona.

La neurona cuenta con conjunto de pesos $\bar{W} = (w_1, w_2, \ldots, w_M)$ y un sesgo común b donde, ante una serie de M datos de entrada $\bar{X} = (x_1, x_2, \ldots, x_M)$, se establece la siguiente relación algebraica:

$$z = \sum_{i=1}^{M} w_i \cdot x_i + b = \bar{W}^T \cdot \bar{X} + b. \tag{2.1}$$

Una vez realizada esta combinación, la neurona producirá la salida $\hat{y}$ tras aplicar la función de activación a z,

$$\hat{y} = f(z). \tag{2.2}$$

La función de activación introduce no linealidad en la salida de la neurona artificial, actuando como un interruptor que decide cuándo enviar una señal. Esto permite a la neurona capturar relaciones complejas en los datos de entrada. En problemas de regresión, se utilizan diferentes tipos de funciones de activación. Por mencionar algunas de las más utilizadas:

- **Lineal:** En algunas ocasiones, no interesa modificar la señal de salida de la neurona. Su ecuación es:

$$f(z) = \text{lineal}(z) = z. \tag{2.3}$$

- **Sigmoid:** Es útil cuando queremos que la salida de la neurona esté en un rango entre 0 y 1. Reduce los valores atípicos extremadamente grandes o pequeños sin eliminarlos. Su ecuación es:

$$f(z) = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}. \tag{2.4}$$

- **Tanh:** La función tangente hiperbólica (tanh) también limita la salida de la neurona, pero en un rango entre -1 y 1. Es útil cuando queremos que la salida sea simétrica alrededor de cero. Su ecuación es:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.5}$$

- **ReLU (*Rectified Linear Unit*):** ReLU es simple y ampliamente utilizada. Si la entrada es positiva, la neurona se activa; de lo contrario, no. Su ecuación es:

$$f(z) = \text{ReLu}(z) = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}. \tag{2.6}$$

En la Figura 2.2, se representan las señales de activación, mientras que la Figura 2.3 muestra el funcionamiento del perceptrón según se ha descrito.

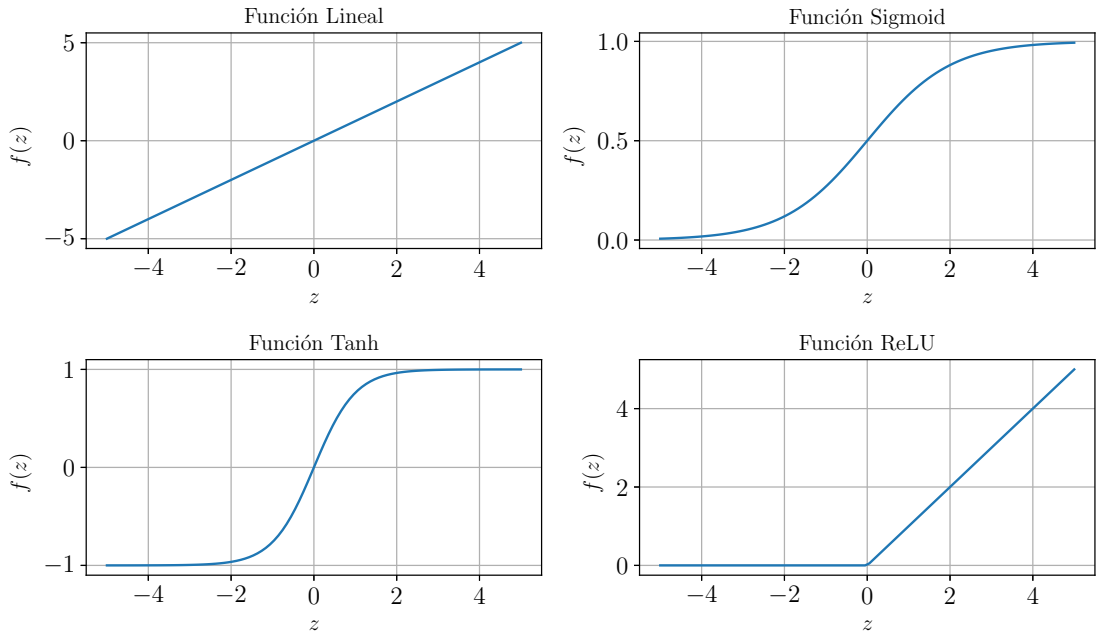


Figura 2.2: Representación de las distintas señales producidas por diferentes funciones de activación.

2.2.2. Arquitecturas de redes neuronales

Una red neuronal, por tanto, es el resultado de una superposición de capas de neuronas interconectadas. Una determinada capa está formada por N neuronas que utilizan todos o

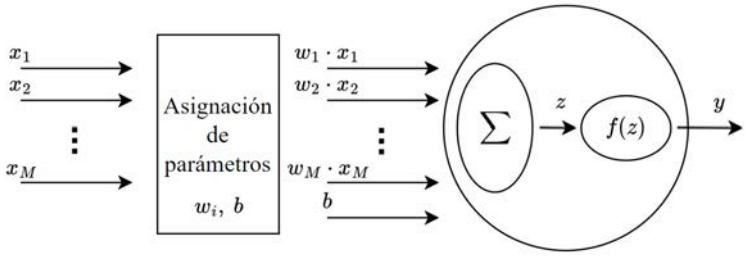


Figura 2.3: Esquema del funcionamiento de una neurona artificial simple.

algunos de los M datos de entrada para generar cada una de sus salidas $\hat{y_j}$, es decir,

$$\hat{y_j} = f\left(\sum_{i=1}^{M} w_{i,j} \cdot x_i + b_j\right) = f\left(\bar{W}_j^T \cdot \bar{X} + b_j\right) \quad \forall\ j = 0, 1, \ldots, N-1. \tag{2.7}$$

De esta manera, las salidas de una capa k, serán utilizadas como entradas de la siguiente capa $k+1$,

$$\hat{\bar{Y}}^{(k)} \equiv \bar{X}^{(k+1)} \tag{2.8}$$

La Figura 2.4 muestra la estructura de un modelo que puede predecir P características diferentes utilizando M características de entrada. Según cómo se conecten las distintas capas entre sí, es decir, la cantidad de datos que son utilizados como entrada a cada una de las neuronas, se define la arquitectura de la red. Cada arquitectura tiene sus ventajas e inconvenientes dependiendo del tipo de datos que procesa y la naturaleza del problema. Desde las más simples, como las densamente conectadas (DNN) [57] hasta las más complejas, como las *Transformers* [58] y las Redes Generativas Antagónicas (GANs, por sus siglas en inglés) [59], cada una de estas arquitecturas ha sido diseñada para resolver un conjunto específico de problemas o mejorar el rendimiento en tareas específicas. Si bien se podría entrar en detalle en las múltiples arquitecturas que se han empleado hasta ahora [60], a continuación únicamente se presentan tres de las más simples, pero que resultan ser suficientes y adecuadas para demostrar los avances que se logran en esta tesis. Además, son utilizadas como base para otras arquitecturas más complejas.

Redes neuronales densamente conectadas (DNNs)

En este tipo de redes, todas las neuronas de una capa k están conectadas a través de los pesos $\bar{\bar{W}}^{(k)}$ con la capa anterior y conectadas con las neuronas de la siguiente mediante los pesos $\bar{\bar{W}}^{(k+1)}$. Es decir, el tamaño de $\bar{\bar{W}}^{(k)}$ vendrá marcado por el número de neuronas de la capa anterior y de la capa k. Por este motivo, $\bar{\bar{W}}^{(k)}$ adquiere notación de tensor de dos dimensiones. Estos serán todos los parámetros entrenables de esa capa, junto con el vector de sesgos correspondiente $\bar{B}^{(k)}$.

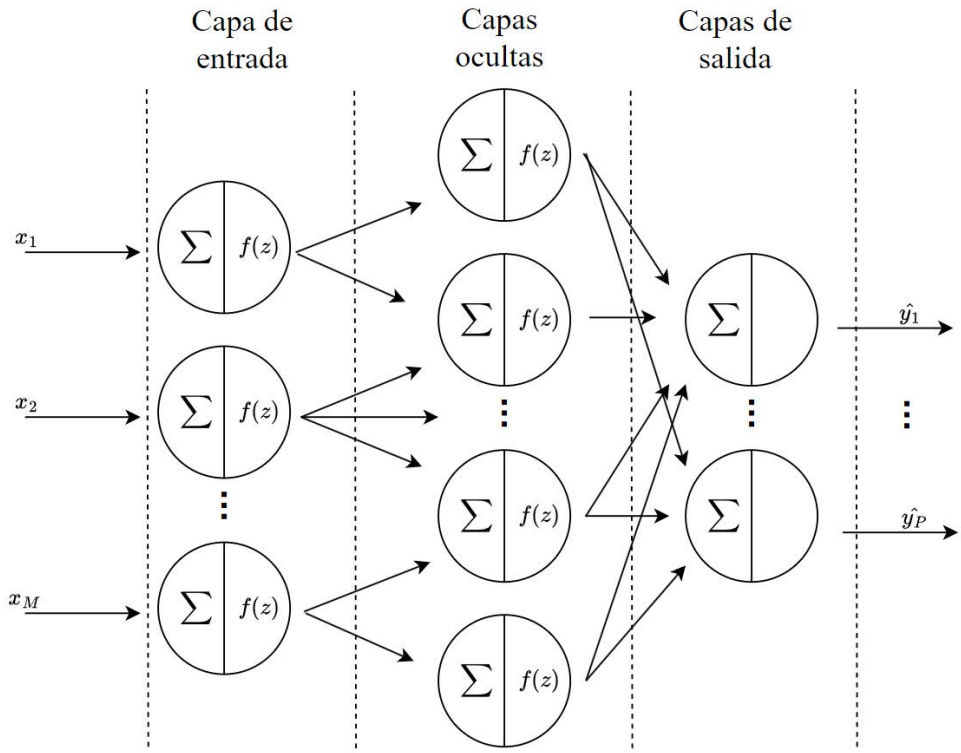


Figura 2.4: Esquema del funcionamiento de una red neuronal.

La función de activación más usada en este tipo de arquitecturas, salvo para la capa de salida, es la función "ReLu" de la ecuación (2.6).

Los datos de entrada pueden venir estructurados en varias dimensiones, pero es conveniente alinearlos para tenerlos en forma de vector para este tipo de arquitecturas,

$$\mathbf{X} \to \bar{X} = \text{flatten}(\mathbf{X}). \tag{2.9}$$

Al hacer esto y al considerar todas las posibles conexiones entre neuronas, este tipo de redes presentan como inconveniente la pérdida de información sobre la estructura espacial de los datos. En la Figura 2.5 se observa, de una manera intuitiva, la cantidad de parámetros que presentan de estas redes.

Redes neuronales convolucionales (CNNs)

Las CNN muestran muchas similitudes con las DNN. Están formadas por neuronas que tienen parámetros como pesos y sesgos. La principal diferencia es que están diseñadas

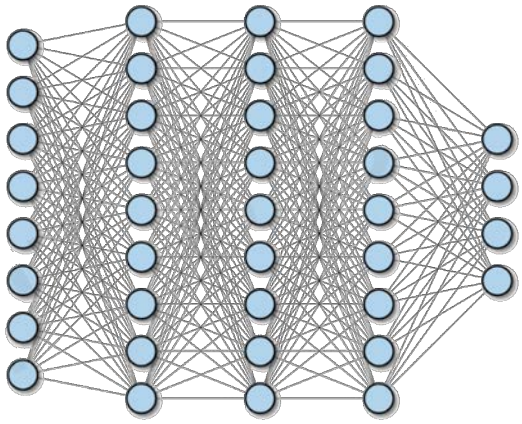


Figura 2.5: Representación de una DNN.

principalmente para el procesamiento de imágenes o datos con una dependencia espacial,[2] como imágenes en dos dimensiones, donde el color de un píxel depende enormemente de los píxeles de alrededor.

Las CNNs son capaces de encontrar relaciones en datos que son consistentes, sin importar su localidad espacial dentro en una imagen o tensor. La Figura 2.6 muestra cómo se realizan las conexiones entre una capa y la siguiente para detectar ciertos patrones. Se observa que el procedimiento es conectar un bloque de neuronas a una única de la capa siguiente, usando una ventana deslizante que define los valores de dichas conexiones, es decir, los pesos. A esta ventana se le conoce como filtro y esos mismos pesos se aplican a cada una de las neuronas de la capa de entrada. El resultado para una neurona (p, q) de la capa de salida, es la operación de convolución (Figura 2.6a):

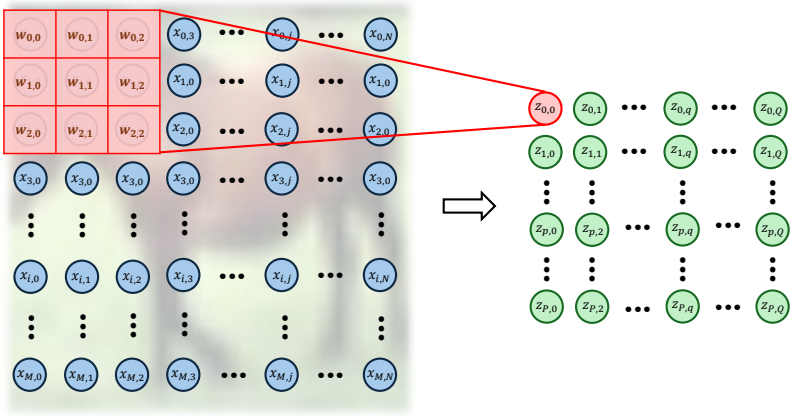
$$z_{p,q} = \sum_{i=1}^{M_f} \sum_{j=1}^{N_f} w_{i,j}\, x_{i+s\cdot p, j+s\cdot q} + b_{p,q}, \tag{2.10}$$

usando un filtro de tamaño $M_f \times N_f$ que se desliza con un paso de s neuronas ($s = 1$ en el caso de la Figura 2.6b) y añadiendo un sesgo $b_{p,q}$. A continuación, se le aplica una función de activación, generalmente ReLu, para introducir la no linearidad. Se hace de una manera similar a las DNNs.

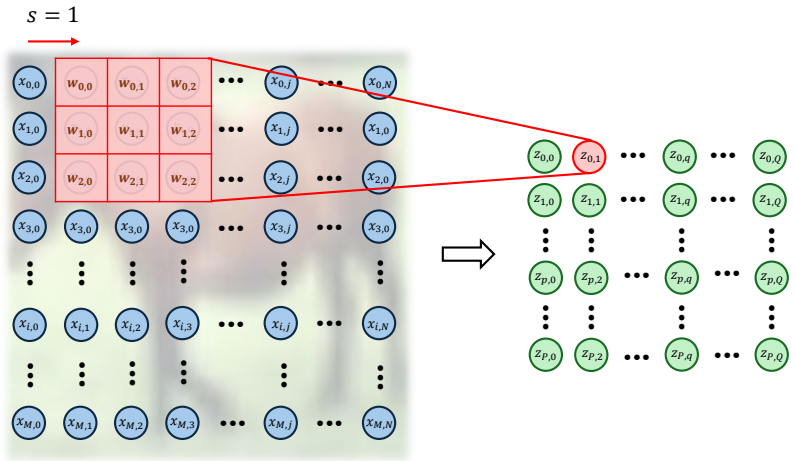
Para extraer distintos patrones o características, se pueden emplear múltiples filtros (Figura 2.6c). Por tanto, la salida de la neurona (p, q) asociada a un filtro f, resulta:

$$\hat{y}_{p,q}^{(f)} = g\left(\sum_{i=1}^{M_f} \sum_{j=1}^{N_f} w_{i,j}^{(f)}\, x_{i+s\cdot p, j+s\cdot q} + b_{p,q} \right), \tag{2.11}$$

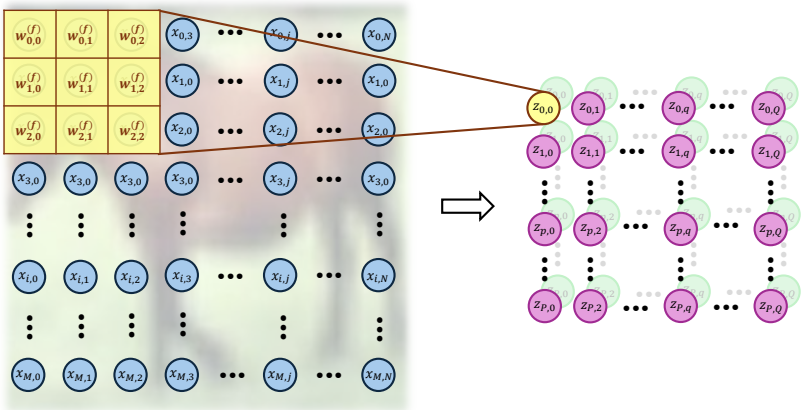
[2]La dependencia espacial se refiere a la posición dentro del tensor de datos, por lo que sería también aplicable a series temporales.



(a) Operación de convolución



(b) Se desliza el mismo filtro con un determinado paso s.



(c) Se pueden aplicar varios filtros a los datos de entrada.

Figura 2.6: Convolución de filtros de pesos en las CNN.

siendo g la nomenclatura utilizada para la función de activación elegida, para diferenciarla de la del filtro.

Se observa que, aunque tras aplicar la operación de convolución se reduce el número de neuronas de salida, si se aplican F filtros el tamaño de la capa de salida puede aumentar considerablemente ($P \times Q \times F$). Por consiguiente, es habitual aplicar una operación de *Pooling* tras las capas convolucionales para reducir las dimensiones. Un tipo de *Pooling* muy utilizado consiste en quedarse con el mayor valor de un pequeño grupo de neuronas. De igual forma que con el filtro, se debe definir el tamaño de la operación y el paso s con el que se aplica. Con este método, se logra reducir enormemente el número de parámetros de la red y se proporciona información espacial para las siguientes capas.

El procedimiento anteriormente descrito puede ser extrapolado a el caso unidimensional o tridimensional, así como a varios canales. Como canales se entienden las distintas características asociadas a un elemento del tensor en concreto, como podrían ser los canales RGB de una imagen o las imágenes resultantes de los filtros que se habían aplicado a una capa anterior. A modo de ejemplo, en la Figura 2.7 se representa una capa de convolución unidimensional para una serie temporal con varias características i. En ella, se ve cómo se aplica un filtro de tamaño $N_f = 3$ a los instantes de tiempo t_1, t_2 y t_3, siendo el mismo filtro para todas las características i. Tras aplicar el sesgo (*bias*) $b = 1$, se obtiene el resultado de la convolución para el tiempo central t_2.

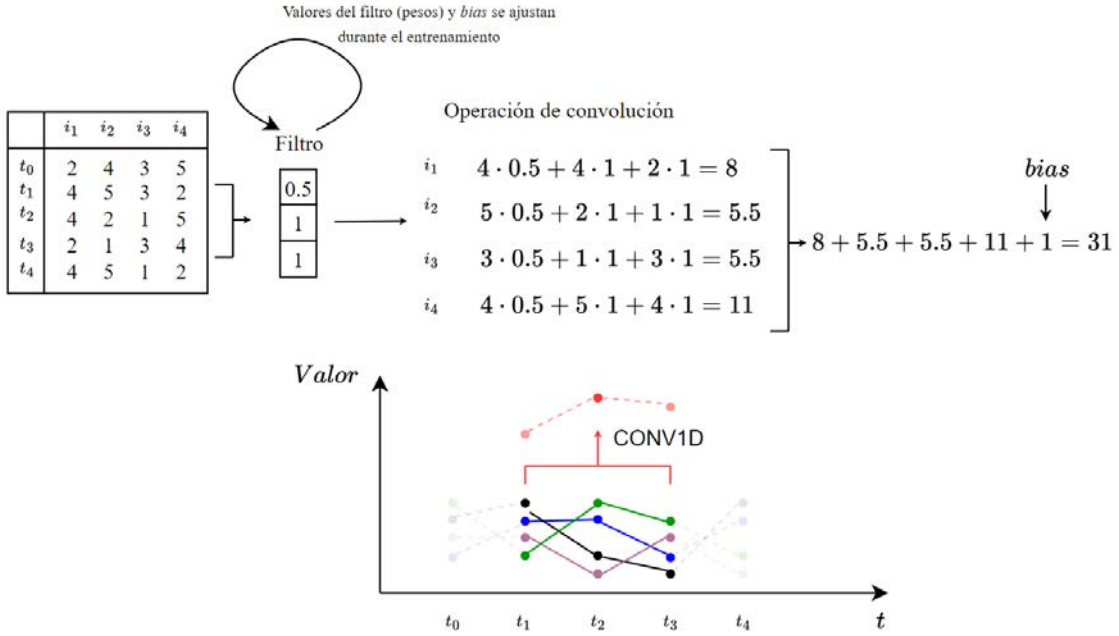
Redes neuronales *autoencoders* (ANNs)

Los *autoencoders* [61] son la base de los modelos generativos de DL. Se tratan de redes neuronales que tienen por objetivo generar nuevos datos de manera artificial, primero comprimiendo la entrada en un espacio de variables latentes y luego reconstruyendo la salida en base a la información subyacente adquirida.

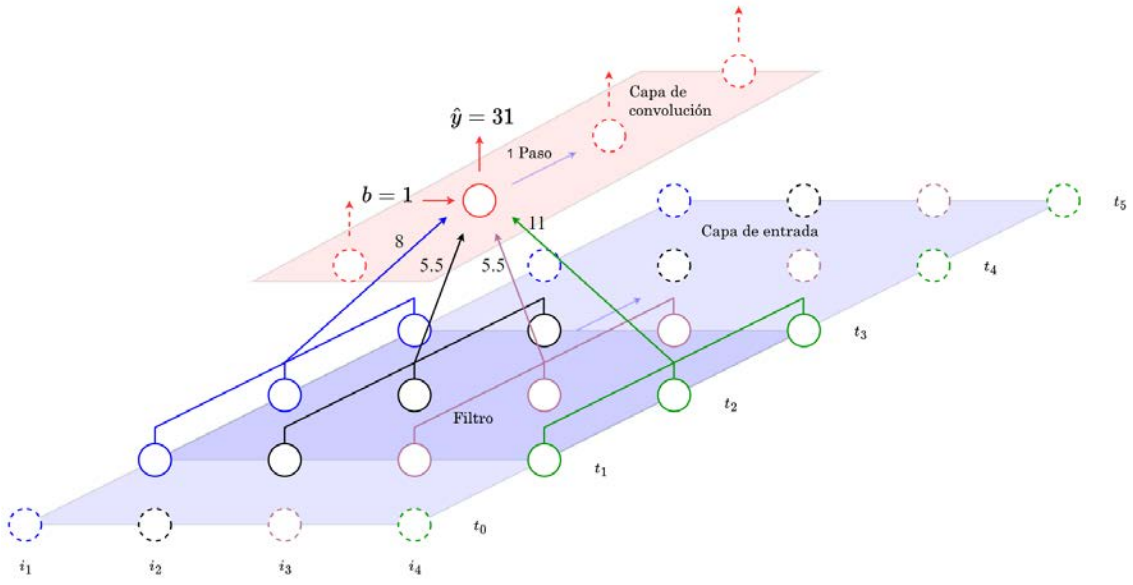
Se definen como variables latentes a aquellas características más generales que se pueden deducir tras observar varios datos concretos y distintos. Por ejemplo, tras haber visto varias imágenes de personas diferentes, se deducen algunas características diferenciadoras a nivel global, como podría ser el sexo, el tono de piel, el color del pelo o la edad.

Por tanto, esta arquitectura puede emplearse para capturar las características más relevantes de los datos de entrada y obtener proyecciones de los mismos en subespacios de mayor interés. Por mencionar algunas de las aplicaciones más comunes: generación artificial de rostros de personas que no existen, detección de valores atípicos o *clustering*.

Se basan en una estructura de codificador-decodificador, según se muestra en la Figura



(a) Operación de convolución 1D



(b) Representación de una capa de convolución 1D en una red neuronal.

Figura 2.7: Representación del proceso de convolución en redes de predicción de series temporales multivariables.

2.8. Las neuronas de la capa de entrada $\bar{X}$ se conectan a capas progresivamente más pequeñas (codificador ϕ) hasta llegar a una capa muy reducida, el espacio de variables latentes $\bar{Z}$. A continuación, se reconstruye de nuevo la información aumentando progresivamente el número de neuronas de cada capa (decodificador θ). La estructura del decodificador debe ser la inversa del codificador. La salida de la red $\hat{\bar{X}}$ son los datos artificiales generados. Las conexiones pueden densas, parciales o convolucionales.

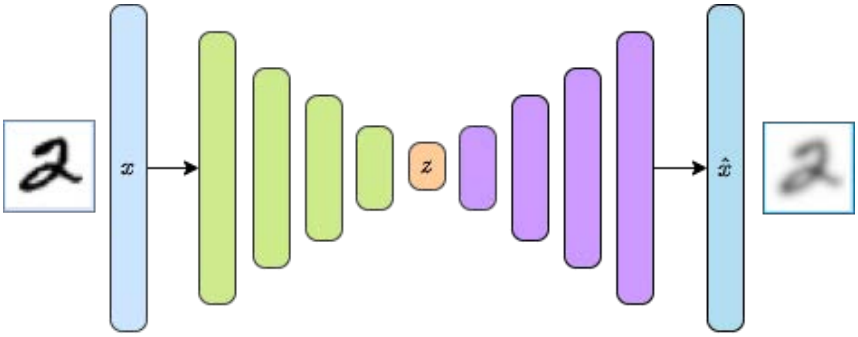


Figura 2.8: Representación de una ANN.

Este tipo de redes tuvo una gran repercusión tras introducir un enfoque probabilista al espacio latente [62]. La mejora consiste en muestrear a partir de una media μ y una desviación típica σ para calcular el espacio latente,

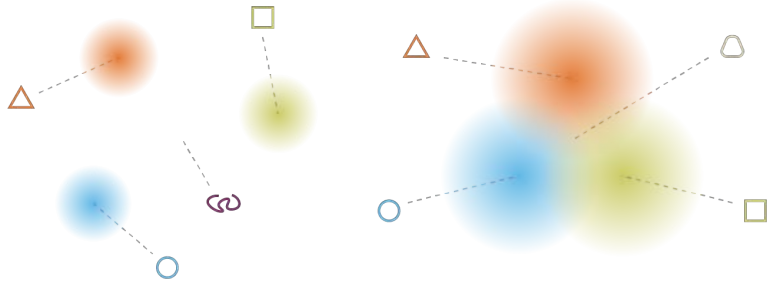
$$z \sim \mathcal{N}(\mu, \sigma^2), \tag{2.12}$$

donde $\mathcal{N}$ representa la distribución normal.

Así, el codificador calcula una probabilidad $q_\phi\big(\bar{Z}|\bar{X}\big)$, mientras que el decodificador calcula $p_\theta\big(\hat{\bar{X}}|\bar{Z}\big)$. Esto incita al codificador a crear distribuciones alrededor del centro del espacio latente, penalizando a la red si trata de agrupar puntos en regiones específicas del mismo, por ejemplo, memorizando los datos. Con esta regularización, además, se introducen dos propiedades clave en el espacio latente: **continuidad**, haciendo que dos puntos próximos en el espacio latente sean similares tras la decodificación, e **integridad**, haciendo que las muestras obtenidas del espacio latente tengan un sentido significativo. La Figura 2.9 representa gráficamente estas propiedades descritas.

2.2.3. Entrenamiento de una red neuronal

En el proceso de configuración y evaluación de modelos en ML, generalmente se dividen los datos en tres conjuntos: entrenamiento, validación y prueba. En la Figura 2.10 se observa un esquema de la división de los datos y el propósito de la misma [63].



(a) Espacio latente no regularizado (b) Espacio latente regularizado

Figura 2.9: Propiedades de continuidad e integridad que se obtienen al introducir el enfoque probabilista del espacio latente.

- **Datos de Entrenamiento:** se utilizan para ajustar los parámetros del modelo. Son los datos con los que el modelo ha sido entrenado y que conoce a fondo, ya que ha aprendido de ellos.
- **Datos de Validación:** se utilizan las métricas para evaluar el desempeño del modelo ante datos de entrada que no conoce. Se emplean para ajustar los hiperparámetros, propios del algoritmo de aprendizaje o la estructura, antes de repetir el proceso de entrenamiento.
- **Datos de Prueba:** se reservan exclusivamente para evaluar el rendimiento final del modelo después de haber sido entrenado y validado.

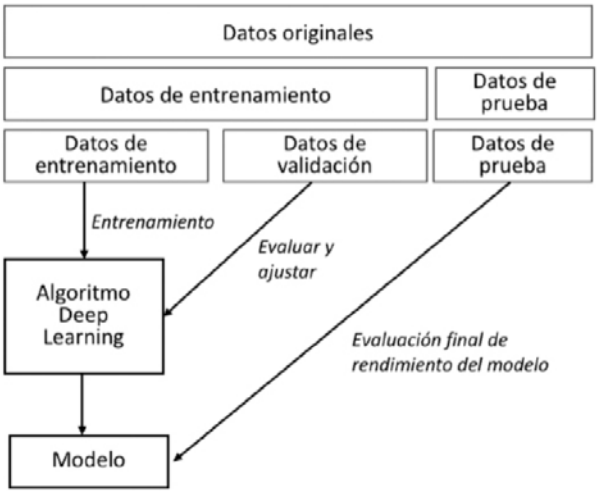


Figura 2.10: Esquema de la segmentación de los datos y el propósito de cada uno de ellos [63].

Se distinguen principalmente tres tipos de aprendizaje, dependiendo del problema a resolver y la naturaleza de los datos, pero en todos hay una serie de pasos comunes [63]:

1. Inicializar los valores (normalmente de manera aleatoria) de los pesos y el sesgos.

2. Introducir en la red un conjunto de datos de entrada (lotes) y pasarlos a la red para obtener su predicción.

3. Calcular el error cometido mediante la función de pérdida con respecto a una salida esperada.

4. Propagar hacia atrás el error para que llegue a cada uno de los parámetros que componen la red neuronal.

5. Usar esta información propagada con un algoritmo de descenso de gradiente para actualizar los parámetros de la red neuronal buscando reducir el error.

6. Continuar iterando los siguientes pasos, hasta que el modelo sea adecuado para resolver el problema de aplicación.

Aprendizaje supervisado

Este tipo de entrenamiento se caracteriza en que los datos incluyen la soluciones deseadas, denominadas etiquetas $\bar{Y}$, que son comparadas con las salidas generadas por la red $\hat{\bar{Y}}$.

Así, dada una red neuronal definida por sus pesos y sesgos $\mathbf{W}$ y un determinado dato de entrada $\bar{X}^{(i)}$ con su correspondiente etiqueta $\bar{Y}^{(i)}$, se define la función de pérdida o error como

$$\mathcal{L}\left(f(\bar{X}^{(i)};\, \mathbf{W}),\, \bar{Y}^{(i)}\right), \tag{2.13}$$

y será el parámetro que mide cómo de incorrecta ha sido la predicción realizada por la red. Existen numerosas funciones de pérdida según la aplicación deseada. Por ejemplo, las regresiones multivariables suelen utilizar el Error Cuadrático Medio (MSE) y los problemas de clasificación, el Error de Entropía Cruzada. Las funciones de pérdida que se han usado en esta tesis se definirán en sus correspondientes casos de estudio.

La función de pérdida suele promediarse para un conjunto de datos de entrenamiento con el fin asegurar la estabilidad del aprendizaje. Sin embargo, promediar con todo el conjunto de datos $\mathbf{X}$ puede ser computacionalmente inabordable, por lo que generalmente se agrupan en lotes de un tamaño concreto B. A esta función de error promediado, se le conoce como función de coste:

$$J(\mathbf{W}) = \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}\left((\bar{X}^{(i)};\, \mathbf{W}),\, \bar{Y}^{(i)}\right). \tag{2.14}$$

Aprendizaje no supervisado

En este caso, los datos de entrenamiento no incluyen etiquetas, y será el algoritmo el que intentará clasificar la información por sí mismo. De esta manera, la red neuronal no puede saber exactamente cómo se denomina a un determinado dato de entrada, pero sí puede deducir similitudes o diferencias entre ellos.

Para este tipo de entrenamiento se suelen utilizar como base las ANN para encontrar la información subyacente de los datos, en el espacio de variables latentes $\bar{Z}$. Debido a que no existen etiquetas, se compara la información reconstruida por el decodificador $\hat{\bar{X}}$ con la propia entrada $\bar{X}$. Por ejemplo, se podría usar el error cuadrático medio para la función de **pérdida de reconstrucción**,

$$\mathcal{L}_x\left(\bar{X};\hat{\bar{X}}\right) = \left|\left|\bar{X} - \hat{\bar{X}}\right|\right|^2. \tag{2.15}$$

Sin embargo, al dar un enfoque estadístico al espacio latente, debe tenerse en cuenta también una función de **pérdida por regularización**. Se busca comparar la función del codificador $q_\phi\big(\bar{Z}|\bar{X}\big)$ con algún tipo de distribución estadística deseada $p\big(\bar{Z}\big)$,

$$\mathcal{L}_s = D\left(q_\phi\big(\bar{Z}|\bar{X}\big) \,||\, p\big(\bar{Z}\big)\right), \tag{2.16}$$

donde D representa una divergencia entre dos distribuciones de probabilidad. Si se toma la distribución normal estándar como la función estadística,

$$p\big(\bar{Z}\big) = \mathcal{N}\big(\mu = 0, \sigma^2 = 1\big), \tag{2.17}$$

la pérdida por regularización resulta la divergencia de Kullback-Leibler [64],

$$\mathcal{L}_{KL} = -\frac{1}{2}\sum_{j=1}^{K}\left(\sigma_j + \mu_j^2 - 1 - \log\sigma_j\right). \tag{2.18}$$

Generalmente, este error se multiplica por un factor β, conocido como coeficiente de desenredo [65], para que los nuevos datos generados no tiendan a parecerse a los de entrenamiento. Así, la función de coste resulta:

$$J(\mathbf{W}) = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_x\left(\bar{X}^{(i)},\hat{\bar{X}}^{(i)}\right) + \beta\cdot D\left(q_\phi\big(\bar{Z}|\bar{X}^{(i)}\big) \,||\, p\big(\bar{Z}\big)\right). \tag{2.19}$$

A esta modificación de las ANNs, se le conoce como *Variational Autoencoder* (VAE).

El esquema de la Figura 2.11 muestra cómo se relacionan todos los conceptos que se abordan en el aprendizaje no supervisado aplicado a los VAEs, siguiendo el procedimiento

propuesto en Amini, A. *et al* (2019) [66]. Incluye algunos parámetros que serán abordados en apartados posteriores.

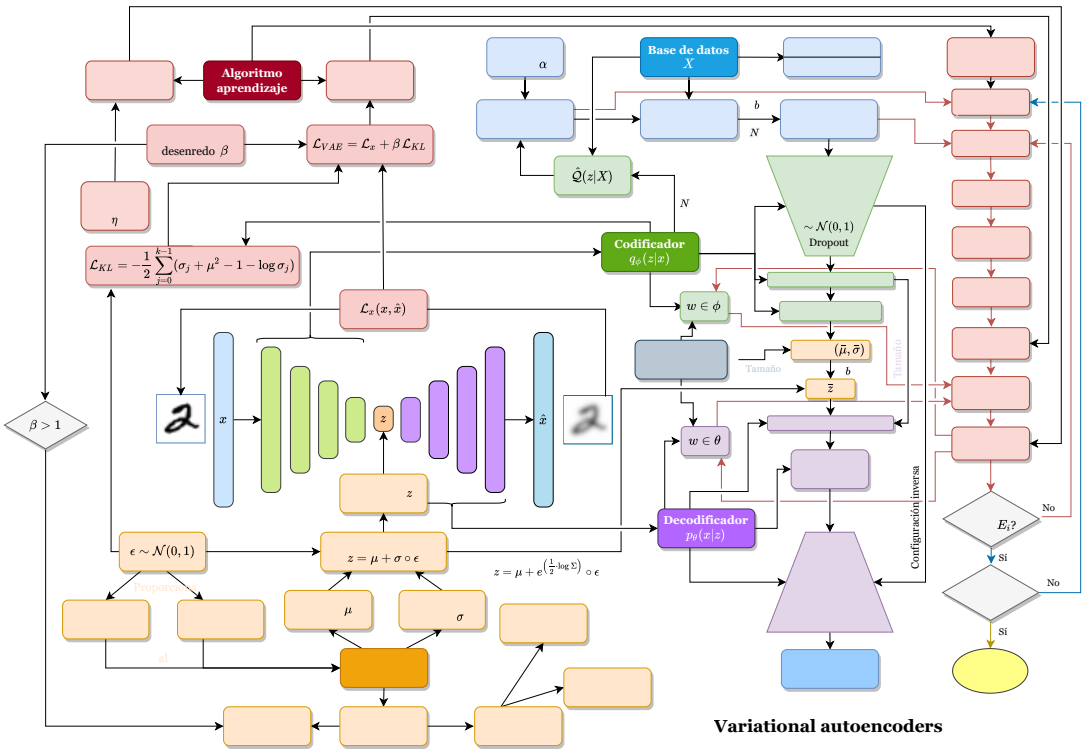


Figura 2.11: Esquema resumen de un proceso de aprendizaje no supervisado para un VAE.

Aprendizaje por refuerzo

El Aprendizaje por Refuerzo o *Reinforcement Learning* (RL) es un área de *Machine Learning* que consiste en determinar qué acciones debe realizar un agente en un entorno dado con el fin de maximizar una "recompensa". Se usa en redes neuronales para que aprendan de entornos dinámicos gobernados por una serie de reglas, como, por ejemplo, sistemas físicos o juegos. Un ejemplo de aplicación de este tipo de aprendizaje podría ser la circulación de un vehículo autónomo en carretera.

En este caso, se desea entrenar un determinado **agente** que ejerce una influencia sobre un sistema como el de la Figura 2.12, a partir de una serie de pares estado-acción, con el fin de maximizar futuras recompensas a lo largo del tiempo. Estos modelos están compuestos por:

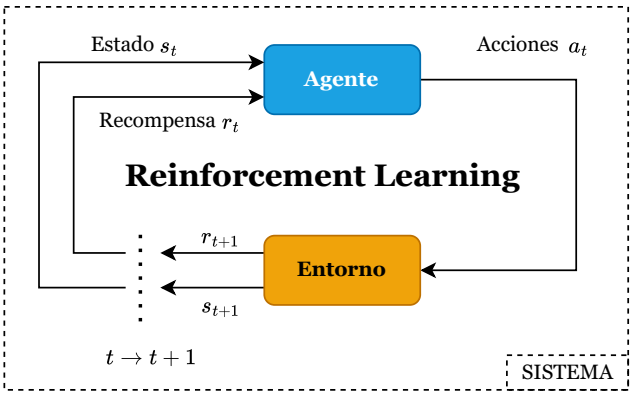


Figura 2.12: Sistema físico controlado por una red neuronal entrenada por refuerzo.

- **Acciones** (a_t): Grados de libertad que posee el agente para modificar el entorno.
- **Estados** (s_t): Son situaciones que el agente percibe u observa.
- **Recompensas** (r_t): Realimentación que mide el éxito o fracaso de cada acción.
- **Agente**: Es quien realiza las acciones. Se trata de una red neuronal profunda, cuya entrada son los estados y sus salidas, las acciones. Se entrena mediante el error de cada acción ponderado con su recompensa. Posee una memoria que recuerda la recompensa o errores que obtuvo con cada acción.
- **Entorno**: El mundo donde existe el agente y opera. Las acciones del agente pueden alterarlo y darán lugar a un nuevo estado. Posee unas reglas que determinan la recompensa de cada acción, como podrían ser unas leyes físicas.

En estos casos, el algoritmo de aprendizaje, mostrado en el esquema de la Figura 2.13, entra en juego cuando se sobrepasan las limitaciones impuestas del entorno, lo que marcaría el fin de un determinado **episodio**. Utilizando la memoria del agente de todas las acciones, observaciones y recompensas a lo largo del episodio, se calcula el error total. Este se obtiene de la probabilidad de haber ejecutado cada acción, de la finalmente tomada entre dichas opciones y la recompensa obtenida por ella. No se entrará en detalle de la función de error que se suele emplear en estos algoritmos, al no ser el tipo de entrenamiento utilizado para los casos de estudio de esta tesis.

2.2.4. Descenso del gradiente

En el DL se utiliza un proceso de optimización iterativo donde se ajusta gradualmente los parámetros del modelo con el objetivo de minimizar la función de coste sobre el conjunto

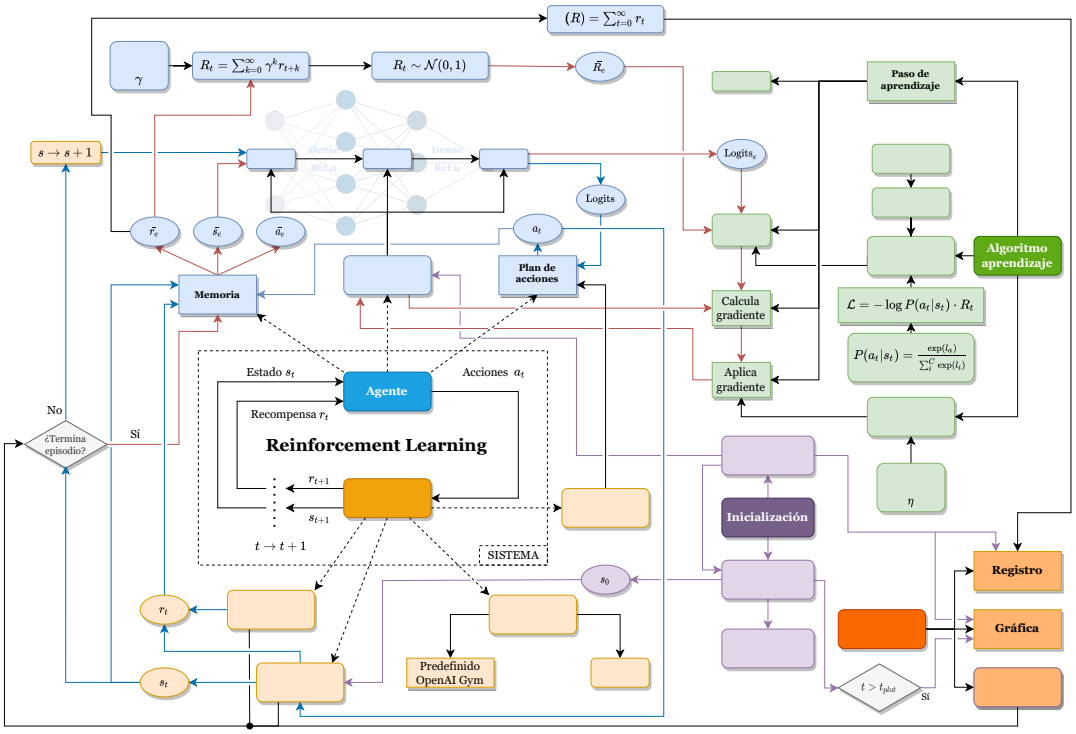


Figura 2.13: Esquema resumen de un proceso de aprendizaje por refuerzo.

de datos de entrenamiento. El descenso del gradiente usa la primera derivada de la función de coste respecto al conjunto de parámetros,

$$\nabla J(\mathbf{W}) = \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}, \tag{2.20}$$

lo implica aplicar la regla de la cadena [67] para calcular cómo cambia la función de coste con respecto a los parámetros de las más alejadas de la salida. A continuación, se actualizan los parámetros en la dirección opuesta al gradiente:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla J(\mathbf{W}), \tag{2.21}$$

siendo η la tasa de aprendizaje, que se definirá más adelante.

En la Figura 2.14 se muestra un esquema del proceso del descenso del gradiente para una red que dispone de los parámetros de solo una neurona artificial. Sin embargo, los modelos de DL están compuestos por numerosas capas de numerosas neuronas, por lo que la función de coste resulta mucho más compleja, con una infinidad de mínimos locales que

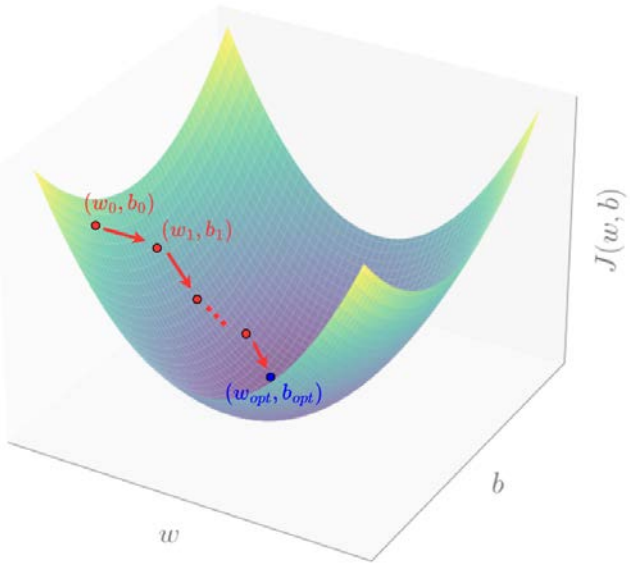


Figura 2.14: Esquema del descenso del gradiente de la función de coste para un modelo con una única neurona donde $J(y, \hat{y}(w, b)) = J(w, b)$.

dificultan el aprendizaje.[3]

Comúnmente, se emplea la técnica de mini lotes (*batches*), que consiste en dividir los datos de entrenamiento en grupos pequeños. En lugar de utilizar todos los datos a la vez para ajustar un modelo, se calcula y aplica el gradiente a cada uno de estos grupos más pequeños, lo que acelera el entrenamiento y mejora la generalización si los datos están bien distribuidos.

Optimizadores

Los optimizadores son variantes u optimizaciones del algoritmo de descenso del gradiente. Los optimizadores más utilizados en la literatura son: SGD [68], RMSprop [69], AdaGrad [70] o Adam [71, 72], entre muchos otros. Esta tesis utiliza únicamente el algoritmo optimizador Adam, que actualmente es el más frecuentemente usado en DL.

Adam[4] utiliza una estimación del momento y de la magnitud de los gradientes anteriores para actualizar los parámetros del modelo en cada iteración. Sin embargo, en lugar de

[3] El descenso del gradiente es, intuitivamente, como encontrar el camino más rápido, para bajar de una montaña, estando totalmente a ciegas, y siguiendo la pendiente más pronunciada para llegar a la parte más baja dando pequeños pasos.

[4] Empleando el entorno de Tensorflow, con la API de Keras, nunca se tendrá que emplear directamente este algoritmo, por lo que ahondar en su parte matemática carece de sentido, mas allá de comprender que la optimización esta basada en gradientes.

utilizar una tasa de aprendizaje constante para todos los parámetros, Adam adapta la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y de la magnitud del gradiente.[5] Esto permite que el modelo se ajuste de manera más eficiente y efectiva a los datos de entrenamiento, lo que puede llevar a una mayor precisión de la predicción en comparación con otros métodos de optimización.

2.2.5. Parámetros e hiperparámetros

Los **parámetros** del modelo son los parámetros internos que se obtienen o se estiman durante el proceso de entrenamiento, y son, por ejemplo, los pesos o sesgos de las neuronas. Estos parámetros, una vez el modelo ya esta entrenado, se utilizan para realizar predicciones.

Los **hiperparámetros** son parámetros externos al modelo establecidos por el programador, por ejemplo las funciones de activación a emplear o el tamaño del lote. Tienen un gran impacto en la red, y encontrar sus valores óptimos no es algo trivial, y en la mayoría de los casos se consigue por prueba y error.

A continuación, se muestran los principales hiperparámetros, clasificados en dos grandes grupos:

Hiperparámetros de estructura y topología de la red neuronal

- **Funciones de activación:** Las funciones utilizadas para calcular la salida de cada neurona en una capa de la red.
- **Número de capas de neuronas:** La cantidad de capas ocultas en la red, junto con el número de neuronas en cada capa, define la profundidad y complejidad de la red, lo que puede influir en su capacidad para modelar relaciones en los datos.
- **Inicialización de pesos:** La manera en que los pesos de las conexiones entre las neuronas se establecen al principio del entrenamiento, normalmente de manera aleatoria.
- **Tamaño de la capa de salida:** El número de neuronas en la capa de salida de la red. Depende del número de clases del problema de clasificación o de la estructura deseada de los datos a generar.

[5] Términos como tasa de aprendizaje o momento son ejemplos de hiperparámetros de una red neuronal. En el siguiente apartado se entrará más en detalle.

- **Función de pérdida:** La función utilizada para medir la discrepancia entre las predicciones del modelo y los valores reales del conjunto de entrenamiento.
- **Regularización:** La técnica utilizada para prevenir el sobreajuste, como la regularización L1 o L2, y los hiperparámetros asociados, como el factor de regularización (λ)).
- ***Dropout*:** La fracción de neuronas que se "apagan" aleatoriamente durante el entrenamiento para prevenir el sobreajuste.
- **Otros hiperparámetros para redes especializadas:** En las redes convolucionales se debe indicar el tamaño de filtro, la cantidad de filtros, el paso, *padding*, etc. En las ANN, se especifica el factor de desenredo, el tamaño del espacio latente, la función estadística deseada, etc.

Hiperparámetros de algoritmo de aprendizaje

- **Número de épocas (*epochs*):** Es el número de veces que los datos de entrenamiento han pasado por la red neuronal. Encontrar el número óptimo es crucial, ya que un valor muy bajo puede limitar el potencial del modelo, mientras que un valor muy alto puede provocar sobreajuste u *overfitting*, donde el modelo se vuelve demasiado especializado en los datos de entrenamiento y no generaliza bien a nuevos datos.
- **Tamaño de lote (*batch size*):** A la hora de entrenar un modelo se suelen introducir los datos de entrenamiento en lotes. Por ejemplo, en el contexto de series temporales, el *batch size* se refiere a el número de ventanas que se utilizan en cada lote durante el entrenamiento. El valor óptimo del *batch size* depende de varios factores; entre ellos la capacidad de memoria del computador o el número total de datos de entrenamiento.
- **Tasa de aprendizaje (*learning rate*):** El vector de gradiente tiene una dirección y una magnitud. Los algoritmos de gradiente descendiente multiplican la magnitud del gradiente por un escalar η conocido como tasa de aprendizaje, generalmente menor que 1. Si se elige una tasa muy alta, el entrenamiento se realizará de manera más rápida, pero puede llevar a realizar saltos inestables en la función de error y, por consiguiente, no converger. En cambio, si se elige una tasa de aprendizaje excesivamente baja, el entrenamiento puede ser muy lento y quedar atrapado en mínimos locales.
- **Decaimiento de la Tasa de aprendizaje (*learning rate decay*):** Este hiperparámetro reduce, en cada época, el valor de la tasa de aprendizaje para lograr un entrenamiento más rápido al principio y, a medida que avanza, ajustes más pequeños.
- **Momento (*Momentum*):** Permite que el optimizador "recuerde" las actualizaciones de peso anteriores y las utiliza para mantener el impulso y superar obstáculos

en el camino hacia el mínimo global.[6] En situaciones donde la función objetivo tiene múltiples mínimos locales, el uso de *momentum* ayuda a evitar quedar atrapado en mínimos locales subóptimos.

[6]Este hiperparámetro se puede entender como una canica que rueda cuesta abajo. Cuando la canica encuentra un obstáculo en el camino, su impulso acumulado le ayuda a superarlo en lugar de quedarse atascada.

Capítulo 3

Metodología

Este capítulo describe la estrategia general empleada en las distintas líneas de investigación abordadas en esta tesis, alineándose con las etapas del ciclo de vida de la ciencia de datos. Cada sección detalla los métodos aplicados, desde la comprensión del problema hasta la creación de modelos de aprendizaje profundo y su implementación en aplicaciones prácticas. La metodología se estructura de manera que permite integrar técnicas avanzadas de procesamiento y análisis de datos en GPUs, maximizando la eficiencia y el rendimiento de los modelos propuestos.

3.1. Análisis contextualizado del problema

En esta sección se presenta una visión general del contexto en el que se enmarca la investigación, destacando cómo surgen las líneas abordadas durante la tesis y los desafíos identificados.

3.1.1. Origen de las líneas de investigación

Las líneas de investigación que sustentan esta tesis doctoral están directamente relacionadas con tres casos de estudio principales: el descubrimiento de fármacos, el cómputo intensivo de datos mediante SkewEngine y la detección de epilepsia. Estas líneas se originaron en el marco de los contratos de colaboración llevados a cabo con el grupo de Supercomputación-Algoritmos de la Universidad de Almería, el grupo de Arquitecturas y algoritmos paralelos de la Universidad de Málaga y con el hospital Carlos Haya de Málaga, permitiendo el acceso a conjuntos de datos diversos y especializados. En cada caso, los

problemas identificados y las metas científicas planteadas marcaron el rumbo de la investigación. Lo que se describe en esta subsección, corresponde a la primera fase del ciclo de vida de la ciencia de datos.

Descubrimiento de fármacos

El grupo de Supercomputación-Algoritmos (TIC146) de la Universidad de Almería lanza un proyecto, financiado por la Junta de Andalucía, titulado: *Inteligencia Computacional en descubrimiento de fármacos* con referencia P18-RT-1193. En él se pretende usar redes neuronales que permitan el cribado virtual de moléculas para encontrar similitudes en forma. Se desea crear aplicaciones o modelos que sean comparables con los algoritmos de optimización desarrollados por el mismo grupo de investigación, en concreto, con el algoritmo Optipharm [73].

SkewEngine

En el grupo de Arquitecturas y algoritmos paralelos (TIC113) de la Universidad de Málaga se pretendía generalizar el uso de una herramienta que había permitido la aceleración del cálculo de la cuenca visual del terreno en trabajos anteriores [74]. Todo ello en el marco del proyecto titulado *Arquitecturas y Programación Avanzadas para Aplicaciones Intensivas en Datos*, con referencia PID2022-136575OB-I00. Se desea demostrar que la herramienta es extensible a numerosas aplicaciones e incluso a tres dimensiones, como imágenes hiperespectrales.

Detección de epilepsia

De nuevo, el grupo de investigación Supercomputación-Algoritmos de la Universidad de Almería recibe una subvención por parte del Ministerio de Ciencia e Innovación para el proyecto *Pulsera inteligente para la predicción, detección y notificación de ataques epilépticos (EPILSERA)*, con referencia PDC2022-133370-I00. El objetivo global del proyecto es el desarrollo de un dispositivo tipo *smarwatch* (EPILSERA) que sea capaz de detectar y predecir crisis epilépticas en tiempo real. Para ello, se desea comparar simultáneamente las medidas de los sensores que llevará la EPILSERA con los datos de un Electroencefalograma (EEG), etiquetado por el equipo médico del hospital Carlos Haya de Málaga (unidad de Neurofisiología). El equipo médico del hospital, por su parte, informa de la necesidad de una herramienta que permita el etiquetado automático de las crisis epilépticas, ya que esta tarea resulta ser tediosa y prolongada.

3.1.2. Fuentes de datos

La recopilación de datos corresponde a la segunda etapa del Ciclo de Vida de la Sección 1.1. Los datos empleados en esta tesis provienen de diversas fuentes que garantizan la calidad y relevancia de los mismos:

- **Bases de datos.** De la infinidad de conjunto de datos disponibles en la web, se optó por Drugbank [75] para las estructuras tridimensionales de las moléculas y por los conjuntos CHB-MIT [76] y Siena Scalp EEG [77] para los registros de los EEG de pacientes que padecen epilepsia.
- **Hospital.** Grabaciones de EEG en formato EDF, obtenidas en sesiones clínicas de pacientes con epilepsia. También se midieron otros sensores de manera simultánea, cuyos registros no son objeto de esta tesis.
- **Departamentos universitarios.** Modelos digitales de elevación procedentes de servicios web (WMS y WCS) [78], utilizados en el desarrollo de herramientas de análisis intensivo.

Cada conjunto de datos implicó retos específicos, desde su recolección hasta su preparación para los modelos de aprendizaje profundo.

3.1.3. Problemas identificados

En una primera fase del análisis de los datos obtenidos (etapa 3 del Ciclo de Vida de la Sección 1.1), se identificaron los siguientes desafíos en las líneas de investigación:

- **Volumen de datos:** Tanto por exceso como por deficiencia. En algunos casos, como en las bases de datos moleculares, el número de moléculas es insuficiente para la aplicación del DL. En otros casos, como con las señales médicas, la elevada frecuencia de muestreo hace que la cantidad de datos sea inabordable.
- **Calidad de los datos:** Se hace necesaria una limpieza y normalización rigurosa para garantizar la calidad de las predicciones, eliminando, por ejemplo, ruidos y artefactos no deseados.
- **Procesamiento eficiente:** La necesidad de acelerar cálculos intensivos utilizando GPUs para mejorar los tiempos de ejecución y la escalabilidad.

3.2. Transformación de los datos

Este apartado se corresponde con la cuarta etapa del Ciclo de Vida (Sección 1.1). En ella garantiza la preparación adecuada de los datos para su uso en modelos de aprendizaje profundo. Inicialmente, los datos recopilados presentaban características que requerían diversos niveles de procesamiento:

- En el ámbito molecular, la generación de proyecciones a partir de geometrías tridimensionales fue un paso clave. Este proceso, llevado a cabo con algoritmos específicos optimizados para GPUs, utilizó espirales de Fibonacci para garantizar una distribución uniforme de los ángulos de proyección, maximizando la diversidad de los datos generados. Este enfoque permitió crear cientos de imágenes únicas por molécula, lo que incrementó significativamente la riqueza del conjunto de datos para el entrenamiento. De esta manera, se logró que los datos fueran compatibles con las redes neuronales convolucionales (CNN).

- El uso de SkewEngine resultó especialmente útil para transformar los datos en representaciones más relevantes en determinados contextos. Por ejemplo, en un problema de regresión multivariable para estimar el precio de un inmueble, podría ser útil transformar los modelos de elevación del terreno en indicadores de visibilidad total. Esto proporcionaría un conjunto de características más interpretables y directamente vinculadas a las variables objetivo. SkewEngine permitió reorganizar las mallas de datos según direcciones espaciales preferentes, lo que mejoró la eficiencia en operaciones aritméticas complejas y redujo significativamente el tiempo computacional.

- En el caso de las señales EEG, las grabaciones en formato EDF contenían ruido y redundancia que debían eliminarse para facilitar su uso en GPUs. Por ello, se aplicó la Transformada de Fourier de Tiempo Corto (STFT) y Transformada Wavelet Discreta (DWT), lo que permitió reducir el tamaño de los datos en un 99 % sin pérdida significativa de información relevante. Este enfoque, además de mejorar la calidad, optimizó los tiempos de procesamiento y la capacidad de almacenamiento.

Adicionalmente, se emplearon técnicas de aumento de datos para ampliar y diversificar los conjuntos existentes. Estas técnicas incluyeron transformaciones geométricas como rotaciones, escalados y traslaciones. En el caso de las moléculas, estas transformaciones fueron esenciales para garantizar que los modelos aprendieran a reconocer patrones sin depender exclusivamente de configuraciones específicas.

Lo más importante de esta etapa, y que además es el objeto de la tesis, es el uso de las GPUs para acelerar los cálculos de manera eficiente.

3.3. Creación de modelos de Deep Learning

Aunque el uso de las GPUs ya está ampliamente extendido en esta quinta etapa, no debe olvidarse que se trata de una de las más delicadas y exigentes para lograr buenos resultados. Por tanto, a continuación se describen algunas estrategias generales seguidas en esta tesis.

3.3.1. Canalización de entrada

Es esencial que los conjuntos de datos contengan las imágenes o tensores de datos generados como los metadatos relacionados como podría ser la propia etiqueta en un aprendizaje supervisado. Y por ello, es necesario, que antes de su entrada a la red neuronal, se ejecuten sucesivas funciones de preprocesamiento para extraer dicha información adicional. Además, dado que, generalmente, el conjunto de datos se encontrará estructurado en carpetas que contienen cada una de las clases, se requiere desordenar los datos para mejorar el proceso de entrenamiento de la red.

Esta canalización de entrada a la red neuronal, ejecutada en una CPU, se suele convertir en el cuello de botella del entrenamiento, ya que también implica operaciones de entrada y salida al almacenamiento interno. Esto deriva en que, mientras la CPU realiza las operaciones mencionadas, la GPU se mantiene a la espera, y viceversa, durante el paso de entrenamiento.

Las tareas anteriores no tienen por qué ser secuenciales, salvo para el primer paso de entrenamiento, por lo que, mientras trabaja la GPU, se puede ir procesando la canalización de entrada. Esto se logra configurando el conjunto de datos con la función `prefetch` de TensorFlow. También es conveniente la paralelización de la CPU en el proceso de carga de las imágenes o tensores y de extracción de los metadatos proporcionados, por ejemplo, por la ruta del archivo.

Por último, como tarea previa es muy importante el agrupamiento en lotes de los datos de entrenamiento, permitiendo sacar el máximo partido a la GPU, ya que se podrá realizar un paso de entrenamiento con múltiples imágenes simultáneamente. El tamaño del lote deberá elegirse conforme a dos aspectos fundamentales: el coste de cálculo del gradiente y la exactitud del mismo. Así, entrenar con datos individuales hace que el gradiente sea más fácil de calcular, pero no permite aprovechar la capacidad de paralelización de la GPU y el gradiente resultante será muy caótico. Por el contrario, entrenar con un lote muy grande, será computacionalmente más costoso y sobrepasará los límites de la memoria de la GPU. En este sentido, el gradiente sería más representativo de la función global objetivo. En la

Tabla 3.1 se recogen las pruebas con distintas configuraciones de la canalización de entrada que se han realizado para decidir cuál será la óptima para el entrenamiento.

Este estudio preliminar se ha realizado sobre 4096 imágenes durante 3 épocas (*epochs*). El método `map` hace la llamada a las funciones de la carga de las imágenes y la extracción de los correspondientes metadatos, y se puede configurar de manera secuencial o paralela. Se colorean aquellos datos que suponen una mejora (verde) o perjuicio (rojo) significativo respecto a la configuración anterior. Es evidente, según los resultados, que precargar los datos reduce significativamente el tiempo de procesamiento de entrada, pero afecta a otros tiempos de ejecución pues no se observan mejoras significativas del tiempo de cómputo global.

Tabla 3.1: Canalización de entrada, en tiempo global, tiempo medio por paso de entrenamiento y peso de la canalización de entrada medido con TensorBoard.

Configuración	**Duración**	**Paso**	**Entrada**
`ds.map(seq).suffle`	0:02:52.2	11.3 ms	6,2 %
`ds.map(parallel).suffle`	0:02:39.2	10,1 ms	5,0 %
`.prefetch`	0:02:38.8	10,1 ms	1,0 %
`.batch(32)`	0:01:09.9	35,6 ms	53,1 %
`.batch(64)`	0:01:08.0	72,5 ms	52,1 %
`.batch(128)`	0:01:12.5	136,3 ms	61,8 %
`.batch(32)`	0:01:09.9	35,6 ms	53,1 %
`.prefetch`	0:01:10.0	26,4 ms	1,1 %
`.batch(64)`	0:01:08.0	72,5 ms	52,1 %
`.prefetch`	0:01:02.9	46,3 ms	28,3 %
`.batch(128)`	0:01:12.5	136,3 ms	61,8 %
`.prefetch`	0:01:08.3	84,2 ms	35,5 %

Por otra parte, se comprueba que el tiempo de entrenamiento es altamente sensible al tamaño del lote, ya que permite a la GPU procesar diferentes cantidades de datos simultáneamente. Sin embargo, un mayor tamaño del lote también implica más llamadas a las funciones de preprocesamiento en la CPU y una mayor transferencia de datos entre GPU y CPU. Esto podría perjudicar la canalización de entrada notablemente, como se puede comprobar en la Tabla 3.1, donde se muestra que hay un mayor peso de la entrada (porcentaje) cuanto mayor tamaño tenga el lote .

3.3.2. Estructura y configuración de las redes neuronales

Para lograr mejores resultados de precisión y para evitar el sobre-entrenamiento de la red, se realizan las siguientes configuraciones:

- Reescalar los valores de la imagen de 0 a 1, para garantizar una aplicación estable del gradiente.
- En el tamaño del filtro de convolución es conveniente seleccionar algunos más grandes para capturar agrupaciones locales en lugar de detalles marginales.
- Tasa de abandono en algunas capas de la red para evitar el sobre-entrenamiento.
- Reducción del tamaño de los datos durante el entrenamiento, ajustando el paso de los filtros o mediante una función de `Pooling`.
- Uso de la normalización por lotes que reduce el problema de la dependencia de los cambios de las capas anteriores durante el entrenamiento, llamado "cambio de covariable interno" por los autores [79]. Con esta regularización de las entradas a una determinada capa, permite usar tasas de entrenamiento más bajas, coordina el entrenamiento entre distintas capas y asegura una convergencia más rápida durante el proceso de aprendizaje. Se recomienda la transformación de la normalización por lotes tras la capa de convolución y antes de la función de activación lineal rectificada o `ReLU` [80]. Además, es una manera de evitar el sobre-entrenamiento de forma más rápida que la tasa de abandono, por lo que los autores anteriores recomiendan no usarlo de manera simultánea.
- El número de filtros durante las etapas de convolución permite capturar más o menos mapas de características, por lo que afinar este parámetro será decisivo para que sea capaz de reconocer las moléculas que son pero también para encontrar similitudes con otras.
- Incrementar el número de capas de la red, es decir, hacerla más profunda, reduce notablemente el error. Sin embargo, cuando el número de capas es muy alto, aparece el problema de desvanecimiento del gradiente, lo que produce un aumento del error. Para ello, se emplean las redes residuales o *ResNets* [81] que permiten aumentar el número de capas, mitigando el efecto del desvanecimiento del gradiente.

3.3.3. Proceso de entrenamiento

En el proceso de entrenamiento de una red neuronal, el enfoque habitual es el método de prueba y error, donde se ajustan diversos hiperparámetros y se revisan las métricas

de rendimiento para evaluar y mejorar el modelo. El objetivo es encontrar la combinación de parámetros que permita a la red generalizar correctamente sobre datos no vistos. Este proceso requiere iteraciones continuas y una evaluación rigurosa de diversas métricas.

Dependiendo del tipo de problema y los objetivos específicos, las métricas utilizadas para evaluar el rendimiento del modelo pueden variar significativamente. En dominios como el descubrimiento de fármacos, donde se busca una clasificación precisa entre compuestos, métricas como la función de **coste** de la ecuación (2.14) y la **exactitud** (precisión global) son comúnmente empleadas. Además, la matriz de confusión puede ser útil para estudiar las similitudes y diferencias entre las moléculas.

Sin embargo, en aplicaciones como la detección de epilepsia, que a menudo involucra datos desbalanceados donde las clases de interés (como las crisis epilépticas) son mucho menos frecuentes, es necesario emplear métricas más específicas. En estos casos, es recomendable usar métricas como la **recuperación**, **precisión** y el área bajo la **curva de precisión-recuperación**. Estas métricas son especialmente relevantes cuando la clase positiva es rara, ya que proporcionan una medida más detallada de la capacidad del modelo para identificar correctamente los casos de interés sin ser influenciado por el desequilibrio entre las clases.

Este proceso de evaluación iterativa es clave para la mejora continua del modelo, ya que las métricas seleccionadas influirán directamente en la forma en que el modelo se ajusta a los datos y en su capacidad para hacer predicciones útiles en escenarios prácticos. Las métricas más adecuadas varían según el dominio de aplicación, el tipo de datos y los objetivos específicos, por lo que se detallarán más en cada línea de investigación.

3.3.4. Validación y test

En algunas líneas se presentan, además, herramientas creadas a partir de los modelos entrenados para comprobar que efectivamente cumplen con los objetivos específicos de cada problema, como comparadores gráficos de moléculas o detectores de crisis epilépticas. Esta corresponde a la sexta etapa del Ciclo de Vida mencionado en la Sección 1.1.

3.4. Desarrollo de aplicaciones

Como etapa final, se desarrollaron aplicaciones prácticas basadas en los resultados de los modelos y metodologías previas, destacando su utilidad en entornos reales. En algunos casos, se implementaron librerías dinámicas en C++ para mejorar la eficiencia de cálculos

específicos y facilitar su integración con otras aplicaciones y lenguajes de programación. En otros, se opta por aplicaciones web con el fin de ofrecer una interfaz accesible y funcional. De esta manera, se han podido incorporar comparadores y herramientas de visualización o gráficas para que el ojo humano evalúe la relevancia y precisión de los modelos. Todos los códigos, en sus diferentes lenguajes (C++, Python, PHP, javascript...) han sido convenientemente documentados para IntelliSense y publicados en GitHub, facilitando así su libre distribución. En cualquier caso, las soluciones desarrolladas integran todo el flujo de trabajo, desde el preprocesamiento hasta la interpretación de resultados, utilizando GPUs para optimizar cada etapa y garantizar una experiencia fluida para el usuario final.

Capítulo 4

Experimentación

En este capítulo se presentan los casos de estudio desarrollados para validar la metodología propuesta y alcanzar los objetivos específicos que se plantearon en la Sección 1.3. Cada caso de estudio aborda un problema concreto y relevante en el ámbito del ML, demostrando cómo la optimización del preprocesamiento y postprocesamiento mediante GPUs mejora significativamente el rendimiento computacional y la aplicabilidad de los modelos a nuevos problemas.

Los casos incluyen el descubrimiento de fármacos, donde se optimiza la generación y análisis de proyecciones moleculares; SkewEngine, una herramienta diseñada para el procesamiento eficiente de datos en mallas regulares; y la detección de epilepsia, enfocada en la reducción y transformación de datos biomédicos de un EEG para su clasificación automática. A través de estos ejemplos, se evidencian los beneficios prácticos y la escalabilidad de las técnicas propuestas en escenarios reales.

4.1. Descubrimiento de fármacos

El descubrimiento de fármacos es un proceso complejo debido a la enorme cantidad de moléculas posibles y a la necesidad de identificar aquellas con potencial terapéutico. Este proceso incluye múltiples etapas, desde la identificación de blancos moleculares hasta la optimización de moléculas afines y la validación experimental. Sin embargo, las metodologías tradicionales son limitadas frente a la inmensidad del espacio químico, estimado en hasta 10^{200} compuestos. Por ello, se han desarrollado técnicas computacionales como el Cribado Virtual (*Virtual Screening*, VS) [29], que permiten reducir significativamente el número de compuestos candidatos y acelerar la identificación de fármacos efectivos.

El VS se divide en dos enfoques principales: basado en estructuras (*Structure-Based Virtual Screening*, SBVS) [82, 83] y basado en ligandos (*Ligand-Based Virtual Screening*, LBVS) [30, 84]. El LBVS es particularmente útil cuando no se dispone de la estructura tridimensional del blanco molecular. Este enfoque se fundamenta en el principio de similitud molecular, que postula qué compuestos con estructuras similares tienen propiedades biológicas similares. En este contexto, la similitud de forma tridimensional y el potencial electrostático son descriptores clave para evaluar y comparar moléculas.

En esta línea de investigación, el enfoque se centra en la metodología VS, específicamente en la técnica de LBVS basada en la similitud de forma (*shape similarity*) [85]. Este método analiza las estructuras tridimensionales de las moléculas y calcula su grado de solapamiento utilizando métricas como el coeficiente de Tanimoto [86]. Adicionalmente, la similitud en potencial electrostático [87], que evalúa cómo se distribuyen las cargas en las moléculas, complementa esta caracterización y permite identificar candidatos con una mayor probabilidad de interacción efectiva con el blanco terapéutico. Sin embargo, en este caso de estudio en concreto, se presenta una técnica novedosa que utiliza un elevado número de proyecciones bidimensionales de cada una de las moléculas de la base de datos `DrugBank` [75] para que una red neuronal artificial aprenda a identificarlas, e incluso sea capaz de localizar moléculas cuya forma coincida con alguna o algunas de las ya existente en la base de datos. Al ser un estudio preliminar, no hay objetivos claramente determinados, aunque un ejemplo podría ser la identificación de fármacos con estructuras espaciales similares a otras moléculas conocidas, que eventualmente pudieran tener la misma función aunque con menos efectos secundarios sobre el organismo.

El reconocimiento visual de un objeto tridimensional es un proceso que requiere, en la inmensa mayoría de los casos, de una proyección bidimensional de dicho objeto como una etapa de "preprocesamiento" antes de ejecutar el propio reconocimiento. Esto ocurre, por ejemplo, en el fondo del ojo y en la retina, como un paso previo casi imprescindible antes de que el cerebro procese la información recibida. Esto puede realizarse con o sin la ayuda de otras imágenes, como las captadas por el segundo ojo, u otros órganos sensoriales. También ocurre en la captura de la imagen en una película fotográfica o un sensor CCD, antes de que, ya sea una máquina o persona, se continúe con el mecanismo final de reconocimiento.

La capacidad de la mayoría de los animales para reconocer los objetos, independientemente de la posición espacial en la que se haya obtenido la proyección, incluyendo los seis grados de libertad, es realmente impresionante. Cuando vemos, por ejemplo, un mando a distancia, inmediatamente lo reconocemos, independientemente de su posición sobre la mesa o la distancia a la que se encuentre. Es cierto que existen posiciones en las que el reconocimiento es más probable que finalice con éxito, como la perspectiva canónica [88], aunque también existe la posibilidad de cometer errores. Sin embargo, a veces, no sólo podemos aprender de los errores, sino que también podemos aprovecharlos en nuestro favor,

para identificar, por ejemplo, objetos similares.

No es éste el primer trabajo que utiliza las diferentes perspectivas de un objeto para mejorar el reconocimiento mediante inteligencia artificial. De hecho, hay cientos de estudios, muchos de ellos recopilados en [89] en los que se utilizan diferentes fotografías o renderizaciones tomadas alrededor de un objeto. Sólo algunos de estos estudios, como RotationNet o VERAN [90,91], utilizan la renderización de objetos desde diferentes coordenadas en una esfera. Sin embargo, su alcance es muy limitado por el reducido número de perspectivas y ninguna explota el paralelismo computacional para el cálculo de las proyecciones, que es un punto crítico en el preprocesamiento para el aprendizaje profundo.

Para este fin se han empleado proyecciones isométricas. Esta decisión se debe a la necesidad crítica de gestionar los costes computacionales y el tamaño del conjunto de datos, que son consideraciones fundamentales en las aplicaciones de aprendizaje profundo. Las proyecciones isométricas garantizan la conservación de las propiedades geométricas esenciales en la representación 2D. Alternativamente, podrían utilizarse colectores topológicos o riemannianos [92] para representar moléculas 3D, ya que estos enfoques proporcionarían una representación más rica y detallada de las estructuras 3D. Sin embargo, la naturaleza detallada y altamente dimensional de estas representaciones requeriría algoritmos más sofisticados y una mayor potencia computacional, lo que conllevaría costes más elevados y conjuntos de datos más pequeños.

La proyección de moléculas en imágenes de baja resolución da lugar a una reducción significativa de la precisión de la información cuando se considera una sola imagen de forma aislada. Sin embargo, de forma similar a la tomografía discreta [93] y la transformada de Radon [2], cientos o incluso miles de imágenes permiten a la red neuronal recuperar una cantidad sustancial de información asociada a la estructura original. Este enfoque también es análogo a la reconstrucción geométrica a partir de teledetección utilizando técnicas de ML (como *Random Forests* y *Support Vector Machines*) [94], donde la integración de datos de múltiples proyecciones o imágenes mejora la precisión de la reconstrucción tridimensional y su análisis.

No se ha tenido conocimiento de ningún trabajo hasta la fecha que utilice esta técnica o parecida en el reconocimiento de moléculas o fármacos.

4.1.1. Proyección multidireccional de la molécula

Cada molécula puede ser representada como un conjunto de átomos y enlaces en un espacio tridimensional. Matemáticamente, podemos representar una molécula M como un diagrama $G = (V, E)$ donde:

- V es el conjunto de vértices (átomos) con sus respectivas coordenadas (x_i, y_i, z_i).
- E es el conjunto de enlaces entre los átomos.

Las proyecciones de la molécula desde diferentes ángulos pueden ser modelada como transformaciones lineales. Sea P_i la proyección de la molécula M en el plano desde el punto i. La transformación puede representarse mediante matrices de rotación R_i,

$$P_i = R_i \cdot M. \tag{4.1}$$

Como se indicó anteriormente, uno de los objetivos de este trabajo es generar un conjunto de imágenes de baja resolución (inferior a 256×256 píxeles), obtenidas mediante las proyecciones ortogonales multidireccionales de cada molécula sobre un plano. Es decir, como si se hubieran capturado desde cámaras situadas en una esfera de radio infinito, con distancia focal infinita, y que estuvieran equitativamente distribuidas en dicha esfera. Y el primer paso de este mecanismo es la selección de dichos puntos y los correspondientes ejes de proyección.

La búsqueda de un conjunto de N puntos, o teselas, que estén distribuidos equitativamente en la superficie de una esfera es uno de los grandes retos de la geometría de todos los tiempos, no solo por su relevancia en muchas áreas de la ciencia, como la astronomía, la meteorología, la agricultura o la energía, sino también por mera curiosidad. Ya Platón, en la antigua Grecia, propuso una posible solución para valores pequeños de N, como el tetraedro (4), el dodecaedro (12) o el icosaedro (20). Pero ni siquiera los sólidos platónicos (o más bien, su proyección en la esfera en la que se inscriben) son capaces de dar una respuesta correcta a una pregunta que resulta ambigua ya en su propia formulación [95]. ¿Qué se considera una distribución equitativa? En la mayoría de los casos, la solución se busca usando distancias euclídeas (como la que plantea el problema de *Tammes*), aunque también se pueden utilizar energías o campos (como en el problema de *Thomson*, que distribuye los electrones en su correspondiente modelo atómico) [96].

Sea como sea, son muy numerosos los algoritmos que generan soluciones aproximadas para el cálculo de puntos o mallas esféricas, especialmente por su relevancia en la resolución numérica de ecuaciones diferenciales, en computación paralela, en gráfica computacional e incluso en exploración espacial, por citar algunos [97–99]. No hay que olvidar que proyectos tan extraordinariamente costosos, como los programas GPS, Galileo o Starlink dependen en gran medida de una distribución equitativa de los satélites [100]. Sin embargo, no siempre es razonable incurrir en un alto coste para el cálculo de una malla equitativa, siendo preferible utilizar algoritmos simples, como el método de Bauer [101], o incluso la elegante y eficiente espiral esférica de Fibonacci [102].

En dicha espiral, usando coordenadas esféricas, el seno del ángulo polar θ entre puntos consecutivos están equitativamente distribuidos en el rango $\{-\pi/2, \pi/2\}$,

$$\Delta \sin\theta = \frac{2}{N-1}, \tag{4.2}$$

por lo que el ángulo polar de un punto i, viene dado por:

$$\theta_i = \arcsin\left(-1 + \frac{2i}{N-1}\right). \tag{4.3}$$

La *razón áurea*,

$$\phi = \frac{\dot{(}1-\sqrt{5})}{2}, \tag{4.4}$$

se usa para calcular el ángulo azimutal del punto,

$$\varphi_i = 2\pi\phi i. \tag{4.5}$$

Por lo tanto, las coordenadas cartesianas de cada eje de proyección se calculan por las transformaciones:

$$x_i = \cos\theta_i \cos\varphi_i; \quad y_i = \cos\theta_i \sin\varphi_i; \quad z_i = \sin\theta_i. \tag{4.6}$$

En la Figura 4.1 se muestra una espiral de Fibonacci con 50 puntos [103]. El conjunto de puntos de la espiral esférica de Fibonacci no es, probablemente, la mejor distribución posible. Por ejemplo, no coincide con los cuerpos platónicos en sus correspondientes valores. Pero con la espiral se obtienen excelentes propiedades de muestreo y es extremadamente simple de construir en contraste con otros esquemas de muestreo esféricos más sofisticados.

Elegir la cantidad exacta de puntos en la esfera para proyectar las moléculas es difícil y obviamente requiere de un mecanismo de ensayo–error. Parece evidente que 10 puntos o menos es claramente insuficiente, y que hacer muchas más de 1000 proyecciones no tiene demasiado sentido, más aún cuando las imágenes proyectadas no tendrán resolución para que merezca la pena. También existe la posibilidad de descartar los ejes de una semiesfera, al ser las proyecciones bastante parecidas a un lado u otro. En este trabajo se ha desechado esta idea por tres motivos. En primer lugar, los ejes no son simétricos en la secuencia de la espiral esférica de Fibonacci. En segundo lugar, y debido al mecanismo de proyección elegido, los ejes de la segunda semiesfera producen imágenes invertidas que mejoran los resultados del entrenamiento de la red neuronal. Finalmente, los átomos de las moléculas se pueden renderizar con ocultación dependiente de la distancia al observador.

Una vez elegido el conjunto de ejes de observación, la renderización de la proyección es una tarea relativamente simple, que implica:

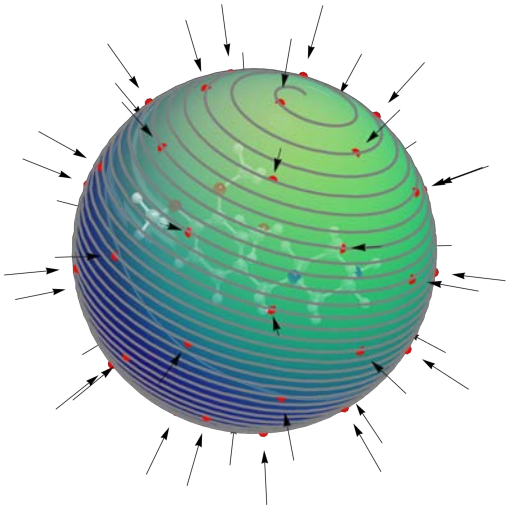


Figura 4.1: Ejes de proyección de una molécula utilizando una espiral de Fibonacci de 50 puntos sobre una esfera unitaria.

- Calcular las coordenadas de cada molécula en el nuevo sistema de ejes, siendo Z' el eje de proyección.
- Opcionalmente, ordenar las moléculas respecto al eje de proyección (de lejos a cerca del observador), para determinar el orden de renderizado.
- Elegir un eje vertical para la imagen a renderizar, dentro de ese plano.
- Dibujar los segmentos correspondientes a los enlaces moleculares (opcional).
- Elegir una escala y tamaño para la imagen.
- Dibujar cada molécula sobre la imagen, de lejos a cerca, considerando, opcionalmente, una cierta transparencia, un color y un tamaño.

Obsérvese que hay varias opciones a la hora de construir la imagen proyectada, y que, a priori, no se sabe cómo podrían afectar al proceso de reconocimiento. Sin embargo, la intuición dice que la red neuronal artificial se va a comportar de forma parecida a nuestro cerebro, por lo que dar más información de calidad optimizará el proceso de reconocimiento. Un claro ejemplo se observa en la facilidad que tiene nuestro cerebro a la hora de identificar los tres anillos de la molécula de Ademetionina de la Figura 4.2 gracias a la presencia de los enlaces atómicos.

El tamaño y el color de los átomos también es opcional, aunque es evidente que añade información de gran relevancia para diferenciar, por ejemplo los distintos anillos hetero–aromáticos presentes en los compuestos. En este trabajo, el diámetro se calcula a partir

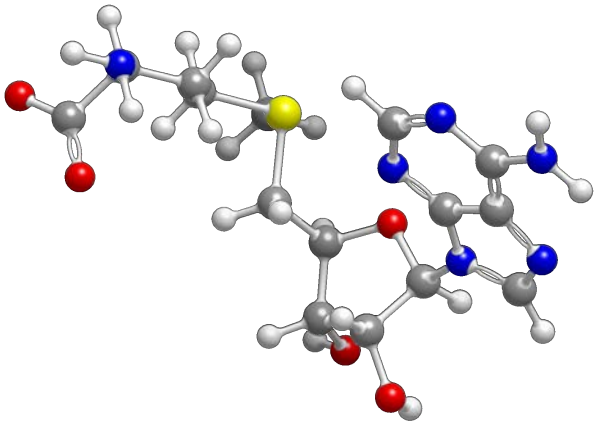


Figura 4.2: Molécula de Ademetionina (Número de acceso DrugBank:: DB00118), renderizada con la aplicación VIDA.

de la masa atómica de los isótopos, mientras que se utiliza una paleta de colores similar a la de la aplicación VIDA [73], con colores dependientes del número atómico, aunque en los átomos de mayor tamaño se ha incluido una cierta transparencia exterior que permita visualizar parcialmente los átomos que se encuentran detrás (véase, por ejemplo, la Figura 4.3).

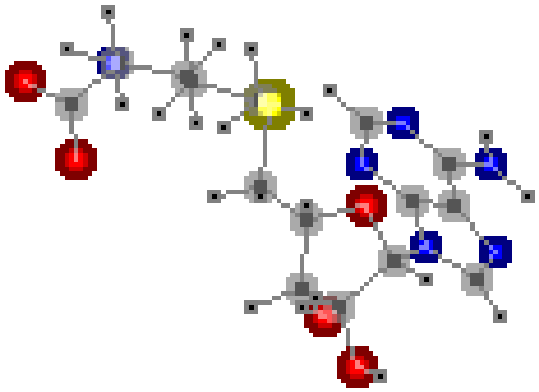


Figura 4.3: La molécula de Ademetionina de la Figura 4.2, renderizada en una imagen de 256x256 píxeles (recortada) utilizando el eje número 144 de una serie de Fibonacci de 512 puntos.

Respecto al dibujo de los enlaces, se hizo un primer experimento sin ellos, pero los resultados (durante el reconocimiento) fueron claramente de peor calidad. Además, la renderización de los enlaces apenas implica coste computacional, ya que los segmentos se dibujan directamente entre átomos (con el algoritmo de Bresenham) que previamente han sido proyectados a sus correspondientes píxeles en dos dimensiones.

Tamaño de las imágenes y jerarquía de memoria

Mediante experimentos preliminares en CPUs se demostró que la generación de imágenes resultaba considerablemente más costosa en comparación con el tiempo requerido para proyectar las moléculas sobre el mapa bidimensional. Este mapa es el que, posteriormente, sería convertido en imágenes. Este resultado fue inicialmente sorprendente, ya que la proyección de moléculas estaba dominada por operaciones trigonométricas de alta complejidad. En contraste, la generación de imágenes implicaba principalmente operaciones de lectura y almacenamiento en memoria. El motivo principal se encuentra en el acceso cuasi–aleatorio al mapa de píxeles y en el propio tamaño del mapa.

Para mejorar la resolución (y mejorar el posterior entrenamiento de la red) se eligió un mapa de 256×256 píxeles, ya que es la imagen más grande en la que la posición de los átomos podría indexarse con un sólo byte (este aspecto es crítico en varios aspectos, como por ejemplo, en el coste de las transferencias entre GPU y CPU). Tamaños inferiores, como 128×128, proporcionarían insuficiente resolución, mientras que tamaños sólo algo superiores dispararán los tiempos de transferencia, almacenamiento, y especialmente, los de reconocimiento, debido probablemente[1] a la superación del indexado mediante byte.

Tabla 4.1: Arquitecturas utilizadas en la generación de imágenes.

	Máquina 1	Máquina 2	Máquina 3
Nombre	Desktop–PC	Server	HPC node
CPUs	1x Intel	32x Intel	40x Intel
	i7-10700K	Xeon E5-2698 v3	Xeon E5-2698 v4
GPUs	1x NVidia	4x NVidia	8x NVidia
	Ampere RTX3080	Maxwell GTX980	Volta V100

Sin embargo, el tamaño 256^2 del mapa de píxeles (65536) estaba provocando un desbordamiento de la cache L1 de datos, ya que ésta, en la inmensa mayoría de procesadores, y por supuesto, en los utilizados en este estudio (Tabla 4.1), tiene un tamaño igual o inferior a 64kB. El desbordamiento se produce porque dentro de cada núcleo del procesador, y pese a que el vector de bytes con la imagen se reutilice, la imagen (más, al menos, el vector con las coordenadas de los átomos) no tiene cabida en la L1. Este problema se multiplica cuando se utiliza una profundidad de colores de 4 bytes, como se planteó inicialmente.

La solución se encuentra precisamente en crear la imagen mediante una paleta indexada con una profundidad de sólo 16 colores (4 bits), que necesita sólo un byte para cada dos píxeles, y que es más que suficiente (usando un color específico para cada uno de los números

[1] El análisis del código fuente de TensorFlow para conocer la causa por la que se dispara el tiempo de reconocimiento (al superar las imágenes un cierto umbral) excede los objetivos de este trabajo.

atómicos que más se repiten, como los de Hidrógeno, Carbono, etc.).

Tabla 4.2: Comparación entre las dos profundidades de color empleadas en la imagen (4 byte (RGBA) vs 4 bits).

Etapa	RGBA	Paleta 4-bit
Proyección átomos	0.20 s	
Asignación a píxeles	11.8 s	0.28 s
Codificación png	285 s	54 s
Tamaño dataset	572MB	559MB

Para comparar ambas codificaciones de imágenes se construyeron 256 imágenes de una selección de 1969 moléculas, y se utilizó un ordenador personal de sobremesa (Máquina 1 en la tabla 4.1) con 8 núcleos, y 8 *threads*. Como en el resto de los cálculos, el lenguaje de programación es C++, usando la librería lodepng para codificar las imágenes y compiladores GNU.

En la Tabla 4.2 se observa un espectacular descenso del tiempo de ejecución al reducir el tamaño del vector de píxeles por debajo de los 32kB. También se reduce (en más de un 80 %) el tiempo de codificación de las imágenes en formato png. Además, la reducción del tiempo se concentra en la fase en la que el mapa de píxeles se rellena (de 10.7s a 0.03s)[2], lo que demuestra, no solo que la causa del bajo rendimiento está en los fallos de cache, sino que además estaba afectando al nivel L2 (de 256kB por cada núcleo, en el procesador empleado). Sin embargo, la reducción del conjunto de datos comprimidos es de apenas un 3 %, lo que no deja de ser una buena noticia, ya que es un indicio de que la cantidad de información se mantiene, a pesar de la significativa reducción de la profundidad de colores.

Escalado de las imágenes

Otro aspecto a tener en cuenta es la escala. Es evidente que la visión estereoscópica del ojo humano contribuye a determinar la diferencia entre, por ejemplo, la maqueta de un Boeing 747 respecto al avión original, pero eso no es posible en imágenes 2D. En nuestro problema, podríamos dibujar todas las moléculas a la misma escala, pero eso plantearía un serio problema, ya que la escala debería ajustarse para que no recortara el peor caso (la molécula de mayor tamaño, proyectada en el eje perpendicular al eje principal [3]). Sin embargo, eso obligaría a dibujar moléculas pequeñas (el ácido fluorhídrico, por ejemplo)

[2]Estos tiempos difieren a los de la Tabla 4.2 en que esta incluye también otras operaciones adicionales como, por ejemplo, la asignación de memoria.

[3]Se considera que el eje principal de una molécula es la línea que une los dos átomos más alejados entre sí

con un tamaño ínfimo, y con la correspondiente pérdida de calidad visual. Para resolver este problema, hemos considerado que todas las moléculas de tamaño inferior a cierto umbral se dibujan a la misma escala, mientras que las moléculas más grandes se ajustan en función de su tamaño. Durante el reconocimiento, la red neuronal utilizará la etiqueta de escala según se describe en la Sección 4.1.3.

Número de ejes y rotación de la cámara

El último aspecto a considerar, quizás el más crítico, es el número de ejes de proyección. Hemos considerado razonable hacer un estudio que considere entre 10^2 y 10^3 ejes, como rango razonable, para determinar el número adecuado.

El objetivo es maximizar la precisión del reconocimiento minimizando el número de proyecciones necesarias. Sea $f(P)$ la precisión de reconocimiento en función del conjunto de proyecciones P. Nuestro objetivo es maximizar $f(P)$ minimizando el número de proyecciones $|P|$.

El problema de optimización puede formularse como:

$$\text{máx}\, f(P), \tag{4.7}$$

sujeto a:

$$|P| \leq K, \tag{4.8}$$

donde K es el número máximo de proyecciones permitidas por limitaciones de tiempo o computacionales.

Podemos utilizar algoritmos de optimización discretos como Hill Climbing, Algoritmos Genéticos o métodos de Optimización Estocástica para encontrar el conjunto óptimo de proyecciones P. Sin embargo, la complejidad de estos algoritmos puede hacerlos computacionalmente prohibitivos cuando se evalúan muchos recuentos de proyecciones posibles. Al seleccionar potencias de dos, aprovechamos su crecimiento exponencial para cubrir una amplia gama de posibles recuentos de proyecciones con un número mínimo de pruebas. Este enfoque reduce significativamente la carga computacional sin dejar de explorar un subconjunto amplio y representativo del espacio de proyección. Los estudios empíricos en contextos de optimización similares muestran a menudo que las mejoras de rendimiento tienden a estancarse más allá de ciertos puntos. Probando potencias de dos, podemos identificar rápidamente si hay un punto en el que aumentar el número de proyecciones produce rendimientos decrecientes. Si no se observan mejoras sustanciales entre estas potencias, es probable que los valores intermedios también muestren ganancias mínimas. En campos como el DL y el procesamiento de señales, es práctica común utilizar potencias de dos para

parámetros como el tamaño de los lotes, el número de nodos o las dimensiones, debido a su alineación con las optimizaciones de hardware subyacentes. Este precedente respalda este enfoque desde un punto de vista práctico.

Respecto a si una misma imagen debe rotarse sobre su eje o no, parece recomendable hacerlo, ya que múltiples experimentos de reconocimiento con aprendizaje profundo lo demuestran [104, 105]. Pero por otra parte, más información se aporta si en vez de rotar, por ejemplo, $k = 16$ veces una misma imagen desde un mismo eje j, la cámara se girara de forma simultánea al desplazamiento desde un eje j al siguiente (en la espiral) $j + 1$, insertando $k - 1 = 15$ ejes intermedios.

Para valorar el número de imágenes y las rotaciones de la cámara, se han construido 9 conjuntos de datos con 256, 512 y 1024 ejes, en los que la imagen proyectada gira sobre su eje una vuelta cada 10 imágenes, cada 30, y sin giro alguno. En [106] se muestra una animación de 4 moléculas y 1024 ejes, con los tres tipos de giros. Un resumen de los resultados del entrenamiento se muestra en la Tabla 4.3, aunque la descripción del parámetro elegido y una discusión de resultados se detallarán en la Sección 4.1.3.

Tabla 4.3: Comparativa de la precisión del reconocimiento entre números de imágenes y rotación de las proyecciones (Índice de simetría).

Nº Imágenes	Sin rot.	1 r./30ejes	1 r./10ejes
256	68.5 %	82.0 %	78.7 %
512	7.6 %	86.4 %	75.7 %
1024	74.1 %	85.8 %	82.9 %

Estos resultados demuestran que el caso óptimo se obtiene con un *dataset* de 512 imágenes y rotación lenta. Además, utiliza menos de la tercera parte de los recursos que con 1024, por lo que en lo sucesivo se utilizará este *dataset* como referencia. En la Figura 4.4 se muestran las proyecciones sobre 4 ejes consecutivos de la espiral esférica de Fibonacci de la molécula de Ethylhexyl methoxycryleno como ejemplo de las imágenes generadas en este *dataset*.

4.1.2. Computación híbrida del renderizado multi-imagen de las moléculas

La generación de las imágenes que entrenarán la red neuronal requiere un enorme esfuerzo computacional debido a que las bases de datos de moléculas tienen tamaños con

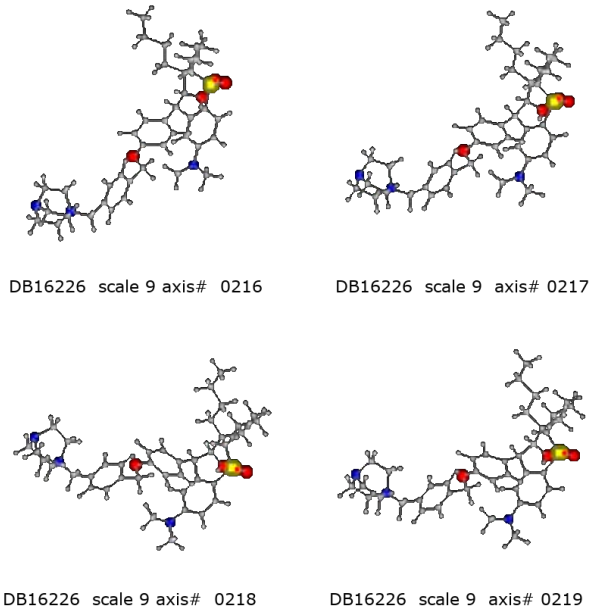


Figura 4.4: Molécula de Ethylhexyl methoxycryleno (Número de acceso DrugBank: DB11226), proyectada sobre 4 ejes consecutivos de la espiral esférica de Fibonacci.

cientos de miles de muestras que, en combinación con los cientos de proyecciones de cada una de ellas derivan en cientos de millones de imágenes que habrán de ser generadas con decenas e incluso cientos de operaciones trigonométricas cada una (dependiendo del número de átomos y enlaces de la molécula). Pero este problema es también un claro ejemplo de lo que se suele denominar *paralelismo embarazoso*, porque se puede descomponer en un elevadísimo número de tareas computacionales independientes de tamaño muy reducido, invirtiendo para ello muy poco esfuerzo de programación adicional. En particular, es obvio que la renderización de imágenes de dos moléculas diferentes se puede implementar con tareas paralelas independientes. Por otra parte, para cada molécula, la proyección en dos ejes diferentes es totalmente independiente por lo que el tamaño de grano del paralelismo que se puede llegar a alcanzar es, como mínimo, la renderización de cada imagen.

Por otra parte, los recursos computacionales disponibles, en la mayoría de los casos (ya sea incluyendo computación en la nube, los de un pequeño laboratorio, e incluso los ordenadores en domicilios privados) permiten no sólo la computación paralela en procesadores multinúcleo, sino también el uso de tarjetas gráficas y sus correspondientes GPUs [107].

Teniendo en cuenta las características del paralelismo sobre GPUs y procesadores multinúcleo, nuestra propuesta consiste en usar el siguiente modelo de paralelismo híbrido anidado:

- En un primer nivel de la jerarquía, las moléculas son asignadas a las GPUs y CPUs

disponibles, mediante un modelo de programación de Granja de Tareas. Es decir, cuando una CPU o GPU está libre, elige una molécula de la base de datos y la procesa.

- Si la tarea es asignada a una CPU, dicha CPU ejecuta la renderización en todos los ejes de proyección mediante un bucle, en el que se garantiza que todos los datos e instrucciones se han almacenado en el nivel L1 de cache, debido al tamaño relativamente pequeño de las moléculas, de la imagen y del propio algoritmo de proyección.
- Si la tarea es asignada a una GPU, entonces se implementa un paralelismo anidado en el que los *threads* de la GPU procesan en paralelo cada una de las proyecciones de la molécula. En particular, las operaciones que se realizan en cada *thread* son exactamente las mismas y en el mismo orden, aprovechando así al máximo la aproximación SIMD. En caso de utilizar un algoritmo de ordenación en el eje de proyección (para implementar correctamente la superposición de átomos), se utilizaría una técnica de ordenación de tiempo constante (p.e, radix sorting).

Optimizaciones al código paralelo

- El conjunto de ejes de proyección se calcula y es transferido a la memoria compartida de las GPUs antes de iniciar los cálculos.
- Las coordenadas de las moléculas son transferidas de forma compacta a la GPU.
- El resultado del renderizado de cada molécula en cada eje se compacta igualmente, con una lista ordenada de índices de moléculas y los píxeles que le corresponde.
- Las moléculas se renderizan con imágenes de 256x256 píxeles lo que permite indexar las coordenadas con un sólo byte. Al tener todas las moléculas menos de 256 átomos, el índice también ocupa un sólo byte.
- En una fase de posprocesado, la lista de moléculas y píxeles se convierte en una imagen utilizando exclusivamente las CPUs, debido a que su construcción en la GPU no compensaría el elevado coste de enviar la información al *host*.
- Los enlaces se dibujan en la fase de posprocesado, y sobre ellos, se dibujan los discos que corresponden a los átomos de la molécula.

4.1.3. Aprendizaje supervisado de la red neuronal

Como se menciona previamente, un posible objetivo del trabajo es la identificación de similitudes en la estructura espacial entre las distintas moléculas. Es por ello por lo que

se propone un clasificador basado en una red neuronal convolucional (CNN), que, desde su descubrimiento [108], se ha convertido en la estructura más eficiente y utilizada para la identificación de imágenes. Siguiendo con la similitud con el cerebro humano, que primero captaría la información de las imágenes y posteriormente se interesaría por la escala, se ha decidido incorporar este dato como entrada tras la fase de convolución; es decir, cuando las neuronas estén totalmente conectadas.

Dado que el conjunto de datos, creado según el procedimiento de la Sección 4.1.1, permite extraer gran cantidad de datos etiquetados, donde la etiqueta corresponde al índice de la molécula en la base de datos DrugBank de la que se ha obtenido cada imagen, se puede escoger un método de aprendizaje supervisado. Así, si se introduce como entrada una imagen bidimensional de una molécula cualquiera, se obtiene un vector de probabilidades de que dicha imagen corresponda a cada una de las moléculas con las que se ha entrenado la red.

La evaluación de la similitud entre moléculas se realiza mediante una matriz de confusión [109, 110], una técnica de aprendizaje automático muy utilizada para visualizar clases relacionadas y comparar probabilidades como puntuaciones de similitud.

Preprocesamiento del conjunto de datos generado

Considerando los resultados de la Sección 3.3.1 y siguiendo el mismo procedimiento, se decide evaluar el tiempo por época para todo el conjunto de datos durante 5 épocas para distintos tamaños de lote. De este estudio, mostrado en la Figura 4.5, se deduce que el tamaño de lote óptimo para reducir los tiempos de entrenamiento es el de 128, aunque ello conlleve a un aumento del peso de la canalización de entrada. Si se entrena para un gran número de épocas, podría ser conveniente cambiar a un tamaño del lote de 256, debido al mayor aumento de la precisión por época entrenada.

Estructura del modelo y parámetros relevantes

El tipo de red neuronal por excelencia para el reconocimiento de imágenes utiliza operaciones de convolución, aplicando diferentes filtros a la imagen de entrada. La estructura comprenderá, por tanto, esta parte convolucional para extraer las características fundamentales de la imagen y, posteriormente, tres capas totalmente conectadas para la clasificación en función de la clase (molécula) a la que corresponda. Justo antes de las capas totalmente conectadas, se introduce como entrada una información adicional de la imagen, la escala con la que fue creada.

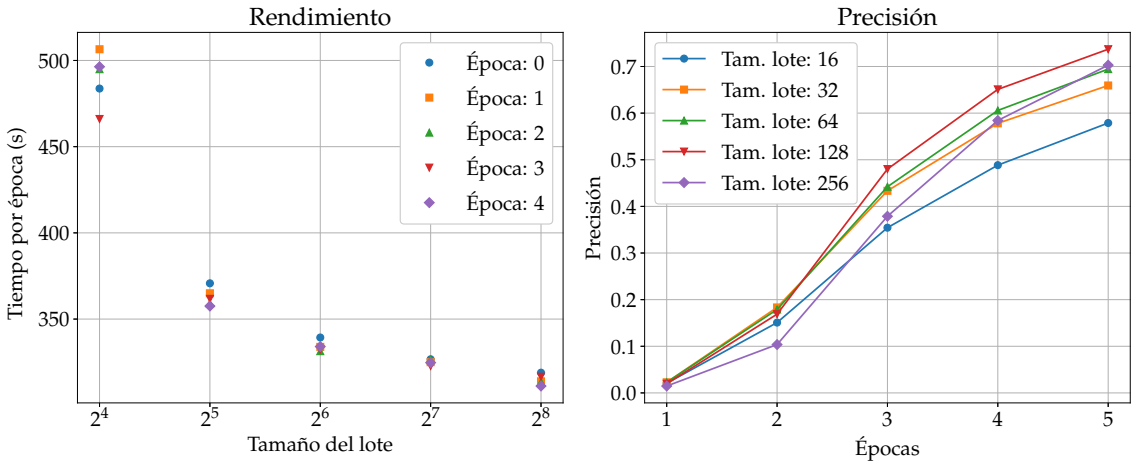


Figura 4.5: Evaluación de rendimiento en tiempo (izquierda) y precisión (derecha) en función del tamaño del lote.

Para comprobar si la red neuronal podía reconocer las moléculas, se tomaron como punto de partida algunas estructuras utilizadas para el reconocimiento de objetos 3D por multivista [89]. Estas estructuras se construyeron a partir de otras preentrenadas para las capas convolucionales, algunas de las cuales fueron ganadoras del concurso Imagenet, como las redes VGG [111], ResNets [81, 112, 113], y EfficientNets [114]. Se obtuvieron resultados prometedores con todas estas redes preentrenadas en términos de similitud recíproca entre moléculas, error y sobreentrenamiento. Sin embargo, los resultados no variaban mucho con respecto a arquitecturas mucho más simples en contraposición al coste computacional en cuanto a memoria (requería disminución de la precisión y del tamaño del lote) y tiempo (entrenamiento del orden de 5 veces más lento). Por tanto, y teniendo en cuenta la simplicidad de las imágenes, se optó por una red neuronal personalizada, la cual hemos denominado MolNet2D, en la que se comprobó que la configuración más eficiente es la que se enumera a continuación y se muestra en la Figura 4.6:

- El tamaño del filtro inicial de 9x9 captura bastante bien las agrupaciones de átomos. Se aplican reducciones de filtro sucesivas a mayor profundidad. Se añaden ceros a los bordes para mantener el tamaño.
- La tasa de abandono solo se incluye en las capas totalmente conectadas y se establece al 50 %.
- La normalización por lotes reduce el sobre-entrenamiento y es conveniente seguir las indicaciones de otros autores [80].
- Un número de filtros inicial de 16 es adecuado para capturar similitudes comunes entre moléculas. Se aplican más filtros a mayor profundidad.

- Es más eficiente aumentar el paso de los filtros que emplear las funciones de `Pooling` [115], excepto para la última capa de convolución que emplea un `Pooling` de media.

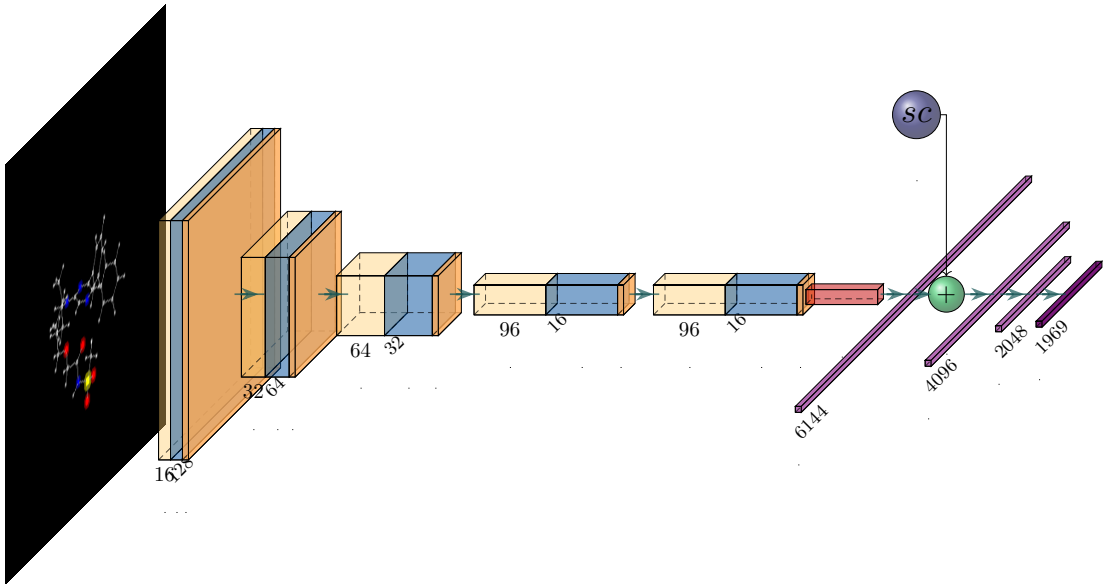


Figura 4.6: Estructura de la red neuronal. En amarillo: capas de convolución; en azul: normalización por lotes, en naranja: función de activación ReLU;en rojo: función de `Pooling`; en morado: capas totalmente conectadas.

Entrenamiento de la red neuronal

La finalidad del entrenamiento es reducir la función objetivo de una red neuronal con una serie de pesos $\mathbf{W}$,

$$J(W) = \frac{1}{B}\sum_{i=1}^{B} L\left(f\left(x^{(i)}; W\right), y^{(i)}\right), \tag{4.9}$$

donde B es el tamaño del lote, $f(x^{(i)}; \mathbf{W})$ es la predicción para una determinada entrada $x^{(i)}$, $y^{(i)}$ su etiqueta real, y L es la función del error. Para este problema de clasificación, se opta por el error de entropía cruzada categórica,

$$L = -\sum_{i=1}^{C} t_i \log(s_i), \tag{4.10}$$

teniendo en cuenta que la etiqueta utiliza una codificación *one-hot*,

$$y^{(i)} = [t_1, t_2, \ldots, t_C], \tag{4.11}$$

y que la predicción es un vector de las probabilidades de cada clase,

$$f(x^{(i)}; W) = [s_1, s_2, \ldots, s_C], \tag{4.12}$$

debido a que la función de activación de la salida es la función `softmax`,

$$s_i \leftarrow f(s_i) = \frac{\exp(s_i)}{\sum_{j=1}^{C} \exp(s_j)}. \tag{4.13}$$

El error se obtiene tras la entrada de un lote de imágenes y el cálculo de las predicciones con el paso hacia adelante de la información. A continuación, se calcula el gradiente del error $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ y el optimizador lo aplica para actualizar los pesos,

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}, \tag{4.14}$$

donde η es la tasa de aprendizaje.

Se escoge el optimizador `Adam` [116], ya que estabiliza el entrenamiento. Se han realizado varias pruebas con distintas tasas de aprendizaje y se ha llegado a la conclusión de que $\eta \sim 10^{-4}$ acelera el entrenamiento garantizando la estabilidad. Se registra la precisión como métrica interesante del entrenamiento.

Evaluación del clasificador

El conjunto de datos se divide en un subconjunto para entrenamiento (80 %) y en otro para evaluación (20 %). Durante cada época se mide el error de entropía cruzada de la ecuación (4.10) y la tasa de acierto o precisión (clase i con mayor porcentaje de salida s_i corresponde con $y^{(i)}$). Al final de la época se realizan las mismas mediciones para el subconjunto de validación y, así, comprobar que no existe sobre-entrenamiento o memorización del subconjunto de entrenamiento. Esta es una manera de comprender también el número de épocas con las que es necesario entrenar a la red. Un entrenamiento excesivamente corto hace que el clasificador no llegue a distinguir unas moléculas de otras y uno excesivamente prolongado aumenta el riesgo de sobre-ajuste.

Entrenando los 9 conjuntos de datos para el mismo número de épocas (6) y la misma estructura y configuración de entrenamiento mencionadas en las secciones anteriores, se obtiene el error de entropía cruzada en función del número de proyecciones por molécula y de la tasa de rotación aplicada que se muestra en la Figura 4.7. Se incluye tanto las mediciones del entrenamiento (líneas punteadas) como las de validación (en continua).

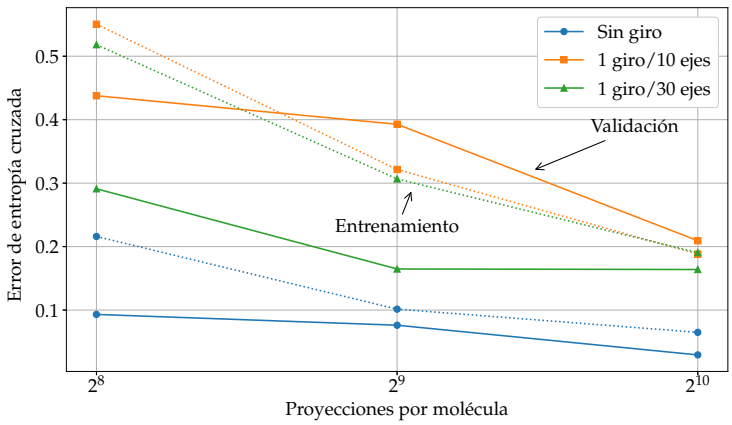


Figura 4.7: Error de entropía cruzada en función del número de proyecciones por moléculas para distintas tasas de rotación aplicadas.

La primera conclusión que se extrae de este análisis, es que una mayor tasa de rotación en las proyecciones empeora el rendimiento del clasificador, tanto en entrenamiento como en validación. Es evidente que, con rotación, las proyecciones se parecen menos entre sí al incluir un grado de libertad más, por lo que el clasificador tiene más dificultades para aprender. Cabe mencionar que, cuando las imágenes son más parecidas con respecto a las proyecciones anteriores y próximas (conjunto sin rotación), los subconjuntos de validación y test son más similares, por lo que este método, comúnmente utilizado para medir el sobre-entrenamiento, no es del todo representativo. Otro aspecto que llama la atención es que a mayor tamaño del conjunto de datos, mejores son los resultados del clasificador, como es de esperar del aprendizaje profundo. Por último, destacar que cuando el error de entrenamiento se encuentra muy por encima del de test, podría significar que se puede aumentar aún más el número de épocas de entrenamiento sin riesgo de sobre-ajuste, como se puede ver en los conjuntos de 256 y 512 proyecciones con rotación suave (en verde).

Por ello, se deduce a priori que el mejor conjunto de datos para la aplicación es el de 1024 proyecciones por molécula sin rotación. Sin embargo, un conjunto de datos tan grande implica un coste computacional mucho mayor, como se muestra en la Figura 4.8, por lo que un tamaño de 512 proyecciones resulta más conveniente. Además, como se mencionó en el párrafo anterior, el conjunto sin rotación podría no ser adecuado para ser auto-evaluado debido a la mayor similitud entre los subconjuntos de validación y entrenamiento, por tanto, a continuación se propone un método alternativo.

El principal objetivo de este caso de estudio es encontrar similitudes entre moléculas. Una de las maneras para realizar esta comparativa es crear una matriz de similitud Ψ, donde cada elemento $\psi_{i,j}$ es la media de las probabilidades $s_{i,j}$ de que la imagen de la molécula i sea una de las proyecciones de la molécula j. Se construirán 9 matrices $\Psi_k \; \forall \; k \in \{1, 2, \ldots, 9\}$

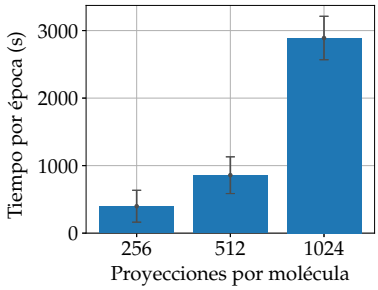


Figura 4.8: Tiempos por época para distintos tamaños del conjunto de datos, medidos en proyecciones por moléculas.

para los 9 clasificadores entrenados con los 9 conjuntos de datos, pero las probabilidades $s_{i,j}$ se calcularán a partir de los conjuntos de 1024 proyecciones con las tasas de rotación distintas a las del clasificador k. Esto permitirá detectar aún mejor el sobre-entrenamiento. Se considera que el clasificador muestra resultados más fiables cuando $\psi_{i,j} \simeq \psi_{j,i}$, es decir, cuando la matriz tiene una forma simétrica. Para medir la simetría de Ψ, se utiliza la expresión [117]

$$S_\Psi = \frac{\|\Psi_{sym}\| - \|\Psi_{anti}\|}{\|\Psi_{sym}\| + \|\Psi_{anti}\|}, \tag{4.15}$$

con

$$\Psi_{sym} = \frac{1}{2}\left(\Psi + \Psi^T\right) \tag{4.16}$$

y

$$\Psi_{anti} = \frac{1}{2}\left(\Psi - \Psi^T\right) \tag{4.17}$$

Por tanto, $-1 \leq S_\Psi \leq 1$, saturando por el límite superior cuando es completamente simétrica.

Es evidente que, cuanto más capaz sea el clasificador de identificar una molécula como la que realmente es, más contribuirá a la simetría de la matriz, pues los elementos estarán dispuestos en la diagonal principal. Por ello, se considera conveniente también medir el reconocimiento

$$R_\Psi = \frac{1}{C}\sum_{i=1}^{C} \psi_{i,i}, \tag{4.18}$$

para saber si la mayor simetría se debe a una matriz más diagonal. En la Figura 4.9 se comprueba la afirmación anterior y, además, se observa que para los conjuntos sin rotación el sobre-entrenamiento es muy acusado, especialmente para el de 512 proyecciones por molécula. Por tanto, finalmente se escoge el conjunto de 512 proyecciones con rotación suave (1 rotación por cada 30 ejes) como el conjunto óptimo para futuros estudios.

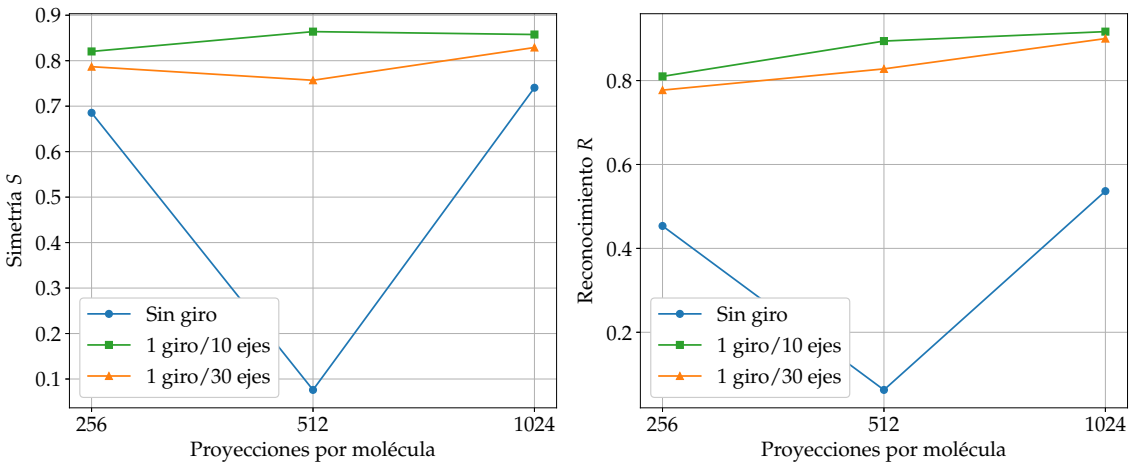


Figura 4.9: Simetría S_Ψ (arriba) y reconocimiento R_Ψ (abajo) de los 9 conjuntos de datos.

Evaluación del clasificador

Una vez seleccionado el conjunto de datos óptimo, se entrenaron varias redes troncales preentrenadas como extensión de MolNet2D para evaluar la fiabilidad de los resultados. Dado que no es posible obtener la verdad fundamental, aunque la similitud de forma entre moléculas puede estimarse utilizando algoritmos de optimización como Optipharm [73], se tomó la Matriz de Confusión Media ($\bar{\Psi}$), calculada a partir de los resultados de todos los experimentos de la red troncal, como caso de referencia. Para determinar si existían diferencias significativas entre las distintas redes neuronales, se calculó una distancia relativa ϵ como la diferencia entre cada Matriz de Confusión (Ψ) y la Matriz de Confusión Media ($\bar{\Psi}$) de la siguiente manera:

$$\epsilon = \frac{\|\Psi - \bar{\Psi}\|}{\|\bar{\Psi}\|} \tag{4.19}$$

Como se muestra en la Tabla 4.4, los resultados son relativamente similares, logrando un valor de ϵ incluso inferior al de la matriz de identidad, que correspondería a un clasificador perfecto. Aunque MolNet2D tuvo un rendimiento aceptable en cuanto a tiempo de evaluación del dataset, los resultados obtenidos mediante la red preentrenada efficientNetV2B0 parecen ser más similares a lo que se ha definido como verdad fundamental, sin aumentar significativamente el coste computacional.

En la práctica, se obtuvieron las mismas moléculas más similares independientemente de la red troncal o el conjunto de datos elegidos, pero con distintas probabilidades o porcentajes de similitud. Se compararon las similitudes obtenidas con las que indicaban otros algoritmos de optimización como Optipharm [73], utilizando el conjunto de datos con 512 proyecciones por molécula, tasa de rotación suave y la EfficientNetV2B0 como red troncal

Tabla 4.4: Comparativa entre las distintas redes troncales en términos de tiempo global de evaluación del conjunto de datos, simetría de la matriz, reconocimiento y distancia relativa con respecto a la Matriz de Confusión Media.

Red	**Tiempo (s)**	$\mathbf{S_\Psi}$	$\mathbf{R_\Psi}$	ϵ
MolNet2D	4026	0.866	0.894	0.120
VGG19 [111]	12858	0.861	0.875	0.125
efficientNetV2B0 [114]	6798	0.907	0.958	0.083
ResNet50 [81]	14787	0.871	0.927	0.094
ResNet50V2 [112]	9840	0.881	0.931	0.082
ResNetRS50 [113]	12192	0.889	0.954	0.107
Identity matrix	-	1	1	0.176

(Tabla 4.5). Se escogen tres moléculas de referencia para demostrar tres comportamientos distintos:

- Para la molécula DB01365, el caso más similar en forma (la salida con mayor probabilidad excluyéndose a sí misma) es la misma que la que indica Optipharm (DB00191).
- Para la molécula DB00728, la más similar según la red neuronal (DB04834) no es la misma que la obtenida mediante Optipharm (DB01339), aunque la red neuronal coincide en que es bastante similar.
- Para la molécula DB00320, son varias las moléculas que tienen una probabiliodad bastante mayor que la indicada por Optipharm (DB01413).

Tabla 4.5: Salidas de la red neuronal con mayor probabilidad comparadas con los resultados del algoritmo de Optipharm (Oph).

Molécula DB01365		Molécula DB00728		Molécula DB00320	
Oph	DB00191	**Oph**	DB01339	**Oph**	DB01413
Red Neuronal		**Red Neuronal**		**Red Neuronal**	
DB00191	9.25 %	DB00728	96.35 %	DB00320	57.49 %
DB01037	0,23 %	DB01336	0.13 %	DB00696	27.89 %
DB01365	87.28 %	DB01337	0.42 %	DB09238	0.24 %
DB01577	1.76 %	DB01339	1.14 %	DB11274	8.79 %
DB09571	1,06 %	DB04834	1.56 %	DB13345	4.56 %

Dado que se requiere la opinión de los expertos para determinar qué algoritmo es más representativo de la realidad, se muestra una comparativa gráfica en la Figura 4.10 para que

sea el ojo humano el que juzgue la similitud entre las moléculas. Los valores de similitud mostrados en la figura han sido reescalados para reflejar la correlación con respecto a la molécula original.

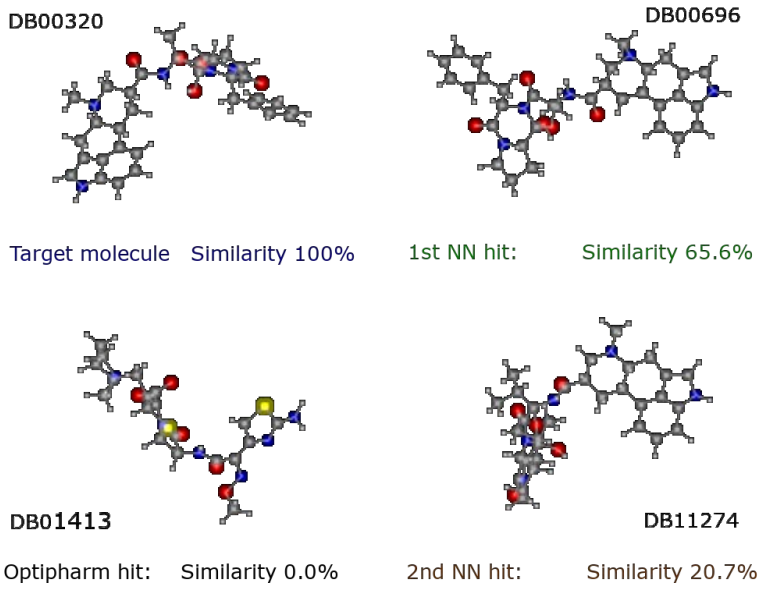


Figura 4.10: Similitudes con respecto a la molécula DB00320 según la red neuronal.

Para demostrar la relevancia de este enfoque innovador para el reconocimiento visual de objetos 3D, en particular en el descubrimiento de fármacos, se midió el rendimiento en tiempo utilizando todo el conjunto de datos DrugBank [75]. Se incluyeron 9213 moléculas, lo que significa que la capa de salida de la red neuronal debe tener este tamaño (número de neuronas). Esto aumenta necesariamente el número de parámetros de entrenamiento y provoca problemas de memoria en la GPU (NVIDIA GeForce RTX 3080). Para solucionar este problema, se redujo la resolución de las imágenes de proyección a 128×128 px, lo que sorprendentemente dio como resultado casi la misma respuesta de reconocimiento.

Basándose en los resultados del estudio, se creó un nuevo conjunto de datos a partir de las 9213 moléculas, que contenía 512 imágenes de proyección de 128×128 px cada una con una tasa de 1 rotación por cada 30 ejes. Utilizando la red troncal EfficientNetV2B0, cada época se completó en aproximadamente 21.000 segundos. Sin embargo, esta inversión de tiempo es justificable, ya que sólo debe realizarse una vez, tras lo cual la estructura y los pesos de la red neuronal pueden importarse en un código ejecutable para la predicción y el descubrimiento de similitudes entre moléculas. En la Tabla 4.6 se muestran los tiempos obtenidos con el programa, llamado *Molecule Finder*, cuando se ejecuta una vez. Se midieron y promediaron diez muestras aleatorias para tener en cuenta la variabilidad de los tiempos entre moléculas. Esto se repitió para diferentes números de proyecciones por molécula. Los resultados se muestran en la Figura 4.11.

Tabla 4.6: Muestra de tiempos medidos en el programa *Molecule Finder*.

Tarea	**Tiempo de cómputo (s)**
Importar Tensorflow	10,4
Construcción de la red	2,40
Carga de pesos	4,96
Generar 32 proyecciones aleatorias (1ª vez)	0,83
Predecir 32 proyecciones aleatorias (1ª vez)	7,38
Generar 32 proyecciones aleatorias (consecutivas)	0,20
Predecir 32 proyecciones aleatorias (consecutivas)	0,34

La generación de las proyecciones en tiempo real se consideró relevante por dos razones: ahorra al usuario la descarga de más de 4 millones de imágenes del conjunto de datos y evita el uso de las mismas imágenes para el entrenamiento. Basándonos en las barras de error de la Figura 4.11, se puede deducir que el rendimiento del clasificador no depende de la molécula, a diferencia de otros algoritmos de optimización, y puede alcanzar tiempos hasta 1000 veces más rápidos [73]. Además, se ha comprobado que su rendimiento es acorde con los resultados.

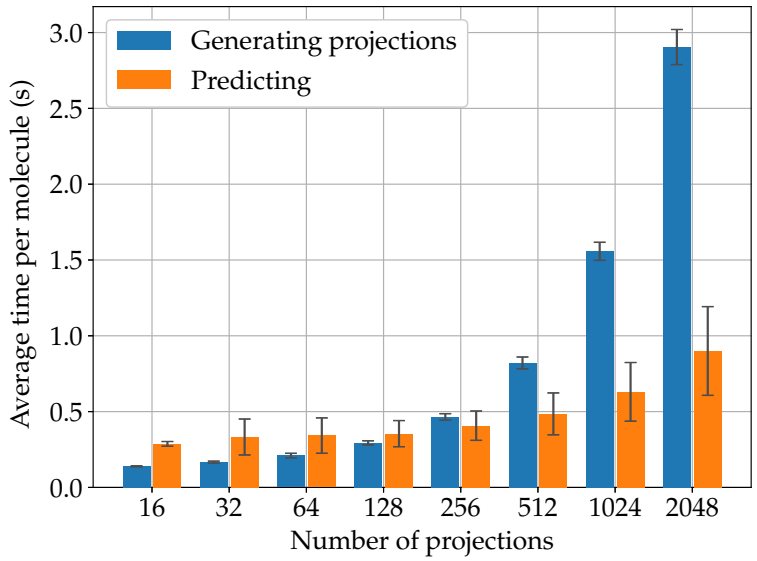


Figura 4.11: Tiempo de cómputo promediado de 10 moléculas aleatorias para distintos números de proyecciones.

4.1.4. Aplicaciones

A continuación se presenta la aplicación *MolFinder*, un motor de búsqueda molecular, desplegada en un servidor web, que permite a los usuarios introducir el código de una molécula perteneciente a una base de datos y calcular su porcentaje de similitud con respecto a otras moléculas de la misma base de datos.

La aplicación integra varios módulos interconectados para este fin:

- Una interfaz gráfica, desarrollada con PHP y JavaScript ES6, que permite al usuario introducir la molécula y configuración deseada, así como la visualización de los resultados.
- Un **generador de proyecciones** a partir de las coordenadas tridimensionales de los átomos de una molécula, desarrollado en C++ en forma de librería dinámica.
- Una **red neuronal** entrenada con Tensorflow de Python contruida a partir de la estructura descrita en la Sección 4.1.3 y despegada en un servidor mediante Flask.

Interfaz web

La interfaz del sitio web ha sido diseñada para enviar parámetros a la red neuronal, que se ejecuta en un servidor independiente. Muestra los resultados obtenidos en gráficos y un visor de moléculas obtenido de ChemDoodle, lo que permite a los expertos evaluar la fiabilidad de la aplicación. El uso de la aplicación es muy sencillo, como puede verse en las instrucciones de la aplicación de la Figura 4.12.

Únicamente hay que introducir las configuraciones deseadas: la **molécula** concreta dentro una **base de datos**, por un lado, y tamaño del lote o número de proyecciones, por otro. Se puede seleccionar las entradas de la red neuronal en la pestaña de entradas o deslizar el panel de configuración ubicado a la izquierda (véase la Figura 4.13). Se observan los mismos campos independientemente del método. Tras introducir la molécula y el número de proyecciones automáticamente hace una llamada a la API del generador de proyecciones para mostrarlas en el visor de la misma pestaña (Figura 4.14). Estas son las imágenes que serán usadas como entrada a la red neuronal. En la pestaña de cálculos, mostrada en la Figura 4.15, se pueden consultar los resultados obtenidos de la red neuronal. El valor de similitud se muestra en el gráfico de la derecha, donde se puede ver el porcentaje de similitud de las moléculas más parecidas según la red neuronal. El porcentaje de duda que se muestra arriba indica la precisión de la red neuronal. La red neuronal proporciona la probabilidad de que las imágenes de entrada sean las proyecciones de cada una de las

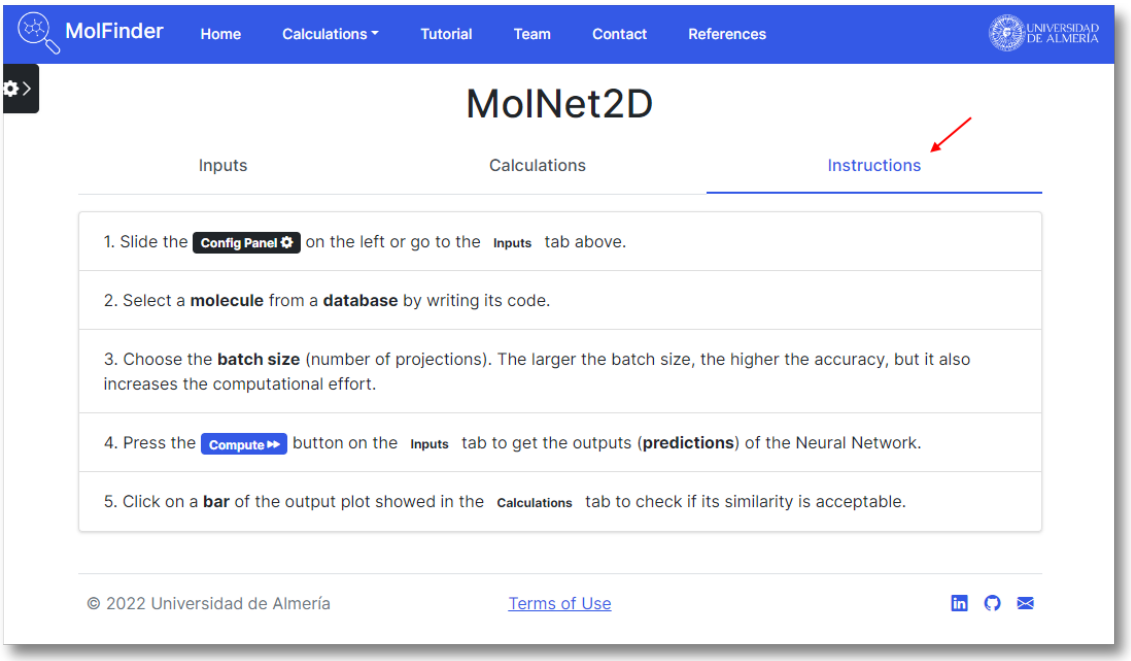


Figura 4.12: Instrucciones de uso de la aplicación en la interfaz web.

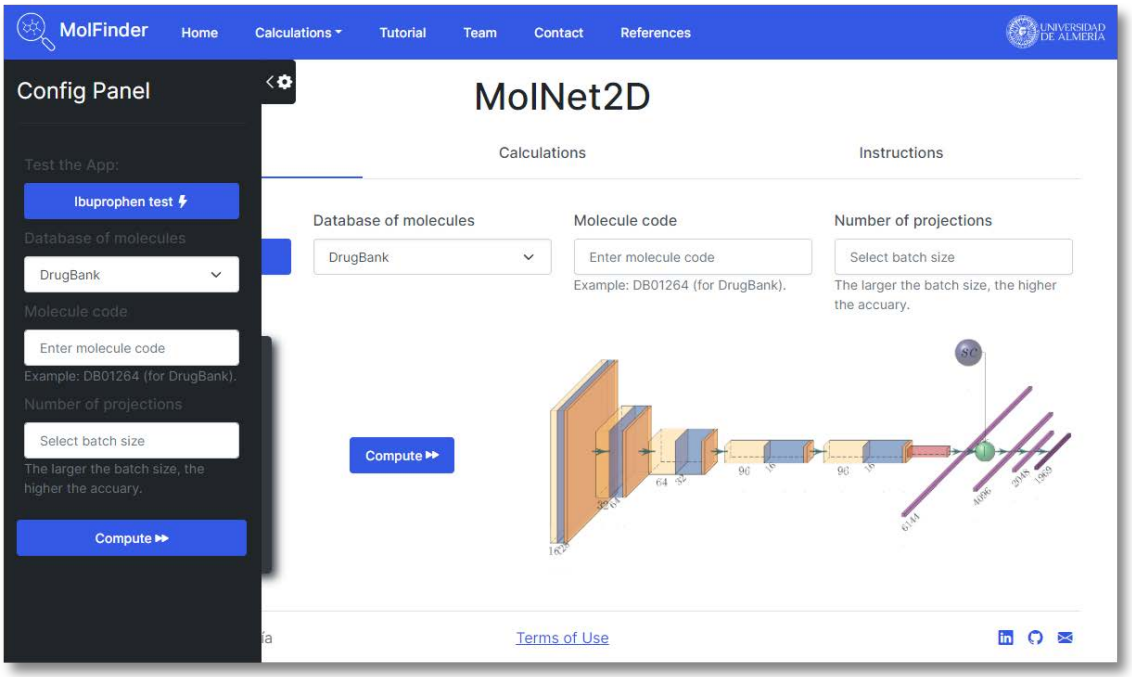


Figura 4.13: Configuración de usuario para las entradas a la red neuronal.

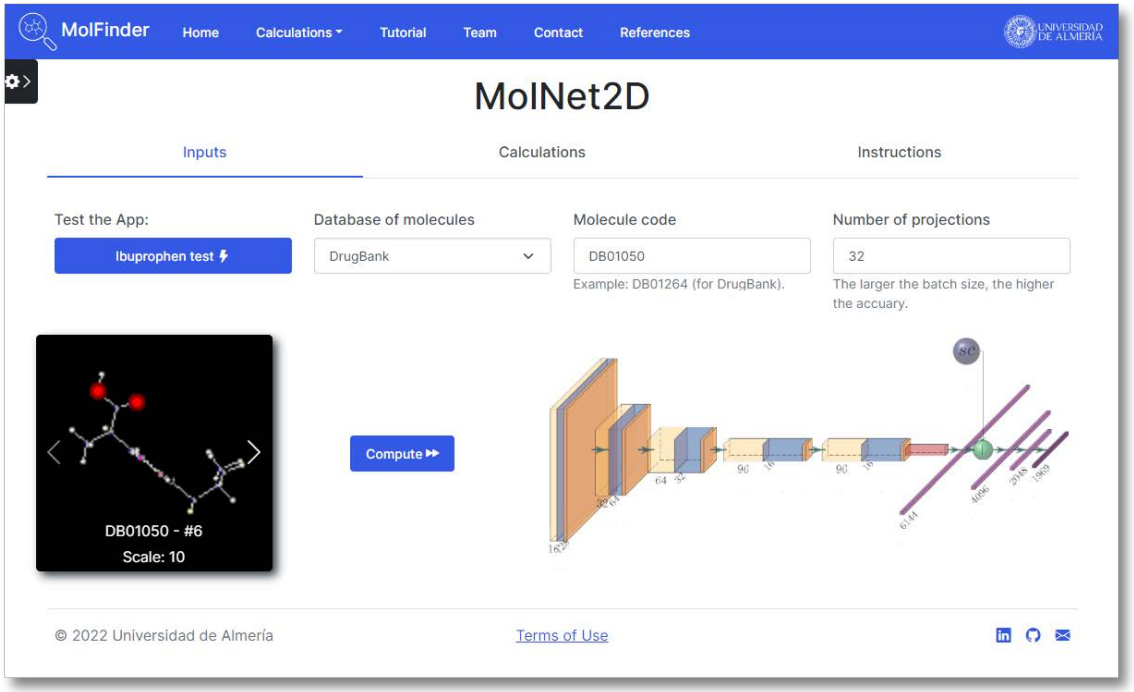


Figura 4.14: Proyecciones generadas dada una molécula y un número de proyecciones.

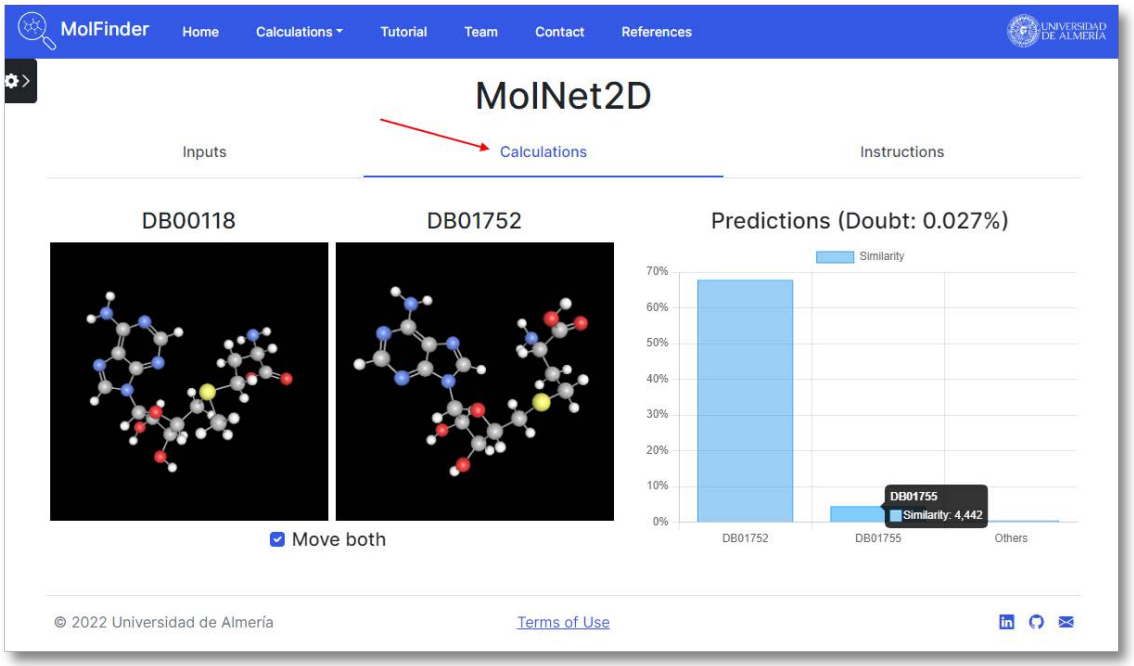


Figura 4.15: Visualización de los resultados obtenidos por la red neuronal.

moléculas entrenadas. Para obtener el valor de similitud, cada probabilidad se reescala utilizando ese porcentaje de duda. Cuanto mayor sea la duda, más dificultad tiene la red neuronal para distinguir las moléculas entre ellas. Esto también podría indicar que se tiene un alto grado de similitud entre las moléculas de salida.

A la izquierda se incluye un visualizador de moléculas en 3D (del software ChemDoodle) para evaluar la similitud real. Se puede interactuar con los gráficos para reorientar cada molécula. Al marcar la casilla "mover ambas", las dos moléculas se moverán simultáneamente. De forma predeterminada, la primera molécula corresponde a la que se introdujo en la red neuronal y la segunda molécula es la que tiene el mayor porcentaje de similitud, aunque la segunda molécula cambiarse seleccionando cualquiera de las barras del gráfico de la derecha.

Generador de proyecciones

La distribución de puntos sobre una esfera se realiza de varias formas. Por ejemplo, utilizando una espiral de Fibonacci, el código utilizado para calcular n ejes se muestra a continuación:

Código 4.1: Código de generación de ejes de proyección en la espiral de Fibonacci

```
int stepsPerLap=30; // Velocidad de rotacion
double twistStep=2*pi/stepsPerLap;
double golden = (1.0 + sqrt(5.0)) / 2.0;

for (int i = 0; i < n; i++) {
    double the = 2.0 * pi * i / golden;
    double phi = acos(1-2*(i+0.5)/n);
    double sphi = sin(phi);
    x = sphi * cos(the);
    y = sphi * sin(the);
    z = cos(phi);
    double a=i*twistStep;
    x_axes[i]={x, y, z, a};
}
```

El código anterior genera un conjunto de ejes distribuidos uniformemente sobre la superficie de una esfera, garantizando que las proyecciones de las moléculas estén equidistribuidas desde distintos ángulos. Los ejes son almacenados en una estructura de datos que incluye las coordenadas tridimensionales y el ángulo de rotación, como se muestra a continuación:

Código 4.2: Estructura de almacenamiento de ejes de proyección

```
typedef struct PointXYZA {
    double x,y,z,a;
}PointXYZA ;

PointXYZA *x_axes = new PointXYZA[n];
// "x" se refiere a "extendido".
```

Se han desarrollado tres herramientas principales para la generación masiva de imágenes y la creación de proyecciones moleculares.

cudamol: Generador masivo de imágenes Esta herramienta genera cientos de imágenes (512 proyecciones) de todas las moléculas disponibles en la base de datos. Ha sido implementada en CUDA y principalmente se emplea para la generación de un conjunto de datos completo para el entrenamiento.

mol2image: Generador de imágenes de una sola molécula Esta versión simplificada genera vistas aleatorias de una molécula y se utiliza principalmente para generar nuevas proyecciones con las que no haya sido entrenada la red.

El uso básico de esta herramienta es:

`mol2image [-v -Q -H -d` *`database`* `-s` *`seed`*`] molecule_file [number=8]`

mol2image: Librería dinámica para generación de imágenes La librería dinámica permite la generación de imágenes de moléculas desde sitios web implementados en Plesk o PHP, y está disponible en los siguientes formatos:

- En Linux: `libmol2image.so`
- En Windows: `mol2image.dll`

Red neuronal entrenada

La red neuronal se despliega en un servidor con Flask y Python. Se ha optado por esta tecnología frente a un servidor de NodeJS debido a que en Python existe una mayor variedad de tipos de capas, permitiendo implementar arquitecturas más complejas como

la EfficientNetV2B0 que ofrecía los mejores resultados para este clasificador de moléculas. En este código principalmente se realizan las siguientes operaciones:

- **Configuración del entorno de GPU:** Se establece la configuración inicial para utilizar dispositivos GPU específicos, lo que mejora el rendimiento del cálculo intensivo.
- **Carga de TensorFlow:** Se inicializa el entorno de TensorFlow y se verifica la disponibilidad de las GPUs físicas en el sistema.
- **Inicialización de la red neuronal:** Se construye la red neuronal entrenada con un conjunto de datos de 9213 moléculas.
- **Carga de pesos preentrenados:** Los pesos de un modelo preentrenado son importados desde un archivo local.
- **Generación de proyecciones aleatorias:** Se ejecutan proyecciones aleatorias de una molécula específica. Este proceso utiliza la librería dinámica, mencionada anteriormente, que genera las imágenes necesarias para el análisis.
- **Predicción de la molécula:** Las imágenes generadas son procesadas por la red neuronal, que aplica una activación *softmax* para obtener las predicciones de similitud con otras moléculas en la base de datos.
- **Cálculo de la similitud:** La red neuronal devuelve un vector de probabilidades que indica la similitud de la molécula procesada con las moléculas conocidas, normalizando los resultados para obtener un valor de similitud del 100 % para la molécula original.
- **Respuesta de la API:** El servidor web, implementado con Flask, permite a los usuarios cargar moléculas y obtener predicciones directamente a través de solicitudes HTTP.

4.1.5. Publicaciones

Los resultados expuestos en este apartado (4.1) han sido publicados en el *Journal of Optimization Theory and Applications*, Q2 en la categoría *Applied Mathematics* [3]. También han sido aceptados y presentados varios trabajos relacionados al congreso EUROPT 2022, celebrado en Lisboa [8], y a las Jornada Sarteco 2022 de Alicante [13].

4.2. SkewEngine

Existen problemas, en muchos y muy diversos campos, en los que es necesario realizar un cálculo extremadamente intensivo sobre cada uno de los puntos de una malla de datos 2D o 3D, como por ejemplo, sobre cada uno de los píxeles de una imagen, sobre cualquier punto de un mapa, o en los datos de una resonancia magnética.

En algunos casos, la intensidad aritmética es tan alta, que hace que el problema sea intratable, desde el punto de vista computacional. Se propone como ejemplo el cálculo de la cuenca visual de un punto específico, en un modelo digital de elevaciones (DEM). Se toma un escenario en el que el objetivo implica determinar la extensión observable de un territorio dado desde una ubicación designada dentro de ese territorio, como se muestra en la Figura 4.16.

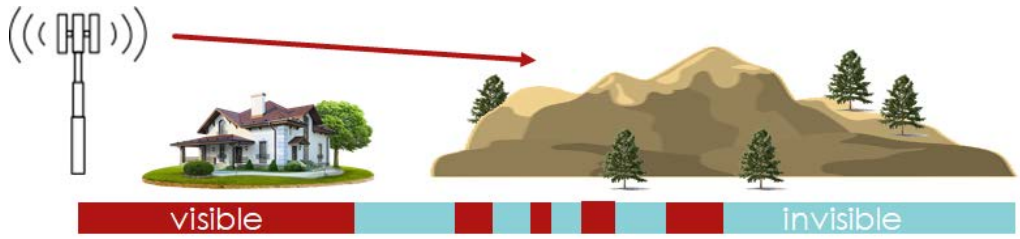


Figura 4.16: Ejemplo autoexplicativo de la cuenca visual 1-D vista desde una torre.

Como es obvio, para saber si desde un punto A se puede ver B (donde B puede ser cualquier otro punto de la geografía considerada), habría que tener en cuenta la altura del resto de puntos del modelo, ya que cualquiera, a priori, puede ser un obstáculo para la visión. Así, en un DEM con $N = \dim_x \times \dim_y$ datos, la complejidad del problema, usando la notación big-O, sería de orden $O(N^2)$, aunque es evidente que dicha complejidad se puede reducir si sólo se consideran como obstáculos los puntos C que están en la línea A-B. Aún así, la complejidad del problema, $O(N^{1,5})$, es bastante alta. La conocida aplicación Google Earth, por ejemplo, tarda varios segundos en obtener un resultado aproximado, como el que se muestra en la Figura 4.17.

Cuando se desea calcular la cuenca visual no sólo ya de un observador situado en un lugar concreto de un territorio, sino la que se obtendría desde una trayectoria arbitraria que transcurre por el terreno, o la que se observaría desde una región, o incluso desde todo el territorio, donde cualquier punto de un DEM se convierte en observatorio, la complejidad se dispara a $O(N^{2,5})$. Incluso con baja precisión, se requieren meses de CPU para un cálculo de un modelo sencillo [74].

Afortunadamente, este tipo de problemas pertenece a una categoría en la que los parámetros de estudio están afectados por un decaimiento (de la relevancia con respecto al

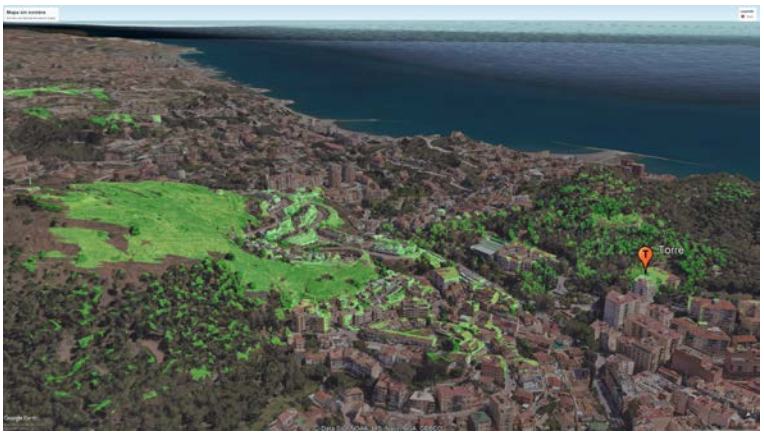


Figura 4.17: Cuenca visual obtenida con Google Earth desde un punto de la ciudad de Málaga.

cálculo) que depende de la distancia geométrica en la malla de datos, ya sea lineal o cuadrático, siendo esto último lo más frecuente. Son esos problemas en la que la influencia de un punto alejado, no afecta, o al menos inmediatamente, a lo que sucede en el otro extremo de la geometría. Este tipo de problemas se suele simplificar mediante un procesamiento en un conjunto discreto de direcciones radiales respecto a un punto de estudio, ya que los radios están más próximos en el punto de origen, y por tanto, la malla es más fina en el punto de estudio.

Sin embargo, en este tipo de problemas, donde la localidad espacial de la información es un factor crítico, existe la necesidad de trasladar dicha localidad espacial a los propios datos que representan a la información, y muy en particular, al almacenamiento en memoria de los mismos. Se propone un ejemplo para explicar este punto. Supóngase que en la imagen de la Figura 4.18 (izquierda) se desea aplicar un filtro en todas las direcciones paralelas a la de la flecha negra. Es evidente que cualquier algoritmo de cierta complejidad de los que suelen aplicarse en gráfica computacional, como la FFT, sería mucho más eficiente si los datos estuvieran alineados en memoria, como ocurriría en la imagen deformada de la derecha.

En este escenario, la localización espacial de los datos contribuye significativamente a la optimización, lo que permite un cálculo más eficiente gracias a la estructura de memoria alineada. Esta alineación facilita el funcionamiento de algoritmos computacionalmente exigentes, mejorando en última instancia la eficiencia de las operaciones de procesamiento de datos en problemas que dependen del espacio.

La propuesta de este trabajo consiste en analizar y explotar los beneficios de una reorganización en memoria, pero sobre todo demostrar que, en problemas de muy elevada complejidad, la reorganización de la información mediante una interpolación “distorsiona-

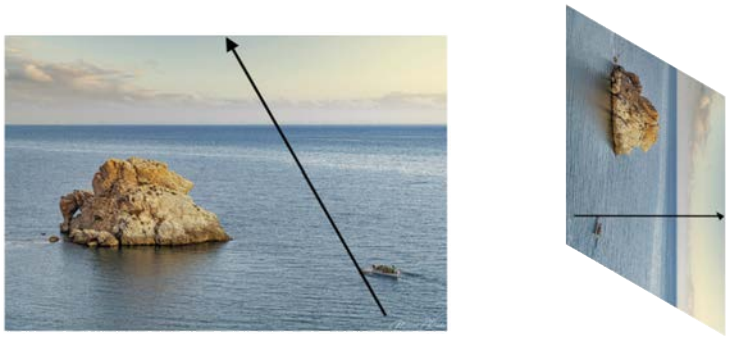


Figura 4.18: Reorganización de datos con sesgo.

da" de los datos es extremadamente beneficiosa.

Los algoritmos de malla estructurados y no estructurados son una estrategia popular para resolver problemas en una amplia gama de aplicaciones, desde el mapeo de texturas hasta la mecánica de fluidos computacional, y a menudo están restringidos por la sobrecarga de computación y memoria. Por un lado, mejorar la organización en memoria para cálculos intensivos es un componente fundamental de la computación de alto rendimiento y el procesamiento paralelo. Por otro lado, optimizar el diseño de la memoria implica estructurar estratégicamente los datos en ella para reducir el tiempo dedicado al acceso y maximizar la proximidad de los datos relevantes, lo que lleva a mejoras significativas en la velocidad y la eficiencia computacional. El trabajo presentado aquí no es el primero que reescribe una malla para optimizar el acceso a la memoria [118–120]. Y, en el ámbito geográfico, tanto el software QGIS [121] como el CDO [122] ofrecen herramientas de línea de comandos para rotar y reorganizar las mallas de conjuntos de datos geográficos. Sin embargo, a través de la investigación sistemática y la evaluación empírica, esta línea de investigación busca establecer que la implementación de la interpolación de datos sesgados en memoria ofrece mejoras sustanciales en la eficiencia computacional y la eficacia en la resolución de problemas. En problemas marcados por la intensidad computacional, esta técnica resulta altamente beneficiosa, optimizando la utilización de los recursos de memoria y acelerando el cálculo al explotar las relaciones espaciales inherentes dentro de los datos.

4.2.1. Antecedentes: El algoritmo sDEM

Tabik *et al* (2013) [74] publican un algoritmo que considera la dependencia radial en el cálculo de la cuenca visual, para crear un algoritmo que la calcula para todos los puntos de un modelo digital, en un conjunto discreto de direcciones alrededor de cada punto ($s = 360$, normalmente). Es decir, para cada sector angular de un grado, sólo se tienen en cuenta los puntos en el eje central del sector, reduciendo la complejidad del problema de

$O(N^{2,5})$ a $O(s \cdot N^{1,5})$. Esta optimización algorítmica mejora significativamente la eficiencia computacional y, al mismo tiempo, preserva un nivel significativo de precisión en el cálculo de la cuenca visual en todo el DEM. El trabajo de Tabik *et al.* (2013) [74] es un avance fundamental que muestra cómo las estrategias mencionadas, como la dependencia radial y el análisis direccional discreto, pueden producir mejoras sustanciales en la solución de problemas espaciales complejos.

Además, mediante una simple inversión de los lazos, en lugar de procesar todos los puntos del DEM con un bucle externo y luego calcular la cuenca visual unidimensional en todos los sectores con el bucle interno, el código aprovecha que todos los datos están ya alineados en una determinada dirección para hacer el cálculo de la cuenca visual desde todos los puntos de la línea en esa dirección. En la Figura 4.19, las imágenes de la fila inferior muestran cómo, al intercambiar los lazos, se aplica el algoritmo a todos los puntos, aprovechando que se encuentran alineados en la misma dirección del lazo exterior. En la fila superior, se han elegido cuatro puntos del territorio, y se ha calculado la cuenca visual en 4 direcciones alrededor de los puntos, mientras que en la fila inferior, para cada una de las cuatro direcciones de cálculo se calcula la cuenca visual a los puntos de estudio. Se observa que algunos puntos se aprovechan del alineamiento de los datos empleados en otros cálculos.

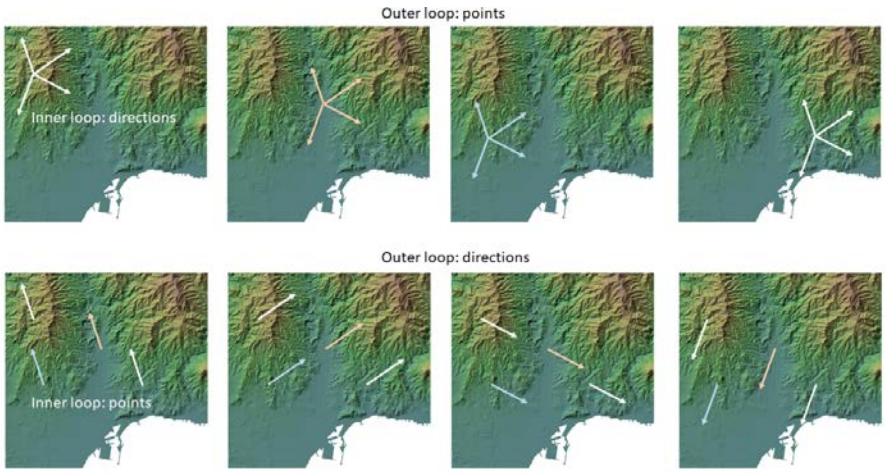


Figura 4.19: Intercambio de bucles. En la fila superior, para cada punto, se calcula la visibilidad en todos los sectores. En la fila inferior, para cada sector, se calcula la visibilidad 1-D para todos los puntos, aprovechando el alineamiento de los datos.

Almacenamiento sesgado de los datos

Recientemente, Sánchez-Fernández, A. *et al* (2021) [123] publican una importante modificación del algoritmo en el que se consideran dos aspectos claves para mejorar aún más el rendimiento:

1. Explotación de la localidad espacial: Dado un sector específico, como solo se utilizan los puntos restantes dentro de la misma línea, la propuesta sugiere alinear los datos correspondientes a lo largo de la misma línea de memoria. Este enfoque aprovecha la localidad espacial, minimizando los retrasos en el acceso a la memoria y optimizando el procesamiento de datos.

2. Paralelismo y uso de GPU: el algoritmo modificado aprovecha el paralelismo al considerar un sector en el que solo se utilizarán puntos a lo largo de la misma línea (eliminando así las dependencias entre líneas). Esto implica procesar simultáneamente todas las líneas, aprovechando las capacidades de las GPUs para lograr una mayor eficiencia computacional.

Al abordar estratégicamente estos aspectos, el algoritmo revisado muestra un avance notable, que demuestra el potencial de mejoras sustanciales en la gestión de la memoria y la velocidad computacional. La integración de la localización espacial y el procesamiento paralelo, junto con la utilización de la GPU, representa un enfoque sofisticado para abordar problemas espaciales complejos con mayor eficiencia y eficacia.

Nace así el algoritmo sDEM (de skew-DEM, modelo sesgado de elevaciones), que se basa en la idea de colocar los datos en memoria de una forma apropiada para el cálculo en una determinada dirección. El coste asociado a la "deconstrucción y reconstrucción del mapa" se justifica plenamente dado el incremento en la eficiencia computacional que se logra al realizar cálculos intensivos sobre los datos en su estado "sesgado".

4.2.2. Un motor para optimizar los datos en memoria

La propuesta de este estudio en esta tesis consiste en generalizar esa idea y extenderla a cualquier algoritmo, mediante una plantilla que sea independiente del algoritmo. Para ello, se propone la herramienta *skewEngine*: un código encapsulado en una clase C++ diseñado para simplificar los aspectos más complejos y laboriosos de esta gestión óptima de la memoria y del que puedan hacer uso los numerosos algoritmos que trabajen en problemas similares. En concreto, el motor sería el encargado de la reorganización de los datos de mallas regulares para que estén alineados en memoria (considerando la necesaria interpolación), y que, al final del algoritmo, reubique los datos a su ubicación original, mediante una interpolación en sentido inverso. En particular, la clase está diseñada para considerar la siguiente secuencia de etapas:

1. Lectura de la imagen o el modelo.

2. Preparación de datos para cada dispositivo (CPU o GPU). Se discute en la Sección 4.2.3.

3. Puesta en marcha de los hilos necesarios.

4. Puesta en marcha del iterador (por ejemplo, sobre 360 direcciones). Cada iteración es asignada a un dispositivo. Así, cada CPU/GPU recibirá varios sectores sobre los que va a realizar distintas operaciones:

   a) Preparación de datos (interpolación *skew*).

   b) Procesamiento en CPU o GPU.

   c) Restauración de datos (interpolación *deskew*).

5. Recolección de resultados de los dispositivos.

Se propone asimismo que la clase (skewEngine) sirva para cualquier tipo de datos (enteros, flotantes, double, o píxeles, por ejemplo), utilizando C++ *templates*, y que se encargue de todo lo relacionado con la preparación de los datos y la recolección de resultados.

Operaciones de la clase SkewEngine

Esta clase, en concreto, realizará sus operaciones en las siguientes etapas:

- Etapa 3: Se creará un objeto `skewEngine` en cada hilo, al que le corresponde un dispositivo de cálculo. Dichos objetos se denominarán "motores".
- En la etapa 4.a, el motor se encarga de "sesgar" los datos de entrada.
- En la etapa 4.b, una función externa aplica el algoritmo supuestamente costoso:

  ```
  skewOutput = FuncionCostosa(skewInput);
  ```

  siendo, por ejemplo:

  ```
  FuncionCostosa = cuencaVisualTotal-1D
  FuncionCostosa = radonTransform-1D
  FuncionCostosa = Identity
  ```

- En la etapa 4.c, el motor se encarga de "reparar" los resultados sesgados.
- En la etapa 5, una sección crítica recupera la información de los motores, y los destruye.

Obsérvese que, entre las funciones costosas elegidas, se puede implementar una función identidad de coste cero. Dicha función cumpliría un triple propósito. En primer lugar desempeña un papel en la depuración, dado que un volumen de datos deconstruido y posteriormente reconstruido debería conservar una correspondencia casi exacta con el original. En segundo lugar, permite identificar los errores de redondeo implicados los procesos de interpolación. Finalmente, facilita la estimación del sobrecoste derivado de la reorganización de datos y, en consecuencia, la evaluación de la conveniencia de utilizar el algoritmo.

4.2.3. Implementación de skewEngine

Consideremos una imagen o mapa 2D (todo será extrapolable a 3D, de forma anidada, como se explica más adelante) a la que queremos aplicar el algoritmo. En primer lugar, hay que tener en cuenta que:

- Los datos alineados se pueden procesar en doble sentido, por lo que sólo se considerarán 180 direcciones en los cálculos de 1° de precisión.
- Los 180 sectores se clasifican en cuatro bloques, con el objeto de hacer el algoritmo más simple y eficiente. Los límites de los cuatro bloques dependen de la relación de aspecto de los datos de entrada. En particular, la variable $\alpha = \text{atan}(\text{dim}_y/\text{dim}_x)$ determina que:

```
set0=[0,$\alpha$),
set1=[$\alpha$,90),
set2=[90,180-$\alpha$),
set3=[180-$\alpha$,180)
```

La Figura 4.20 muestra, para una imagen, 4 ángulos de cada uno de los conjuntos.

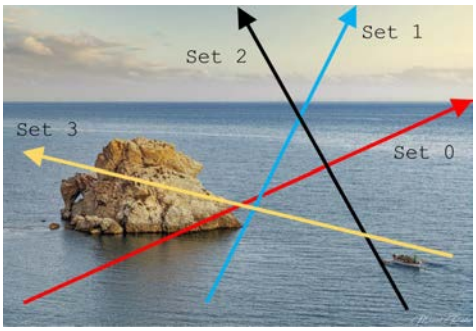


Figura 4.20: Cuatro ángulos correspondientes a diferentes conjuntos.

Antes de proceder al sesgado, se preparan 4 versiones de la imagen o modelo de entrada, a las que se podría denominar Normal-Normal (`input0`), Transpose-Normal (`input1`), Transpose-Mirror (`input2`) y Normal-Mirror (`input3`), y que se corresponderán con las entradas que necesitaría el algoritmo para los ángulos de cada uno de los conjuntos de sectores. Estas variaciones se observan en la Figura 4.21.

Figura 4.21: Transformaciones de imagen correspondientes a los 4 ángulos de los diferentes conjuntos.

Con estas 4 versiones del modelo, el algoritmo no tiene que distinguir entre signos para la tangentes, ni tipos de acceso, ni está limitado a modelos cuadrados. El procesamiento es similar en los cuatro casos. Consiste, básicamente, en sesgar los datos de entrada, mediante un mecanismo simple de interpolación, representado en la Figura 4.22.

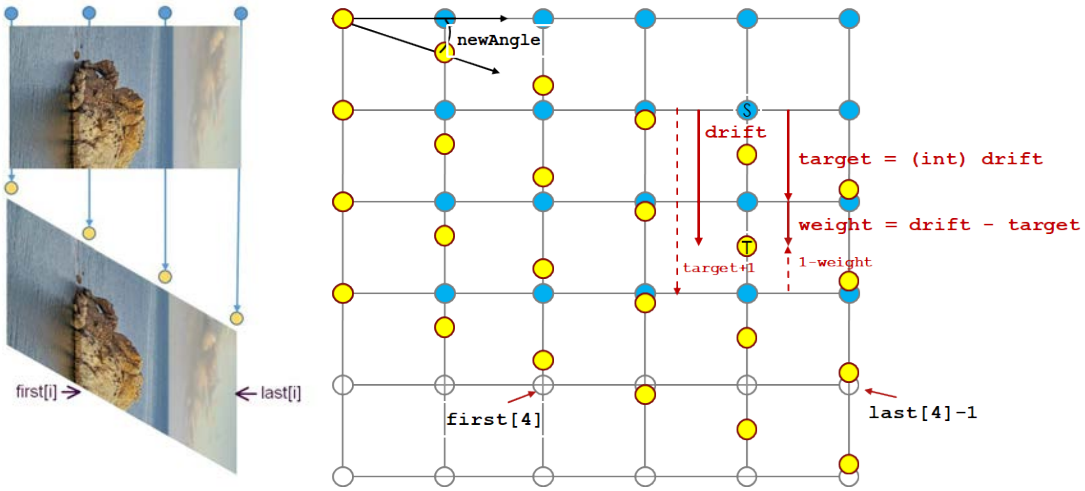


Figura 4.22: Representación gráfica de los parámetros necesarios para sesgar un modelo. En azul, los datos originales; en amarillo, los datos tras sesgar o reconstruir el modelo.

En concreto, será necesario calcular varios parámetros simples, como el ángulo de sesgo (`newAngle`), las dimensiones verticales y horizontales de la correspondiente versión de entrada (dim_o, dim_i) (subindexados con o por *outer*, i por *inner*), el sesgado (`skewness` = tan(`newAngle`)), y el desplazamiento vertical de la última columna (`offset` =

$\dim_i \cdot$ `skewness`).

Hay que tener en cuenta que al sesgar los datos de entrada (al pasar del rectángulo al romboide), cada elemento de entrada se va a ubicar en la misma columna, pero en un lugar intermedio entre dos filas del array destino. Por ese motivo, necesitamos calcular un par de vectores que sólo dependen del índice de columna: `target` y `weight`, siendo `target` el número de filas que avanza un determinado punto hacia abajo (redondeado a inferior), y `weight` un factor de ponderación, según la ubicación sesgada del punto, entre `target` y `target` + 1. En la misma figura se representan en rojo dichos parámetros, y que son independientes de la fila.

Asimismo, se calculará el número de filas que se van a procesar sobre el modelo sesgado, `skewHeight`, así como los límites de cada fila (`first[`i`]` y `last[`i`]`), que dependen de `newAngle`, `offset`, y de $\dim_i$, $\dim_o$ (Algoritmo 4.1). En resumen, estos dos vectores, definen la forma del romboide, como se muestra en la Figura 4.22, a la izquierda. Por último, el modelo es sesgado (Algoritmo 4.2).

Algoritmo 4.1 Cálculo de los límites del sesgado.

```
1: for i ← 0 to skewHeight − 1 do
2:     first[i] ← 0
3:     last[i] ← dim_i
4:     if i < offset then
5:         last[i]← (i + 1)/skewness + 1
6:     end if
7:     if i > dim_o then
8:         first[i] ← (i − dim_o)/skewness + 1
9:     end if
10: end for
```

Algoritmo 4.2 Sesgado del modelo.

```
1: for i ← 0 to skewHeight · dim_i do
2:     skewInput[i] ← 0
3: end for
4: source ← isTranspose?(isMirror?input2 : input1) : (isMirror?input3 : input0)
5: for i ← 0 to dim_o − 1 do
6:     for j ← 0 to dim_i − 1 do
7:         row ← i + target[j]
8:         skewInput[row · dim_i + j] ← (1,0 − weight[j]) · source[dim_i · i + j]
9:         skewInput[(row + 1) · dim_i + j] ← weight[j] · source[dim_i · i + j]
10:    end for
11: end for
```

Computación intensiva tras el sesgado

Una vez preparados los datos, el algoritmo intensivo consiste en un bucle simple que se puede ejecutar con paralelismo anidado (y embarazoso), ya que cada fila es independiente, como se muestra en el Algoritmo 4.3. Es importante no olvidar que el espacio se ha deformado, por lo que las distancias que se apliquen en el algoritmo (si se necesitan) tienen un factor de escala 1 en vertical y 1/cos(`newAngle`) en horizontal. Por tanto, la interpolación de *skewEngine* es bilineal.

Algorithm 4.3 Ejecución del *kernel*.

1: **for** $i \leftarrow 0$ **to** skewHeight $- 1$ **do**
2: skewOutput$[i] \leftarrow$ kernel(skewInput$[i]$, first$[i]$, last$[i]$)
3: **end for**

Reducción

Finalmente, cuando una CPU o GPU termina de procesar los sectores que le han sido asignados, el proceso de interpolación de los resultados se simplifica considerablemente, ya que no es necesario calcular nuevos parámetros. A continuación, el algoritmo tiene que pasar a la fase crítica de reducción. El correspondiente *thread* puede tener datos de sectores pertenecientes a más de uno de los cuatro conjuntos, por lo que se apoya en cuatro variables booleanas. En esta etapa, los datos traspuestos o en espejo recuperan la ubicación original.

La implementación de *skewEngine* se ha realizado en lenguaje C++, utilizando la librería OpenMP para la distribución de las direcciones de procesamiento entre los distintos núcleos. Normalmente se trabaja con 180 sectores, por lo que el grado de paralelismo es suficiente para que apenas haya desequilibrio de carga significativo en ordenadores de 16 o menos núcleos. El código implementa también la transferencia opcional de los modelos sesgados a una GPU, para que el *kernel* de un determinado ángulo se pueda ejecutar en la misma. Para GPUs, *skewEngine* se ha programado tanto en CUDA como en OpenCL.

También se ha implementado un *kernel* unitario, en el que, básicamente, el Algortimo 4.3 se reduce a un bucle con la simple igualdad skewOutput$[i][j]$ = skewInput$[i][j]$. Además, se han implementado diferentes casos de estudio, y en todos ellos, tan sólo se necesita definir un método de acceso de tipo void, cuyo único argumento es el objeto de la clase *skewEngine* que corresponde al sector. En la práctica, el programador solo necesita modificar la línea `skewer->kernel = funcionElegida` para adaptarla a su caso de estudio específico. Por ejemplo, para la Cuenca Visual Total:

```
skewer->kernel=isGPU?viewshedGPU:vieshedCPU;
```


El código *skewEngine* está disponible públicamente en un repositorio Github [124].

Extensión de skewEngine al caso 3D

Si el procesamiento intensivo de datos, con una dependencia de todos con todos en cualquier dirección sobre una malla de datos bidimensional es muy costoso en dos dimensiones, su extensión a tres dimensiones puede ser excepcionalmente costosa, por lo que es evidente que cualquier forma de abordar el problema, debe partir de dos premisas similares a la propuesta anterior en dos dimensiones.

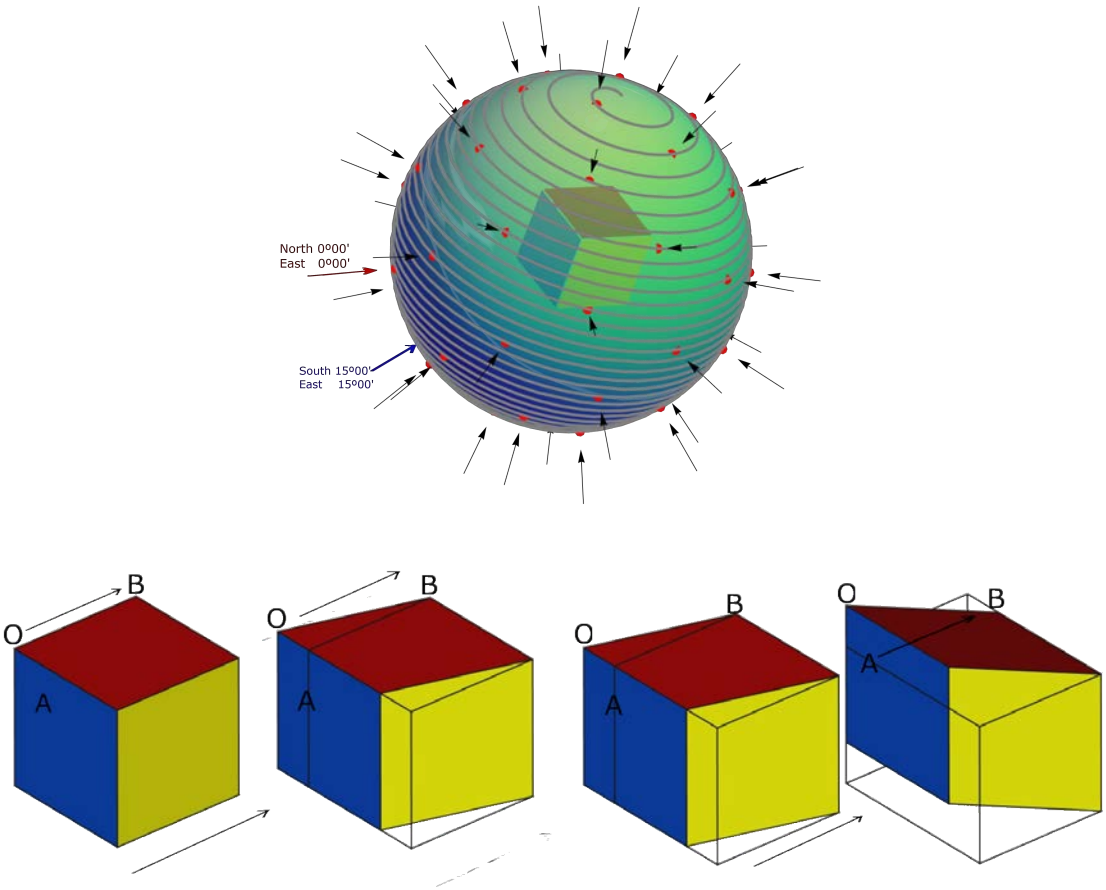
En primer lugar, el infinito número de direcciones en el que podríamos procesar los datos debe reducirse, de una forma similar a como ya se hizo con la discretización azimutal en 2D con *skewEngine*, y que por defecto, dividía el espacio en 360°. En segundo lugar, los datos deben alinearse en memoria, e incluso sería altamente recomendable un almacenamiento alineado de la información en memoria secundaria, también usando criterios similares a *skewEngine*.

El primer problema, se resuelve utilizando un elegante y sencillo método de discretización del espacio 3D: la espiral esférica de Fibonacci, como se realizó en la Sección 4.1.1 para la proyección de moléculas.

Haciendo un símil de la esfera con nuestro planeta, las latitudes de los diferentes ejes están equitativamente distribuidos, de forma que si el número de ejes fuera $N = 180$, habría una separación exacta de un grado de latitud entre ejes (latitudes desde 0.5 a 179.5, para ser precisos). Por otra parte, a cada "latitud discreta" le corresponde una longitud exclusiva que depende de la razón áurea, que se mostró en la ecuación (4.4). Siguiendo con el símil, podría ser que a la latitud 36.5° le corresponda la longitud de Málaga, mientras que justo la anterior de la serie, a 35.5°, podría coincidir con Tokio.

Podemos aprovechar entonces la distribución equitativa de latitudes para resolver el problema del alineamiento de los datos en 3D mediante la misma reutilización del código para *skewEngine* en 2D. De esta forma, los N_x planos de datos (de dimensiones $N_y \times N_z$) serían reproyectados con el algoritmo *skewEngine* generando un cubo que estaría sesgado perpendicularmente respecto al eje z.

Considérese el siguiente ejemplo: una malla tridimensional de datos, como la representada en el interior de la esfera en la Figura 4.23(a), dispone los datos alineados en memoria siguiendo la dirección marcada en rojo. Es decir, la arista que separa la cara de color amarillo de la de color rojo almacena datos de manera consecutiva. A esta configuración inicial se le asignan las coordenadas de latitud y longitud cero.



(b) Alineamiento en longitud. (c) Alineamiento en latitud.

Figura 4.23: Algoritmo *skewEngine* en 3 dimensiones aplicado a un eje de la secuencia de Fibonacci.

Sin embargo, queremos aplicar un algoritmo intensivo que procese los datos en la dirección marcada por la flecha azul, de latitud -15° y longitud 15°. Para alinear la información, se realizaran dos pasos. En uno de ellos se alinearía en latitud (Figura 4.23(b)), mediante la aplicación del algoritmo *skewEngine* aplicado a cada plano Z, y en el segundo paso, al cubo regular que contiene al cubo original sesgado, se le aplicaría el sesgo de latitud (Figura 4.23(c)) o viceversa.

4.2.4. Aplicaciones

Los excelentes resultados obtenidos por el algoritmo sDEM [125] para el cálculo de la cuenca visual total fueron la motivación principal para el desarrollo de la herramienta *skewEngine*. Sin embargo, en sDEM se detectaron algunas deficiencias que se han corregido

en este trabajo, y que, básicamente se centran en que sDEM no hace una preparación de los datos antes del procesamiento, sino que simultáneamente hace el trabajo de sesgado, aplica el algoritmo, y reconstruye los datos originales. Además de ser un código tremendamente complejo y poco exportable a otros casos, sDEM no diferencia previamente entre los 4 conjuntos que se representaron en la Figura 4.20, por lo que incluye numerosas bifurcaciones que ralentizan el código, especialmente en GPUs. La implementación de la cuenca visual total con *skewEngine* ha sido precisamente (tras la obvia implementación de la identidad) el primer caso de estudio considerado. Sin embargo, para comprobar la facilidad de la implementación de otros códigos con el modelo propuesto, se han elegido otros dos casos adicionales: el filtrado de imágenes borrosas y la transformada Radon. Por otro lado, en la Tabla 4.7 se muestran las arquitecturas utilizadas en este trabajo.

Tabla 4.7: Arquitecturas utilizadas en los casos de estudio.

	Máquina 1	Máquina 2	Máquina 3
Nombre	Desktop–PC	Server	HPC node
CPUs	16x Intel	32x Intel	64x Intel
	i7-10700K	Xeon E5-2698 v3	Xeon E5-2698 v4
GPUs	1x NVidia	4x NVidia	8x NVidia
	Ampere RTX4080	Maxwell GTX980	Volta V100

En los siguientes apartados se muestra un breve resumen de los resultados de los casos de estudio. Sin embargo, es muy importante aislar previamente el tiempo empleado en las dos fases en las que está realmente implicada la herramienta *skewEngine*, pues es lo que servirá para identificar las aplicaciones en las que merezca la pena utilizarla.

Identidad

Para estimar el coste de las etapas *skew* y *deskew*, se ha utilizado un *kernel* identitario (los datos se deconstruyen y reconstruyen sin cálculos intermedios), en sus versiones para CPU y GPU. Se han seleccionado 180 sectores, y se ha aplicado a dos conjuntos de datos diferentes: una fotografía RGB de 2122×2122 píxeles, y un DEM de una zona montañosa de 2000×2000 puntos. Para simplificar, se muestran solo los resultados obtenidos, en tiempo de ejecución, para la fotografía, ya que el escalado del tiempo con el tamaño es prácticamente lineal (un 12,5 % más lenta la imagen que el DEM). Los resultados se muestran en la Tabla 4.8.

Tabla 4.8: Tiempos de ejecución del kernel Identidad.

	Máquina 1	Máquina 2	Máquina 3
tiempo CPUs	2.34 s.	1.3 s.	0.45 s.
tiempo GPUs	0.24 s.	0.46 s.	0.10 s.

Teniendo en cuenta la perfecta escalabilidad de la herramienta respecto al número de núcleos o GPUs, se han utilizado sólo 15, 30 y 60 núcleos de los disponibles en las respectivas máquinas, para que sean divisores del número de sectores (180) y descontar así el desequilibrio de carga. Por otra parte, el código puede elegir una CPU o GPU mediante un paradigma de granja de tareas, por lo que los mejores tiempos (siempre con GPUs) podrían reducirse si reciben la colaboración de las CPUs, aunque apenas merece la pena en ninguna de las arquitecturas. En cualquier caso, se observan tiempos muy razonables, especialmente para algoritmos que pueden tardar minutos, horas e incluso días, en conjuntos de datos de tamaños similares.

Cuenca visual total

Como se ha descrito en la Sección 4.2.1, la cuenca visual total determina la superficie de un territorio que es visible por un observador, calculada para todas sus posibles ubicaciones. Dada una ubicación cualquiera de este observador, su visibilidad se determina en un conjunto discreto de s direcciones radiantes equitativamente distribuidas. Con *skewEngine*, y dada una dirección discreta, se ha visto que fácilmente se puede calcular la cuenca visual a todos los puntos que estén en la misma línea, reutilizando todos los datos de elevación.

En trabajos previos [125–127], la mayoría de los modelos de elevación utilizados tienen tamaños de alrededor de 2500×2500 puntos. Estas dimensiones son suficientes para cubrir, por ejemplo, la superficie del Parque Nacional Sierra de las Nieves, con una resolución de 10 metros. Teniendo en cuenta que una línea de datos tiene un tamaño de (a lo sumo) 2 a 4 mil datos de elevación, y que normalmente cada dato es de sólo 2 bytes, lo normal es que toda la información de una línea quepa en la caché L1 de un núcleo. Sin embargo, por la propia naturaleza del algoritmo, el número de operaciones es de cientos de millones de FLOPs por línea, que al ejecutarse sin apenas fallos en la L1, produce unos rendimientos altísimos en todas las arquitecturas empleadas en este trabajo.

Sin entrar en detalles concretos de los resultados, cabe destacar que en el peor de los casos (el ordenador de sobremesa, utilizando sólo las CPUs) el tiempo de ejecución fue de 59 segundos para el modelo de 25M de puntos del Parque Sierra de las Nieves, y mejorando

un 10 % los resultados de sDEM. Sin embargo, la GPU hizo los cálculos en apenas 3,5 segundos, mientras que sDEM requiere 10,2 segundos. Hay que tener en cuenta que las herramientas de cálculo utilizadas en los Sistemas de Información Geográfica, como gdalviewshed o GRASS [78, 121], hacen sus operaciones en pocos segundos para el cálculo de una única cuenca visual. En este caso, se está realizando el cálculo de 25 millones de ellas. Esto supone que el modelo que aquí se presenta es de 6 a 7 órdenes de magnitud más rápido. Merece la pena destacar que el *kernel* de la cuenca visual solo necesita 35 líneas de código.

Transformada Cepstrum para filtrado del desenfoque por movimiento

Pero el objeto de este estudio no es demostrar el rendimiento de una aplicación como la referenciada en el apartado anterior, sino su utilidad para implementar rápidamente versiones más eficientes de otros algoritmos. Y para ello, se han elegido dos aplicaciones que requieren un cálculo muy intensivo. La primera de ellas es la transformada Cepstrum local [128], aplicada a una imagen borrosa por el movimiento. La transformada Cepstrum es una técnica matemática utilizada para analizar la estructura espectral de una señal en el dominio cepstral. Se define como la transformada de Fourier del logaritmo del espectro de potencia de la señal. Su fórmula se expresa de la siguiente manera:

$$C(\tau) = \mathcal{F}^{-1}\left[\log\left(\left|\mathcal{F}[x(t)]\right|^2\right)\right], \tag{4.20}$$

donde $x(t)$ representa la señal en el dominio del tiempo y τ es la variable de retardo. El uso de la transformada cepstrum es común en aplicaciones de procesamiento de señales y análisis espectral [129, 130]. En imágenes borrosas por movimiento, el dominio cepstral ayuda a identificar los coeficientes que identifican la dirección e intensidad del movimiento que ha provocado que la imagen esté borrosa. No se ha podido encontrar ningún experimento en la literatura que haya calculado la transformada cepstral centrada en cada uno de los píxeles de una imagen, posiblemente por su elevado coste computacional. El análisis cepstral solo se ha aplicado a toda la imagen, por lo que sólo se utilizaba para detectar el movimiento de la cámara, y no de los diferentes objetos de la cámara, como ocurre en la imagen de la Figura 4.24, en la que distintos objetos se vuelven borrosos en direcciones y velocidades diferentes.

Con *skewEngine* se ha implementado la transformada Cepstrum aplicada a cada píxel de la imagen, en 360 direcciones diferentes, y con ventanas de 32, 64 y 128 píxeles de radio. Los tiempos de ejecución oscilan entre 1 y 3 minutos para una imagen de 4M píxeles, aunque lo más sorprendente es que se ha podido programar en apenas unas horas, gracias a la herramienta.

Figura 4.24: Imagen borrosa por objetos con diferentes movimientos.

Transformada Radon

La transformada de Radon [131] es una herramienta matemática utilizada en radiología para la reconstrucción de imágenes de objetos a partir de mediciones de rayos X o de otras formas de radiación. En radiología, la transformada de Radon mide la cantidad de radiación que atraviesa el objeto desde todas las direcciones posibles y convierte esta información en una imagen bidimensional o tridimensional del objeto. Este proceso se llama tomografía y se utiliza comúnmente en medicina para visualizar estructuras internas del cuerpo humano y en otras áreas de la ciencia para la inspección de materiales o la investigación de la naturaleza de un objeto. La transformada Radon se define como la integral de la imagen a lo largo de todas las rectas que pasan por un punto fijo del espacio. En otras palabras, para cada dirección, se mide la cantidad de radiación que atraviesa el objeto a lo largo de esa dirección y se integra esta información a lo largo de todas las rectas de esa dirección.

El resultado de la transformada de Radon es una función que representa la suma de las proyecciones a lo largo de cada dirección posible. Esta función se llama sinograma y se utiliza como entrada para la reconstrucción de la imagen. La reconstrucción se realiza mediante una técnica llamada retroproyección filtrada, que consiste en tomar cada proyección del sinograma, rotarla y luego "proyectarla hacia atrás" a lo largo de la dirección correspondiente en el espacio. El resultado de este proceso, tras aplicarlo en todas las direcciones, se acumula para producir la imagen reconstruida del objeto.

La Transformada Radon Local (LRT, por sus siglas en inglés) [130] es una variante de la Transformada de Radon que se utiliza para analizar imágenes en dominios locales y no globales. A diferencia de la transformada de Radon estándar, que utiliza la información de la proyección de la imagen en todas las direcciones posibles, la LRT utiliza una ventana de análisis local en la imagen para calcular la transformada de Radon. Esto permite analizar

la imagen de forma más detallada y adaptativa a las características locales de la imagen. La LRT tiene varias aplicaciones, como el análisis de texturas en imágenes, la detección de bordes y la segmentación de imágenes. También se utiliza en campos como la inspección no destructiva de materiales, la visualización médica y la astronomía. La LRT tiene un alto coste computacional en comparación con la transformada de Radon estándar. Esto se debe a que la LRT requiere el cálculo de la transformada de Radon para cada ventana local en la imagen, lo que puede ser computacionalmente intensivo, por lo que SkewEngine se presenta como la herramienta más adecuada para ello.

En este trabajo se ha implementado tanto la transformada Radon local como la general o estándar, aunque sólo presentamos datos de comparación con la última, ya que es la única de la que existe un software disponible en la bibliografía. En particular, hemos comparado nuestros resultados con las dos aplicaciones más utilizadas y eficientes: SciKit [132] (en Python) y Astra Toolbox [133], para Matlab. En ambos casos, el código binario de los respectivos *kernel* está en C++, y también en CUDA, para el caso de Astra.

La imagen que se ha elegido para la transformada Radon es una fotografía de 1500×1000 píxeles. Sin embargo, una de las aplicaciones mencionadas internamente extiende las imágenes para que estén circunscritas en un círculo, y éste, a su vez, en un cuadrado, por lo que hemos preferido crear una imagen ficticia de 2122×2122 píxeles para que la comparación sea en iguales condiciones, a pesar de que perjudica los resultados de *skewEngine*, que trabaja con los límites ajustados. En todas las aplicaciones, la imagen resultante tras la aplicación del algoritmo ha sido idéntica (utilizando un filtro rampa), como se muestra en la Fig. 4.25.

Figura 4.25: Imagen original, e imagen restaurada a partir del sinograma.

La Tabla 4.9 muestra un resumen de los resultados de la comparación, en las que se demuestra el alto rendimiento de *skewEngine*, a pesar de que no es precisamente un algoritmo especialmente costoso desde el punto de vista computacional.

Tabla 4.9: Tiempos de ejecución de la transformada Radon.

	SKE (Server)	SKE (PC)	Scikit (PC)	Astra (PC)
CPUs	1.46 s.	2.03 s.	10.58 s.	-
GPUs	0.49 s.	0.317 s.	-	0.316 s.

Como se puede observar, Astra Toolbox, que utiliza CUDA, es la que obtiene (muy ligeramente) mejores resultados, y obviamente, esto se debe a que el coste de la reorganización de los datos en memoria, que en este caso supone casi el 70 % del coste total, no merece la pena para una aplicación con tan poco coste computacional. Pero si sorprende que tan sólo se hayan necesitado apenas cuatro líneas de código para su implementación con *skewEngine*:

```
for (int i=start;i<end;i++)
    sum+=skewInput[row *dim_i+i];
for (int i=start;i<end;i++)
    skewOutput[row*dim_i+i]=sum;
```

Poniéndose así de manifiesto no sólo el rendimiento de la herramienta, sino también la altísima productividad en la programación de aplicaciones que se aprovechen de la misma.

4.2.5. Publicaciones

Los resultados expuestos en este capítulo han sido publicados en el *Journal of Supercomputing* [2], Q2 en las categorías:

- *Computer Science, Hardware & Architecture*,
- *Engineering, Electrical & Electronic* y
- *Computer Science, Theory & Methods*.

También han sido aceptados y presentados varios trabajos relacionados al congreso CMMSE 2023, celebrado en Rota (Cádiz) [7], y a las Jornada Sarteco 2023 de Ciudad Real [12].

4.3. Detección de epilepsia

El estudio de señales cerebrales de los pacientes se realiza en un electroencefalograma (EEG). Este consiste en una serie de electrodos colocados en la cabeza del paciente y que miden 148 señales eléctricas procedentes de distintas regiones del cerebro. Los equipos médicos homologados pueden recoger estas señales a frecuencias de entre 256 y 512 Hz. En la mayoría de los casos, estas señales se guardan en un formato de archivo conocido como Formato Europeo de Datos (EDF, por sus siglas en inglés).

La detección de crisis epilépticas en señales de EEG ha sido estudiada a fondo en la literatura [134, 135], proponiendo el análisis de señales en diferentes dominios. El primero es el análisis temporal [136], utilizado habitualmente por los médicos para interpretar el EEG epiléptico mediante la visualización de las series temporales de las señales. Por lo tanto, la detección de convulsiones en el dominio del tiempo podría llevarse a cabo por sistemas automatizados, sin necesidad de cambiar de dominio. En otros trabajos, en lugar de analizar la señal temporal en bruto, se extraen algunas características utilizando análisis de componentes principales (PCA, por sus siglas en inglés) [137]. Sin embargo, las señales del EEG pueden contener muchos artefactos y ruido [138], lo que hace que su clasificación sea una tarea difícil, incluso para los expertos humanos. Por este motivo, se han propuesto filtros paso banda de Butterworth [139] que los eliminan. Por otro lado, se ha empleado también el análisis en el dominio de la frecuencia [140, 141]. La transformada rápida de Fourier (FFT, por sus siglas en inglés) se utiliza para calcular la amplitud de las distintas frecuencias, lo que permite identificar patrones útiles para detectar crisis epilépticas. M. Zhou *et al* (2018) [140], calculan la FFT en ventanas temporales de un segundo, pero no realizan un submuestreo posterior, por lo que se mantiene la cantidad de datos que deben ser almacenados y procesados. Por último, volviendo al dominio temporal, existen numerosos estudios centrados en el análisis de propiedades de las señales no estacionarias, como tendencias, discontinuidades y patrones repetitivos del EEG [142]. Entre los diferentes métodos, se ha popularizado la Transformada Wavelet Discreta (DWT, por sus siglas en inglés) [143, 144]. Este enfoque, ha permitido demostrar el buen rendimiento de los métodos de ML cuando se entrenan con características de tiempo-frecuencia [145–147].

En cualquier caso, la adquisición de datos de EEG a altas frecuencias de muestreo (512 Hz) presenta varios retos computacionales. A menudo se subestiman en la literatura existente, pero son de vital importancia, especialmente en escenarios de monitorización en tiempo real. Por ejemplo, un solo canal de datos de EEG muestreado a esta frecuencia produce 512 puntos de datos por segundo. En escenarios prácticos, en los que se utilizan múltiples canales de EEG (existen sistemas de hasta 256 canales), el volumen de datos se vuelve prácticamente inabordable.

Para hacer frente a esta situación es necesario utilizar eficazmente los recursos informá-

ticos. El uso de las GPUs ha resultado ser crucial. Las GPUs están diseñadas para tareas de procesamiento paralelo, lo que las hace ideales para manejar los grandes volúmenes de datos generados por las grabaciones de EEG de alta frecuencia. Pueden mejorar significativamente el rendimiento computacional distribuyendo las tareas entre numerosos núcleos, lo que acelera el procesamiento de señales y la extracción de características.

Por ejemplo, Pavel Dohnálek *et al* (2013) [148] usan GPUs para paralelizar el cálculo de métricas basadas en frecuencias a partir de datos de EEG (principalmente utilizando CUBLAS). Los resultados muestran que el procesamiento paralelo de las diferentes fases (FFT y cálculo de características, clasificación de datos, etc.) consigue un procesamiento en tiempo real, permitiendo el despliegue en escenarios reales. Mientras que Lizhe Wang *et al* (2012) [149] afrontan el mismo problema, pero aplican el algoritmo *Ensemble Empirical Mode Decomposition* (EEMD). Se usa una arquitectura con 256 CPUs para construir un clúster Hadoop [150] y aplicar un enfoque de *map-reduction* para paralelizar el costoso procesamiento. También Felipe Muñoz en su tesis doctoral (2024) [151] emplea un clúster de 160 nodos con 128 *cores* cada uno para la detección de epilepsia por coincidencia de patrones, utilizando la distancia con respecto a la DWT.

Esta línea de investigación abarca todo el procedimiento descrito en el Capítulo 3, con el objetivo específico de desarrollar un modelo capaz de detectar y clasificar crisis epilépticas a través de los datos medidos por un EEG. En las siguientes secciones se describirá:

1. Cómo están dispuestos los datos en los archivos que contienen las señales biomédicas de los pacientes.
2. Los problemas con los que habitualmente se encuentra el equipo médico para analizar las señales.
3. La librería de lectura eficiente que se ha desarrollado y empleado para leer y analizar los datos de los archivos.
4. Las tareas de preprocesamiento, paralelizadas en GPU, para resolver los problemas mencionados y para que los datos sean fácilmente entendibles para las redes neuronales y, por último,
5. La creación, entrenamiento y validación de los modelos de DL.

4.3.1. El formato de los datos de la electroencefalografía

Como se mencionó anteriormente, los datos obtenidos de la video-electroencefalografía se almacenan en ficheros EDF. Estos son conocidos por su sencillez, eficacia y compatibilidad

con una amplia gama de dispositivos y aplicaciones software. Facilitan el intercambio y el procesamiento estandarizado de datos fisiológicos, lo que los convierte en un formato ampliamente aceptado en el campo de la asistencia sanitaria, la investigación y la ingeniería biomédica.

Se han desarrollado varias bibliotecas software para leer y analizar archivos EDF. Entre ellas se encuentran EDFlib para C/C++ y Java [152, 153], el paquete edfReader [154] en R, y PyEDFlib [155] y MNE toolbox [156] para Python. Algunas de estas bibliotecas son muy sensibles a los errores causados por diversos factores, como archivos temporalmente discontinuos o anotaciones con un formato incorrecto. Además, muchos de los archivos son generados por equipos médicos que pueden no estar familiarizados con las especificaciones de este formato de datos [157]. Por otra parte, estas bibliotecas están diseñadas para mostrar los datos de forma intuitiva y continua para cada señal. Sin embargo, los archivos EDF están organizados en bloques de datos discontinuos, lo que puede dar lugar a una lectura ineficiente de la información por parte de las bibliotecas actuales debido a los numerosos cálculos de posición de punteros y saltos.

La estructura de un archivo EDF es sencilla, aunque no especialmente intuitiva. Consta de una sección de cabecera que contiene metadatos con información crucial sobre la grabación, como la información del paciente, la hora de inicio de la grabación y detalles sobre cada canal de señal, como el tipo de señal, la etiqueta y las unidades físicas. Tras la cabecera, las muestras de datos reales de cada canal se almacenan de manera discontinua, en grabaciones de aproximadamente un segundo que contienen cada uno de los canales. Cada grabación, denominada registro de datos, contiene una serie de muestras N_s para cada canal, ordenadas consecutivamente. Normalmente, N_s es consistente en todos los canales excepto en las anotaciones. La frecuencia de muestreo del EEG puede determinarse en función de la duración de la grabación t_{dr} en segundos y de N_s,

$$f_z = \frac{N_s}{t_{dr}}. \tag{4.21}$$

Es importante tener en cuenta que no se puede acceder a los datos de forma bidimensional debido a la agrupación de los datos en grabaciones separadas. Por ejemplo, la muestra con índice N_s del primer canal no está en la ubicación del archivo consecutiva a la del índice $N_s - 1$. La Figura 4.26 muestra cómo se estructura un fichero EDF, apreciándose lo que se comenta en la frase anterior: en el registro de datos 0, la muestra d_{N_s} del canal 0 (tomando una notación global) no se encuentra ubicada en el archivo de manera consecutiva a la anterior, sino que existe un cambio de canal.

Existen varias aplicaciones informáticas que permiten al personal médico visualizar y analizar gráficamente estas señales, realizando anotaciones para identificar momentos de interés, como las crisis epilépticas.

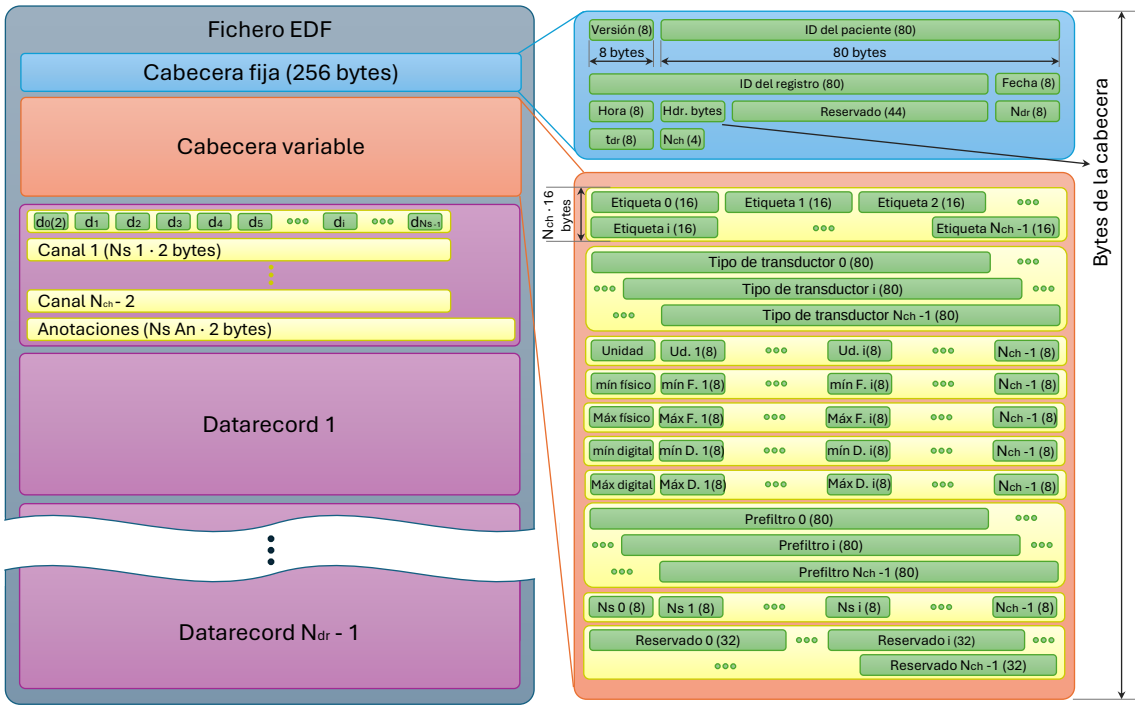


Figura 4.26: Estructura del fichero EDF.

4.3.2. Simplificación del fichero EDF

Una medición de datos de EEG de un día con 148 señales (y anotaciones), frecuencia de muestreo de 512 Hz y precisión de datos de 2 bytes supera fácilmente los 12 GB.

Se toma como ejemplo un fichero EDF recortado, facilitado por el servicio de neurofisiología del hospital Carlos Haya de Málaga, que consta de 2.919.552 muestras en cada uno de los 148 canales, con sus correspondientes anotaciones. Sin tener en cuenta estas anotaciones ni el tamaño de la cabecera, el tamaño de referencia a reducir serían 824 MB. Se propone entonces un método para reducir el espacio ocupado por estos datos sin perder información significativa, facilitando así su transferencia[4] y posterior tratamiento.

Inicialmente, este método consistía en submuestrear las variables para reducir la canti-

[4]En un hospital, la conexión a internet suele estar restringida debido a estrictos procedimientos de seguridad diseñados para protegerse contra ataques informáticos y salvaguardar la información sensible de los pacientes. En España, este tipo de instituciones implementan sistemas de seguridad de múltiples niveles, como *firewalls*, redes aisladas y políticas de acceso restringido, que garantizan la privacidad de los datos sanitarios conforme a la normativa vigente, como el RGPD. Cualquier solicitud de acceso a una página web o servidor externo requiere un procedimiento altamente complejo y supervisado, que incluye justificantes específicos, autorizaciones administrativas y una validación técnica exhaustiva para minimizar riesgos de ciberseguridad.

dad de datos por segundo, hacer la media y determinar el rango de valores dentro de una ventana de datos dada. Una menor frecuencia de submuestreo f_t suponía un menor tamaño de los datos, pero también una pérdida de precisión, especialmente teniendo en cuenta que las ondas cerebrales humanas pueden alcanzar hasta 90 Hz, por lo que una red neuronal artificial no sería lo suficientemente efectiva para detectar crisis epilépticas. En la Figura 4.27 se puede apreciar estas consecuencias del submuestreo. Además, tras ejecutar la FFT a toda la serie temporal de cada señal, se observaron unos cuantos picos en los resultados, que se denominaron frecuencias dominantes, y que eran superiores en muchos casos a las frecuencias de submuestreo que se estaban realizando. Los resultados del análisis de dos de estas señales se muestran en la Figura 4.28.

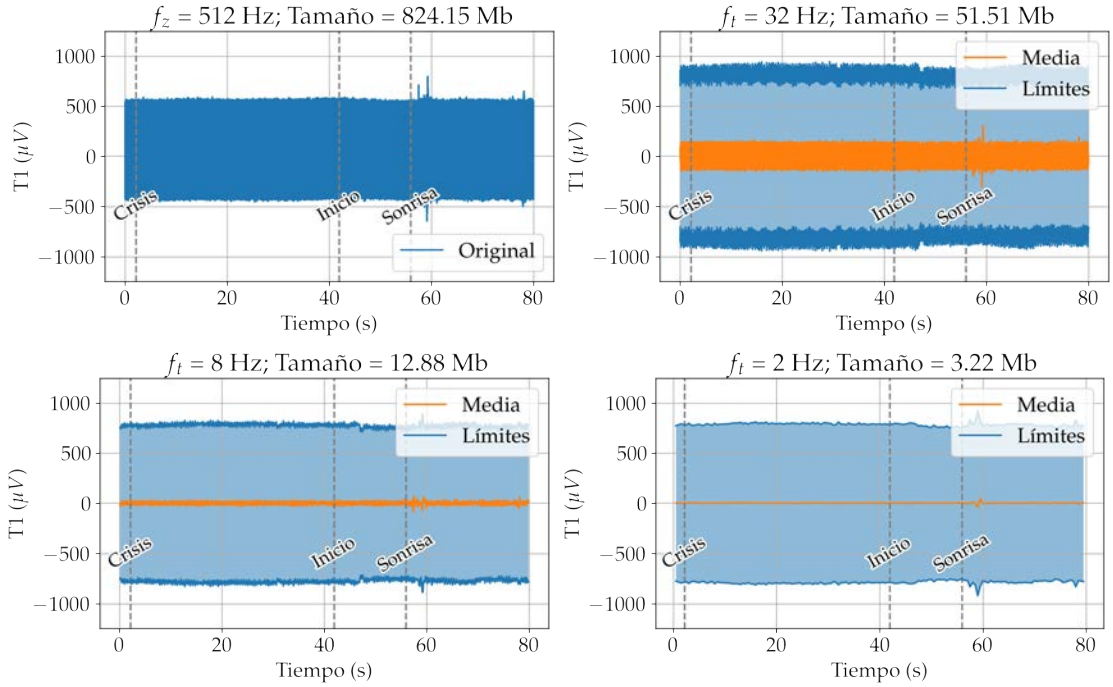


Figura 4.27: Reducción de tamaño y pérdida de información tras realizar la técnica de submuestreo a distintas frecuencias.

Concretamente, teniendo en cuenta la naturaleza caótica del cerebro, se decidió calcular la FFT en ventanas de tiempo más pequeñas, según se muestra en la Figura 4.29. Esta técnica, comúnmente conocida como Transformada de Fourier de Tiempo Corto (STFT) ha sido ampliamente utilizada para la detección de crisis epilépticas [158, 159]. El tamaño de datos de cada ventana se decidió como un múltiplo o un divisor del número de muestras dentro de un registro de datos N_s. Como este método sólo podía reducir la cantidad de información de cada ventana de tiempo a la mitad, por la simetría de los resultados de la FFT, se tomó únicamente la amplitud de las tres frecuencias dominantes, descartando el resto al considerarse menos representativo.

Tras realizar la STFT y visualizar las amplitudes de las frecuencias dominantes a lo largo

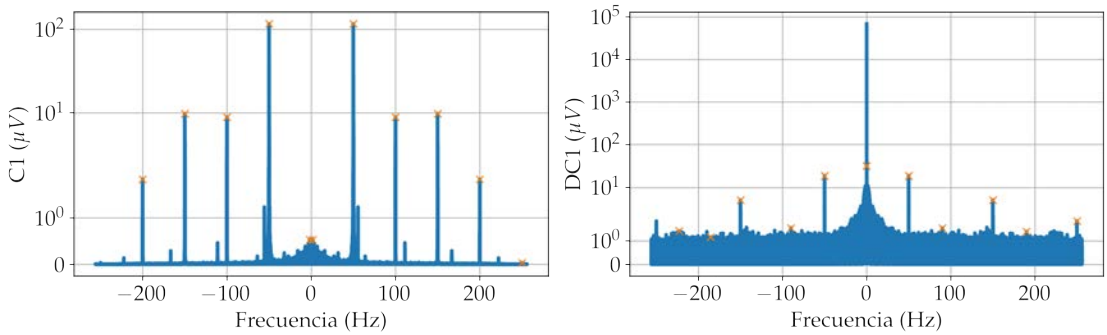


Figura 4.28: FFT realizada sobre la señal C1 del EEG, cuyas frecuencias dominantes son múltiplos de 50 Hz, y sobre la señal DC1, que tiene una fuerte componente de corriente continua.

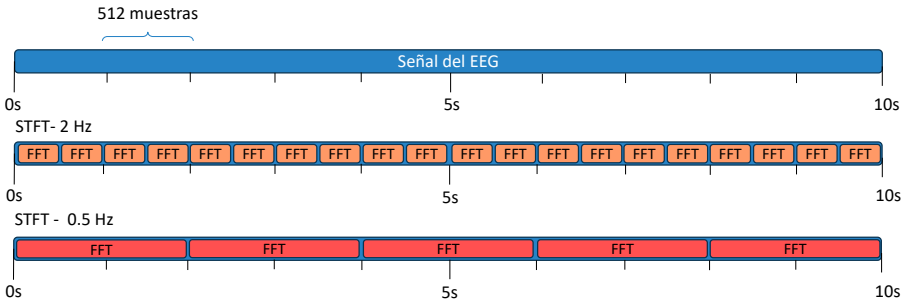


Figura 4.29: Método FFT por ventanas (STFT). La frecuencia de la ventana afecta a la resolución de los resultados de salida, así como al tamaño de los datos.

del tiempo, como se ilustra en la Figura 4.30, observamos estabilidad en las amplitudes de algunas de las señales durante los ataques epilépticos. En particular, estas señales coincidían con las identificadas por los profesionales médicos. Por el contrario, durante la actividad cerebral normal, el caos vuelve a ser evidente. Este enfoque facilita la exclusión de señales no representativas, como la señal T1.

A pesar de los prometedores resultados de aplicar la STFT para la detección de crisis, se decidió probar con otra técnica muy utilizada en la bibliografía, la DWT, que combina el análisis de la frecuencia y del tiempo. El motivo principal fue que las frecuencias que se estaban analizando con la STFT eran superiores a las que analiza el equipo médico para la detección de la epilepsia, que suelen estar por debajo de las ondas gamma (30 Hz) [160]. En otros trabajos se ha empleado también esta metodología para simplificar las señales de un EEG antes de que sean procesadas por una red neuronal convolucional [146, 147].

Con esta otra técnica se pretende un análisis de las señales del EEG para descomponerlas en diferentes niveles de frecuencia, correspondientes a las bandas delta, theta, alfa, beta y gamma. La DWT permite analizar señales no estacionarias, separando sus componentes

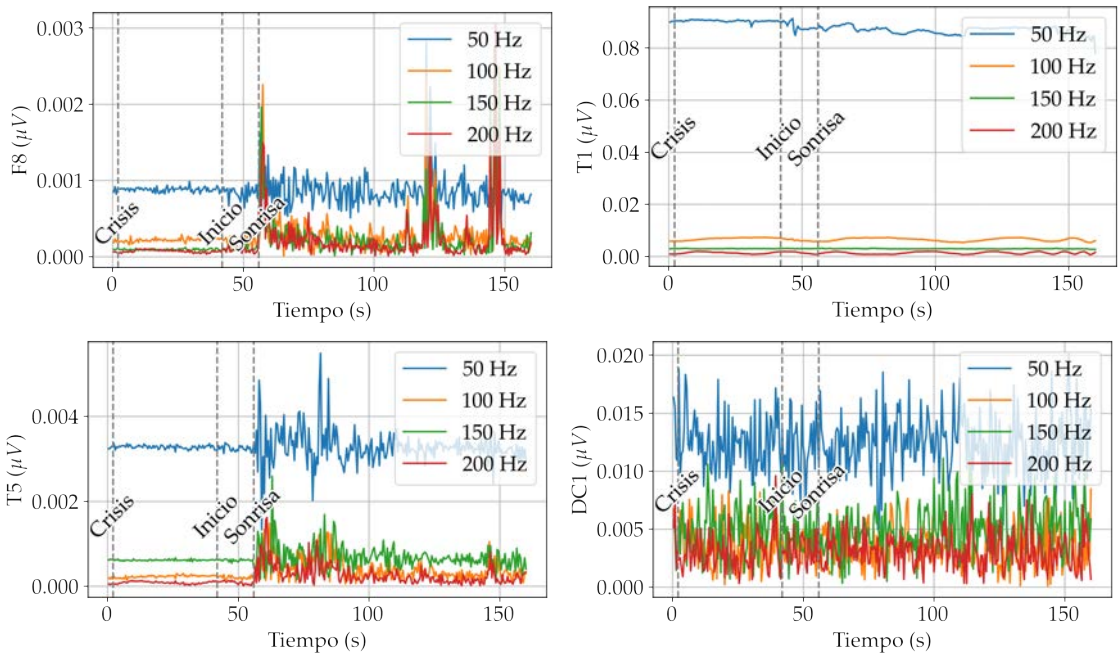


Figura 4.30: Amplitudes de las frecuencias dominantes a lo largo del tiempo en las señales F8, T1, T5 Y DC1. Se observa una estabilidad durante los ataques epilépticos en algunas de las señales.

de baja y alta frecuencia en cada nivel de descomposición. Los coeficientes obtenidos en cada nivel representarán la actividad en las distintas bandas, facilitando el estudio de la distribución de energía en la señal y su relación con los diferentes tipos de ondas cerebrales. Estos coeficientes se obtienen mediante un proceso iterativo que combina convolución y submuestreo.

En primer lugar, la señal original se convoluciona con una onda tipo. En este caso, la más adecuada para la detección de epilepsia ha resultado ser la *wavelet sym24* [161]. En su forma discreta, esta convolución se traduce en dos tipos de filtro, uno de paso bajo (para obtener los coeficientes de aproximación) y otro de paso alto (para los coeficientes de detalle), los cuales se muestran en la Figura 4.31. Después de aplicar cada filtro, se reduce la resolución temporal de los resultados al tomar solo uno de cada dos puntos, dividiendo la señal en dos conjuntos más pequeños. Esto es lo que se conoce como submuestreo.

Este proceso se repite en varios niveles, utilizando las aproximaciones de cada nivel como entrada para el siguiente, generando coeficientes que representan diferentes rangos de frecuencia.

En la Figura 4.32 se pueden apreciar estos coeficientes para un canal T7-P7 en una ventana de tiempo donde ocurre una crisis epiléptica. De ella también se deduce que esta transformada es útil para la detección de crisis. Sin embargo, en este caso se estaría au-

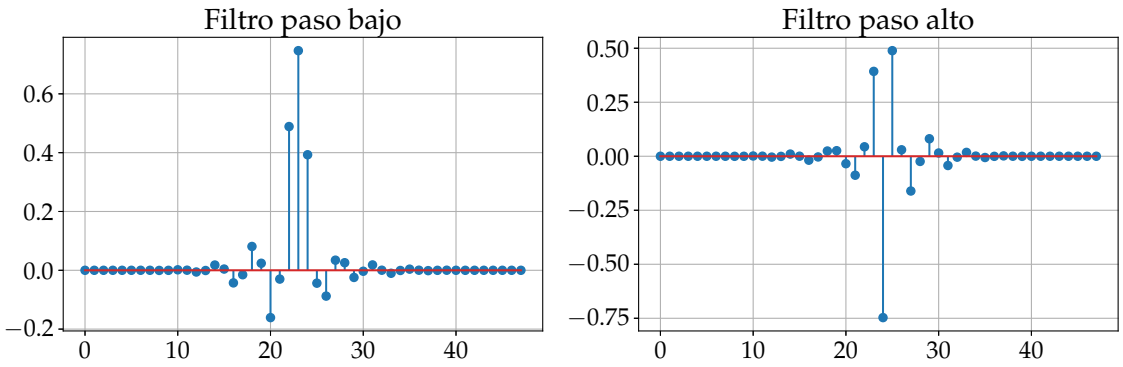


Figura 4.31: Filtros paso bajo y paso alto de la *wavelet sym24*.

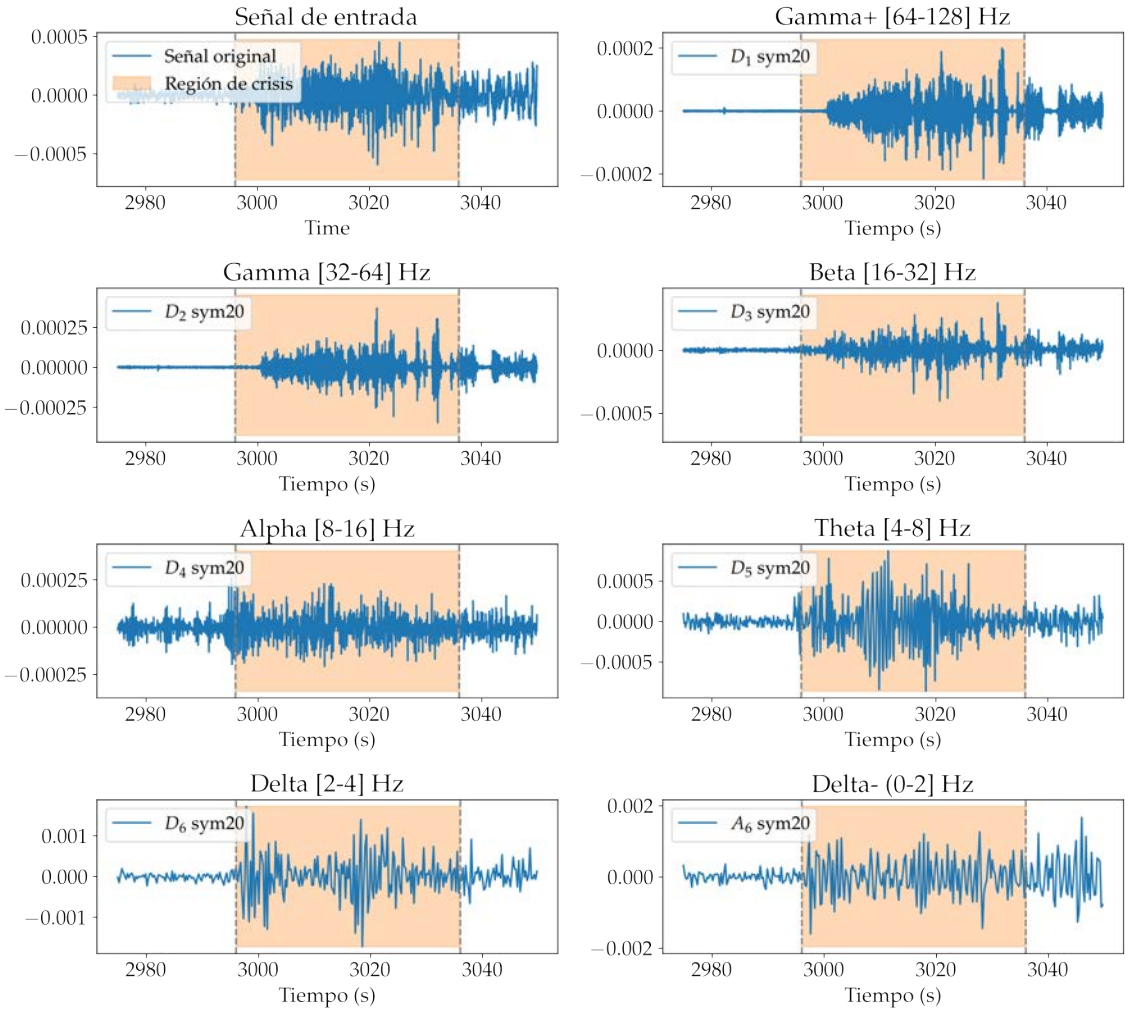


Figura 4.32: Comparativa entre la señal original y los coeficientes de detalle y aproximación a distintos niveles tras realizar la DWT.

mentando la cantidad de información, al generarse nuevos datos para distintos rangos de frecuencia de una señal. Por ello se propone, como realizan en otros trabajos [162, 163], calcular la entropía de Shannon en ventanas de tiempo W,

$$H(X) = -\sum_{W} p(x) \log_2 p(x) \tag{4.22}$$

donde $p(x)$ representa la probabilidad de ocurrencia de cada resultado posible x en una señal X, dada por el histograma de valores de la ventana. Así, de nuevo, por cada ventana de tiempo se almacenan únicamente los valores de entropía H para los rangos de frecuencia que sean más relevantes como, por ejemplo, las ondas beta, alfa y theta.

No obstante, la gran cantidad de información a tratar para llevar a cabo los distintos procesamientos hace necesario usar computación paralela, moviendo partes del proceso a GPU, como se indicará más adelante.

4.3.3. Lectura eficiente de los archivos de datos

Debido a la ineficiencia de las bibliotecas de software actuales para la lectura de los ficheros EDF, así como de los errores ocurrentes al intentar tratar aquellos facilitados por el hospital, se decidió desarrollar una nueva biblioteca. Se ha implementado en el lenguaje de programación C++ debido a la necesidad de paralelización con la GPU. Aunque existen métodos para utilizar la GPU desde otros lenguajes, como la librería pyCUDA para Python, no son tan eficientes como hacerlo en C++.

Como se mencionó, ya existe una biblioteca escrita en este lenguaje, EDFlib [152]. Sin embargo, ésta presenta varios problemas:

- Dificultades para gestionar distintas disposiciones de los datos, como al unir EDFs independientes.
- Lectura de bytes de forma individual, calculando nuevos saltos de posición a la hora de cambiar de registro de datos o de canal.

Más concretamente, para obtener todos los datos de un cierto canal en esta biblioteca, excluyendo las anotaciones, un bucle externo debe iterar a través de cada canal. Al llegar a un nuevo registro de datos, el puntero del archivo se establece en una nueva posición mediante la función `fseek`. Además, la biblioteca lee bytes de uno en uno con la función `fgetc` en lugar de leer varias muestras con `fread`. El proceso que sigue EDFlib se detalla en el Algoritmo 4.4 y se explica, de manera más intuitiva, en la Figura 4.33. El número

total de llamadas se calcula como $N_{ch}\, N_{dr}\, (2\, N_s$ `fgetc` + `fseek`), considerando que cada dato ocupa 2 bytes.

Algoritmo 4.4 Función de lectura de la librería EDFlib.

```
1:  function EDFREAD(edfFile, channel, n, buffer)
2:      header ← edfFile.header
3:      offset ← getPosition(header, channel)
4:      fseek(edfFile, offset, SEEK_SET)
5:      N_s ← header.smp_per_record
6:      jump ← header.recordsize - (2 · N_s)
7:      for i ← 0 to n step 1 do
8:          if i % N_s and i != 0 then
9:              fseek(edfFile, jump, SEEK_CURR)
10:         end if
11:         buffer[i] ← fgetc(edfFile)<<8 + fgetc(edfFile)
12:     end for
13: end function
14:
Require: edfFile, data, N_s, N_ch, N_dr
15: for i ← 0 to N_ch step 1 do
16:     edfRead(edfFile, i, N_dr · N_s, &data[i·N_dr · N_s])
17: end for
```

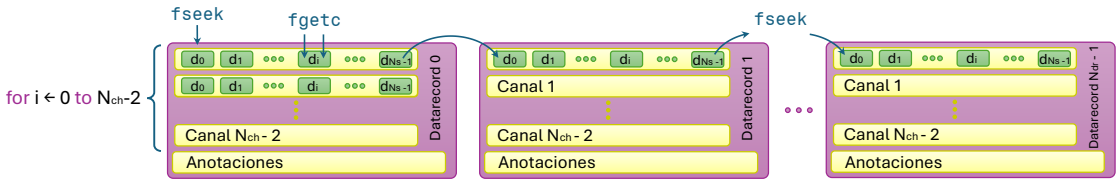


Figura 4.33: Proceso de lectura de los *datarecords* del EDF con la biblioteca EDFlib.

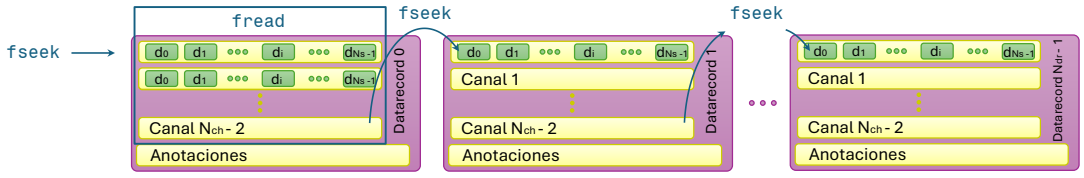
En cambio, la biblioteca propuesta implementa el Algoritmo 4.5, representado en la Figura 4.34. Como se puede apreciar, supone un nuevo enfoque en el que se leen los datos en el mismo orden en el que aparecen en el archivo. Esto se consigue utilizando la función `fread` para leer registros de datos completos y evitando anotaciones mediante saltos con `fseek`. Esta estrategia reduce considerablemente el número de llamadas, que pasa a ser $N_{dr} \cdot$ (`fread` + `fseek`). Además, si no hay anotaciones en el archivo, se reduce aún más, a $1 \cdot$ (`fread` + `fseek`).

Sin embargo, este método almacena la información en memoria de una forma menos intuitiva. En concreto, para acceder a una muestra del canal `ch` perteneciente a un instante `t`, las bibliotecas existentes solo requieren un acceso bidimensional:

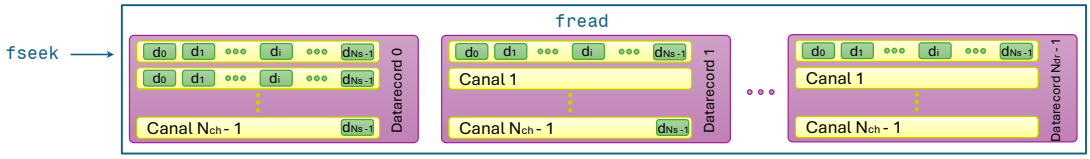
```
data[ch*N_ch + t]
```

Algoritmo 4.5 Función de lectura de archivos de la librería propuesta.

```
1: function EDFREAD(filename, buffer)
2:     file ← fopen(filename)
3:     header ← readHeader(filename)
4:     offset ← header.headerBytes
5:     N_s ← header.smp_per_record
6:     N_dr ← header.num_records
7:     N_ch ← header.num_channels
8:     annotSize ← header.annotSize
9:     if annotSize > 0 then
10:        recordSize ← (N_ch − 1) · N_s
11:        for i ← 0 to N_dr step 1 do
12:            fseek(file, offset+i·2·recordSize+annotSize, SEEK_SET)
13:            fread(&buffer[i·recordSize], sizeof(short int), recordSize, file)
14:        end for
15:    else
16:        fseek(file, offset, SEEK_SET)
17:        fread(buffer, sizeof(short int), N_dr · N_ch · N_s, file)
18:    end if
19:    fclose(file)
20: end function
21:
Require: filename, data
22: edfRead(filename, data)
```



(a) Archivos con anotaciones.



(b) Archivos sin anotaciones.

Figura 4.34: Proceso de lectura de los *datarecords* del EDF con la biblioteca propuesta.

En cambio, según el método propuesto, es necesario determinar el registro de datos `i` al que pertenece y la muestra `s` dentro de ese registro de datos.

```
i = int(t/N_s)
s = t % N_s
data[i*N_ch*N_s + ch*N_s + s]
```

El nuevo enfoque de lectura ha dado lugar a factores de aceleración o *speedup* superiores a 350 en comparación con EDFlib para C++, como se muestra en la Figura 4.35.

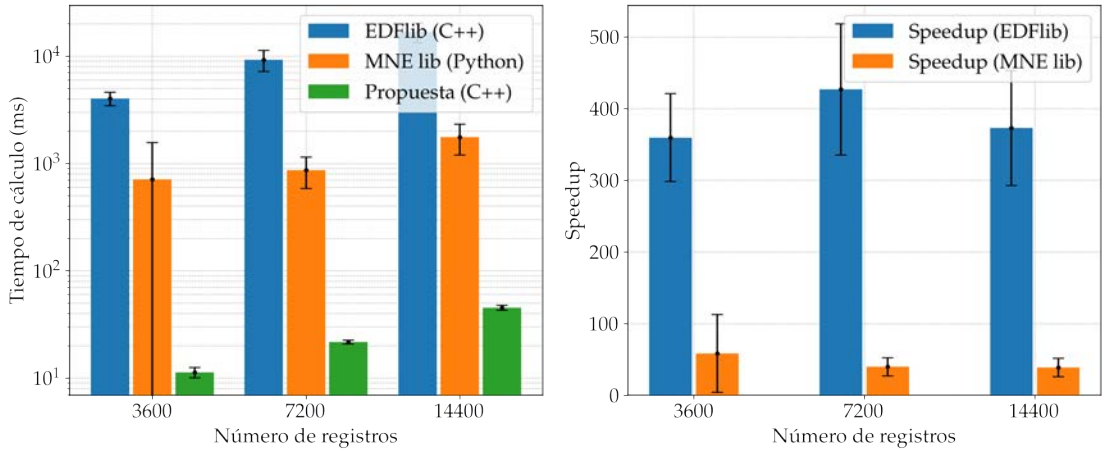


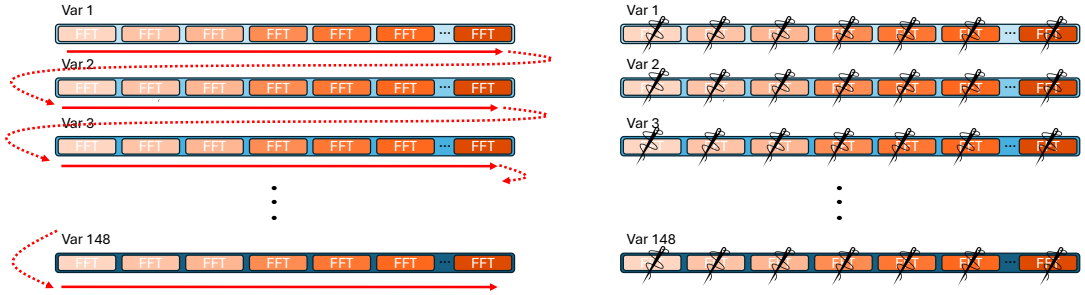
Figura 4.35: Aceleración del proceso de lectura al utilizar la biblioteca propuesta.

4.3.4. Computación paralela para la simplificación de los datos

La necesidad de paralelización en GPU empieza a ser notable a partir de los 10 millones de datos. Por tanto, el uso de estos dispositivos está justificado, teniendo en cuenta la frecuencia de muestreo, la duración del diagnóstico y el número de señales en el estudio de la epilepsia de los pacientes, donde se pueden llegar a superar fácilmente los 5 mil millones de datos para un único paciente por día. Aun realizando cálculos relativamente complejos como son la STFT y la DWT con entropía en ventanas de tiempo reducidas, el número de ventanas sigue siendo lo suficientemente elevado, además de ser independientes entre ellas a la hora de ejecutar las operaciones correspondientes. A continuación se detallan estos dos *kernels* implementados.

Transformada de Fourier de Tiempo Corto (STFT)

En este caso, cada *thread* se encargará de realizar los cálculos de la FFT en una ventana en concreto, aprovechando que no existen dependencias entre ellas, y se almacenan los resultados de las amplitudes de las frecuencias deseadas en un *array* de salida. En la Figura 4.36 se puede ver la ventaja de ejecutar esta técnica de manera paralela en GPU frente a la secuencialidad que ofrece un único *core* de la CPU cuando existe una gran cantidad de datos.



(a) CPU (b) GPU

Figura 4.36: Cálculo de la STFT mediante CPU, trabajando de manera secuencial, y GPU, donde cada *thread* se encarga de una ventana específica.

El Algoritmo 4.6 muestra la función que se ejecuta como un *kernel* de CUDA, donde cada *thread* procesa una porción de los datos. El *kernel* funciona de la siguiente manera:

1. Cada *thread* calcula su índice de la ventana e índice del canal basándose en sus IDs de bloque y *thread* [2-3]. Los parámetros `blockIdx`, `blockDim`, y `threadIdx` se utilizan para determinar la posición del *thread* dentro de la malla de *threads* o *grid*.
2. Determina N_w (número de ventanas) asegurándose que sea un entero [6-8]. Es importante saber el número de ventanas que hay dentro de un registro (a), en caso de que el número de muestras en el registro sea mayor que el tamaño de la ventana, o el número de registros que abarca una ventana (b), en caso contrario.
3. Itera sobre N_{ch} si el número de canales supera el tamaño de la malla [9].
4. Itera sobre N_s, si el número de ventanas supera el tamaño de la malla [10].
5. Crea una matriz temporal compleja `windowData` para almacenar la submatriz seleccionada [11].
6. En caso de que haya varias ventanas en un registro de datos, se debe determinar a cuál de ellas pertenece `w` [12].

7. Se asignan los valores correspondientes de la matriz de entrada `input` a `windowData` [13-17]

8. Se aplica la FFT a `windowData` [18].

9. Se calculan los valores absolutos de los componentes de la FFT [22].

10. Los resultados de la FFT para las frecuencias especificadas (`freqs`) se almacenan en `output` [20-23].

Algoritmo 4.6 Kernel para la FFT por ventanas en GPU.

```
1: function STFT(input, output, N_dr, N_ch, N_s, N_ch, W_s, f_z, freqs)
2:     ventana ← blockIdx.x · blockDim.x + threadIdx.x
3:     canal ← blockIdx.y · blockDim.y + threadIdx.y
4:     stride.x ← blockDim.x · gridDim.x
5:     stride.y ← blockDim.y · gridDim.y
6:     a ← max(1, N_s/W_s)                    ▷ Ventanas por registro (si N_s > W_s)
7:     b ← max(1, W_s/N_s)                    ▷ Registros que hay por ventana (si N_s < W_s)
8:     N_w ← N_dr· a/b
9:     for ch ← canal to N_ch step stride.y do
10:        for w ← ventana to N_w step stride.x do
11:            Crear windowData[W_s]          ▷ Array temporal para la señal de entrada
12:            subventana ← w % a
13:            for i ← 0 to W_s − 1 do
14:                registro ← w · b/a + i/N_s
15:                windowData[i].x ← input[registro·N_ch · N_s + ch·N_s + subventana·W_s +i]
16:                windowData[i].y ← 0,0
17:            end for
18:            FFT(windowData, W_s)
19:            N_f ← len(freqs)
20:            for i ← 0 to N_f do
21:                i_freq ← freqs[i]·W_s/f_z
22:                output[N_f · (N_ch·w+ch)+i] ← abs(windowData[i])
23:            end for
24:        end for
25:    end for
26: end function
```

La Figura 4.37 ilustra la ventaja obtenida en los tiempos de ejecución al optar por calcular en la GPU frente a la CPU. Se concluye que, para ventanas pequeñas, esta paralelización es más efectiva. A medida que la ventana se amplía (se reduce la frecuencia de cada ventana), la carga de trabajo de cada *thread* aumenta, lo que se traduce en una ligera pérdida de rendimiento en la GPU. También debe tenerse en cuenta que cuanto más ventanas se calculen, mayor será la precisión, pero mayor será la cantidad de información almacenada.

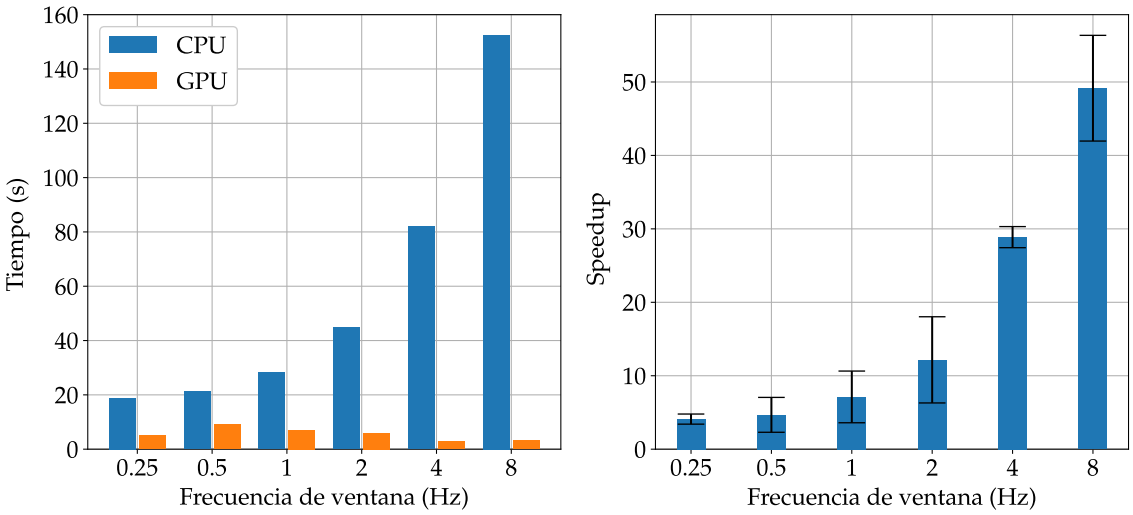


Figura 4.37: Aceleración medida a partir de distintas frecuencias de ventana para el método de la STFT. Cuanto mayor sea la cantidad de datos y más estrecha la ventana, más eficaces serán los resultados de la paralelización.

Por otra parte, como cada *thread* trabaja sobre ventanas independientes, no es necesario disponer los datos de manera bidimensional, por lo que se puede también aprovechar la aceleración conseguida con la librería de lectura desarrollada. Así, se consigue de una manera extremadamente eficiente una reducción de la información del EEG pero manteniendo su relevancia para detectar crisis epilépticas. Se puede generar, incluso, un nuevo EDF de tamaño reducido al implementar esta técnica. En la Figura 4.38 se aprecia cómo podría generarse este nuevo EDF. Se observa que la ventana sobre la que trabaja un determinado *thread* (recuadro rojo), puede corresponder a parte de un registro, a uno completo o a varios de ellos, pero los resultados de la FFT se almacenan en un único registro . El número de registro de datos resultante podría variar en función del tamaño de ventana deseado W_s y el número de canales aumenta en función del número frecuencias a analizar (en este caso, 3). El factor de reducción de la información en este ejemplo es $3/W_s$. El dato representado por $r_{w,ch,f}$ corresponde a la amplitud de la frecuencia f deseada, del canal o señal ch para la ventana w calculada por un *thread* de la GPU.

Transformada Wavelet Discreta (DWT) con Entropía

La principal razón para realizar el submuestreo es reducir el tamaño de los datos en cada nivel de la transformada, ya que a menor frecuencia, los datos se vuelven redundantes. Es decir, aunque se realice el proceso de la DWT, este es completamente reversible. Además, al eliminar esta redundancia de los datos, se consigue una mayor eficiencia computacional, al reducir la cantidad de cálculos para el siguiente nivel. Sin embargo, existen situaciones

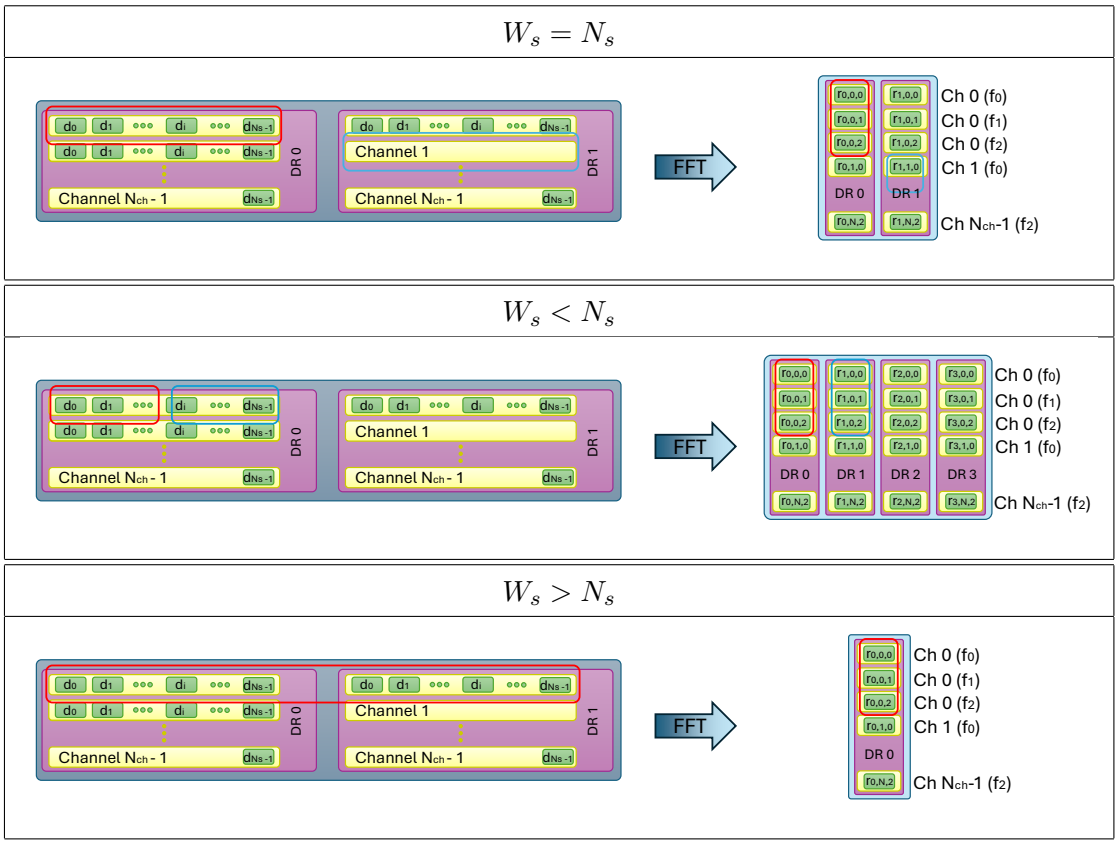


Figura 4.38: Generación de un nuevo EDF tras aplicar el *kernel* de la STFT.

en las que esta reducción de resolución puede no ser deseable o necesaria:

- **Preservación de la redundancia**: En algunas aplicaciones, es crucial preservar toda la información para una eliminación del ruido más robusta.
- **Análisis de señales no estacionarias**: para señales no estacionarias, donde el contenido de frecuencia cambia con el tiempo, puede ser beneficioso mantener la resolución completa para capturar con precisión las características de la señal [164].
- **Detección de artefactos:** En el caso de los EEG, puede ser necesario mantener la frecuencia de muestreo original para detectar anomalías sutiles con precisión [165].
- **Requisitos de alta resolución**: ciertas aplicaciones pueden requerir un análisis de alta resolución, donde el submuestreo podría degradar la calidad de la señal procesada [166].
- **Regularidad de los datos**: Puede ser útil mantener la resolución de cada nivel para tareas de preprocesamiento consecutivas para una mayor generalización. En este caso, el cálculo de la entropía.

La GPU, por tanto, se hace imprescindible si se quiere evitar el submuestreo, ya que este preprocesamiento, si se realiza de manera secuencial, puede ser extremadamente costoso debido a la cantidad ingente de datos de los ficheros EDF generados de los EEG y el número de niveles necesarios para llegar a la frecuencia de las ondas beta, alfa y theta. Se emplea un algoritmo de convolución relativamente sencillo (Algoritmo 4.7), que funciona de la siguiente manera:

1. Cada *thread* calcula su índice del registro, índice del canal e índice de muestra dentro del canal basándose en sus IDs de bloque e *thread* [2-4].
2. Itera sobre N_{dr}, N_{ch} y N_s si se sobrepasa el tamaño de la malla [8-10].
3. Se calcula el índice `out_idx` para el que se calcula la convolución [11].
4. Se determina el tamaño de los filtros de paso alto (`hpf`) y paso bajo (`lpf`) [12].
5. Se itera a lo largo del filtro, seleccionando los puntos de la curva de entrada `input` alrededor de `out_idx` [15-22].
6. En caso que el filtro se deba aplicar a un dato de otro registro hay que ver cuántos registros se ha saltado [16], teniendo en cuenta que si el dato pertenece a un registro anterior [17], se deberá restar 1 al número de saltos [18] (de lo contrario, el salto sería 0).
7. El número de saltos permitirá situar en puntero en el registro correcto en el se encuentra el dato [19]. Además, en estos casos, hay que cambiar de canal en sentido contrario [20], para que `s+i` lo sitúe de nuevo en el canal de la señal para la que se quiere calcular la convolución.
8. Con estos ajustes, se seleccionan los puntos temporalmente contiguos para cada señal [21].
9. Se convolucionan los filtros `hpf` y `lpf` para los datos seleccionados, asegurando que no se produce un desbordamiento de memoria [22-25].
10. Se calculan los coeficientes de detalle y aproximación para el índice en cuestión, manteniendo el mismo tipo de datos que los de entrada [27-28].

Como se desea reducir el tamaño de los datos, se realiza el cálculo de la entropía por ventanas, al igual que se hizo con la STFT, utilizando también la GPU mediante el kernel mostrado en el Algoritmo 4.8, que es similar al Algoritmo 4.6, pero con algunas diferencias:

1. Crea un histograma que se empleará como la función de probabilidad $p(x)$ de la ecuación (4.22) [11].

Algoritmo 4.7 Kernel para la Transformada Discreta de Wavelet (DWT) en GPU.

```
1: function DWT(input, detail, approx, hpf, lpf, N_dr, N_ch, N_s)
2:     registro ← blockIdx.x · blockDim.x + threadIdx.x
3:     canal ← blockIdx.y · blockDim.y + threadIdx.y
4:     muestra ← blockIdx.z · blockDim.z + threadIdx.z
5:     stride.x ← blockDim.x · gridDim.x
6:     stride.y ← blockDim.y · gridDim.y
7:     stride.z ← blockDim.z · gridDim.z
8:     for dr ← registro to N_dr step stride_x do
9:         for ch ← canal to N_dr step stride_y do
10:            for s ←muestra to N_s step stride_z do
11:                out_idx ← dr · N_ch · N_s + ch · N_s + s
12:                F_s ← len(hpf)                          ▷ Tamaño del filtro
13:                approxSum ← 0.0
14:                detailSum ← 0.0
15:                for i ← −F_s/2 to F_s/2 do
16:                    saltos ← int((s + i)/N_s)
17:                    esAnterior ← (s+i) < 0
18:                    saltos ← saltos - esAnterior
19:                    dr ← dr + saltos
20:                    ch ← ch - saltos
21:                    idx ← dr · N_ch · N_s + ch · N_s + (s + i)
22:                    if idx ≥ 0 and idx < N_dr · N_ch · N_s then
23:                        approxSum ← approxSum + lpf[i+F_s/2] · input[idx]
24:                        detailSum ← detailSum + hpf[i+F_s/2] · input[idx]
25:                    end if
26:                end for
27:                approx[out_idx] ← castToInt(approxSum)
28:                detail[out_idx] ← castToInt(detailSum)
29:            end for
30:        end for
31:    end for
32: end function
```

2. Se calcula a qué barra correspondería el valor de sa señal de entrada, normalizando con un valor máximo `max` y un valor mínimo `min`, teniendo en cuenta el número de barras deseado, `PRECISIÓN` [15].

3. Se actualiza la frecuencia de la barra en concreto [17].

4. Se calcula la entropía de Shannon según la ecuación (4.22) [19].

5. Se almacena el resultado en un *array* de salida `output` para la ventana, canal y nivel de la DWT [20].

Algoritmo 4.8 Kernel para la entropía por ventanas en GPU.

```
1: function CALCULARENTROPÍA(input, output, N_dr, N_ch, N_s, N_ch, N_l, W_s, max, min, nivel)
2:     ventana ← blockIdx.x · blockDim.x + threadIdx.x
3:     canal ← blockIdx.y · blockDim.y + threadIdx.y
4:     stride.x ← blockDim.x · gridDim.x
5:     stride.y ← blockDim.y · gridDim.y
6:     a ← max(1, N_s/W_s)                          ▷ Ventanas por registro (si N_s > W_s)
7:     b ← max(1, W_s/N_s)                          ▷ Registros que hay por ventana (si N_s < W_s)
8:     N_w ← N_dr· a/b
9:     for ch ← canal to N_ch step stride.y do
10:        for w ← ventana to N_w step stride.x do
11:            histograma[PRECISIÓN] ← 0            ▷ Inicializa el histograma de valores
12:            subventana ← w % a
13:            for i ← 0 to W_s − 1 do
14:                registro ← w · b/a + i/N_s
15:                barra ← (input[registro·N_ch · N_s + ch·N_s + subventana·W_s +i]-min)/(max-min)·(PRECISIÓN-1)
16:                barra ← min(temp, PRECISIÓN-1)
17:                histograma[barra] ← histograma[barra] + 1
18:            end for
19:            H ←ShannonEntropy(histograma, W_s)
20:            output[N_l · (N_ch·w+ch)+nivel] ← H
21:        end for
22:    end for
23: end function
```

Integrando los dos *kernels* mencionados, el procedimiento se recoge en el Algoritmo 4.9, donde se aplica la DWT a N_l niveles [2], se calcula la entropía de los coeficientes de detalle en ventanas de tamaño W_s [3] y se utilizan los coeficientes de aproximación como entrada al siguiente nivel [4]. La Figura 4.39 muestra un ejemplo de cómo se generaría un nuevo EDF con este método para $W_s = N_s$ y tres niveles de aplicación de la DWT. De nuevo, en este ejemplo el factor de compresión es $3/W_s$. El dato representado por $r_{w,ch,n}$ corresponde al nivel n de la DWT, del canal o señal ch para la ventana w calculada por un *thread* de la GPU.

Algoritmo 4.9 Procedimiento de simplificación de los datos por el método de DWT + entropía.

Require: input, output, detail, approx, hpf, lpf, N_{dr}, N_{ch}, N_s, N_l, W_s, max, min

1: **for** nivel $\leftarrow 0$ **to** N_l **do**
2: DWT(input, detail, approx, hpf, lpf, N_{dr}, N_{ch}, N_s)
3: calcularEntropía(detail, output, N_{dr}, N_{ch}, N_s, N_{ch}, N_l, W_s, max, min, nivel)
4: swap(input, aprox)
5: **end for**

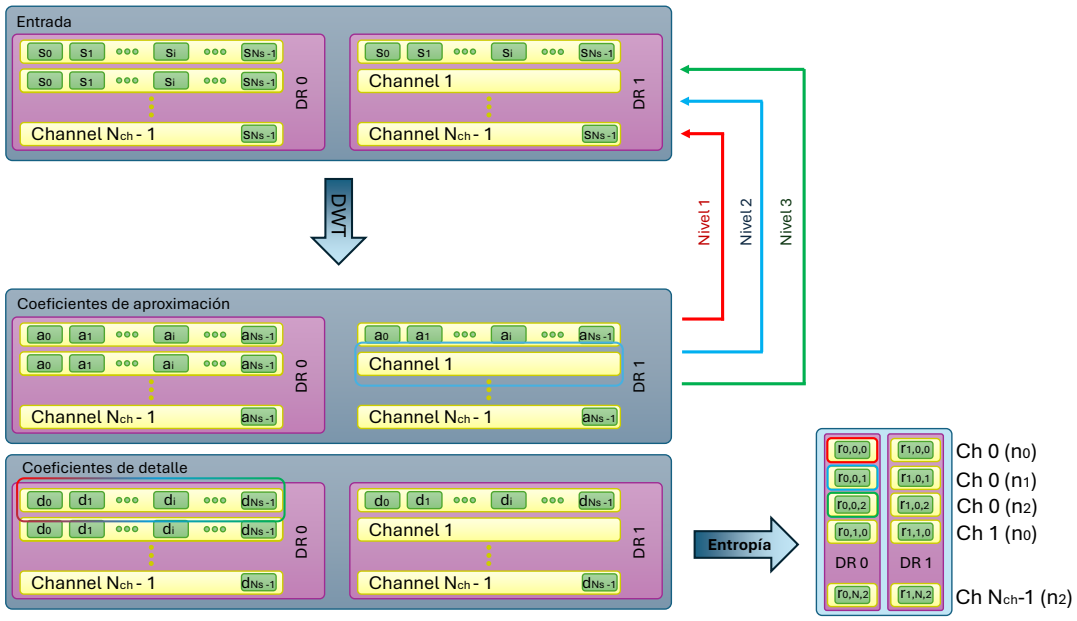


Figura 4.39: Generación de un nuevo EDF tras aplicar el kernel de la DWT a tres niveles junto con la entropía por ventanas de tamaño N_s.

La aceleración en GPU frente a la CPU con este enfoque es claramente determinante en este caso, como se aprecia en la Figura 4.40, incluso para un relativamente pequeño número de registros.

4.3.5. Visión artificial en la epilepsia

Una vez se ha conseguido reducir el tamaño de los datos para poder almacenarlos y se ha comprobado que siguen siendo válidos para determinar las crisis epilépticas, se quiere entrenar una red neuronal que pueda detectarlas o incluso estudiar su naturaleza.

Teniendo en cuenta el gran potencial que tienen las redes de convolución, se decidió transformar los datos, obtenidos del preprocesamiento de la STFT o de la DWT+entropía,

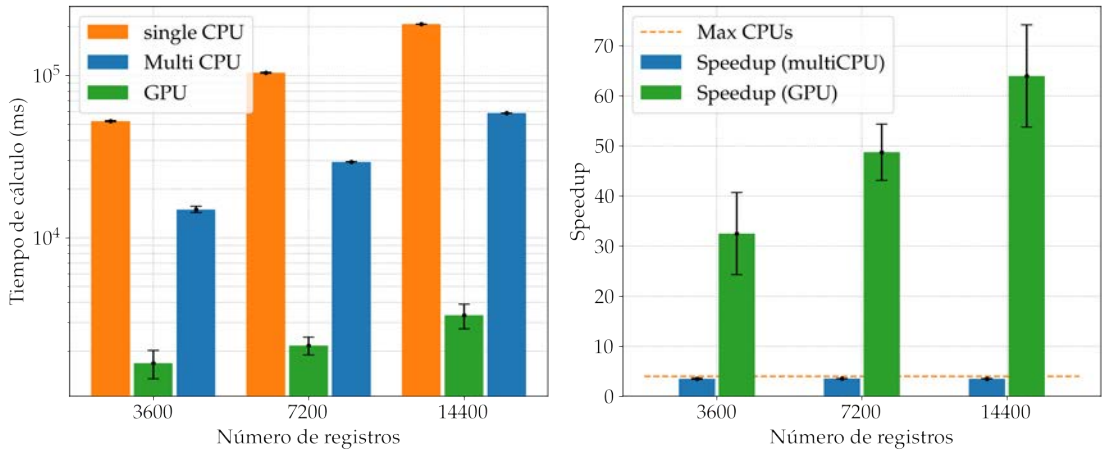


Figura 4.40: Aceleración del método de la DWT con entropía medida en archivos de distintos N_{dr}.

en imágenes. De esta manera, en lugar de generar un nuevo EDF de salida, por cada EDF de entrada se generan una serie de archivos con formato PNG, que corresponden a ventanas de tiempo de varios segundos de duración. Para cada imagen:

- Un determinado píxel representa el resultado del preprocesamiento para una ventana calculada por un *thread*. Para el kernel de la STFT, cada canal de color RGB es el resultado normalizado de la amplitud de una determinada frecuencia. Para el de la DWT+entropía, es el resultado normalizado de un nivel de descomposición (por ejemplo, rojo para ondas beta, verde para alfa y azul para theta).
- Las distintas filas de píxeles, representan cada una de las señales o canales del EEG.
- Las columnas de píxeles representan el tiempo, de manera que si el kernel se ejecuta con una frecuencia de ventana de 1 Hz, cada una de ellas representa el estado del paciente en cada segundo.
- Si el archivo EDF original tiene anotaciones de los momentos de crisis, se hace distinción de las imágenes. Así, se almacenan en rutas diferentes en función de si en la ventana temporal que abarca la imagen existen crisis epilépticas o no. Esto permite almacenar las imágenes de forma etiquetada para un entrenamiento supervisado.

El resultado, son imágenes como las que se muestran en la Figura 4.41, obtenidas de una de las grabaciones del paciente con id *chb01* del conjunto de datos CHB-MIT [76]. Ya se intuye en este caso que si el ojo humano es capaz de apreciar visualmente esta región de crisis, una red neuronal entrenada también puede ser capaz de distinguir estas situaciones.

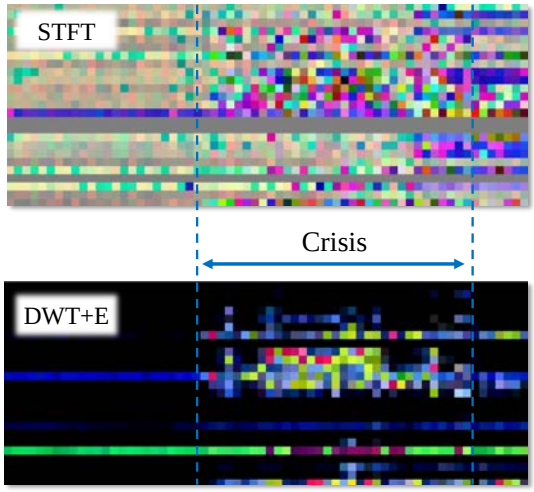


Figura 4.41: Imágenes generadas tras aplicar los dos tipos de preprocesamientos para un mismo momento de una grabación, apreciándose un momento de crisis según indican las anotaciones del EDF original.

Aprendizaje supervisado para la detección de crisis

Se construyó una red neuronal convolucional relativamente sencilla debido al tipo de imágenes que se estaba empleando. No se ha profundizado en la estructura, ya que se quiere probar únicamente la validez del preprocesamiento de datos empleado. La única diferencia con respecto a la arquitectura básica de referencia de este tipo de redes [167], es la incorporación de una serie de metadatos de entrada, como la edad o el género, de manera previa a las capas densamente conectadas. La etiqueta $\bar{Y}$ en este caso es un vector de un único valor y, que indica si la imagen pertenece a una ventana donde se da una crisis (1) o no se da (0). Tanto la etiqueta como los metadatos de entrada, se incorporan en la ruta del archivo. Para el entrenamiento, se han empleado imágenes obtenidas con el kernel de la DWT a partir del conjunto de datos CHB-MIT [76].

Debido a la existencia de una gran desproporción entre la cantidad de imágenes con crisis respecto a las que no, se han seguido varias estrategias para el entrenamiento:

- Se ha aplicado *data augmentation* únicamente a las ventanas que mostraban crisis, generando imágenes donde los píxeles aparezcan temporalmente desplazados (eje X de la imagen).

- Se penalizan más los errores de las imágenes con crisis, añadiendo proporcionalmente más peso a esta clase (parámetro `class_weight`).

- El sesgo de la neurona de salida se inicializa de manera previa al entrenamiento:

$$b = \log\left(\frac{\sum y_i = 1}{\sum y_i = 0}\right) \tag{4.23}$$

La función de coste de la ecuación (2.14) empleada en este caso, utiliza el error binario de entropía cruzada,

$$J(\mathbf{W}) = \frac{1}{B}\sum_{i=1}^{B} y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i). \tag{4.24}$$

Para estimar la bondad del entrenamiento se utilizan métricas que son más adecuadas para casos de clases desproporcionadas.

La **exactitud** (o *accuracy*, en inglés) es la proporción de todas las clasificaciones correctas, ya sean positivas o negativas,

$$A = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}, \tag{4.25}$$

donde T_p y T_n son los aciertos de las imágenes de crisis y no crisis, respectivamente, mientras que F_p y F_n son los fallos de esas dos clases.

La **recuperación** (o *recall*, en inglés) es la proporción de todos los positivos reales que se clasificaron correctamente como positivos,

$$R = \frac{T_p}{T_p + F_n}. \tag{4.26}$$

La **precisión** es la proporción de todas las clasificaciones positivas del modelo que realmente son positivas,

$$P = \frac{T_p}{T_p + F_p}. \tag{4.27}$$

La **curva precisión-recuperación** es la relación gráfica entre la precisión y la recuperación evaluados en múltiples umbrales de decisión. Se puede descomponer en los cálculos fundamentales de precisión y recuperación para una época t, y luego extenderlo al cálculo del área bajo la curva (AUC-PRC). De forma discreta, aplicando la regla del trapecio:

$$\text{AUC-PRC} = \sum_{i=1}^{t-1}(R_{i+1} - R_i) \cdot \frac{P_i + P_{i+1}}{2}. \tag{4.28}$$

Estas tres últimas métricas, son más relevantes para este tipo de entrenamientos. Así, en la Figura 4.42 se puede comprobar que este clasificador de crisis epilépticas desarrolla un

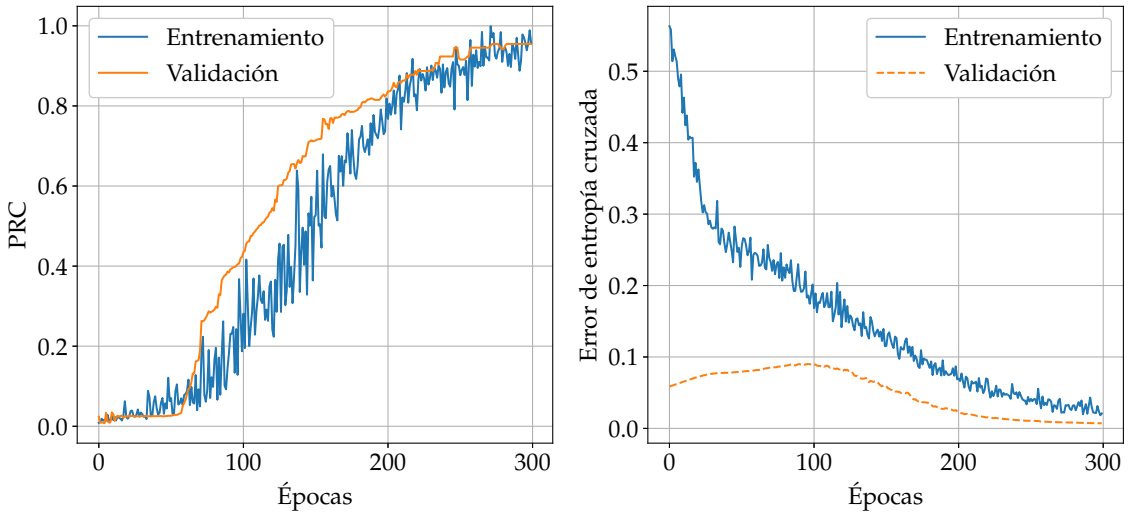
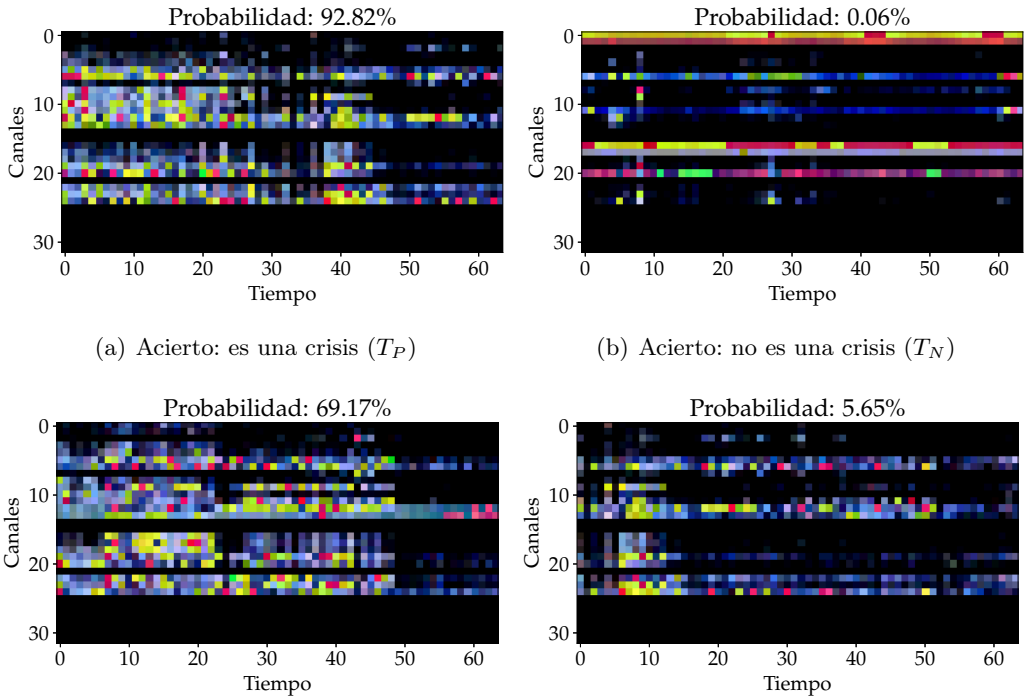


Figura 4.42: Entrenamiento supervisado para la CNN que detecta crisis epilépticas.

comportamiento satisfactorio. Por otra parte, la Figura 4.43 muestra ejemplos de los cuatro casos posibles, de aciertos y fallos del clasificador, para imágenes obtenidas del paciente con id *chb09* del conjunto de datos CHB-MIT [76].



(a) Acierto: es una crisis (T_P)

(b) Acierto: no es una crisis (T_N)

(c) Fallo: no es una crisis (F_P)

(d) Fallo: es una crisis (F_N)

Figura 4.43: Desarrollo del clasificador ante distintas imágenes del paciente con id *chb09* del conjunto de datos CHB-MIT.

Aprendizaje no supervisado para clasificación del tipo de crisis

Para este tipo de entrenamiento, se empleó una ANN muy básica:

- Un codificador con tres capas convolucionales, duplicando el número de filtros por cada capa.
- Un espacio latente de dos variables, con el fin de poder representarlo en un plano bidimensional. Se le aplica el enfoque estadístico mencionado en la Sección 2.2.3.
- Un decodificador con la estructura inversa al codificador.
- La función de coste típica de las ANNs con enfoque estadístico, indicada en la ecuación 2.19.

De nuevo, no se trata de encontrar una clasificador que realmente permita agrupar los datos en distintos grupos o *clusters* que distingan los distintos tipos de crisis epilépticas, sino únicamente de validar el método. Por ello, se realizó, a modo de prueba, el *clustering* representado en la Figura 4.44. Tan solo con este ensayo, ya se puede distinguir que los parámetros de edad y género del paciente, o la escala de magnitud (valor físico máximo), son influyentes en las señales medidas por el EEG. Se aprecia que algunas de las imágenes de crisis epilépticas son claramente distinguibles del resto (zona superior de las gráficas), pero hay muchas otras que podrían ser más difíciles de identificar. Esta representación del espacio latente corresponde al entrenamiento de la Figura 4.45, donde se ha utilizado un coeficiente de desenredo $\beta = 1$.

Con estas dos redes entrenadas y validadas se da por concluido, para este caso de estudio, el Ciclo de Vida del desarrollo de los modelos que se mencionaba en la Sección 1.1. Se ha podido ofrecer una visión íntegra de todas las etapas:

1. Comprensión del problema tras varias entrevistas con el personal médico del hospital.
2. Recopilación de archivos EDF de distintas bases de datos y otros generados por el EEG del hospital Carlos Haya de Málaga.
3. Desarrollo de librerías más eficientes para la lectura de los archivos.
4. Transformación de los datos en imágenes, más comprensibles para las CNN.
5. Entrenamiento de una CNN y un VAE.
6. Validación para comprobar que los modelos detectan las crisis epilépticas y las pueden clasificar.

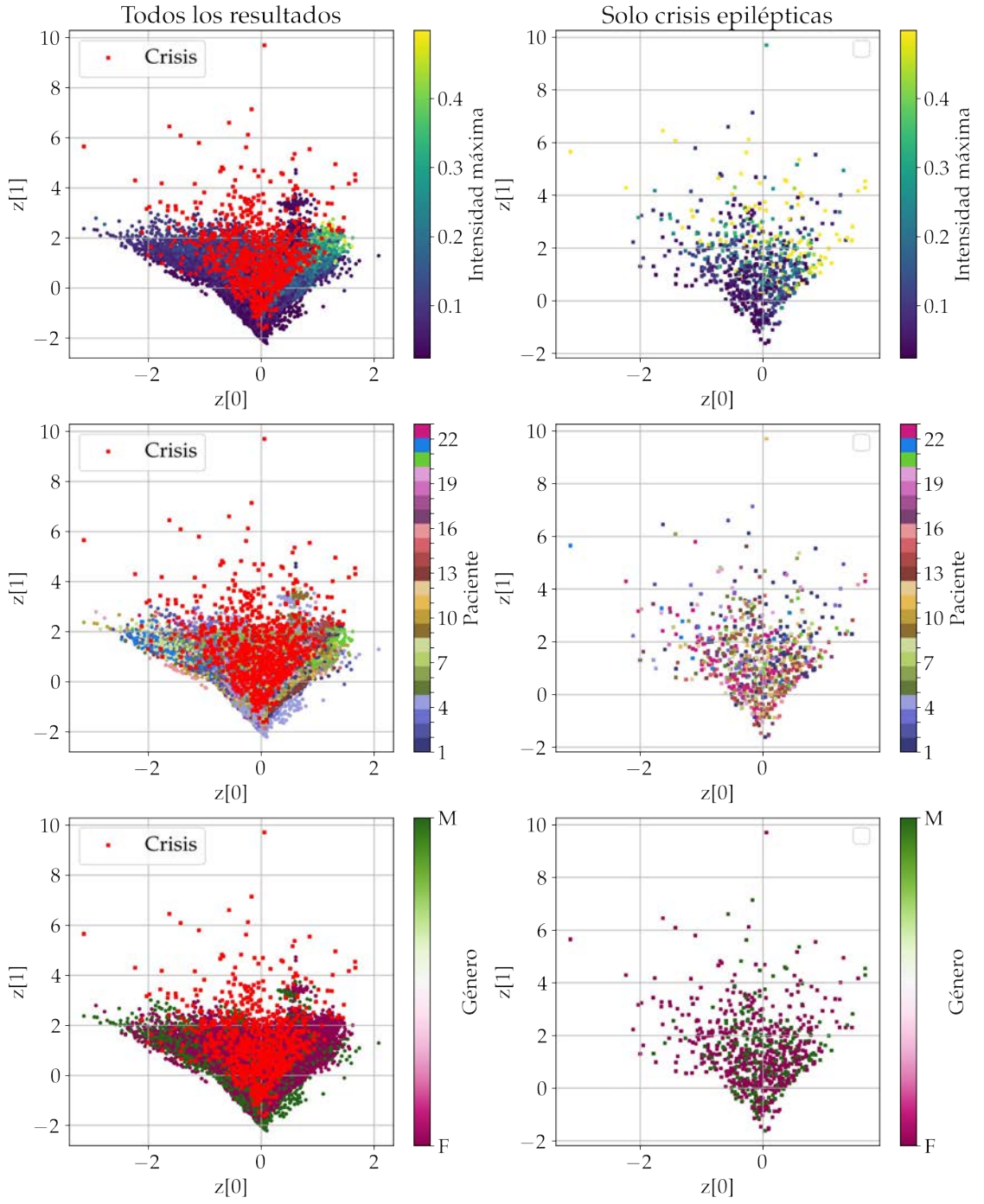


Figura 4.44: *Clustering* resultante del entrenamiento no supervisado con las imágenes generadas, visualizado para distintos parámetros.

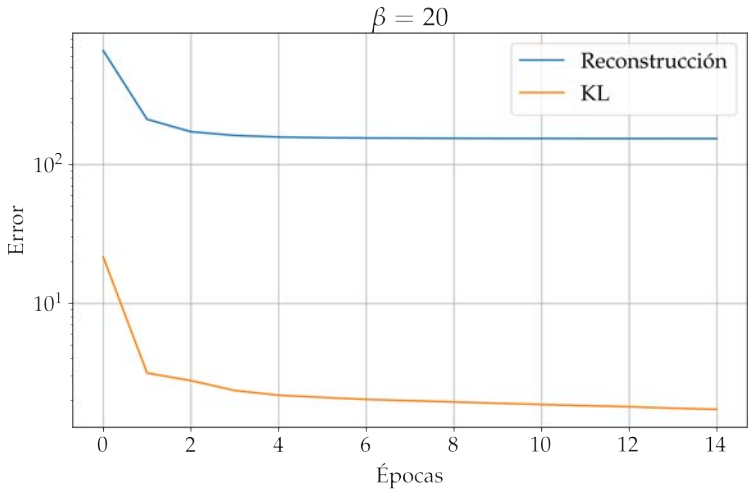


Figura 4.45: Entrenamiento no supervisado para la ANN que clasifica las imágenes generadas.

7. Desarrollo de librerías dinámicas para su futura implementación en aplicaciones.

Las conclusiones específicas de esta línea de investigación se discutirán en la Sección 5.2.3.

4.3.6. Publicaciones

La gestión de los entrenamientos de las redes neuronales de esta línea de investigación se llevaron a cabo mediante un administrador de ejecución bajo el sistema de colas Slurm, publicado en el *Acta Polytechnica Hungarica*, Q2 en las categorías *Engineering* y *Multidisciplinary* [1]. El proceso de simplificación de los EDF y la nueva librería de lectura acelerada se publicaron en las actas del congreso Euro-PAR 2024, celebrado en Madrid [4]. La recolección y tratamiento de datos de distintas fuentes médicas se publicaron en las actas del congreso UCAmI 2023, celebrado en la Riviera Maya (México) [5]. También han sido aceptados y publicados varios trabajos relacionados al congreso CMMSE 2023, celebrado en Rota (Cádiz) [6], a las Jornada Sarteco 2023 de Ciudad Real [11] y al CEDI 2024 de La Coruña [9].

Capítulo 5

Conclusiones

En este capítulo se presentan las conclusiones derivadas de esta tesis, estructuradas en dos niveles. En primer lugar, se exponen las conclusiones generales, que abarcan los aspectos más amplios y transversales del estudio, destacando las contribuciones principales y los avances alcanzados. Posteriormente, se detallan las conclusiones particulares para cada uno de los casos de estudio abordados, con un enfoque específico en los resultados obtenidos, su relevancia en cada contexto y una propuesta de posibles trabajos futuros.

5.1. Generales

En esta tesis doctoral se han desarrollado y estudiado metodologías innovadoras para la optimización del posprocesamiento y preprocesamiento de datos en aplicaciones de DL mediante GPUs. A través de los casos de estudio abordados, se ha demostrado que la integración de estos dispositivos en otras etapas del Ciclo de Vida de la ciencia de datos puede mejorar significativamente la eficiencia computacional y la aplicabilidad de los modelos de inteligencia artificial.

Tradicionalmente, las GPUs se han empleado principalmente en las fases de entrenamiento y validación de modelos de aprendizaje profundo. En esta tesis, se ha demostrado que su aplicación en etapas de preprocesamiento y posprocesamiento es, no solo viable, sino altamente eficiente. La aceleración conseguida en tareas como la normalización, el aumento de datos y el análisis post-entrenamiento permite reducir significativamente los tiempos totales del ciclo, haciendo posible abordar proyectos más ambiciosos y con mayores demandas de computación.

Además, las técnicas de preprocesamiento propuestas en esta tesis han permitido superar una de las principales limitaciones del DL: la insuficiencia de datos en ciertas aplicaciones. Al generar nuevos conjuntos de datos mediante estrategias como el aumento de datos y el filtrado, se ha logrado habilitar el uso de redes neuronales profundas en dominios donde anteriormente no era viable, ampliando así el alcance del DL a problemas con datos limitados o incompletos. Los resultados obtenidos en esta tesis contribuyen al avance del conocimiento en el ámbito de la computación de altas prestaciones y su aplicación en problemas complejos, así como de proporcionar una base metodológica que puede ser adaptada y extendida a otros dominios de la ciencia de datos.

5.2. Específicas

5.2.1. Descubrimiento de fármacos

En esta línea de investigación se han transformado miles de objetos tridimensionales en cientos de imágenes de baja resolución por cada uno de ellos, aprovechando los recursos informáticos disponibles mediante computación heterogénea. Las imágenes de cada objeto se capturaron proyectándolas desde un conjunto de puntos equidistantemente espaciados sobre una esfera hipotética de distancia focal infinita, incluyendo la rotación de la cámara sobre su eje, mediante una técnica denominada «FSST» (*Fibonacci Spherical Spiral with Twist*). La imágenes así generadas se emplearon para entrenar una red neuronal, utilizando un conjunto de datos de moléculas de una base de datos farmacéutica. Los resultados del entrenamiento fueron totalmente satisfactorios, lo que demuestra que la técnica propuesta representa un avance significativo en el reconocimiento de objetos 3D. También se demostró que las proyecciones de baja resolución apenas afectan a la capacidad de reconocimiento, mientras que reducen drásticamente la eficiencia del modelo. La aplicación, disponible en un repositorio público y en una aplicación web, se comporta como una capa de preprocesamiento que mejora la versatilidad de las redes neuronales tradicionales para el reconocimiento bidimensional de imágenes.

Trabajos futuros

Este trabajo abre un nuevo enfoque para el descubrimiento de fármacos. En este sentido, se proponen algunas ramas para futuras investigaciones:

- *Clustering* de moléculas similares utilizando ANNs [168]. Estas redes permitirían entender qué variables sigue la red neuronal para clasificar, además de proporcionar

una forma alternativa de encontrar similitudes de forma.

- Añadir *Evidential Deep Learning* al final de la red neuronal [169]. En este caso, la red también calculará el nivel de confianza en la matriz de probabilidad de salida. Por ejemplo, si se presenta a la red neuronal una molécula nueva, que no se ha utilizado para el entrenamiento, una probabilidad alta no tiene por qué significar que las moléculas sean similares. Estas técnicas ya se han aplicado al DL para el cribado virtual de moléculas [170].

- Explorar posibles sinergias entre técnicas de reconocimiento de objetos 3D mediante IA y algortimos de optimización, utilizando estos últimos para reformular la función de error de entrenamiento y hacerla más precisa.

5.2.2. SkewEngine

El correcto alineamiento de datos en la memoria del computador es un factor cada vez más importante en el diseño de algoritmos eficientes que procesen datos estructurados de una forma intensiva. En el caso de mallas regulares de dos o más dimensiones, los algoritmos que realizan operaciones en una dirección concreta que sea diferente a la dirección principal de alineamiento (la que coincide con la secuencia de almacenamiento de los correspondientes datos), tendrán un patrón de acceso a los datos que, desde la perspectiva hardware, resulta casi completamente aleatorio, y en consecuencia, provoca innumerables fallos de cache que pueden ser catastróficos, especialmente en el caso de grandes volúmenes de datos, como aquellos generados en un TAC o en radioastronomía.

El re-alineamiento de datos en la dirección correspondiente a las operaciones del algoritmo es una operación relativamente costosa si el volumen de datos es elevado, pero como su complejidad computacional es lineal y por tanto, escala bastante bien, puede merecer la pena el coste de la reorganización de la información, siempre y cuando las operaciones de los algoritmos tengan complejidades superiores. En los experimentos mostrados en esta línea de investigación, correspondientes a tres problemas computacionales de alta y media complejidad, como son la cuenca visual total en modelos digitales de elevación, la parametrización de imágenes borrosas por movimiento, y la transformada Radon bidimensional, se ha demostrado que el re-alineamiento de la información ya merece la pena en algoritmos de complejidad media, superando incluso a los mejores resultados publicados, y produce espectaculares mejoras, de varios órdenes de magnitud, en los problemas de mayor complejidad.

Para facilitar el uso de los algoritmos de interpolación y extrapolación de las mallas en las N direcciones elegidas (que son las etapas que preceden y suceden a cualquier algoritmo que realice su operación en una dirección concreta), se ha elaborado una clase en lenguaje

C++, denominada *skewEngine*, que gracias a la explotación del paralelismo embarazoso de la interpolación mediante OpenMP y OpenCL o CUDA, hace que la parte más laboriosa de la propuesta de este casi de estudio sea simple y rápida. Además, *skewEngine* está diseñado de una forma que facilita la implementación de otros algoritmos de cálculo intensivo de datos en mallas regulares usando direcciones arbitrarias de procesamiento.

Trabajos futuros

- Los experimentos se han desarrollado mediante cálculos en direcciones equitativamente distribuidas en dos dimensiones (normalmente, en $N = 360$ direcciones, separadas un grado), pero también se presentan las directrices de una implementación de la herramienta *skewEngine* en 3 dimensiones, utilizando, igualmente, un conjunto equitativo de direcciones de búsqueda tridimensional que se calculan utilizando la espiral esférica de Fibonacci, al igual que en el caso de estudio del descubrimiento de fármacos. Se propone, para validar la utilidad de esta extrapolación de *skewEngine* , su uso para el proyecto del *Square Kilometre Array* (SKA) [171], participando en uno de los *SKA Data Challenges*.
- Crear un modelo de red neuronal que tome como entrada los datos generados por *skewEngine*. Por ejemplo, se podría estudiar mediante una regresión multivariable, la dependencia de la visibilidad total con respecto al precio de los inmuebles en una zona geográfica.

5.2.3. Detección de epilepsia

La detección de ataques de epilepsia usando EEG ha sido un problema ampliamente estudiado, desarrollando soluciones avanzadas (principalmente usando Deep Learning) que logran un rendimiento sobresaliente en conjuntos de datos disponibles en la literatura. Sin embargo, la gran cantidad de datos de entrenamiento y las capacidades de cómputo necesarias hacen que ésta sea una tarea muy desafiante.

En esta línea de investigación se han utilizado algunas técnicas para reducir la cantidad de datos de entrenamiento y procesarlos de forma óptima utilizando GPUs. En los resultados experimentales se utilizó la Transformada de Fourier de Tiempo Corto (STFT) y la Transformada Wavelet Discreta (DWT) para reducir el problema de almacenamiento. Este método permite comprimir los datos hasta un 99 %, pero manteniendo la información significativa que permite detectar ataques epilépticos. El uso de GPUs mejora significativamente la eficiencia computacional. Los resultados muestran que, utilizando la GPU, el tiempo de procesamiento de un lote de datos registrado durante dos días se reduce de 23 minutos a 3 segundos.

La detección de crisis epilépticas se ha demostrado con el entrenamiento de una red neuronal que analiza imágenes generadas con las técnicas de la STFT y la DWT, que representan ventanas de tiempo del estado cerebral del paciente, y predice si en ellas se produce crisis o no. También se han empleado redes *Autoencoders* para clasificar los distintos tipos de crisis observadas en los distintos pacientes. Además, se ha desarrollado en C++ un nuevo enfoque para la lectura de archivos EDF, que mejora cientos de veces el tiempo computacional empleado por otras librerías existentes.

Trabajos futuros

En vista a los resultados obtenidos, se propone:

- Desarrollar una librería dinámica y una aplicación web que permita al equipo médico o bien simplificar un fichero EDF obtenido del EEG o bien detectar los momentos de crisis mediante una red neuronal.
- Implementar nuevos *kernels* en GPU utilizando otras técnicas de la bibliografía. Por ejemplo, generando gráficos rasterizados en formato `.tiff` a partir de los resultados completos de la STFT en lugar de únicamente sus amplitudes dominantes [172]. En ellos, cada capa del gráfico podría pertenecer a un canal concreto del EEG. Este tipo de archivos podrían ser fácilmente convertidos a tensores de entrada para las CNN y, por tanto, mejorar aún más la validez de los modelos para que puedan ser utilizados en Medicina.

Referencias

[1] M. Lupión, N. C. Cruz, F. Romero, J. Sanjuan, and P. M. Ortigosa. A lightweight execution manager for training Tensorflow models under the Slurm Queuing System. *Acta Polytechnica Hungarica*, 22:63–78, Jan 2025.

[2] F. Romero, P. M. Ortigosa, G. Bandera, and L. F. Romero. Skewengine: enhancing performance of intensive calculations on regular meshes. *The Journal of Supercomputing*, 80:1–19, Feb 2024.

[3] F. Romero, L. Romero, J. L. Redondo, and P. M. Ortigosa. Drugs discovery by shape similarity using deep learning. *Journal of Optimization Theory and Applications*, 204, Jan 2025.

[4] F. Romero, M. Lupión, N. C. Cruz, L. F. Romero, and P. M. Ortigosa. On the use of GPU computing for accelerating EEG preprocessing. In J. Carretero, S. Shende, J. Garcia-Blas, I. Brandic, K. Olcoz, and M. Schreiber, editors, *Euro-Par 2024: Parallel Processing*, pages 270–282, Cham, 2024. Springer Nature Switzerland.

[5] M. Lupión, F. Romero, L. F. Romero, J. F. Sanjuan, and P. M. Ortigosa. Edge IoT system for wearable devices: Real-time data processing, inference, and training for activity monitoring and health evaluation. In José Bravo and Gabriel Urzáiz, editors, *Proceedings of the 15th International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2023)*, pages 131–142, Cham, 2023. Springer Nature Switzerland.

[6] F. Romero, LF. Romero, M. Lupión, Sanjuan J. F., and P. M. Ortigosa. Accelerating Epilepsy Diagnosis and Prediction on EEG Signals with GPU-Based Frequency Domain Analysis and Resampling. In *CMMSE 2023*, Rota, Spain, July 3-8, 2023.

[7] F. Romero, P. M. Ortigosa, G. Bandera, and L. F. Romero. SkewEngine: Enhancing Performance of Intensive Calculations on Regular Mesh Data. In *CMMSE 2023*, Rota, Spain, July 3-8, 2023.

[8] F. Romero, L. F. Romero, J. L. Redondo, and P. M. Ortigosa. Drugs discovery by shape similarity using Deep Learning. In *Europt 2022*, Caparica, Portugal, 20-22 July, 2022.

[9] F. Romero, M. Lupión, N. C. Cruz, L. F. Romero, and P.M. Ortigosa. Optimización de la lectura de archivos EDF en C++: Avances para el procesamiento eficiente en GPU de grandes conjuntos de datos de EEG. In *CEDI 204*, A Coruña, Spain, 17-21 June, 2024.

[10] M. Lupión, N. C. Cruz, F. Romero, J. F. Sanjuan, and P. M. Ortigosa. Herramienta de gestión de entrenamientos de TensorFlow en un clúster de HPC gestionado con SLURM. In *CEDI 204*, A Coruña, Spain, 17-21 June, 2024.

[11] F. Romero, M. Lupión, L. F. Romero, J. F. Sanjuan, and P. M. Ortigosa. Metodología de adquisición y procesamiento de datos para la detección de crisis epilépticas utilizando dispositivos IoT y GPUs. In *Jornadas Sarteco 2023*, Ciudad Real, Spain, 20-22 September, 2023.

[12] F. Romero, L. F. Romero, and G. Bandera. SkewEngine: Reorganización de mallas regulares para cálculo intensivo. In *Jornadas Sarteco 2023*, Ciudad Real, Spain, 20-22 September, 2023.

[13] F. Romero, L. F. Romero, H. Pérez-Sánchez, J. L. Redondo, and P. M. Ortigosa. Reconocimiento de fármacos mediante Inteligencia Artificial. In *Jornadas Sarteco 2022*, Alicante, Spain, September 21-23, 2022.

[14] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2021.

[15] D. L. Poole and A. K. Mackworth. *Artificial Intelligence*. Cambridge University Press, 2017.

[16] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, 1998.

[17] N. J. Nilsson. *The Quest for Artificial Intelligence*. Cambridge University Press, 2009.

[18] W. Ertel. *Introduction to Artificial Intelligence*. Springer, 2018.

[19] G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson, 6 edition, 2009.

[20] K. Warwick. *Artificial Intelligence: The Basics*. Routledge, 2012.

[21] M. A. Boden. *AI: A Very Short Introduction*. Oxford University Press, 2018.

[22] S. Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Viking, 2019.

[23] M. Johnson, R. Jain, and P. Brennan-Tonetta. Impact of big data and artificial intelligence on industry: Developing a workforce roadmap for a data driven economy. *Global Journal of Flexible Systems Management*, 22:197–217, 2021.

[24] NVIDIA Corporation. GPU-based Deep Learning inference: A performance and power analysis, 2015.

[25] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, Mar 2008.

[26] A. Paszke et al. Pytorch: An imperative style, high-performance deep learning library, 2019. Conference on Neural Information Processing Systems (NeurIPS) 2019.

[27] M. Abadi, others, A. Agarwal, P. Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[28] J. M. Wing. The Data Life Cycle. *Harvard Data Science Review*, 1(1), Jul 2019.

[29] D. Horvath. A virtual screening approach applied to the search for trypanothione reductase inhibitors. *Journal of medicinal chemistry*, 40(15):2412–2423, 1997.

[30] S. Puertas-Martín. *Computación de Altas Prestaciones para la Resolución de Problemas de Optimización en Bioinformática*. PhD thesis, Universidad de Almería, 2020.

[31] W. Berger, B. Ralph, M. Kaczocha, J. Sun, T. Balius, R. Rizzo, S. Haj-Dahmane, I. Ojima, and D. Deutsch. Targeting Fatty Acid Binding Protein (FABP) Anandamide transporters – A novel strategy for development of anti-inflammatory and anti-nociceptive drugs. *PloS one*, 7:e50968, Dec 2012.

[32] A. Dalby, J. G. Nourse, W. D. Hounshell, A. K. I. Gushurst, D. L. Grier, B. A. Leland, and J. Laufer. Description of several chemical structure file formats used by computer programs developed at molecular design limited. *Journal of Chemical Information and Computer Sciences*, 32(3):244–255, 1992.

[33] World Health Organization. Epilepsy. URL. Accessed: 2023-05-29.

[34] E. Beghi. The epidemiology of epilepsy. *Neuroepidemiology*, 54(2):185–191, 2020.

[35] C. E. Elger and C. Hoppe. Diagnostic challenges in epilepsy: Seizure under-reporting and seizure detection. *The Lancet Neurology*, 17(3):279–288, 2018.

[36] J. J. Shih, N. B. Fountain, S. T. Herman, A. Bagic, F. Lado, S. Arnold, M. L. Zupanc, E. Riker, and D. M. Labiner. Indications and methodology for video-electroencephalographic studies in the epilepsy monitoring unit. *Epilepsia*, 59(1):27–36, Jan 2018.

[37] L. J. Hirsch. Continuous EEG monitoring in the intensive care unit: an overview. *Journal of Clinical Neurophysiology*, 21(5):332–340, 2004.

[38] ACE Cloud. The evolution of GPU, 2025. Accessed: 2025-02-11.

[39] C. McClanahan. History and evolution of GPU architecture a paper survey, 2011.

[40] D. B. Kirk and W. H. Wen-mei. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, 2016.

[41] Girolino. Nvidia GPU evolution: From geforce to ai powerhouse, 2025. Accessed: 2025-02-11.

[42] NVIDIA Corporation. GPU-based deep learning inference: A performance and power analysis. *NVIDIA Developer Blog*, 2019.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[44] J. D. Owens, M. Houston, D. Luebke, et al. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[45] NVIDIA. GeForce RTX 5090 & GeForce RTX 5080 out now, featuring game-changing AI and neural rendering capabilities and DLSS 4 with multi frame generation. URL, 2025. Accessed: 2025-02-24.

[46] AMD. AMD Announces New Gaming Products for Ultimate Gameplay Experience at CES. URL, 2025. Accessed: 2025-02-24.

[47] NVIDIA Corporation. *CUDA C++ Programming Guide*, 2023. Accessed: 2025-02-11.

[48] Khronos Group. *The OpenCL Specification*, 2021. Accessed: 2025-02-11.

[49] NVIDIA Corporation. *NVIDIA cuDNN Dodumentation*, 2023. Accessed: 2025-02-11.

[50] NVIDIA Corporation. *NVIDIA TensorRT Documentation*, 2023. Accessed: 2025-02-11.

[51] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, 38(3):157–174, 2012.

[52] C. Li, Y. Sun, L. Xu, Z. Cao, P. Fan, D. Kaeli, S. Ma, Y. Guo, and J. Yang. Priority-based PCIe scheduling for multi-tenant multi-GPU systems. *IEEE Computer Architecture Letters*, PP:1–1, Nov 2019.

[53] S. J. Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[54] J. Torres-Viñals. *Deep Learning : Introducción práctica con Keras*. Kindle Direct Publishing, Barcelona, 2018.

[55] W. S McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[56] F. Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[57] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[58] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[59] A. Aggarwal, M. Mittal, and G. Battineni. Generative Adversarial Network: An overview of theory and applications. *International Journal of Information Management Data Insights*, 1(1):100004, 2021.

[60] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, Cham, Switzerland, 2 edition, 2023.

[61] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[62] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[63] J. Torres Viñals. *Python deep learning : introducción práctica con Keras y TensorFlow 2*. Marcombo, Barcelona, 2020.

[64] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[65] R. TQ Chen, X. Li, R. B. Grosse, and D. K. Duvenaud. Isolating sources of disentanglement in variational autoencoders. *Advances in neural information processing systems*, 31, 2018.

[66] A. Amini, A. P. Soleimany, W. Schwarting, S. N. Bhatia, and D. Rus. Uncovering and mitigating algorithmic bias through learned latent structure. In *Proceedings of the*

2019 AAAI/ACM Conference on AI, Ethics, and Society, AIES '19, page 289–295, New York, NY, USA, 2019. Association for Computing Machinery.

[67] T. M. Apostol. *Calculus, Volume 1*, chapter "The Chain Rule for Differentiating Composite Functions', pages 174–179. John Wiley & Sons, 2nd edition, 1991.

[68] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28(3) of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, Jun 2013. PMLR.

[69] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[70] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, Jul 2011.

[71] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[72] J. R. Sashank, Satyen K., and Surinder K. On the convergence of Adam and beyond. *ArXiv*, abs/1904.09237, 2018.

[73] S. Puertas-Martín, J. L. Redondo, P. M. Ortigosa, and H. Pérez-Sánchez. Optipharm: An evolutionary algorithm to compare shape similarity. *Scientific Reports*, 9(1):1398, Feb 2019.

[74] S. Tabik, E. L. Zapata, and L. F. Romero. Simultaneous computation of total viewshed on large high resolution grids. *International Journal of Geographical Information Science*, 27(4):804–814, 2013.

[75] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. Lo, A. Marcu, J. R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda, N. Assempour, I. Iynkkaran, Y. Liu, A. Maciejewski, N. Gale, A. Wilson, L. Chin, R. Cummings, D. Le, A. Pon, C. Knox, and M. Wilson. DrugBank 5.0: a major update to the DrugBank database for 2018. *Nucleic Acids Research*, 46(D1):D1074–D1082, Nov 2017.

[76] J. Guttag. CHB-MIT Scalp EEG Database (version 1.0.0), 2010.

[77] P. Detti. Siena Scalp EEG Database (version 1.0.0), 2020.

[78] GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2020.

[79] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, pages 317–319. The MIT Press, 2016.

[80] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[81] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[82] P. D. Lyne. Structure-based virtual screening: an overview. *Drug Discovery Today*, 7(20):1047–1055, Oct 2002.

[83] E. Lionta, G. Spyrou, D. Vassilatis, and Z. Cournia. Structure-based virtual screening for drug discovery: Principles, applications and recent advances. *Current Topics in Medicinal Chemistry*, 14(16):1923–1938, Oct 2014.

[84] P. Ripphausen, B. Nisius, and J. Bajorath. State-of-the-art in ligand-based virtual screening. *Drug Discovery Today*, 16(9-10):372–376, May 2011.

[85] P. C. D. Hawkins, A. G. Skillman, and A. Nicholls. Comparison of shape-matching and docking as virtual screening tools. *Journal of Medicinal Chemistry*, 50(1):74–82, Jan 2007.

[86] P. Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37:241–72, Jan 1901.

[87] S. Puertas-Martín, J. L. Redondo, H. Pérez-Sánchez, and P. M. Ortigosa. Optimizing electrostatic similarity for virtual screening: A new methodology. *Informatica*, pages 1–19, 2020.

[88] S. Palmer. Canonical perspective and the perception of objects. *Attention and performance*, pages 135–151, 1981.

[89] S. Qi, X. Ning, G. Yang, L. Zhang, P. Long, W. Cai, and W. Li. Review of multi-view 3d object recognition methods based on deep learning. *Displays*, 69:102053, 2021.

[90] A. Kanezaki, Y. Matsushita, and Y. Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[91] S. Chen, L. Zheng, Y. Zhang, Z. Sun, and K. Xu. Veram: View-enhanced recurrent attention model for 3d shape classification. *IEEE transactions on visualization and computer graphics*, 25(12):3244–3257, 2018.

[92] S. Tanimura and T. Iwai. Reduction of quantum systems on riemannian manifolds with symmetry and application to molecular mechanics. *Journal of Mathematical Physics*, 41(4):1814–1842, 2000.

[93] O. Diner, C. Diner, A. Doğan, G. Weber, F. Özbudak, and A. Tiefenbach. On the applied mathematics of discrete tomography. *Vychislitel'nye Tekhnologii*, 9, Jan 2004.

[94] S. Kuter. Completing the machine learning saga in fractional snow cover estimation from modis terra reflectance data: Random forests versus support vector regression. *Remote Sensing of Environment*, 255:112294, Mar 2021.

[95] E. B. Saff and A. B. J. Kuijlaars. Distributing many points on a sphere. *The mathematical intelligencer*, 19(1):5–11, 1997.

[96] A. Katanforoush and M. Shahshahani. Distributing points on the sphere, i. *Experimental Mathematics*, 12(2):199–209, 2003.

[97] G. Chukkapalli, S. Karpik, and C. Ethier. A scheme for generating unstructured grids on spheres with application to parallel computation. *Journal of Computational Physics*, 149:114–127, Feb 1999.

[98] R. Swinbank and R. Purser. Fibonacci grids. In *AMS 13th Conference on Numerical Weather Prediction*, Jan 1999.

[99] T. Enomoto. Quasi-uniform grids using a spherical helix. In *Workshop on the Partial Differential Equations on the Sphere*, page 1, 2014.

[100] K. M. Gorski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2), 2005.

[101] R. Bauer. Distribution of points on a sphere with application to star catalogs. *Journal of Guidance Control and Dynamics*, 23:130–137, Jan 2000.

[102] R. F. Dixon. The mathematics and computer graphics of spirals in plants. *Leonardo*, 16, 1983.

[103] M. Roberts. How to evenly distribute points on a sphere more effectively than the canonical fibonacci lattice. URL, Accessed: 2023-01-04, 2021.

[104] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[105] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5028–5037, 2017.

[106] F. Romero and L. F. Romero. 1024 proyecciones de 4 moléculas. URL, 2022. Accessed: 2022-05-30.

[107] A. J. Sanchez. *Novel Parallel Approaches to Efficiently Solve Spatial Problems on Heterogeneous CPU-GPU Systems*. PhD thesis, University of Málaga, 2022.

[108] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of the 25th International Conference on Neural Inform. Process. Sys. - Volume 1*, 2012.

[109] J. T. Townsend. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, 9(1):40–50, Jan 1971.

[110] G. Liu, H. Zeng, and D. K. Gifford. Visualizing complex feature interactions and feature sharing in genomic deep neural networks. *BMC Bioinformatics*, 20(1):401, Jul 2019.

[111] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[112] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[113] I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T. Lin, J. Shlens, and B. Zoph. Revisiting resnets: Improved training and scaling strategies. *CoRR*, abs/2103.07579, 2021.

[114] Mingxing T. and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[115] H. Gholamalinezhad and H. Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

[116] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[117] O. Lanzi. How symmetric a matrix is. URL, 2016. Accessed: 2025-05-30].

[118] N. Park. *Improving Memory Hierarchy Performance Using Data Reorganization*. PhD thesis, University of Southern California, USA, 2002. AAI3093966.

[119] I. Voutchkov, A. Keane, S. Shahpar, and R. Bates. (Re-) Meshing using interpolative mapping and control point optimization. *Journal of Computational Design and Engineering*, 5(3):305–318, 2018.

[120] S. B. Iryanto, F. H. Muttaqien, and R. Sadikin. Irregular grid interpolation using radial basis function for large cylindrical volume. *Jurnal Ilmu Komputer dan Informasi*, 2020.

[121] QGIS Development Team. *QGIS Geographic Information System*. QGIS Association, 2023.

[122] U. Schulzweida. CDO user guide, October 2022.

[123] A. J. Sanchez-Fernandez, L. F. Romero, G. Bandera, and S. Tabik. VPP: Visibility-based path planning heuristic for monitoring large regions of complex terrain using a UAV onboard camera. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, PP:1–1, Dec 2021.

[124] F. Romero and L. F. Romero. Skewengine: Mesh reorganization for computing intensive applications. URL, 2023.

[125] A. J. Sanchez-Fernandez, L. F. Romero, G. Bandera, and S. Tabik. A data relocation approach for terrain surface analysis on multi-gpu systems: a case study on the total viewshed problem. *International Journal of Geographical Information Science*, 35(8):1500–1520, 2021.

[126] A.R. Cervilla, S. Tabik, J. Vias, M. Merida, and L. F. Romero. Total 3D-Viewshed map: Quantifying the visible volume in digital elevation models. *Transactions in GIS*, 2016.

[127] S. Tabik, L. F. Romero, and E. L. Zapata. High-performance three-horizon composition algorithm for large-scale terrains. *International Journal of Geographical Information Science*, 25(4):541–555, 2011.

[128] Bruce P. Bogert. The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. In *Proc. Symposium Time Series Analysis, 1963*, pages 209–243, 1963.

[129] J. Radon. On the determination of functions from their integral values along certain manifolds. *IEEE Transactions on Medical Imaging*, 5(4):170–176, 1986.

[130] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Pearson, London, UK, 3rd edition edition, 2014s.

[131] J. Radon. On the determination of functions from their integral values along certain manifolds. *IEEE Transactions on Medical Imaging*, 5(4):170–176, 1986.

[132] F. Pedregosa et al. Scikit-Learn: machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[133] W. Aarle, W. Palenstijn, J. De Beenhouwer, T. Altantzis, S. Bals, K. Batenburg, and J. Sijbers. The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography. *Ultramicroscopy*, 157, May 2015.

[134] U. R. Acharya, S. V. Sree, G. Swapna, R. J. Martis, and J. S. Suri. Automated EEG analysis of epilepsy: a review. *Knowledge-Based Systems*, 45:147–165, 2013.

[135] A. Sharmila. Epilepsy detection from EEG signals: A review. *Journal of Medical Engineering & Technology*, 42(5):368–380, 2018.

[136] Z. Wei, J. Zou, J. Zhang, and J. Xu. Automatic epileptic EEG detection using convolutional neural network with improvements in time-domain. *Biomedical Signal Processing and Control*, 53:101551, 2019.

[137] U. R. Acharya, S. V. Sree, A. P. C. Alvin, and J. S. Suri. Use of principal component analysis for automatic classification of epileptic EEG activities in wavelet framework. *Expert Systems with Applications*, 39(10):9072–9078, 2012.

[138] W. O. Tatum, B. A. Dworetzky, and D. L. Schomer. Artifact and recording concepts in eeg. *Journal of clinical neurophysiology*, 28(3):252–263, 2011.

[139] J. Rasekhi, M. R. K. Mollaei, M. Bandarabadi, C. A. Teixeira, and A. Dourado. Preprocessing effects of 22 linear univariate features on the performance of seizure prediction methods. *Journal of Neuroscience Methods*, 217(1-2):9–16, 2013.

[140] M. Zhou, C. Tian, R. Cao, B. Wang, Y. Niu, T. Hu, H. Guo, and J. Xiang. Epileptic seizure detection based on EEG signals and CNN. *Frontiers in Neuroinformatics*, 12:95, 2018.

[141] N. D. Truong, A. D. Nguyen, L. Kuhlmann, M. Reza Bonyadi, J. Yang, S. Ippolito, and O. Kavehei. Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram. *Neural Networks*, 105:104–111, 2018.

[142] A. Subasi and E. Ercelebi. Classification of EEG signals using neural network and logistic regression. *Computer Methods and Programs in Biomedicine*, 78(2):87–99, 2005.

[143] O. Faust, U. R. Acharya, H. Adeli, and A. Adeli. Wavelet-based EEG processing for computer-aided seizure detection and epilepsy diagnosis. *Seizure*, 26:56–64, 2015.

[144] A. B. Geva and D. H. Kerem. Forecasting generalized epileptic seizures from the EEG signal by wavelet analysis and dynamic unsupervised fuzzy clustering. *IEEE Transactions on Biomedical Engineering*, 45(10):1205–1216, 1998.

[145] S. M. Usman, S. Khalid, R. Akhtar, Z. Bortolotto, Z. Bashir, and H. Qiu. Using scalp EEG and intracranial EEG signals for predicting epileptic seizures: Review of available methodologies. *Seizure*, 71:258–269, 2019.

[146] R. Akut. Wavelet based deep learning approach for epilepsy detection. *Health Information Science and Systems*, 7(1), 2019.

[147] A. D. Roy and M. M. Islam. Detection of epileptic seizures from wavelet scalogram of EEG signal using transfer learning with alexnet convolutional neural network.

In *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, pages 1–5, 2020.

[148] P. Dohnálek, P. Gajdoš, T. Peterek, and M. Penhaker. Pattern recognition in EEG cognitive signals accelerated by GPU. In *International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pages 477–485. Springer, 2013.

[149] L. Wang, D. Chen, R. Ranjan, S. U. Khan, J. KolOdziej, and J. Wang. Parallel processing of massive EEG data with mapreduce. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 164–171. Ieee, 2012.

[150] Apache Software Foundation. Hadoop, 2010. Versión 0.20.2.

[151] F. Muñoz. *Accelerating massive sensor-based analytics*. PhD thesis, Universidad de Málaga, 2024.

[152] T. van Beelen. EDFlib, 2024. Versión 1.25.

[153] T. van Beelen. EDFlib for Java, 2022. Versión 1.02.

[154] J. Vis. EDFReader, 2019. Versión 1.2.1.

[155] H. Nahrstaedt and S. Kern and L. Cerina and S. Appelhoff and D. T.H. Kao and C. Franklin and S. Clarke and J. Zitting and D. Ojeda and C. Boulay and O. Moore and D. Kosachev. PyEDFlib, 2021. Versión 0.1.23.

[156] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, and M. S. Hämäläinen. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, 7(267):1–13, 2013.

[157] B. Kemp and J. Olivan. European data format 'plus' (edf+), an edf alike standard format for the exchange of physiological data. *Clinical Neurophysiology*, 114:1755–1761, 2003.

[158] P. Peng, Y. Song, L. Yang, and H. Wei. Seizure prediction in EEG signals using STFT and domain adaptation. *Frontiers in Neuroscience*, 15, 2022.

[159] M. K. Kıymık, İ. Güler, A. Dizibüyük, and M. Akın. Comparison of STFT and wavelet transform methods in determining epileptic seizure activity in EEG signals for real-time application. *Computers in Biology and Medicine*, 35(7):603–616, 2005.

[160] R. Shantha-Selva-Kumari and J. Prabin Jose. Seizure detection in EEG using time frequency analysis and SVM. In *2011 International Conference on Emerging Trends in Electrical and Computer Technology*, pages 626–630, 2011.

[161] M. I. Al-kadi, M. B. I. Reaz, and M. A. Mohd-Ali. Compatibility of mother wavelet functions with the electroencephalographic signal. In *2012 IEEE-EMBS Conference on Biomedical Engineering and Sciences*, pages 113–117, 2012.

[162] I. Wijayanto, R. Hartanto, and H. A. Nugroho. Complexity based multilevel signal analysis for epileptic seizure detection. *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, pages 388–393, 2020.

[163] C. V. K. Latha, G. Aparna, M. K. Joseph, and SrikanthNalluri. Detection of normal and epileptic eeg signals using by lifting based dwt transform and neural network. *Solid State Technology*, 63:5787–5795, 2020.

[164] C. Madhu and T. S. Reddy. MST radar signal processing with fast approximate fourier transform using DWT. *Digital Signal Processing*, 6:126–131, 2014.

[165] P. Fuentealba, R. Salvi, J. Henze, A. Burmann, A. Boese, E. J. Gomes-Ataide, M. Spiller, A. Illanes, and M. Friebe. Carotid sound signal artifact detection based on discrete wavelet transform decomposition. *Current Directions in Biomedical Engineering*, 7:299 – 302, 2021.

[166] Suma. Modelling of efficient medical image compression system based on joint operation of DWT and neural network. *Wireless Personal Communications*, 124:3129 – 3143, 2022.

[167] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[168] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, Jul 2012. PMLR.

[169] A. Amini, W. Schwarting, A. Soleimany, and D. Rus. Deep evidential regression. *Advances in Neural Inform. Process. Sys.*, 33, 2020.

[170] A. P. Soleimany, A. Amini, S. Goldman, D. Rus, S. N. Bhatia, and C. W. Coley. Evidential deep learning for guided molecular property prediction and discovery. *ACS Central Science*, 7(8):1356–1367, 2021.

[171] SKA Organisation. Square Kilometre Array (SKA). URL, 2025. Accessed: 2025-02-05.

[172] N. Truong, L. Kuhlmann, M. R. Bonyadi, D. Querlioz, L. Zhou, and O. Kavehei. Epileptic seizure forecasting with generative adversarial networks. *IEEE Access*, PP:1–1, Sep 2019.