



Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Diseño de un modelo neuronal para la detección y la clasificación de intrusiones en redes informáticas

Alumno:
Hugo López Álvarez

Tutor:
Diego García Álvarez

Al aprendizaje y a la informática, que parecen no tener límites.

Agradecimientos

Mi más sincera gratitud a mi director de este Trabajo Fin de Grado, el Dr. D. Diego García Álvarez, por su dedicación docente, sus conocimientos compartidos y su inestimable guía para la comprensión del campo de estudio. Su paciencia, observaciones, apoyo, implicación y constructivos comentarios han sido cruciales en el desarrollo de este trabajo.

Gracias a todos aquellos que me han enseñado a enfrentar los desafíos que se me plantean en la vida.

A mi familia le agradezco su apoyo en todo lo que me propongo, hacerme sentir protegido y valorado, despertar mi curiosidad por lo que me rodea y contribuir a que disfrute de cada instante de mi vida.

Por último, mi agradecimiento a cada uno de los docentes que he tenido a lo largo de mis estudios por sus aportaciones a este fructífero viaje académico.

Resumen

En la actualidad, las intrusiones en redes informáticas son cada día más frecuentes y variadas. Esto provoca que los equipos de defensa en ciberseguridad de los sistemas tengan que monitorizar los flujos de información que atraviesan sus redes. Una intrusión en un sistema puede suponer desde el espionaje empresarial hasta el secuestro de datos, por lo que es fundamental actuar rápidamente en cuanto se detecta una intrusión.

El principal objetivo de este Trabajo Fin de Grado (TFG) es el desarrollo de modelos neuronales capaces de detectar y clasificar intrusiones en un sistema informático. El modelo propuesto se divide en dos niveles jerárquicos, un modelo dedicado a la clasificación binaria de los flujos de datos (como benignos o malignos) y una vez que se ha detectado un flujo maligno, un segundo modelo de clasificación multiclase capaz de distinguir entre nueve tipos diferentes de ataques entre los que este modelo clasifica el flujo.

Puesto que el trabajo desarrollado se enmarca en un Grado en Ingeniería Informática, el proceso de desarrollo se ha guiado siguiendo CRISP-DM, una metodología específica para proyectos de minería de datos en la industria. Esta metodología abarca desde la comprensión del problema como en diseño de modelos útiles, hasta su evaluación en un entorno real.

Abstract

Nowadays, intrusions into computer networks are becoming more frequent and varied. This forces cybersecurity defense teams to monitor the information flows traversing their networks. An intrusion into a system can range from corporate espionage to data hijacking, making it essential to act quickly once an intrusion is detected.

The main objective of this Bachelor's Thesis (TFG) is the development of neural models capable of detecting and classifying intrusions in a computer system. The proposed model is divided into two hierarchical levels: a model dedicated to the binary classification of data flows (as benign or malicious) and, once a malicious flow is detected, a second multiclass classification model capable of distinguishing between nine different types of attacks into which the model categorizes the flow.

Since this work is part of a Degree in Computer Engineering, the development process has followed CRISP-DM, a specific methodology for data mining projects in the industry. This methodology covers everything from understanding the problem to designing useful models and evaluating them in a real environment.

Índice general

Agradecimientos	III
Resumen	V
Abstract	VII
Lista de figuras	XV
Lista de tablas	XIX
1. Introducción	1
1.1. Contexto	1
1.2. Motivación	2
1.3. Objetivos del proyecto	3
1.4. Objetivos académicos	3
1.5. Estructura de la memoria	3
2. Metodología	5
2.1. CRISP-DM	5
2.2. SCRUM	8
3. Planificación	11
3.1. Planificación temporal	11

3.2. Gestión de riesgos	14
3.3. Estimación de costes	26
3.3.1. Costes materiales	26
3.3.2. Costes humanos	29
4. Entendimiento del problema	33
4.1. ¿Qué es un ataque a un sistema informático?	33
4.2. Tipos de ataque a sistemas informáticos	34
4.3. ¿Qué es TCP?	35
4.3.1. ¿Qué es un segmento TCP?	36
4.4. Importancia de protegerse frente a un ataque	37
4.5. Importancia de detectar los ataques rápidamente	38
4.6. Soluciones comerciales o actuales a estos problemas	38
4.7. Requisitos	39
4.7.1. Requisitos Funcionales	40
4.7.2. Requisitos No Funcionales	40
5. Entendimiennto de los datos	41
5.1. Origen de los datos	41
5.2. Tipos de ataques registrados en los datos de NF-UNSW- NB15-v3	42
5.3. Parámetros de los datos	43
5.4. Selección de atributos	46
5.5. Preparación de los datos	47
6. Modelos	51
6.1. ¿Qué es un modelo neuronal?	51
6.1.1. Origen de los modelos neuronales	51
6.1.2. Funcionamiento básico de una red neuronal	53

6.1.3.	¿Qué tipos de modelos neuronales existen?	59
6.1.4.	Técnicas útiles para obtener modelos que converjan correctamente	60
6.1.5.	Conjunto de datos de entrenamiento	62
6.1.6.	Parámetros e hiperparámetros	63
6.1.7.	PyTorch	64
6.2.	Clasificación con redes neuronales	64
6.2.1.	Matriz de confusión para clasificación binaria	65
6.2.2.	Métricas útiles en clasificación binaria	65
6.2.3.	Matriz de confusión para clasificación multiclase	68
6.2.4.	Métricas útiles en clasificación multiclase	69
6.3.	Arquitectura de los modelos de clasificación binaria desarrollados	74
6.3.1.	Capa oculta con la mitad de neuronas que atributos de entrada	74
6.3.2.	Capa oculta con el mismo número de neuronas que atributos de entrada	76
6.3.3.	Capa oculta con el doble de neuronas que atributos de entrada	77
6.4.	Arquitecturas de los modelos de clasificación multiclase desarrollados	79
6.4.1.	Capa oculta con la mitad de neuronas que atributos de entrada	80
6.4.2.	Capa oculta con el mismo número de neuronas que atributos de entrada	81
6.4.3.	Capa oculta con el doble de neuronas que atributos de entrada	83
6.4.4.	Diseños del modelo de clasificación multiclase descartados	85
6.5.	Implementación del modelo neuronal de clasificación binaria	86
6.5.1.	Propósito del modelo de clasificación binaria	87
6.5.2.	Diseño del modelo e hiperparámetros seleccionados	87
6.6.	Implementación del modelo neuronal de clasificación multiclase	90
6.6.1.	Propósito del modelo de clasificación multiclase	90
6.6.2.	Diseño del modelo e hiperparámetros seleccionados	91
6.7.	Selección de modelos de clasificación binaria	94

6.7.1. Comparación de las arquitecturas seleccionadas de los modelos de clasificación binaria	101
6.8. Selección de modelos de clasificación multiclase	103
6.8.1. Comparación de las arquitecturas seleccionadas de los modelos de clasificación multiclase	109
7. Evaluación	113
7.1. Conjunto de datos de prueba	113
7.2. Proceso de evaluación	114
7.3. Resultados de las métricas en el proceso de evaluación	115
7.3.1. Resultados del modelo de clasificación binaria	115
7.3.2. Resultados del modelo de clasificación multiclase	121
7.4. Análisis de los resultados obtenidos	130
7.4.1. Modelo de clasificación binaria	130
7.4.2. Modelo de clasificación multiclase	131
7.5. Comparación de los resultados obtenidos con trabajos similares	133
7.5.1. Comparación de los modelos de clasificación binaria	133
7.5.2. Comparación de los resultados por clase de los modelos de clasificación multiclase	134
7.5.3. Evaluación comparativa del rendimiento	134
8. Despliegue	137
8.1. Integración en entornos operativos	137
8.2. Consideraciones técnicas	137
8.2.1. Patrones arquitectónicos recomendados: Pipeline y Observador	138
8.3. Monitorización y mantenimiento	138
8.4. Impacto y aplicación en el mundo real	139
9. Conclusiones	141
9.1. Conclusiones	141

9.2. Trabajo futuro	143
10.Apéndice	145
Bibliografía	147

Lista de Figuras

2.1. Esquema del ciclo CRISP-DM estándar [1]. 8

2.2. Esquema del ciclo de vida de scrum [2]. 9

3.1. Diagrama de Gantt para la planificación del proyecto. 13

4.1. Esquema del *three way handshake* de TCP [3]. 35

4.2. Esquema segmento TCP [4]. 36

4.3. Formato de la cabecera en IPv4 [5]. 37

5.1. Función de transformación para los parámetros IPv4. 48

6.1. Esquema del funcionamiento de una neurona artificial [6]. 52

6.2. Esquema de redes neuronales [7]. 53

6.3. Esquema de algoritmo de optimización usando los gradiente calculados por la retropropagación [8]. 57

6.4. Esquema de redes neuronal convolucional [9]. 60

6.5. Arquitectura del modelo con $n/2$ neuronas en la capa oculta siendo n el número de parámetros de entrada. 75

6.6. Arquitectura del modelo con n neuronas en la capa oculta siendo n el número de parámetros de entrada. 76

6.7. Arquitectura del modelo con $2n$ neuronas en la capa oculta siendo n el número de parámetros de entrada. 78

6.8. Arquitectura del modelo con $n/2$ neuronas en la capa oculta siendo n el número de parámetros de entrada. 80

6.9. Arquitectura del modelo con n neuronas en la capa oculta siendo n el número de parámetros de entrada.	82
6.10. Arquitectura del modelo con $2n$ neuronas en la capa oculta siendo n el número de parámetros de entrada.	84
6.11. Definición de la clase del modelo de clasificación binaria.	87
6.12. Definición de la clase <code>DatasetTFG</code>	89
6.13. Bucle de entrenamiento del modelo de clasificación binaria.	90
6.14. Estructura del modelo de clasificación multiclase diseñada.	91
6.15. Bucle de validación del modelo de clasificación multiclase.	93
6.16. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el MCB25 durante la fase de búsqueda.	97
6.17. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el MCB49 durante la fase de búsqueda.	99
6.18. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el MCB98 durante la fase de búsqueda.	101
6.19. Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria.	102
6.20. Matriz de confusión de la mejor configuración de hiperparámetros encontrada en la fase de búsqueda para el MCM25.	105
6.21. Matriz de confusión de la mejor configuración de hiperparámetros encontrada en la fase de búsqueda para el MCM49.	107
6.22. Matriz de confusión de la mejor configuración de hiperparámetros encontrada en la fase de búsqueda para el MCM98.	109
6.23. Mejores diez configuraciones de hiperparámetros del modelo de clasificación multiclase.	110
7.1. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 3º MCB25 durante la fase de evaluación.	117
7.2. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 2º MCB49 durante la fase de evaluación.	119
7.3. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 3º MCB98 durante la fase de evaluación.	121
7.4. Matriz de confusión de la mejor configuración de hiperparámetros obtenida con en el 3º MCM25 durante la fase de evaluación.	123

7.5. Matriz de confusión normalizada de la mejor configuración de hiperparámetros obtenida en el 3º MCM25 durante la fase de evaluación.	124
7.6. Matriz de confusión de la mejor configuración de hiperparámetros obtenida con en el 5º MCM49 durante la fase de evaluación.	126
7.7. Matriz de confusión normalizada de la mejor configuración de hiperparámetros obtenida en el 5º MCM49 durante la fase de evaluación.	127
7.8. Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 4º MCM98 durante la fase de evaluación.	129
7.9. Matriz de confusión normalizada de la mejor configuración de hiperparámetros obtenida en el MCM98 durante la fase de evaluación.	130
7.10. Mejores cinco modelos en la fase de evaluación del modelo de clasificación binaria.	131
7.11. Mejores cinco modelos en la fase de evaluación del modelo de clasificación multiclase.	133
8.1. Ejemplo de despliegue de los modelos en un sistema informático.	139

Lista de Tablas

3.1. Cronograma de hitos y horas de trabajo. 14

3.2. Matriz Probabilidad-Impacto. 16

3.3. R01: Conflicto con periodos de exámenes y entregas de otras asignaturas. . . 18

3.4. R02: Fallos hardware en equipos de desarrollo. 19

3.5. R03: Limitaciones de capacidad de procesamiento para el entrenamiento de los modelos. 20

3.6. R04: Pérdida o corrupción del código. 21

3.7. R05: Disponibilidad limitada del tutor académico. 22

3.8. R06: Desviaciones en la planificación temporal inicial. 23

3.9. R07: Cambios en los requisitos técnicos. 24

3.10. R08: Dependencia de tecnologías inestables o no documentadas. 25

3.11. R09: Problemas de licencia de software. 26

3.12. Costes de Software. 28

3.13. Costes de Profesionales - Fase 1. 29

3.14. Costes de Profesionales - Fase 2. 30

3.15. Costes de Profesionales - Fase 3. 30

3.16. Coste Total por Profesional. 30

5.1. Clasificación de amenazas de seguridad del dataset NF-UNSW-NB15-v3. 43

6.1. Matriz de confusión para clasificación binaria. 65

6.2.	Matriz de confusión para clasificación con 9 clases.	69
6.3.	Resultados de los diferentes diseños probados para el modelo de clasificación multiclase.	86
6.4.	Valores de los hiperparámetros utilizados en los experimentos del modelo de clasificación binaria.	88
6.5.	Valores de los hiperparámetros utilizados en los experimentos del modelo de clasificación multiclase.	93
6.6.	Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria con una capa oculta de 25 neuronas ($n/2$).	96
6.7.	Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria con una capa oculta de 49 neuronas (n).	98
6.8.	Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria con una capa oculta de 98 neuronas ($2n$).	100
6.9.	Mejores cinco configuraciones de hiperparámetros del modelo de clasificación multiclase con una capa oculta de 25 neuronas.	104
6.10.	Mejores cinco configuraciones de hiperparámetros del modelo de clasificación multiclase con una capa oculta de 49 neuronas.	106
6.11.	Mejores cinco configuraciones de hiperparámetros del modelo de clasificación multiclase con una capa oculta de 98 neuronas.	108
7.1.	Resultados de la evaluación del modelo de clasificación binaria con $n/2$ neuronas en la capa oculta siendo n el número de atributos de entrada.	116
7.2.	Resultados de la evaluación del modelo de clasificación binaria con n neuronas en la capa oculta siendo n el número de atributos de entrada.	118
7.3.	Resultados de la evaluación del modelo de clasificación binaria con $2n$ neuronas en la capa oculta siendo n el número de atributos de entrada.	120
7.4.	Resultados de la evaluación del modelo de clasificación multiclase con $n/2$ neuronas en la capa oculta siendo n el número de atributos de entrada.	122
7.5.	Resultados de la evaluación del modelo de clasificación multiclase con n neuronas en la capa oculta siendo n el número de atributos de entrada.	125
7.6.	Resultados de la evaluación del modelo de clasificación multiclase con $2n$ neuronas en la capa oculta siendo n el número de atributos de entrada.	128
7.7.	Comparación de modelos de clasificación binaria.	133
7.8.	Comparación de los valores obtenidos en $F1-Score$ por clase.	134

Capítulo 1

Introducción

Este documento corresponde con la memoria del Trabajo de Fin de Grado (TFG) del Grado en Ingeniería Informática de la Escuela de Ingeniería Informática (EII) de la Universidad de Valladolid (UVa). Este trabajo se centra en el diseño e implementación de un modelo neuronal capaz de detectar intrusiones en una red informática. El modelo está dividido en dos niveles de clasificación, un modelo de clasificación binaria que clasifica las conexiones como benignas o malignas, y un modelo de clasificación multiclase que recibe como entradas las conexiones clasificadas por el modelo binario como malignas. Este último modelo distingue en nueve tipos diferentes de intrusiones las conexiones que recibe, es decir, trata de determinar el tipo de intrusión. El uso de redes neuronales para la detección de intrusiones permite identificar patrones complejos y no lineales en el tráfico de red, mejorando la capacidad de detectar ataques desconocidos o sofisticados. Además, las redes neuronales pueden adaptarse y aprender continuamente a partir de nuevos datos, aumentando su precisión con el tiempo.

1.1. Contexto

En la actualidad, los sistemas informáticos reciben muchos más ataques de denegación de servicio y de intrusión que hace unos años, esto se debe en parte a los avances en los modelos de Inteligencia Artificial (IA). Los sistemas informáticos enfrentan actualmente graves amenazas debido al uso malintencionado de la IA por parte de ciberdelincuentes. Una de las principales problemáticas es la automatización de ataques, donde herramientas basadas en IA permiten ejecutar campañas de ataques informáticos con mayor precisión y escala. Estas IAs pueden generar mensajes convincentes, imitar patrones de comportamiento legítimos y evadir medidas de seguridad tradicionales, lo que incrementa la frecuencia y sofisticación de los ataques.

Otro desafío crítico es la explotación de vulnerabilidades mediante IA, que acelera la identificación de fallos en sistemas sin intervención humana. Existen algoritmos de aprendizaje automático (*Machine Learning*, ML) que analizan grandes volúmenes de datos para

descubrir brechas de seguridad en tiempo récord, facilitando ataques dirigidos incluso contra infraestructuras críticas como hospitales. Por estos motivos, la ciberseguridad es un tema de actualidad del que cada vez más se habla con más frecuencia incluso en la prensa general o económica [10].

La IA también complica la defensa, ya que los sistemas de detección tradicionales no siempre pueden anticipar tácticas adaptativas generadas por algoritmos hostiles. Esto obliga a las organizaciones y empresas a invertir en soluciones de IA defensiva, como sistemas de respuesta autónoma. Sin embargo, esto genera una carrera tecnológica desigual donde actores maliciosos aprovechan herramientas accesibles y de bajo costo. La falta de regulación global agrava este escenario, dificultando la mitigación de riesgos asociados.

Además, los modelos neuronales son adaptativos: mejoran su precisión con el tiempo al entrenarse con nuevos datos, lo que es crucial en entornos dinámicos donde los ciberataques evolucionan rápidamente. Por ejemplo, pueden distinguir entre comportamientos legítimos inusuales (como un empleado accediendo a recursos fuera de horario) y actividades maliciosas (como filtración de datos), reduciendo falsos positivos. En cambio, los enfoques tradicionales suelen ser rígidos y requieren ajustes manuales frecuentes para mantener su eficacia.

Sin embargo, el uso de modelos neuronales para la defensa de los sistemas conlleva grandes desafíos, como la necesidad de grandes conjuntos de datos etiquetados y recursos computacionales intensivos. Aun así, en escenarios donde la sofisticación de los ataques supera las capacidades de detección convencionales, los modelos neuronales representan un salto cualitativo en proactividad y escalabilidad.

1.2. Motivación

La principal motivación, que impulsó el desarrollo de este proyecto es facilitar la detección de ataques en redes informáticas, que tantas complicaciones está generando a los encargados de la administración de estos sistemas. Para cumplir con esta motivación, se decidió implementar un modelo neuronal que cumpliera con estos requisitos.

Durante la formación universitaria en el Grado en Ingeniería Informática, los alumnos de la mención de tecnologías de la información, aprenden a administrar grandes sistemas de computación en aspectos como: la seguridad, la garantía de la información, la evaluación de dichos sistemas y el almacenamiento de los datos. Además de contener formación sobre ciertos componentes de desarrollo de software. La falta de formación en algunos aspectos de la IA en dicha mención, desemboca en una de las grandes motivaciones académicas para desarrollar este proyecto, como es la familiarización con las técnicas de ML, en concreto de las redes neuronales, como herramientas útiles.

1.3. Objetivos del proyecto

En esta sección se listan los objetivos del proyecto, que constituyen las metas específicas, medibles, alcanzables, relevantes y con plazos definidos que se persiguen con la ejecución del mismo. Dichos objetivos describen los resultados concretos que se espera lograr al finalizar el proyecto y proporcionan un marco de referencia para la planificación, la ejecución, el seguimiento y la evaluación de su progreso.

1. Diseñar e implementar un modelo capaz de detectar intrusiones en redes informáticas y proporcionar una clasificación previa de la intrusión.
2. Los modelos de detección han de ser modelos neuronales.
3. Evaluar y comparar los modelos generados con un dataset real y complejo .

1.4. Objetivos académicos

En esta sección se enumeran los objetivos académicos del presente estudio, los cuales representan las metas específicas, susceptibles de evaluación, realizables, pertinentes para el ámbito del conocimiento, que se pretenden alcanzar a través del desarrollo de este proyecto.

1. Comprender el funcionamiento de los modelos neuronales através de `PyTorch` y las métricas de evaluación.
2. Aprender las características de varios de los tipo de modelos neruonales que existen.
3. Descubrir el potencial de las redes neuronales para optimizar y mejorar las tecnologías de la información, incluyendo la ciberseguridad de los sistemas.

1.5. Estrucutra de la memoria

Este trabajo fin de grado se estructura de la siguiente manera:

Capítulo 2 Metodología: En este capítulo se definen cuales son las fases de la metodología CRISP-DM que se utiliza como metodología y modelo de proceso para el diseño y evaluación de los modelos neuronales para la detección de intrusiones. Se describe la aplicación de cada una de las seis fases al contexto específico del desarrollo de modelos neuronales.

Capítulo 3 Planificación: Este capítulo presenta la planificación del proyecto. Se definen los recursos necesarios, se identifican las tareas principales, se estiman los plazos y se establece el cronograma. También se aborda la gestión de riesgos inicial y la asignación de roles.

Capítulo 4 Entendimiento del problema: En este capítulo se describe el problema que el proyecto busca abordar. Se presenta el contexto, la relevancia y los objetivos generales.

Capítulo 5 Entendimiento de los datos: Este capítulo se dedica a la exploración y comprensión del conjunto de datos utilizado para el entrenamiento y evaluación de los modelos desarrollados. Se describe la fuente, el formato, el tamaño y las variables de los datos. Se presenta un análisis exploratorio para identificar patrones, problemas de calidad y la distribución de las variables.

Capítulo 6 Modelos: En este capítulo se detallan los modelos de redes neuronales desarrollados y entrenados. Se describe la arquitectura, la justificación de su elección, los hiperparámetros, la función de pérdida y el optimizador. Se incluye la estrategia de entrenamiento y las métricas de evaluación.

Capítulo 7 Evaluación: Este capítulo se centra en la evaluación final de los modelos entrenados. Se describe el conjunto de datos de prueba, el proceso de evaluación, la presentación de los resultados de las métricas y el análisis de las fortalezas y debilidades de los modelos.

Capítulo 8 Despliegue: Este capítulo aborda la fase de despliegue de los modelos entrenados. Se describe la integración en un entorno operativo, las consideraciones técnicas, los posibles desafíos y las estrategias de monitorización y mantenimiento. Debido a la diversidad de entornos existentes en los que se puede realizar el despliegue, se comentarán los pasos generales que habría que seguir para desplegar los modelos en un entorno de producción real, sin entrar en profundidad.

Capítulo 9 Conclusiones: En este capítulo final se presentan las conclusiones del proyecto. Se resumen los principales hallazgos, se evalúa el cumplimiento de los objetivos académicos, se discuten las implicaciones de los resultados, las limitaciones y las posibles líneas de trabajo futuro.

Capítulo 2

Metodología

En este capítulo se explica la metodología CRISP-DM (*Cross-Industry Standard Process for Data Mining*), que se utiliza en el desarrollo del resto del proyecto para alcanzar los objetivos propuestos.

La adopción de metodologías estructuradas es fundamental en el desarrollo de proyectos informáticos, puesto que proporcionan un marco sistemático para garantizar la calidad, eficiencia y trazabilidad del proyecto. En particular, metodologías como CRISP-DM, permiten: alinear objetivos técnicos con necesidades de negocio, reducir riesgos mediante fases iterativas y documentadas, y facilitar la colaboración entre equipos multidisciplinares.

Según algunos estudios, los proyectos que utilizan metodologías estandarizadas incrementan un 35 % su probabilidad de éxito, frente a aproximaciones *ad-hoc*, al minimizar desviaciones en costes y plazos [11]. En el ámbito de la ciberseguridad, donde los requisitos legales y técnicos son críticos, este enfoque metodológico resulta indispensable para asegurar soluciones robustas y auditables.

2.1. CRISP-DM

La metodología CRISP-DM, fue diseñada para guiar proyectos de minería de datos y aprendizaje automático. Su estructura cíclica y flexible, como se puede apreciar en la Figura 2.1 que muestra su ciclo de vida, la hace aplicable en diversos dominios, desde marketing hasta ciberseguridad. Está compuesta por las siguientes fases:

1. Comprensión del negocio: La primera fase de CRISP-DM establece los cimientos estratégicos del proyecto mediante un proceso de alineación entre los objetivos técnicos y las necesidades organizacionales. Para lograr establecer los cimientos, se lleva a cabo un análisis exhaustivo del contexto empresarial para identificar los problemas clave que el proyecto debe abordar, así como las oportunidades de mejora que podrían aprovecharse. Se realiza un proceso de recopilación y documentación de requisitos que involucra a todas las partes interesadas

relevantes. El resultado de esta fase es una definición precisa del alcance del proyecto, que incluye no solo los objetivos cuantificables sino también los criterios de éxito que permitirán evaluar el impacto real de la solución propuesta. Además, se establecen las limitaciones operativas y estratégicas que condicionarán el desarrollo del proyecto, asegurando que todas las fases posteriores se ejecuten dentro de un marco bien definido y alineado con las prioridades organizacionales. Esta fase se detalla en los Capítulos [3 Planificación](#) y [4 Entendimiento del problema](#).

2. Comprensión de los datos: Esta fase se centra en el análisis detallado de los datos disponibles para el proyecto, con el objetivo de evaluar su idoneidad y calidad para abordar los problemas identificados en la fase anterior. Este proceso implica un examen minucioso de las diversas fuentes de información, su estructura y sus características fundamentales. Durante esta etapa, se identifican y documentan aspectos críticos como la complejidad de los datos, la presencia de posibles sesgos y la representatividad de la información en relación con los objetivos del proyecto. La comprensión profunda de los datos permite anticipar desafíos potenciales y establecer estrategias adecuadas para su tratamiento en fases posteriores. Además, esta fase proporciona perspectivas que pueden influir en decisiones técnicas importantes, como la selección de algoritmos o el diseño de características. El resultado es un conocimiento del potencial y las limitaciones de los datos disponibles, que sirve como base para las transformaciones que se realizan en la siguiente fase. En este trabajo, esta fase se documenta en el Capítulo [5 Entendimiento de los datos](#).

3. Preparación de los Datos: Se trata de una fase crítica donde los datos brutos se transforman en un conjunto adecuado para modelado. Esta etapa implica una serie de operaciones fundamentales que garantizan la calidad y consistencia de los datos que alimentan a los modelos analíticos. Las actividades realizadas en esta fase son cruciales para el éxito del proyecto, ya que determinan en gran medida la capacidad de los algoritmos para extraer patrones significativos y generar resultados confiables. Se aplican técnicas especializadas para abordar problemas comunes en los datos, asegurando que la información sea representativa, completa y se encuentre adecuadamente estructurada para los análisis posteriores. Cualquier deficiencia en la preparación de los datos puede comprometer significativamente la efectividad de las siguientes fases. Al finalizar este proceso, se obtiene un conjunto de datos optimizado que conserva la esencia de la información original mientras elimina ruido y distorsiones que podrían afectar negativamente a los resultados del modelado. Al igual que la fase anterior, esta fase se aborda en el Capítulo [5 Entendimiento de los datos](#).

4. Modelado: Constituye el núcleo técnico del proceso CRISP-DM, donde se desarrollan y evalúan los algoritmos diseñados para extraer conocimiento de los datos preparados. Esta etapa comienza con la selección cuidadosa de las técnicas de modelado más apropiadas para los objetivos específicos del proyecto y las características de los datos disponibles. Durante el proceso de modelado, se exploran diferentes enfoques algorítmicos, ajustando meticulosamente sus parámetros para optimizar su rendimiento. En esta fase se incluyen procesos de validación diseñados para garantizar que los modelos desarrollados sean robustos y generalizables, capaces de mantener su efectividad cuando se enfrenten a datos nuevos y no vistos previamente. El modelado es un proceso iterativo que puede requerir volver a fases anteriores para refinar la preparación de datos o incluso reconsiderar algunos aspectos del planteamiento inicial del problema. El resultado de esta fase es uno o varios modelos validados que cumplen con los criterios de calidad establecidos y están listos para su evaluación en el contexto de

los objetivos empresariales definidos inicialmente. Esta fase se documenta en el [Capítulo 6 Modelos](#) de este trabajo.

5. Evaluación: Esta fase representa un examen exhaustivo de los modelos desarrollados, contrastando su desempeño técnico con los objetivos empresariales establecidos en la primera fase del proyecto. Este proceso va más allá de las métricas estadísticas tradicionales para incorporar una valoración del impacto potencial de la solución propuesta. Durante la evaluación, se analiza minuciosamente la capacidad de los modelos para resolver el problema de negocio original, considerando tanto su precisión técnica como su aplicabilidad práctica en el contexto organizacional. Se identifican y documentan las limitaciones de los modelos, así como los posibles riesgos asociados a su implementación. Esta fase también incluye la validación de los resultados con las partes interesadas clave, asegurando que la solución cumpla con las expectativas y requisitos operativos. La evaluación termina con una decisión fundamentada sobre la idoneidad de los modelos para su implementación, junto con recomendaciones para su posible mejora o adaptación a escenarios futuros. También se valida su robustez en escenarios realistas. Esta fase se detalla en el [Capítulo 7 Evaluación](#) de este trabajo.

6. Despliegue: Se trata de la fase final de CRISP-DM, esta se centra en la transición del modelo analítico desde un entorno de desarrollo hasta un entorno de producción real donde pueda generar valor tangible para la organización. Este proceso implica una serie de actividades cuidadosamente planificadas que garantizan la integración efectiva de la solución en los procesos empresariales existentes. El despliegue incluye aspectos técnicos como la implementación de la infraestructura necesaria, el desarrollo de interfaces adecuadas y la creación de mecanismos de monitoreo continuo. También se ha de tener en cuenta la capacitación de los usuarios finales y la documentación exhaustiva de la solución, asegurando su adopción efectiva y su uso óptimo. La fase de despliegue también establece procesos para el mantenimiento y actualización periódica del modelo, puesto que las soluciones analíticas requieren evolución continua para mantener su relevancia y efectividad. Como en el resto de metodologías, se implementan mecanismos para medir el impacto real de la solución una vez en producción, cerrando el ciclo al proporcionar retroalimentación valiosa que puede ser la base de futuros proyectos analíticos. Esta fase no se aborda en el trabajo, pero se le dedica el [Capítulo 8 Despliegue](#), en él que se comentan los pasos generales que habría que seguir para desplegar los modelos en un entorno de producción real, sin entrar en profundidad.

Como se ha explicado, CRISP-DM es una metodología iterativa, esto significa que los resultados de fases posteriores pueden revelar la necesidad de ajustes en etapas anteriores (como recolectar más datos o redefinir objetivos). Su enfoque estructurado minimiza riesgos y maximiza el valor entregado, siendo especialmente útil en proyectos complejos donde la alineación entre técnica y negocio es esencial.

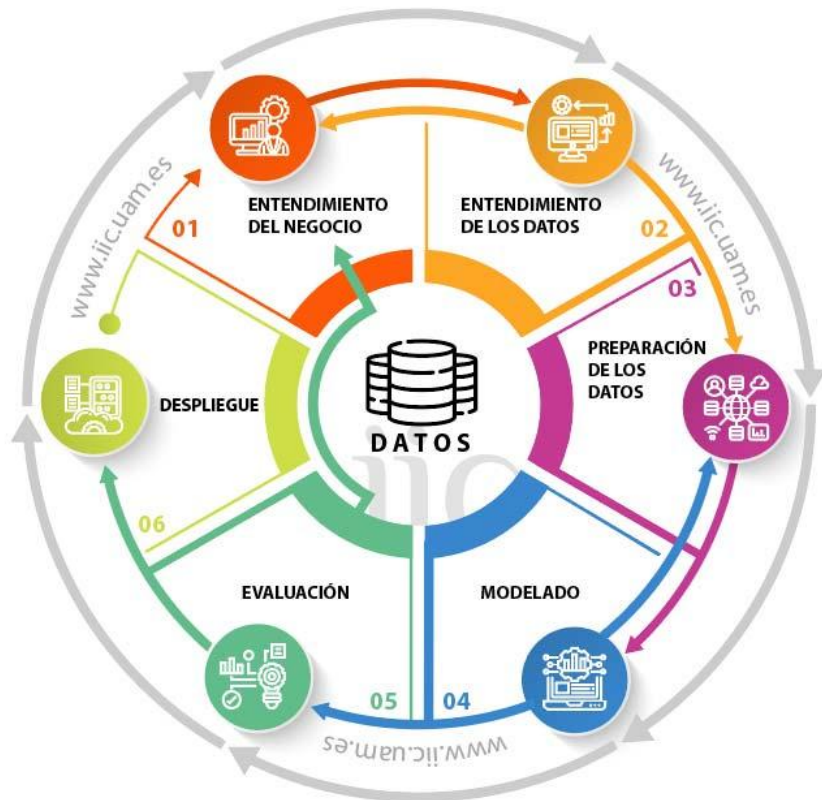


Figura 2.1: Esquema del ciclo CRISP-DM estándar [1].

2.2. SCRUM

Un marco de trabajo ágil es una estructura metodológica utilizada para gestionar proyectos de manera flexible, iterativa y centrada en la entrega continua de valor. Estos marcos se basan en los principios del Manifiesto Ágil, el cual promueve la colaboración entre equipos multidisciplinarios, la respuesta rápida al cambio y la entrega frecuente de productos funcionales al cliente [12].

Los marcos ágiles permiten a los equipos adaptarse a condiciones cambiantes y enfocarse en mejorar constantemente, promoviendo una comunicación fluida, ciclos de retroalimentación cortos y una participación activa del cliente durante todo el desarrollo del producto.

Dentro de estos marcos, uno de los más populares y ampliamente adoptados es Scrum.

Scrum es un marco de trabajo ágil diseñado para ayudar a equipos a desarrollar productos complejos de forma incremental. Se caracteriza por su estructura liviana y fácil de entender, aunque su dominio completo requiere disciplina y experiencia. Scrum se basa en ciclos de

trabajo cortos llamados *sprints*, que típicamente duran entre una y cuatro semanas, durante los cuales se entrega un incremento funcional del producto. La Figura 2.2 muestra el ciclo de vida del marco de trabajo ágil scrum.

Este marco de trabajo ágil, establece roles definidos (*Product Owner*, *Scrum Master* y Equipo de Desarrollo), eventos específicos (como el *Sprint*, la Planificación del *Sprint*, la Revisión, la Retrospectiva y el *Scrum* Diario) y artefactos clave (*Product Backlog*, *Sprint Backlog* e Incremento). Estos elementos promueven la transparencia, la inspección continua y la adaptación dentro del proceso de desarrollo.

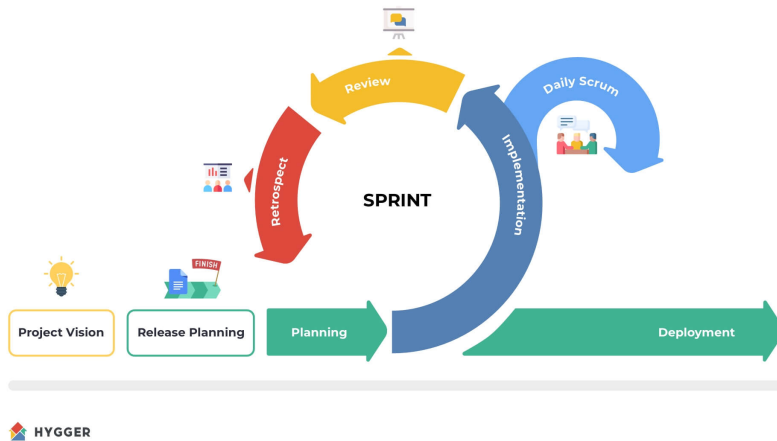


Figura 2.2: Esquema del ciclo de vida de scrum [2].

Capítulo 3

Planificación

Este capítulo aborda la organización detallada del trabajo de fin de grado, cubriendo desde su diseño inicial hasta la implementación y el seguimiento durante su desarrollo. Una planificación rigurosa resulta fundamental para sentar las bases del proyecto, ya que permite definir con claridad los objetivos, los recursos necesarios, los plazos de entrega y las actividades clave para alcanzar los resultados esperados [13].

En primer lugar, se establece una planificación temporal preliminar, donde se estiman los tiempos requeridos para cada etapa. Este cronograma se estructura en torno a las fases de la metodología CRISP-DM, complementadas con etapas específicas propias de un Trabajo de Fin de Grado. A continuación, se realiza un análisis de riesgos exhaustivo, evaluando tanto la probabilidad como el impacto de cada posible contingencia.

Además, se elabora un presupuesto detallado para las tareas del proyecto, abordado desde dos perspectivas. Por un lado, se incluye una estimación realista de los costes asociados a la ejecución del trabajo en el ámbito académico. Por otro lado, se plantea una proyección teórica de los gastos que implicaría un proyecto equivalente en un contexto profesional.

Por último, se contrasta la planificación inicial con el desarrollo real del trabajo, lo que permite evaluar posibles desviaciones y los aprendizajes obtenidos durante el proceso.

3.1. Planificación temporal

La planificación temporal constituye un elemento fundamental en la ejecución de un proyecto fin de grado, ya que permite estructurar de manera sistemática todas las actividades necesarias para alcanzar los objetivos propuestos. En el contexto de un trabajo académico que combine el desarrollo de software con una metodología de investigación, como es el caso de CRISP-DM para el modelo de proceso y SCRUM para la gestión del proyecto, una adecuada planificación garantiza la distribución equilibrada del tiempo disponible entre

las distintas fases del trabajo. Esta organización temporal resulta especialmente relevante cuando se deben coordinar aspectos teóricos, desarrollo técnico y validación de resultados, asegurando que cada componente reciba la atención necesaria sin comprometer la calidad global del proyecto.

El empleo de un diagrama de Gantt como herramienta de planificación ofrece ventajas significativas para visualizar la secuencia de actividades y su superposición temporal. Este tipo de representación gráfica facilita la identificación de hitos críticos y dependencias entre tareas, aspectos particularmente importantes cuando se combinan metodologías diferentes como CRISP-DM y SCRUM. La primera, con sus fases bien definidas, proporciona la estructura para el desarrollo del núcleo analítico del proyecto, mientras que SCRUM, con sus sprints iterativos, permite adaptar el trabajo a los descubrimientos que vayan surgiendo durante la investigación. La integración de ambas aproximaciones en un único cronograma exige una cuidadosa coordinación que el diagrama de Gantt ayuda a materializar de forma clara y comprensible. El diagrama de Gantt utilizado para la planificación de este trabajo se muestra en la Figura 3.1. Este diagrama tiene una duración de 300 horas, que corresponden con las horas de trabajo de alumno de los 12 créditos ECTS que exige el plan de estudios del Grado de Ingeniería Informática de la UVa para el TFG.



Figura 3.1: Diagrama de Gantt para la planificación del proyecto.

3.2. Gestión de riesgos

En este apartado se presentan los principales riesgos potenciales del proyecto junto con sus correspondientes planes de mitigación. Además, en la Figura 3.1 se muestran las horas dedicadas al cumplimiento de cada hito del trabajo. De acuerdo con el PMBOK (*Project Management Body of Knowledge*) [13], los riesgos en gestión de proyectos se clasifican en las categorías de la Sección 3.2.

Hitos del Proyecto

Hito	Horas hasta el hito	Horas acumuladas
Finalización de formación y trabajos previos	20	20
Finalización de la planificación inicial	30	50
Finalización de la comprensión de datos	40	90
Finalización del modelado	80	170
Obtención de modelos óptimos	40	210
Finalización y entrega de la memoria	90	300

Tabla 3.1: Cronograma de hitos y horas de trabajo.

Tipología de Riesgos

1. Riesgos Técnicos:

- Limitaciones en la infraestructura de hardware.
- Incompatibilidad entre sistemas o tecnologías.
- Problemas de rendimiento o escalabilidad.
- Deficiencias en el diseño o implementación de software.

2. Riesgos de Gestión:

- Deficiencias en la comunicación entre los miembros del equipo.
- Modificaciones en los requisitos del proyecto.
- Inestabilidad del equipo por conflictos internos o rotación de personal.
- Retrasos en la disponibilidad de recursos críticos.

3. Riesgos de Mercado:

- Aparición de competencia no anticipada.
- Variaciones en las condiciones del mercado que afectan la demanda.
- Cambios regulatorios que impactan la ejecución del proyecto.

4. Riesgos Financieros:

- Limitaciones en la disponibilidad de fondos.
- Excesos presupuestarios no previstos.
- Fluctuaciones en los tipos de cambio.

5. Riesgos Externos:

- Fenómenos meteorológicos adversos.
- Interrupciones en la cadena de suministro.
- Eventos naturales catastróficos.

Metodología de Evaluación

La identificación y valoración de riesgos se realiza mediante criterios cualitativos, al no disponer de métricas cuantitativas suficientemente fiables para un análisis más exhaustivo. Este enfoque permite priorizar los riesgos según su impacto potencial y probabilidad de ocurrencia [14].

		Impacto				
		Mínimo	Bajo	Medio	Alto	Extremo
Probabilidad	Extrema					
	Alta					
	Media					
	Baja					
	Muy Baja					

Nivel de Riesgo	Color
Extremo	
Alto	
Moderado	
Bajo	
Mínimo	

Tabla 3.2: Matriz Probabilidad-Impacto.

Probabilidad

Grado de posibilidad de que un riesgo se materialice. Se suele cuantificar en escala del 1 (muy improbable) al 5 (casi seguro). En la matriz que se muestra en la Tabla 3.2, determina el eje vertical y se combina con el impacto para priorizar riesgos.

Impacto

Consecuencia o efecto potencial que tendría la materialización del riesgo. Se valora del 1 (impacto mínimo) al 5 (impacto catastrófico). Representa en la matriz que se muestra en la Tabla 3.2, el eje horizontal en la matriz y mide la severidad del riesgo.

Plan de Mitigación

Acciones proactivas para reducir la probabilidad o impacto del riesgo antes de que ocurra. Incluye:

- Prevención: Eliminar las causas del riesgo.
- Reducción: Disminuir su probabilidad o impacto.
- Transferencia: Trasladar el riesgo a terceros.

Plan de Contingencia

Medidas reactivas que se implementan cuando el riesgo se materializa. Contiene:

- Activación: Criterios para ejecutar el plan.
- Respuesta: Acciones específicas de contención.
- Recuperación: Cómo volver a la normalidad.

Nivel de Riesgo

Resultado de multiplicar la probabilidad por el impacto. Clasifica riesgos en:

- Alto (15-25): Requieren acción inmediata.
- Medio (5-14): Necesitan monitoreo.
- Bajo (1-4): Aceptables con supervisión mínima.

Umbral de Riesgo

Límite máximo aceptable de riesgo para el proyecto. Determina cuándo se deben implementar planes de mitigación o contingencia.

Propietario del Riesgo

Persona o equipo responsable de monitorear cada riesgo y ejecutar los planes correspondientes.

Riesgos identificados

1. **R01:** Conflicto con periodos de exámenes y entregas de otras asignaturas (Tabla 3.3).
2. **R02:** Fallos hardware en equipos de desarrollo (Tabla 3.4).
3. **R03:** Limitaciones de capacidad de procesamiento para el entrenamiento de los modelos (Tabla 3.5).
4. **R04:** Pérdida o corrupción del código (Tabla 3.6).
5. **R05:** Disponibilidad limitada del tutor académico (Tabla 3.7).
6. **R06:** Desviaciones en la planificación temporal inicial (Tabla 3.8).
7. **R07:** Cambios en los requisitos técnicos (Tabla 3.9).
8. **R08:** Interpretación errónea de los resultados por falta de formación en técnicas de aprendizaje automático. (Tabla 3.10).
9. **R09:** Problemas de licencias de software (Tabla 3.11).

Riesgo R01	
Título	Conflicto con periodos de exámenes y entregas de otras asignaturas.
Descripción	Sobrecarga académica que dificulta la dedicación y el rendimiento en todas las tareas.
Probabilidad	4 (Alta)
Impacto	3 (Media)
Matriz P/I	Alta/Media (12)
Plan Mitigación	■ Coordinar calendario académico anticipadamente. ■ Avanzar trabajo en periodos de menor carga de trabajo.
Plan Contingencia	■ Dedicar horas extra en los asuntos académicos. ■ Reorganizar prioridades temporales.

Tabla 3.3: R01: Conflicto con periodos de exámenes y entregas de otras asignaturas.

Riesgo R02	
Título	Fallos hardware en equipos de desarrollo.
Descripción	Problemas causados por el mal funcionamiento de los componentes físicos de un ordenador, como la placa base, la tarjeta gráfica, la memoria RAM, el disco duro o la fuente de alimentación en equipos de desarrollo.
Probabilidad	3 (Media)
Impacto	4 (Alto)
Matriz P/I	Media/Alto (12)
Plan Mitigación	■ Mantenimiento preventivo mensual. ■ Uso de equipos redundantes.
Plan Contingencia	■ Utilizar equipos alternativos. ■ Uso de máquinas virtuales del sistema de virtualización de la Escuela de Ingeniería de Informática de Valladolid.

Tabla 3.4: R02: Fallos hardware en equipos de desarrollo.

Riesgo R03	
Título	Limitaciones de capacidad de procesamiento para el entrenamiento de los modelos.
Descripción	Restricciones de hardware (cálculo, memoria) que impactan la velocidad, viabilidad y calidad del entrenamiento, influyendo en el tamaño y complejidad de los modelos.
Probabilidad	4 (Alta)
Impacto	5 (Extremo)
Matriz P/I	Alto/Extremo (20)
Plan Mitigación	■ Optimización temprana del código. ■ Uso de técnicas de muestreo.
Plan Contingencia	■ Utilizar sistema de virtualización de la escuela. ■ Reducir complejidad de modelos.

Tabla 3.5: R03: Limitaciones de capacidad de procesamiento para el entrenamiento de los modelos.

Riesgo R04	
Título	Pérdida o corrupción del código.
Descripción	Extraviación o daño en los conjuntos de datos del proyecto que se utilizan para en entrenamiento y la validación del modelo.
Probabilidad	2 (Baja)
Impacto	5 (Extremo)
Matriz P/I	Baja/Extremo (10)
Plan Mitigación	■ Almacenamiento redundante de los datos en repositorios. ■ Verificación de la integridad de los datos con checksums.
Plan Contingencia	■ Recuperar datasets desde backups externos. ■ Regenerar datos sintéticos.

Tabla 3.6: R04: Pérdida o corrupción del código.

3.2. GESTIÓN DE RIESGOS

Riesgo R05	
Título	Disponibilidad limitada del tutor académico.
Descripción	Restricciones de tiempo y acceso al tutor académico, afectando la frecuencia y profundidad de la retroalimentación y el apoyo al estudiante en su proceso de aprendizaje.
Probabilidad	3 (Media)
Impacto	3 (Media)
Matriz P/I	Media/Media (9)
Plan Mitigación	■ Agendar reuniones con anticipación. ■ Preparar preguntas concretas.
Plan Contingencia	■ Consultar con profesores alternativos. ■ Usar foros académicos.

Tabla 3.7: R05: Disponibilidad limitada del tutor académico.

Riesgo R06	
Título	Desviaciones en la planificación temporal inicial.
Descripción	Variaciones o retrasos respecto al cronograma original, impactando los plazos de entrega, la gestión del tiempo y la consecución de los objetivos previstos.
Probabilidad	4 (Alta)
Impacto	4 (Alto)
Matriz P/I	Alto/Alto (16)
Plan Mitigación	■ Incluir días asignados a descanso como días dedicados al proyecto. ■ Revisiones semanales de progreso.
Plan Contingencia	■ Reorganizar del diagrama de Gantt. ■ Eliminar funcionalidades no críticas.

Tabla 3.8: R06: Desviaciones en la planificación temporal inicial.

Riesgo R07	
Título	Cambios en los requisitos técnicos.
Descripción	Modificaciones o alteraciones en las especificaciones necesarias para un proyecto o tarea, que pueden afectar al diseño, a la implementación, a los recursos y a los plazos.
Probabilidad	4 (Alta)
Impacto	4 (Alto)
Matriz P/I	Alto/Alto (16)
Plan Mitigación	■ Documentar requisitos iniciales con precisión. ■ Establecer procedimiento de cambio formal.
Plan Contingencia	■ Revisar el alcance con tutor. ■ Asignar tiempo adicional para cambios.

Tabla 3.9: R07: Cambios en los requisitos técnicos.

Riesgo R08	
Título	Interpretación errónea de los resultados por falta de formación en técnicas de aprendizaje automático.
Descripción	El desconocimiento de las métricas utilizadas para analizar los resultados como precisión, recall, F1, puede llevar a confiar en modelos con bajo desempeño real o tomar decisiones equivocadas basadas en correlaciones espurias.
Probabilidad	3 (Media)
Impacto	5 (Extremo)
Matriz P/I	Media/Extremo (15)
Plan Mitigación	■ Buscar alternativas para obtener una formación básica en técnicas de aprendizaje automático. ■ Corroborar los resultados obtenidos con el tutor académico.
Plan Contingencia	■ Suspender de manera inmediata las decisiones tomadas basadas en las salidas mal interpretadas del modelo. ■ Reemplazar el modelo mal interpretado por una versión anterior o un modelo base.

Tabla 3.10: R08: Dependencia de tecnologías inestables o no documentadas.

Riesgo R09	
Título	Problemas de licencia de software.
Descripción	Inconvenientes o restricciones legales relacionadas con el uso, la distribución o la activación de software, pudiendo causar interrupciones, costos adicionales o incluso acciones legales.
Probabilidad	2 (Baja)
Impacto	3 (Media)
Matriz P/I	Baja/Media (6)
Plan Mitigación	■ Verificar licencias antes de usarlas para su implementación. ■ Priorizar la utilización software open-source.
Plan Contingencia	■ Buscar alternativas equivalentes. ■ Solicitar licencias académicas.

Tabla 3.11: R09: Problemas de licencia de software.

3.3. Estimación de costes

En esta sección se presenta la estimación de costes, que comprende la identificación y valoración de los recursos necesarios para el desarrollo del proyecto. Este proceso implica la cuantificación de los gastos previsible asociados a materiales y software específico utilizado. También se considera un coste el tiempo dedicado por el estudiante a la realización del trabajo.

La precisión en la estimación de costes facilita la elaboración de un presupuesto realista y la planificación financiera del proyecto. Permite anticipar las necesidades económicas, buscar posibles fuentes de financiación si fuese necesario y gestionar eficientemente los recursos disponibles. Una estimación detallada contribuye a evitar desviaciones presupuestarias y a asegurar la viabilidad económica del proyecto [15].

3.3.1. Costes materiales

A continuación, se hace un recuento de los costes materiales en hardware y software que han sido utilizados para el desarrollo de este proyecto.

Hardware

El hardware comprende el conjunto de componentes físicos y tangibles que constituyen un sistema informático. Proporciona la infraestructura física necesaria para la ejecución del software y el procesamiento de la información.

Para realizar este proyecto se ha utilizado un equipo informático con los siguientes componentes:

- **CPU:** AMD Ryzen 7 6800HS with Radeon Graphics (16) @ 4.785GHz.
- **RAM:** 16GB SO-DIMM DDR5 4800MH.
- **Memoria:** 512GB PCIe® 4.0 NVMe™ M.2 SSD.
- **GPU1:** NVIDIA GeForce RTX 3050 Mobile.
- **GPU2:** AMD ATI Radeon 680M.

Debido al tamaño del conjunto de datos, el componente que más ha ralentizado el proyecto es la RAM, que en ciertas ocasiones durante el entrenamiento de los modelos se quedaba algo escasa en capacidad.

Teniendo en cuenta que el coste del ordenador en el momento de la compra fue de 829 €, que la vida útil aproximada es de 8 años y que para el desarrollo de este proyecto se ha estado utilizando durante 3.5 meses, la amortización del hardware es:

$$\text{Amortización del Hardware} = 829 \text{ €} \cdot \frac{1}{8} \cdot \frac{3,5}{12} = 30,22 \text{ €} \quad (3.1)$$

Además, para la ejecución de los experimentos realizados en el Capítulo 6, se ha utilizado una máquina virtual propiedad del grupo de investigación *Grupo de Sistemas Inteligentes* al que pertenece el tutor de este trabajo D. Diego García Álvarez. La máquina física sobre la que está virtualizada la máquina utilizada, cuenta con las siguientes especificaciones:

- **CPU:** Dos Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz (24 cores y 48 hilos).
- **RAM:** 8 Módulos de 16GB DDR4 a 3200 MT/s (128GB).
- **Memoria:** 14TB de discos SSD en raid 6 con 6 discos.

La máquina virtualizada utilizada posee:

- **CPU:** 8 cores.
- **RAM:** 16GiB.
- **Memoria:** 60GiB.

3.3. ESTIMACIÓN DE COSTES

Por desgracia, debido al desconocimiento de la vida útil de la máquina física sobre la que se ejecuta la virtualización utilizada, no es posible calcular la amortización de esta parte del hardware utilizado en el desarrollo de este trabajo.

Software

El software constituye el conjunto intangible de programas, datos e instrucciones que habilitan el funcionamiento de un sistema informático. Es el responsable de definir la funcionalidad, el comportamiento y la interacción del sistema con el usuario y con otros sistemas. En la Tabla 3.12 se detalla el coste del software utilizado para desarrollar este trabajo.

Funcionalidad	Software	Coste Mensual	Duración	Coste Total
Sistema Operativo (SO)	Kubuntu 24.10 x86_64	0 €	3,5 meses	0 €
Lenguaje (memoria)	Latex	0 €	3 meses	0 €
Editor latex	TexMaker	0 €	3 meses	0 €
IDE	MS Visual Studio Code	0 €	1 mes	0 €
Lenguaje (modelos)	Python	0 €	1 mes	0 €
IDE de Python	Jupyter Notebooks	0 €	1 mes	0 €
Plataforma MLOps ¹	Weights&Biases	0 €	1 mes	0 €
Control de versiones	GitHub	0 €	1 mes	0 €
IA generativa (código)	DeepSeek	0 €	1 mes	0 €
IA generativa (memoria)	Gemini	0 €	2 mes	0 €
Comunicación 1	MS Outlook	0 € ²	3,5 mes	0 €
Comunicación 2	MS Teams	0 €	3,5 mes	0 €

Tabla 3.12: Costes de Software.

Debido a que el proyecto se realiza en un ámbito académico, se han minimizado los costes software del proyecto utilizando exclusivamente herramientas cedidas por la entidad académica o bien herramientas con licencia open-source que no suponen un coste monetario para el desarrollo del proyecto.

¹*Machine Learning Operations*, son un conjunto de prácticas y principios que buscan gestionar de forma eficiente y escalable el ciclo de vida del aprendizaje automático

²licencia académica

3.3.2. Costes humanos

Como proyecto académico, los costes humanos representan el valor del tiempo y el esfuerzo personal invertido en la planificación, investigación, redacción y presentación del trabajo. Estos costes se manifiestan en las horas dedicadas al proyecto, el esfuerzo intelectual requerido, el aplazamiento o anulación de otras actividades personales o profesionales y el estrés asociado al proceso.

En el caso de la simulación de los costes monetarios de un proyecto similar a este, sería necesario contar con personas cualificadas para los siguientes puestos:

- Ingeniero de Aprendizaje Automático.
- Científico de Datos.
- Analista de Datos.

Los costes humanos se dividen en tres fases diferentes:

1. Definición y Preparación: la Tabla 3.13 muestra los costes para esta fase.
2. Desarrollo y Entrenamiento: los costes para esta fase vienen desarrollados en la Tabla 3.14.
3. Definición y Preparación: la Tabla 3.15 detalla los costes de esta fase del proyecto.

Para obtener una visión de conjunto de los costes humanos del proyecto, la Tabla 3.16 refleja la suma de las fases comentadas.

Fase 1: Definición y Preparación (75 horas)

Profesional	€/hora	Horas	Total (€)
Ingeniero ML	42	12	504
Científico Datos	34,5	38	1 311
Analista Datos	18,5	25	462,5
Total Fase 1		75	2 277,5

Tabla 3.13: Costes de Profesionales - Fase 1.

Fase 2: Desarrollo y Entrenamiento (150 horas)

Profesional	€/hora	Horas	Total (€)
Ingeniero ML	42	95	3 990
Científico Datos	34,5	38	1 311
Analista Datos	18,5	17	314,5
Total Fase 2		150	5 615,5

Tabla 3.14: Costes de Profesionales - Fase 2.

Fase 3: Validación y Evaluación (75 horas)

Profesional	€/hora	Horas	Total (€)
Ingeniero ML	42	30	1 260
Científico Datos	34,5	37	1 276,5
Analista Datos	18,5	8	148
Total Fase 3		75	2 684,5

Tabla 3.15: Costes de Profesionales - Fase 3.

Coste Total del Proyecto (300 horas)

Profesional	Coste Total (€)
Ingeniero ML	5 754
Científico Datos	3 898,5
Analista Datos	925
Coste Total del Proyecto	10 577,5

Tabla 3.16: Coste Total por Profesional.

Las estimaciones salariales proporcionadas se basan en los salarios en el sector tecnológico y de análisis de datos en España. Esta información se encuentra publicada en portales de empleo (Talent.com³), escuelas de negocio (Aicad Business School⁴, KSchool⁵, Esden Business School⁶) y noticias del sector (Tokio School⁷).

³<https://es.talent.com/>

⁴<https://www.aicad.es/>

⁵<https://kschool.com/>

⁶<https://esden.es/>

⁷<https://www.tokioschool.com/>

Capítulo 4

Entendimiento del problema

En este capítulo se trata el entendimiento del problema o dominio. Tal y como se comenta en el Capítulo 2, es la fase inicial de la metodología CRISP-DM. A continuación, se alinean los objetivos técnicos con las necesidades del negocio y con el problema a resolver. Se definen requisitos, se identifican métricas de éxito y se trata de dar comprensión sobre el contexto organizacional. La totalidad de la fase de entendimiento del problema de la metodología usada, se distribuye entre este capítulo, el Capítulo 1, donde se fijan los objetivos del proyecto y el Capítulo 3, donde se establece la planificación y los riesgos.

4.1. ¿Qué es un ataque a un sistema informático?

Un ataque a un sistema informático constituye una acción deliberada y no autorizada que explota vulnerabilidades con el objetivo de comprometer la confidencialidad, integridad o disponibilidad de los datos y recursos del sistema. Esta actividad maliciosa puede manifestarse a través de diversas técnicas, incluyendo la inyección de código malicioso, la denegación de servicio, el acceso no autorizado y la ingeniería social. Su ejecución busca obtener beneficios ilícitos, interrumpir operaciones o dañar la infraestructura tecnológica [16].

La consecuencia de un ataque puede variar desde la pérdida o alteración de información sensible hasta la paralización completa de los servicios ofrecidos por el sistema. La identificación, análisis y mitigación de estas amenazas representan un aspecto fundamental en la seguridad informática, requiriendo la implementación de medidas preventivas y reactivas para proteger los activos digitales de una organización o individuo [17].

4.2. Tipos de ataque a sistemas informáticos

Los ataques a sistemas informáticos se pueden clasificar de distintas maneras según su naturaleza. A continuación, se detallan los más comunes y peligrosos:

- **Ataques de Denegación de Servicio (DoS).** Un ataque de Denegación de Servicio tiene como objetivo hacer que un sistema o red no esté disponible para sus usuarios legítimos, interrumpiendo su funcionamiento normal. Estos ataques se llevan a cabo sobrecargando los recursos de un servidor o red con una cantidad excesiva de solicitudes. Un ejemplo de este tipo de ataque es el Ataque de Denegación de Servicio Distribuido (DDoS), en el cual múltiples dispositivos comprometidos atacan al mismo tiempo un servidor, sobrecargándolo hasta que se vuelve inoperativo [18].
- **Ataques de Inyección.** Los ataques de inyección ocurren cuando un atacante introduce código malicioso en un sistema que no valida correctamente las entradas de los usuarios. Un ejemplo común es la inyección SQL, donde un atacante puede insertar comandos SQL en una aplicación web para acceder, modificar o eliminar datos en una base de datos. Este tipo de ataque explota la falta de medidas de validación y filtrado en las entradas del sistema [19].
- **Ataques de *Phishing*.** El *phishing* es un tipo de ataque en el que un atacante engaña a una persona para que proporcione información confidencial, como contraseñas o detalles bancarios. Generalmente, esto se realiza mediante correos electrónicos o sitios web fraudulentos que simulan ser una entidad de confianza, como bancos o servicios en línea. La principal característica del *phishing* es la manipulación psicológica de la víctima [20].
- **Ataques de *Malware*.** El *malware* es un software malicioso diseñado para dañar o robar información de un sistema informático. Entre los tipos más comunes de *malware* se encuentran los virus, gusanos, troyanos y *ransomware*. El *ransomware*, por ejemplo, cifra los archivos de un sistema y exige un pago para liberar el acceso a los mismos. Los ataques de *malware* pueden ser ejecutados a través de archivos adjuntos de correo electrónico, descargas de software malicioso o vulnerabilidades de seguridad.
Los ataques de *Malware* que se tratan en este trabajo son: *Backdoor* y *Worms*.
- **Ataques *Man-in-the-Middle* (MitM).** En un ataque *Man-in-the-Middle* (de intermediario o de hombre en el medio), el atacante intercepta y, a veces, modifica la comunicación entre dos partes sin que ellas lo sepan. Estos ataques son comunes en redes no seguras, como las redes Wi-Fi públicas. El atacante puede obtener información confidencial, como contraseñas y datos de tarjetas de crédito, si no se utilizan medidas de seguridad adecuadas, como cifrado de las comunicaciones [18].
- ***Exploits* de Vulnerabilidades.** Un *exploit* es un ataque que se aprovecha de una vulnerabilidad en el software o hardware de un sistema. Estos ataques se basan en el aprovechamiento de fallos conocidos en programas o dispositivos sin que el fabricante o el usuario los haya corregido. Los *exploits* pueden permitir a los atacantes ejecutar código malicioso en un sistema, obtener acceso a información confidencial o incluso tomar control total de la máquina afectada [20].

Aquellos ataques que pertenecen al tipo *Exploits* de vulnerabilidades que el modelo de clasificación multiclase deberá distinguir son: *Fuzzers*, *Exploits* y *Shellcode*.

4.3. ¿Qué es TCP?

El Protocolo de Control de Transmisión (*Transmission Control Protocol*, TCP) constituye uno de los protocolos fundamentales de la capa de transporte del modelo TCP/IP, sobre el cual se sustenta gran parte de la comunicación en redes IP, incluyendo Internet. Su diseño se orienta a proporcionar un servicio de transferencia de datos fiable, ordenado y con detección de errores entre aplicaciones que se ejecutan en sistemas finales diferentes. Para lograr esta fiabilidad, TCP establece una conexión virtual punto a punto entre las aplicaciones comunicantes mediante un proceso de *three way handshake* (triple apretón de manos) que se muestra en la Figura 4.1, lo que permite la negociación de parámetros de la conexión y la sincronización de los números de secuencia iniciales.

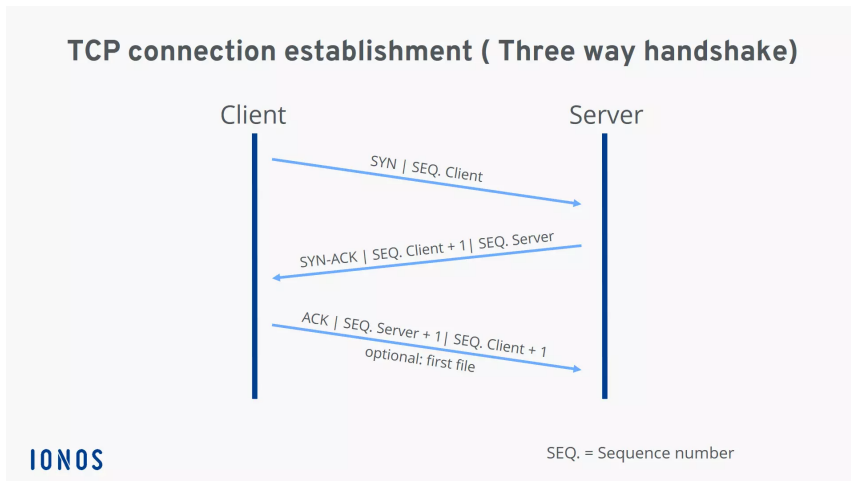


Figura 4.1: Esquema del *three way handshake* de TCP [3].

El protocolo garantiza la entrega ordenada de la información al receptor mediante la asignación de números de secuencia a cada byte transmitido, permitiendo así la reordenación en caso de que la información no llegue al receptor en el orden correcto. La fiabilidad se logra a través de un mecanismo de acuse de recibo (*acknowledgment*, ACK) positivo con retransmisión, donde el receptor confirma la recepción correcta de los paquetes de información, y el emisor retransmite aquellos partes de la información para los que no recibe confirmación dentro de un tiempo límite (*timeout*).

4.3.1. ¿Qué es un segmento TCP?

Una vez establecida la conexión, TCP divide los datos de la aplicación en unidades más pequeñas denominadas segmentos cuya estructura se detalla en la Figura 4.2. Un segmento o paquete TCP constituye la unidad de datos fundamental que se intercambia a través de una red utilizando el mencionado protocolo TCP. Este segmento encapsula una porción de los datos de la capa de aplicación, precedida por una cabecera TCP.

La cabecera TCP contiene información de control esencial para la funcionalidad del protocolo, incluyendo los números de puerto de origen y destino que identifican las aplicaciones comunicantes, los números de secuencia y de acuse de recibo ACK que garantizan la entrega ordenada y fiable, las banderas de control que indican el propósito del segmento (establecimiento de conexión, finalización, ACK, entre otros muchos), y otros campos como la ventana de recepción para el control de flujo y la suma de verificación para la detección de errores.

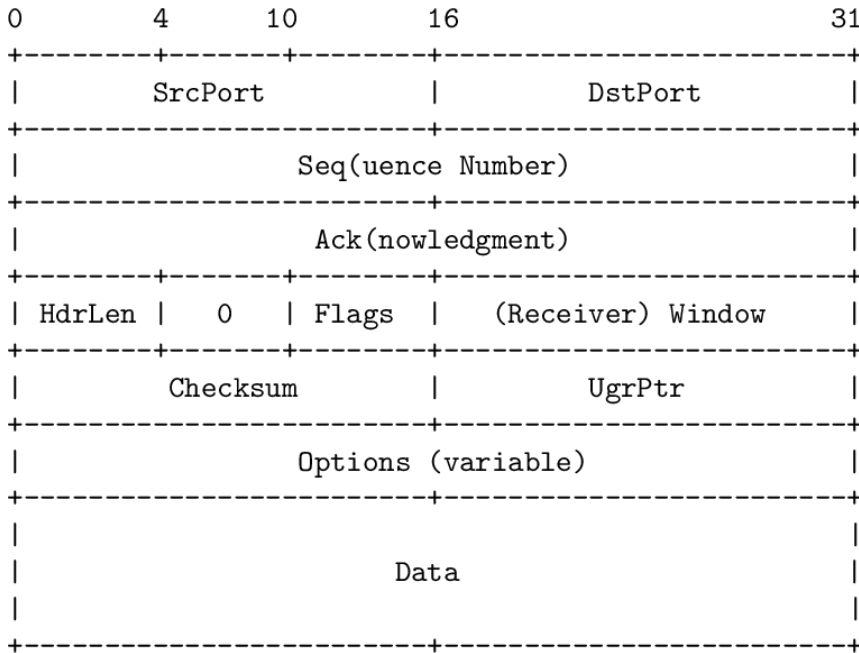


Figura 4.2: Esquema segmento TCP [4].

En el proceso de transmisión, el segmento TCP se encapsula a su vez dentro de un paquete IP (Protocolo de Internet) para su enrutamiento a través de la red. El paquete IP añade su propia cabecera con las direcciones IP de origen y destino, entre otra información necesaria para el transporte a nivel de red. Toda esta estructura se puede observar con detalle en la Figura 4.3.

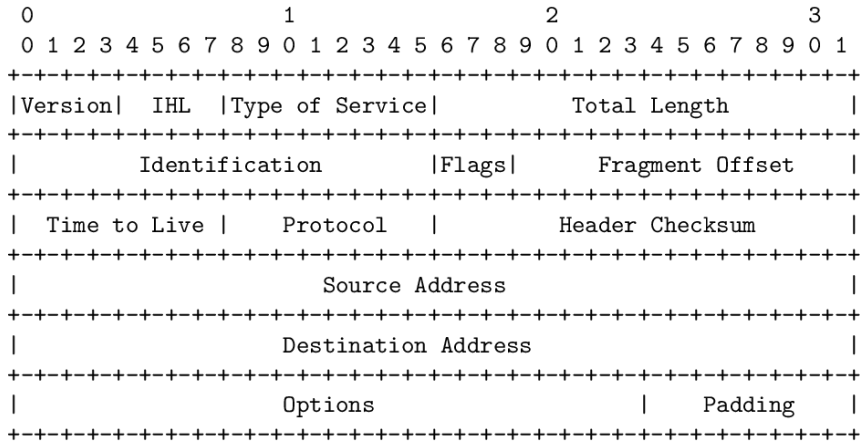


Figura 4.3: Formato de la cabecera en IPv4 [5].

4.4. Importancia de protegerse frente a un ataque

La importancia de protegerse frente a ataques informáticos radica en la salvaguarda de activos digitales críticos, la garantía de la continuidad operativa y la preservación de la confianza y la reputación. En un entorno digital cada vez más interconectado, los ataques informáticos representan una amenaza significativa para individuos, organizaciones y la sociedad en su conjunto, pudiendo acarrear consecuencias devastadoras [21].

Para las organizaciones, las implicaciones de un ataque informático pueden ser aún más costosas. Estas implicaciones incluyen pérdidas financieras directas debido al robo de fondos, la interrupción de las operaciones comerciales, los costes de recuperación y las posibles sanciones regulatorias. Además, se puede producir un daño significativo a la reputación y la pérdida de la confianza de los clientes, lo que a largo plazo afecta la viabilidad del negocio. Los ataques también pueden resultar en el robo de propiedad intelectual, secretos comerciales e información estratégica, otorgando ventajas competitivas a adversarios [22].

Por otra parte, la interrupción de servicios críticos, como energía, comunicaciones o salud, puede tener consecuencias graves para la sociedad en su conjunto.

La protección frente a ataques informáticos no es solo una cuestión de seguridad tecnológica, sino una necesidad imperante en la actualidad para proteger activos valiosos, asegurar la continuidad de las actividades, mantener la confianza de los usuarios y garantizar la estabilidad y el bienestar en el mundo digital actual. La implementación de prácticas de seguridad robustas y la concienciación sobre las amenazas cibernéticas son fundamentales en a la hora de defenderse de estos ataques [23].

4.5. Importancia de detectar los ataques rápidamente

La detección temprana de ataques informáticos constituye un pilar fundamental en la ciberseguridad moderna debido a su capacidad para mitigar consecuencias críticas. Cuando un sistema logra identificar intrusiones o actividades maliciosas en sus fases iniciales, se reducen significativamente los daños operativos y económicos. Esta rapidez de respuesta permite contener amenazas antes de que comprometan infraestructuras completas, preservando tanto la integridad de los datos como la continuidad del negocio [24].

Desde una perspectiva técnica, la identificación inmediata limita la superficie de ataque, impidiendo que los actores maliciosos escalen privilegios o se propaguen lateralmente por la red. En el ámbito regulatorio, cumple con los estrictos plazos que exigen normativas como el Reglamento General de Protección de Datos (RGPD) [25], que obliga todos los países miembros de la Unión Europea a notificar violaciones de seguridad en un máximo de 72 horas. Esta normativa fue promovida por la Unión Europea con el objetivo de fortalecer y unificar la protección de los datos personales en todo el territorio de la UE. El RGPD entró en vigor el 25 de mayo de 2018, después de un período de transición de dos años para que las organizaciones pudieran adaptarse a las nuevas normativas. Además, desde el punto de vista económico, reduce los costes asociados a las reparaciones, que suelen multiplicarse exponencialmente cuando los ataques permanecen indetectados durante largos períodos.

La capacidad de detectar rápidamente anomalías en el tráfico de red, accesos no autorizados o patrones de comportamiento sospechosos no solo protege los activos digitales, sino que también salvaguarda la reputación institucional. Organizaciones con sistemas de detección temprana robustos demuestran proactividad ante clientes y socios comerciales, generando confianza en su capacidad para manejar información sensible. Esta anticipación resulta especialmente crítica en entornos donde la disponibilidad del servicio es primordial, como en las infraestructuras críticas anteriormente mencionadas [26].

4.6. Soluciones comerciales o actuales a estos problemas

En esta sección se comentan algunas de las soluciones y software que se utilizan en la actualidad para detectar y neutralizar posibles ataques informáticos. Estas herramientas protegen los sistemas informáticos analizando y controlando el tráfico de la red.

Los *firewalls* de próxima generación (NGFW) como Palo Alto Networks, Check Point o Cisco Firepower, inspeccionan el tráfico de red a un nivel profundo (Deep Packet Inspection - DPI), analizando el contenido de los paquetes más allá de los puertos y protocolos tradicionales. Este enfoque se sustenta en diversas tecnologías avanzadas, incluyendo técnicas de inteligencia artificial (IA) y aprendizaje automático (ML), que permiten una detección más precisa y dinámica de amenazas. Palo Alto Networks, por ejemplo, emplea IA a través de su sistema *WildFire*, que utiliza análisis de comportamiento y *sandboxing* para identificar *malware* y amenazas avanzadas. Por su parte, Check Point implementa el uso de *ThreatCloud*, una plataforma que aplica algoritmos de IA para la detección y prevención de amenazas, y que se beneficia de la inteligencia colectiva de miles de dispositivos conectados. Cisco Firepo-

wer se apoya en tecnologías de *next-generation IPS* y *URL filtering*, además de usar análisis de tráfico basado en IA para identificar patrones anómalos de tráfico y potenciales riesgos de seguridad. Estas herramientas permiten identificar y bloquear amenazas sofisticadas, *malware*, y tráfico de aplicaciones maliciosas, además de ofrecer funcionalidades como prevención de intrusiones (IPS) y control de aplicaciones [27].

Los sistemas de detección y prevención de intrusiones o IDS e IPS, como: *Snort*, *Suricata* o *Trend Micro TippingPoint*, monitorean el tráfico de red en tiempo real en busca de patrones sospechosos o firmas de ataques conocidos. Los IDS alertan sobre posibles intrusiones, mientras que los IPS tienen la capacidad de bloquear o mitigar activamente el tráfico malicioso detectado, interrumpiendo los ataques en curso. Estas herramientas emplean diversas tecnologías para el análisis del tráfico y la detección de amenazas. *Snort*, por ejemplo, se basa en un sistema de detección de firmas y anomalías, utilizando algoritmos de análisis de tráfico que, aunque tradicionalmente no utilizan IA, han sido extendidos con módulos que incorporan técnicas de aprendizaje automático para mejorar su capacidad de identificar ataques desconocidos. *Suricata*, por su parte, también emplea un sistema de detección basado en firmas y análisis de flujo, pero se distingue por su capacidad para procesar grandes volúmenes de tráfico en paralelo, y su integración con herramientas de análisis de inteligencia de amenazas que utilizan IA para mejorar la precisión en la identificación de ataques avanzados. *Trend Micro TippingPoint* utiliza un enfoque de prevención de intrusiones de próxima generación que integra tecnologías de inspección profunda de paquetes (*DPI*) y análisis de tráfico en tiempo real, además de implementar IA y aprendizaje automático para detectar patrones de comportamiento anómalo y bloquear ataques sofisticados [28].

La microsegmentación de red con herramientas como VMware NSX, Cisco ACI o Illumio, divide la red en segmentos más pequeños y aislados, aplicando políticas de seguridad granular a cada segmento. Esto limita el movimiento lateral de los atacantes dentro de la red una vez que han comprometido un punto inicial. Al controlar el tráfico entre estos segmentos, se reduce la superficie de ataque y se contiene la propagación de las amenazas [29].

4.7. Requisitos

En la fase de entendimiento del problema de la metodología CRISP-DM es fundamental establecer claramente los requisitos que se deben cumplir durante el desarrollo del proyecto. Para definir los requisitos se ha utilizado el estándar ISO/IEC/IEEE 29148:2018 [30] que trata sobre ingeniería de requisitos en sistemas y software. Este estándar especifica los procesos necesarios para desarrollar requisitos a lo largo del ciclo de vida de un proyecto o sistema.

Como se ha comentado en la Sección 1.3 [Objetivos del proyecto](#), el principal objetivo del proyecto es desarrollar un modelo neuronal que detecte la presencia de ataques en una red informática y los clasifique según su tipo. Para cumplir con dicho objetivo, se considera imprescindible cumplir con los requisitos que se listan a continuación.

4.7.1. Requisitos Funcionales

Primera versión de requisitos, no me convencen mucho

- **RF-1:** El sistema deberá detectar cuales de las conexiones podrían ser potenciales intrusiones en la red.
- **RF-2:** El sistema deberá clasificará las conexiones en 10 categorías predefinidas en [5.1 Clasificación de amenazas de seguridad del dataset NF-UNSW-NB15-v3](#).
- **RF-3:** El sistem deberá ser capaz de procesar formatos estándar de logs como son Syslog, NetFlow y PCAP.
- **RF-4:** El sistema deberá diferenciar entre ataques conocidos (basados en firmas) y desconocidos (basados en anomalías).

4.7.2. Requisitos No Funcionales

- **RNF-1:** Latencia < 50 ms en redes de 10Gbps (requisito crítico para SOC [\[31\]](#)).
- **RNF-2:** Interfaz accesible para usuarios no técnicos (evaluado con test SUS [\[32\]](#)).
- **RNF-3:** Se debe utilizar el *dataset* NF-UNSW-NB15-v3 para entrenar los modelos.

Capítulo 5

Entendimiennto de los datos

Este capítulo se corresponde con la segunda etapa de la metodología CRISP-DM, En el se explicará la naturaleza de los datos y sus características, así como los valores atípicos que presentan y sus sesgos.

5.1. Origen de los datos

Para elegir un conjunto de datos que se adapte a las necesidades de los modelos de clasificación que se desarrollan en este trabajo, se realizó una búsqueda y comparación entre diferentes *datasets* que contienen datos de conexiones benignas y malignas a sistemas informáticos. Algunos de los datasets descartados son:

- KDD 1999: es un conjunto de datos creado para la detección de intrusiones en redes, derivado de DARPA 1998. Este *dataset* contiene tráfico simulado con etiquetas de ataques y tráfico normal. Su uso fue descartado para este trabajo por estar desactualizado, tener datos redundantes y no reflejar las amenazas modernas ni el tráfico real de las redes actuales ¹.
- CIC-IDS-2017: es un conjunto de datos generado por la [Universidad de Brunswick](#). El *dataset* se divide en 8 archivos CSV y presenta una escasa variedad de tipos de ataques, lo que no es representativo del tráfico real actual. Además, en ciberseguridad, los tipos de ataques y su funcionamiento varían cada muy poco tiempo, lo que implica que un conjunto de datos con 8 años ya se considera desactualizado ².
- MIT-datasets: el [Instituto de Tecnología de Massachusetts](#) (MIT) cuenta con varios *dataset* con información sobre conexiones malignas en redes. Algunos de ellos se basan en DARPA. La mayoría de los conjuntos de datos son antiguos y no detallan como se

¹Enlace al *dataset* KDD 1999

²Enlace al *dataset* CIC-IDS-2017

obtuvo la información que los compone, por estos motivos, estos *datasets* también fueron descartados ³.

Tras comparar varios conjuntos de datos, los datos que se utilizan para desarrollar este trabajo, se han obtenido de conjuntos de datos diseñados para entrenar Sistemas de Detección de Intrusión de Red (NIDS) basados en el aprendizaje automático. El conjunto de datos en cuestión forma parte de un análisis realizado en la [Universidad de Queensland](#), Australia [33].

El *dataset* utilizado es NF-UNSW-NB15-v3, se trata de una versión basada en NetFlow del conjunto de datos UNSW-NB15, mejorado con características adicionales de NetFlow y etiquetada de acuerdo con sus respectivas categorías de ataque. Su última actualización es del año 2025 y cuenta con una gran variedad de tipos de ataques registrados entre sus datos.

La Universidad de Queensland cuenta con otros *datasets* con características similares, pero de entre los conjuntos de datos desarrollados por esta entidad el más completo y actualizado es el que finalmente se ha elegido para desarrollar los modelos de este proyecto.

5.2. Tipos de ataques registrados en los datos de NF-UNSW-NB15-v3

En esta sección se explican los tipos de datos presentes en el *dataset* que se utiliza para entrenar a los modelos del proyecto. Se explica en que consiste cada tipo de ataque registrado así como el número exacto de ataques de cada tipo presente.

El conjunto de datos consiste en un total de 2 365 424 flujos de datos, donde 127 639 (5,4 %) son muestras de ataque y 2 237 731 (94,6 %) son benignos. Los flujos de ataque se clasifican en nueve clases, cada una representando una amenaza a la red distinta. La Tabla 5.1 proporciona una distribución detallada del conjunto de datos:

³[Enlace al dataset MIT-datasets](#)

Clase	Cantidad	Descripción
Benigno	2 237 731	Flujos normales no maliciosos.
<i>Fuzzers</i>	33 816	Tipo de ataque en el que el atacante envía grandes cantidades de datos aleatorios que hacen que un sistema se bloquee y también apuntan a descubrir vulnerabilidades de seguridad en un sistema.
<i>Analysis</i>	2 381	Un grupo que presenta una variedad de amenazas que se dirigen a aplicaciones web a través de puertos, correos electrónicos y <i>scripts</i> .
<i>Backdoor</i>	1 226	Una técnica que tiene como objetivo eludir los mecanismos de seguridad respondiendo a aplicaciones específicas de clientes construidos.
DoS	5 980	La denegación de servicio es un intento de sobrecargar los recursos de un sistema informático con el objetivo de evitar el acceso o la disponibilidad de sus datos.
<i>Exploits</i>	42 748	Son secuencias de comandos que controlan el comportamiento de un <i>host</i> a través de una vulnerabilidad conocida.
<i>Generic</i>	19 651	Un método que se dirige a la criptografía y causa una colisión con cada cifrado de bloques.
<i>Reconnaissance</i>	17 074	Una técnica para recopilar información sobre un <i>host</i> de red, también se conoce como sonda.
<i>Shellcode</i>	4 659	Un <i>malware</i> que penetra en un código para controlar el <i>host</i> de una víctima.
<i>Worms</i>	158	Ataques que se replican y se extienden a otros sistemas.

Tabla 5.1: Clasificación de amenazas de seguridad del dataset NF-UNSW-NB15-v3.

5.3. Parámetros de los datos

En esta sección se explicarán el significado de cada uno de los 55 atributos que componen cada fila del *dataset* seleccionado. El *dataset*, que es de dominio público se puede descargar en formato .csv donde los atributos se encuentran separados por comas [33].

1. IPV4_SRC_ADDR: Dirección IPv4 de origen.

2. `IPV4_DST_ADDR`: Dirección IPv4 de destino.
3. `L4_SRC_PORT`: Número de puerto de origen de la capa 4.
4. `L4_DST_PORT`: Número de puerto de destino de la capa 4.
5. `PROTOCOL`: Byte identificador del protocolo IP.
6. `L7_PROTO`: Protocolo que opera en la capa 7 del Modelo OSI, también conocida como capa de aplicación.
7. `IN_BYTES`: Número de bytes entrantes.
8. `OUT_BYTES`: Número de bytes salientes.
9. `IN_PKTS`: Número de paquetes entrantes.
10. `OUT_PKTS`: Número de paquetes salientes.
11. `FLOW_DURATION_MILLISECONDS`: Duración del flujo en ms.
12. `TCP_FLAGS`: *Bits* dentro del encabezado TCP que indican el estado o la dirección de una conexión TCP.
13. `CLIENT_TCP_FLAGS`: *Bits* dentro del encabezado TCP que indican el estado de la conexión TCP del cliente.
14. `SERVER_TCP_FLAGS`: *Bits* dentro del encabezado TCP que indican el estado de la conexión TCP del servidor.
15. `DURATION_IN`: Duración del flujo Cliente a Servidor (ms).
16. `DURATION_OUT`: Duración del flujo Cliente a Servidor (ms).
17. `MIN_TTL`: Mínimo valor del TTL (*Time To Live*), que limita la duración de un paquete de datos en una red informática, durante la sesión TCP.
18. `MAX_TTL`: Máximo valor del TTL (*Time To Live*), que limita la duración de un paquete de datos en una red informática, durante la sesión TCP.
19. `LONGEST_FLOW_PKT`: Paquete de mayor tamaño enviado durante la conexión de los sistemas (bytes).
20. `SHORTEST_FLOW_PKT`: Paquete de menor tamaño enviado durante la conexión de los sistemas (bytes).
21. `MIN_IP_PKT_LEN`: Longitud del paquete IP más pequeño del flujo observado.
22. `MAX_IP_PKT_LEN`: Longitud del paquete IP más grande del flujo observado.
23. `SRC_TO_DST_SECOND_BYTES`: *Bytes/segundo* de origen a destino.
24. `DST_TO_SRC_SECOND_BYTES`: *Bytes/segundo* de destino a origen.
25. `RETRANSMITTED_IN_BYTES`: Número de bytes TCP retransmitidos del flujo (src a dst).

26. RETRANSMITTED_IN_PKTS: Número de paquetes TCP retransmitidos del flujo (src a dst).
27. RETRANSMITTED_OUT_BYTES: Número de bytes TCP retransmitidos del flujo (dst a src).
28. RETRANSMITTED_OUT_PKTS: Número de paquetes TCP retransmitidos del flujo (dst a src).
29. SRC_TO_DST_AVG_THROUGHPUT: Tasa de transferencia promedio de origen a destino (bps).
30. DST_TO_SRC_AVG_THROUGHPUT: Tasa de transferencia promedio de destino a origen (bps).
31. NUM_PKTS_UP_TO_128_BYTES: Paquetes cuyo tamaño IP es ≤ 128 .
32. NUM_PKTS_128_TO_256_BYTES: Paquetes cuyo tamaño IP es > 128 y ≤ 256 .
33. NUM_PKTS_256_TO_512_BYTES: Paquetes cuyo tamaño IP es > 256 y ≤ 512 .
34. NUM_PKTS_512_TO_1024_BYTES: Paquetes cuyo tamaño IP es > 512 y ≤ 1024 .
35. NUM_PKTS_1024_TO_1514_BYTES: Paquetes cuyo tamaño IP es > 1024 y ≤ 1514 .
36. TCP_WIN_MAX_IN: Ventana TCP máxima (src a dst).
37. TCP_WIN_MAX_OUT: Ventana TCP máxima (dst a src).
38. ICMP_TYPE: Tipo ICMP por 256 + Código ICMP.
39. ICMP_IPV4_TYPE: Tipo ICMP.
40. DNS_QUERY_ID: ID de transacción de la consulta DNS.
41. DNS_QUERY_TYPE: Tipo de consulta DNS (ej., 1=A, 2=NS..).
42. DNS_TTL_ANSWER: TTL del primer registro A (si existe).
43. FTP_COMMAND_RET_CODE: Código de retorno del comando del cliente FTP.
44. FLOW_START_MILLISECONDS: Marca de tiempo de inicio del flujo en ms.
45. FLOW_END_MILLISECONDS: Marca de tiempo de fin del flujo en ms.
46. SRC_TO_DST_IAT_MIN: IAT mínimo (src a dst).
47. SRC_TO_DST_IAT_MAX: IAT máximo (src a dst).
48. SRC_TO_DST_IAT_AVG: IAT promedio (src a dst).
49. SRC_TO_DST_IAT_STDDEV: Desviación estándar del IAT (src a dst).
50. DST_TO_SRC_IAT_MIN: IAT mínimo (dst a src).
51. DST_TO_SRC_IAT_MAX: IAT máximo (dst a src).
52. DST_TO_SRC_IAT_AVG: IAT promedio (dst a src).
53. DST_TO_SRC_IAT_STDDEV: Desviación estándar del IAT (dst a src).

Este conjunto de datos ya se encontraba previamente etiquetado, lo que permite realizar un trabajo como el propuesto en el tiempo establecido. Típicamente, los conjuntos de datos enfocados a la clasificación suelen tener una única etiqueta, pero este conjunto de datos tiene dos, lo que agiliza el tratamiento del conjunto de los datos para el entrenamiento y la evaluación de los modelos que se proponen en este proyecto.

1. **Label:** Toma valor 0 si el flujo es benigno y 1 si se trata de un ataque.
2. **Attack:** Clase de flujo, perteneciente a las clases mencionadas en [5.1 Clasificación de amenazas de seguridad del dataset NF-UNSW-NB15-v3](#)

5.4. Selección de atributos

En esta sección se comentan los atributos preliminares, así como los valores y los sesgos presentes en el *dataset*. La identificación de estas irregularidades es fundamental para el correcto desarrollo del proyecto.

Al analizar los datos originales del *dataset*, se encuentran características que afectan de forma negativa al entrenamiento del modelo y por lo tanto, a su correcto funcionamiento. A continuación, se mencionan cuales son las características problemáticas de los datos que se utilizan.

Tras estudiar los datos, se descubre que algunos parámetros presentan valores infinitos, estos valores alteran la distribución inherente de las variables, introduciendo valores que no se pueden tratar en el proceso de aprendizaje. Los algoritmos de entrenamiento, diseñados para operar con valores numéricos finitos, pueden comportarse de manera impredecible o inestable ante la presencia de infinitos, dificultando la convergencia hacia una solución óptima. Asimismo, la interpretación de las características con valores infinitos se vuelve ambigua, comprometiendo la capacidad del modelo para establecer relaciones significativas con otras variables y para generalizar correctamente a datos futuros que no contengan tales valores extremos. En consecuencia, la presencia de infinitos puede degradar sustancialmente el rendimiento y la fiabilidad del modelo entrenado.

Como se comenta en la Sección anterior [5.3 Parámetros de los datos](#), existen dos parámetros que registran la IP origen y la IP destino de la conexión. En principio, si los datos no fuesen sintéticos, la dirección IP de la máquina que origina la comunicación sería aleatoria. En este conjunto de datos, se puede observar como todos los ataques provienen de IP con máscara 175.45.176.255, lo que no encaja con la realidad. Independientemente de este patrón, que no se manifiesta de forma natural en las conexiones entre sistemas, el uso de las IPs de las máquinas en el entrenamiento de los modelos provoca que el algoritmo encargado de este entrenamiento asigne pesos de manera incorrecta a un parámetro que no influye en la clasificación que se propone para este trabajo.

5.5. Preparación de los datos

En esta sección se modificarán los datos que presentan patrones, valores atípicos o sesgos que se comentan en la Sección anterior, [5.4 Selección de atributos](#). Así como el tratamiento necesario de los datos para poder entrenar los modelos.

Una de las características problemáticas, es la existencia de parámetros con valores infinito. Para tratar con esta problemática existen dos enfoques, o bien se sustituyen los valores infinitos por el máximo valor posible para ese parámetro, o bien, se elimina el flujo al que pertenece cada valor infinito. Para este trabajo, se opta por eliminar los registros con valores infinitos, ya que asignarles un valor falso, podría crear una relación sintética entre los datos y provocar que el modelo no converja en una solución real.

Para entrenar un modelo es necesario que en primer lugar todos los parámetros sean numéricos. En los datos originales del *dataset*, se encuentran tres parámetros que no presentan un formato numérico: `IPV4_SRC_ADDR`, `IPV4_DST_ADDR` y `Attack`.

- Para dar un formato correcto al parámetro `Attack`, se utiliza el codificador `LabelEncoder` de la biblioteca `sklearn.preprocessing` y el método `fit_transform` de la biblioteca `sklearn`. La primera función, `LabelEncoder` codifica las etiquetas de características categóricas en valores numéricos entre 0 y el número de clases menos 1. Una vez se instancian las categorías del parámetro, el método `fit_transform` permite entrenar el codificador y transformar el conjunto de datos en un único paso.
- Las direcciones IPv4 están formadas por cuatro octetos separados por puntos. Obviamente este formato no es numérico, por ese motivo se opta por dividir las direcciones IPv4 en 4 parámetros diferentes, uno por cada octeto. De esta manera, si el valor de un flujo para el parámetro `IPV4_SRC_ADDR` es 175.45.176.23, se separa en cuatro nuevos valores que son: 175, 45, 176 y 23. Para realizar esta transformación se utiliza la función [5.1 Función de transformación para los parámetros IPv4](#).

```
1 def split_ip_column(df, ip_column_name):
2
3     # Divide la IP en cuatro partes
4     ip_parts = df[ip_column_name].str.split('.', expand=True)
5
6     # Crea nombres de columnas basados en el nombre original
7     new_columns = {
8         0: f"{ip_column_name}_part1",
9         1: f"{ip_column_name}_part2",
10        2: f"{ip_column_name}_part3",
11        3: f"{ip_column_name}_part4"
12    }
13
14    # Se elimina la columna de ip_column_name
15    df = df.drop(columns=[ip_column_name])
16
17    # Añade las nuevas columnas al DataFrame
18    for part, col_name in new_columns.items():
19        df[col_name] = pandas.to_numeric(ip_parts[part]) # Convierte a numérico
20
21    return df
```

Figura 5.1: Función de transformación para los parámetros IPv4.

Una vez que todos los parámetros presentan valores numéricos, se separan los parámetros entre los que formarán parte de la entrada del modelo neuronal como atributos (a partir de ahora se les menciona como X) y los que contienen el valor de la salida que debe proporcionar el modelo neuronal llamados etiquetas (a partir de ahora se les menciona como Y).

Los parámetros que pertenecen a Y son **Label** y **Attack**. Sin embargo, en función del modelo que se entrena se utiliza o bien **Label**, o bien **Attack**, pero nunca los dos al mismo tiempo.

Por su parte, X la conforman el total de los parámetros del *dataset* a excepción de:

- **Label y Attack:** Se trata de los parámetros que conforman Y o etiquetas, si estos parámetros formasen parte de los datos de entrada del modelo, el algoritmo encargado de entrenarlo, les asignaría a estos parámetros unos pesos muy elevados y tendría una tasa teórica de éxito del 100%, puesto que, se estaría introduciendo la respuesta al problema que se aborda directamente como entrada al modelo. En un sistema informático real esta práctica es imposible.
- **IPV4_SRC_ADDR y IPV4_DST_ADDR:** Tal y como se menciona en la Sección [5.3 Parámetros de los datos](#), estos parámetros presentan un patrón irregular y sintético. Al introducir estos parámetros como entradas para el entrenamiento del modelo, de forma análoga a como sucede con los parámetros **Label** y **Attack** se estarían introduciendo valores que no se pueden obtener en una situación realista y condicionando el comportamiento del modelo. Esto implica como se menciona en los siguientes capítulos que el modelo resultante obtendría muy buenas métricas en validación pero muy malas al someterlo a datos que nunca ha visto durante su entrenamiento.

- **FLOW_START_MILLISECONDS y FLOW_END_MILLISECONDS:** A diferencia de los anteriores parámetros, estos muestran valores que puedan condicionar el entrenamiento de los modelos ni sus resultados, pero se trata de parámetros que ofrecen información redundante, pues el parámetro **FLOW_DURATION_MILLISECONDS** es combinación lineal de estos dos. Al eliminar estos parámetros se agiliza el entrenamiento del modelo y su tiempo de respuesta una vez entrenado sin perjudicar su precisión.

Como el conjunto de datos original presentaba 53 atributos de entrada a los modelos y se ha descartado el uso de 4 de ellos, los modelos serán entrenados con 49 atributos de entrada.

Para finalizar con el tratamiento de los datos, se normalizan los valores de X para que no haya tanta varianza entre los parámetros que lo conforman. Para normalizar los valores se utiliza el escalador **MinMaxScaler** que forma parte de la biblioteca **sklearn.preprocessing** junto con el método **fit_transform** que se menciona en la explicación de la transformación del parámetro **Attack**. Llamando a **MinMaxScaler** con un rango $(0,1)$, tras haber tratado los datos con el método **fit_transform**, se obtienen valores escalados en un rango entre 0 y 1, lo que facilita la comparación y el procesamiento por parte de los algoritmos de aprendizaje automático sin alterar las relaciones proporcionales inherentes en los datos originales.

Capítulo 6

Modelos

Este capítulo corresponde con la fase de modelado de la metodología CRISP-DM. En él se se explica como se crearon los primeros modelos, cual es el funcionamiento básico de las redes neuronales y algunos de los tipos de modelos más utilizados. Se detallan los modelos de redes neuronales desarrollados y entrenados. Se describe la arquitectura, la justificación de su elección, los hiperparámetros, la función de pérdida y el optimizador. Se incluye la estrategia de entrenamiento y las métricas de evaluación.

6.1. ¿Qué es un modelo neuronal?

Los modelos neuronales son estructuras computacionales inspiradas en el cerebro humano, diseñadas para procesar información mediante una red de unidades interconectadas que imitan, de manera simplificada, la forma en que las neuronas biológicas se comunican entre sí [34].

6.1.1. Origen de los modelos neuronales

Los descubrimientos del Premio Nobel Premio Nobel en Fisiología o Medicina español, Santiago Ramón y Cajal sobre la estructura y funcionamiento de las neuronas, proporcionaron las bases para la comprensión del sistema nervioso en el que se basan los modelos de inteligencia artificial. A finales del siglo XIX, Cajal formuló la teoría de que las neuronas son células individuales conectadas por sinapsis, una idea que revolucionó la neurociencia y sirvió de inspiración para los modelos computacionales de redes neuronales. Su trabajo permitió comprender cómo las señales eléctricas viajan entre las neuronas y cómo se pueden formar conexiones adaptativas, conceptos que más tarde serían adoptados en el diseño de redes neuronales artificiales [35].

En 1943, los científicos Warren McCulloch y Walter Pitts propusieron el primer modelo matemático de una neurona artificial, como la que se muestra en la Figura 6.1, en su trabajo titulado *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Este modelo representaba las neuronas como unidades lógicas binarias, cuya función de activación dependía de la suma ponderada de las entradas, las cuales podían ser 0 o 1. Cuando la suma de las entradas superaba un umbral determinado, la neurona se activaba (produciendo una salida de 1), mientras que si no se alcanzaba ese umbral, permanecía inactiva (salida 0). Este enfoque permitió representar funciones lógicas complejas mediante un conjunto de neuronas artificiales. El trabajo de McCulloch y Pitts fue fundamental para el desarrollo de las redes neuronales artificiales, ya que sentó las bases para representar las redes neuronales en términos matemáticos y lógicos, lo que eventualmente inspiró el desarrollo de redes más complejas [36].

Frank Rosenblatt desempeñó un papel fundamental en el desarrollo de los modelos neuronales al crear el *Perceptrón* en 1958, el primer modelo computacional de una red neuronal artificial. A diferencia del modelo lógico propuesto por McCulloch y Pitts, el *Perceptrón* consistía en una red neuronal de una sola capa, en la que las neuronas estaban conectadas a través de pesos que podían ajustarse durante el proceso de entrenamiento. El algoritmo de aprendizaje supervisado del *Perceptrón* permitía ajustar estos pesos para minimizar el error entre la salida producida por la red y la salida deseada. Este modelo de entrenamiento representó un avance importante en la teoría de redes neuronales, aunque su capacidad estaba limitada a resolver problemas lineales. A pesar de sus restricciones, el trabajo de Rosenblatt sentó las bases para el desarrollo de redes neuronales más complejas y la aparición del aprendizaje profundo en el futuro [37].

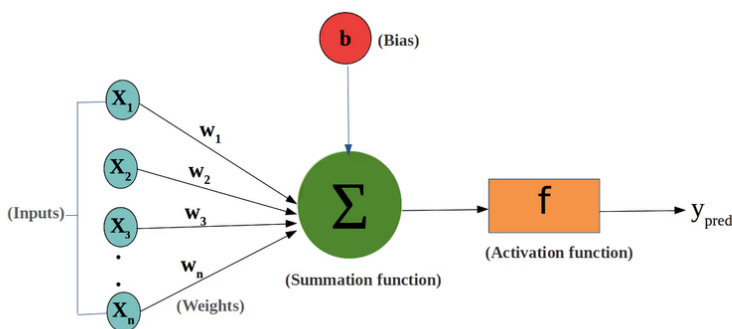


Figura 6.1: Esquema del funcionamiento de una neurona artificial [6].

La función de transferencia, también conocida como función de activación, es un componente esencial en las redes neuronales artificiales. Su principal función es introducir no linealidades en el modelo, permitiendo que la red neuronal aprenda y represente relaciones complejas entre las entradas y las salidas. Sin esta función, la red se limitaría a realizar transformaciones lineales, incapaces de capturar patrones no lineales presentes en los datos

reales [38].

Entre las funciones de activación más utilizadas se encuentran la sigmoide, la tangente hiperbólica ($\tanh$) y la unidad lineal rectificadora (ReLU). La función sigmoide, definida como $f(x) = \frac{1}{1+e^{-x}}$, transforma sus entradas en un rango entre 0 y 1, lo que la hace adecuada para problemas de clasificación binaria. Sin embargo, esta función presenta el inconveniente del desvanecimiento del gradiente, lo cual puede dificultar el entrenamiento de redes profundas [39].

Por otro lado, la función ReLU, expresada como $f(x) = \max(0, x)$, ha ganado popularidad debido a su simplicidad y eficiencia computacional. Esta función devuelve cero para entradas negativas y la entrada misma para valores positivos, lo que facilita una convergencia más rápida durante el entrenamiento y mitiga el problema del desvanecimiento del gradiente [40].

6.1.2. Funcionamiento básico de una red neuronal

En la actualidad, en su configuración más básica, un modelo neuronal es una arquitectura matemática que busca resolver problemas complejos de aprendizaje mediante el procesamiento de datos. Está compuesto por una serie de unidades de procesamiento, conocidas como neuronas artificiales, organizadas en capas. Como muestra la Figura 6.2, cada capa recibe la salida de la capa anterior, aplica una función matemática sobre los datos y transmite el resultado a la siguiente capa. Este proceso se repite de manera secuencial hasta que se alcanza la capa de salida, que proporciona el resultado final del modelo [41].

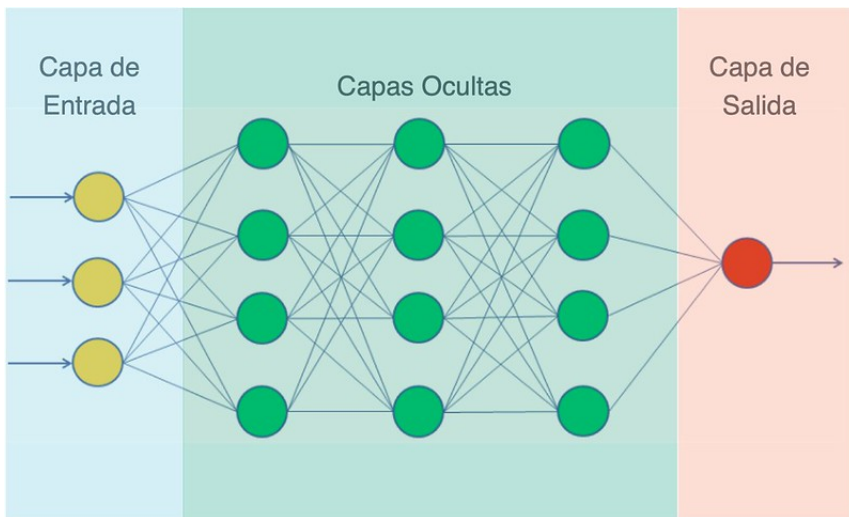


Figura 6.2: Esquema de redes neuronales [7].

Las neuronas dentro de un modelo neuronal están interconectadas mediante enlaces ponderados, conocidos como pesos, los cuales determinan la influencia relativa de cada señal

recibida. A estos pesos se les suma un parámetro adicional llamado *bias* u *offset*, que permite ajustar la salida de la neurona independientemente de las entradas recibidas. Este sesgo proporciona mayor flexibilidad al modelo al permitir que las funciones de activación se desplacen, lo cual facilita la representación de relaciones más complejas. Durante el proceso de entrenamiento, tanto los pesos como los valores de *bias* se modifican con el fin de reducir el error de predicción, mejorando así la capacidad del modelo para aprender de los datos y generalizar sobre nuevas instancias [42].

El entrenamiento de un modelo neuronal se basa en un mecanismo de retroalimentación conocido como *backpropagation*, en el cual el error de salida se propaga hacia atrás a través de la red para ajustar los parámetros del modelo. Este procedimiento optimiza iterativamente los pesos y los *bias* mediante el cálculo de los gradientes de error, permitiendo al modelo adaptarse progresivamente a los datos de entrada. Gracias a este proceso, las redes neuronales son capaces de mejorar su desempeño conforme se expone a mayores volúmenes de datos y tareas de diversa complejidad [43].

Función de pérdida

Una función de pérdida es un componente fundamental en el entrenamiento de redes neuronales, cuya finalidad consiste en cuantificar la discrepancia entre las predicciones generadas por el modelo y los valores reales esperados. Esta medida permite guiar el proceso de optimización, ya que el objetivo durante el entrenamiento es minimizar dicha pérdida para mejorar la precisión del modelo [44].

Las funciones de pérdida desempeñan un papel central en la formulación matemática del aprendizaje supervisado, al establecer una métrica que penaliza el error cometido por el modelo. Esta penalización permite que los algoritmos de optimización, como el descenso por gradiente, ajusten iterativamente los parámetros del modelo en la dirección que reduce la pérdida total. Para que esta estrategia sea efectiva, la función de pérdida debe poseer ciertas propiedades fundamentales.

Una de estas propiedades fundamentales es la diferenciabilidad, que permite el cálculo de gradientes, necesarios para optimizar los parámetros del modelo mediante técnicas basadas en derivadas, como el descenso por gradiente. Otra de las propiedades esenciales es la convexidad, aunque es deseable, favorece la convergencia hacia un mínimo global, esto se traduce en la práctica, en que muchos modelos no lineales presentan funciones de pérdida no convexas. La estabilidad numérica juega un papel crucial al prevenir errores computacionales derivados de valores extremos o transformaciones exponenciales, garantizando que la precisión se mantenga durante el entrenamiento. Por su parte, la sensibilidad al error como propiedad, asegura que los errores más grandes sean penalizados de manera más significativa, lo que orienta el modelo hacia mejores predicciones. Por su parte, la escalabilidad es una característica clave que permite la aplicación eficiente del modelo en contextos con grandes volúmenes de datos y arquitecturas de red complejas. Estas propiedades aseguran que la función de pérdida cumpla su objetivo central, actuar como un mecanismo confiable y eficiente para guiar la actualización de los parámetros del modelo [45].

Las redes neuronales se utilizan principalmente para tareas de *clustering*, clasificación y

regresión. En el caso del *clustering*, las redes neuronales permiten agrupar datos similares, como en la segmentación de clientes en *marketing*. Para la clasificación, las redes asignan etiquetas a los datos, como en el reconocimiento de imágenes, donde se identifican categorías. En regresión, las redes predicen un valor continuo, como el precio de una propiedad. Cada una de estas tareas requiere una función de coste específica: para clasificación se emplea comúnmente la entropía cruzada, mientras que para regresión se utiliza el error cuadrático medio.

En este proyecto, para resolver el problema objetivo de detectar intrusiones en redes, se emplea clasificación. La clasificación binaria consiste en asignar una de dos categorías posibles a una entrada, como la detección de correos electrónicos spam o no spam. Por otro lado, la clasificación multiclase se aplica cuando existen más de dos categorías, como en la clasificación de imágenes en diversas clases (por ejemplo, perro, gato, pájaro). Ambas tareas requieren que la red neuronal distinga entre las distintas categorías, aunque el enfoque y la arquitectura de la red pueden variar dependiendo de si el problema es binario o multiclase.

En el caso específico del modelo de clasificación binaria de este trabajo, se usa la función **BCEWithLogitsLoss** (*Binary Cross Entropy with Logits Loss*). Esta función combina en una sola operación dos pasos fundamentales: la aplicación de la función sigmoide y el cálculo de la entropía cruzada binaria. Al integrar ambos procedimientos en una única función, se obtienen varias ventajas prácticas [46]. En primer lugar, se mejora la estabilidad numérica, ya que evita operaciones redundantes que podrían dar lugar a pérdidas de precisión, especialmente al manejar las salidas no normalizadas del modelo (*logits*) con valores extremos. En segundo lugar, se optimiza el rendimiento computacional, al reducir el número de transformaciones necesarias antes del cálculo de la pérdida. Finalmente, permite trabajar directamente con *logits*, lo cual simplifica la implementación y reduce errores potenciales derivados de transformaciones incorrectas [47].

La función **BCEWithLogitsLoss** se define como:

$$\mathcal{L}(x, y) = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log(\sigma(x_i)) + (1 - y_i) \cdot \log(1 - \sigma(x_i))]$$

donde $\sigma(x_i) = \frac{1}{1+e^{-x_i}}$ es la función sigmoide aplicada al valor *logit* x_i , y $y_i \in \{0, 1\}$ es la etiqueta binaria correspondiente. Sin embargo, en la implementación práctica, para mejorar la estabilidad numérica, esta fórmula se reescribe como:

$$\mathcal{L}(x, y) = \frac{1}{N} \sum_{i=1}^N \left[\max(x_i, 0) - x_i y_i + \log(1 + e^{-|x_i|}) \right]$$

En el caso del modelo de clasificación multiclase, se emplea la función **CrossEntropyLoss**, recomendada específicamente para tareas donde el objetivo es predecir una única clase entre múltiples categorías posibles. Esta función de pérdida, integrada en el entorno de desarrollo **PyTorch**, combina de forma eficiente dos operaciones clave: la aplicación de la función *Softmax* para normalizar los *logits* del modelo en una distribución de probabilidad, y el cálculo de la

entropía cruzada entre dicha distribución y las etiquetas verdaderas codificadas como enteros [48].

La función de pérdida `CrossEntropyLoss` se define como:

$$\mathcal{L}(x, y) = - \sum_{i=1}^C y_i \log \left(\frac{e^{x_i}}{\sum_{j=1}^C e^{x_j}} \right)$$

donde:

- C es el número total de clases,
- x_i es el *logit* correspondiente a la clase i ,
- $y_i \in \{0, 1\}$ es la etiqueta codificada en *one-hot* para la clase verdadera.

En la práctica, si se tiene una etiqueta $y \in \{1, 2, \dots, C\}$ en lugar de codificación *one-hot*, la pérdida se reescribe como:

$$\mathcal{L}(x, y) = - \log \left(\frac{e^{x_y}}{\sum_{j=1}^C e^{x_j}} \right) = -x_y + \log \left(\sum_{j=1}^C e^{x_j} \right)$$

Esta función conlleva múltiples beneficios prácticos. En primer lugar, mejora la estabilidad numérica al evitar la explicitación separada del *Softmax*, que podría introducir errores de redondeo o valores infinitos al procesar *logits* extremos. En segundo lugar, incrementa la eficiencia computacional, al consolidar ambas operaciones en un único paso optimizado internamente por `PyTorch`. Finalmente, facilita la implementación, ya que permite trabajar directamente con salidas no normalizadas del modelo y etiquetas enteras, sin requerir codificaciones adicionales como *one-hot*, reduciendo así la probabilidad de errores en la fase de entrenamiento.

Algoritmo de optimización

Un algoritmo de optimización es un conjunto de procedimientos matemáticos y computacionales cuyo propósito es encontrar el mejor valor posible (o el más cercano a este) de una función para un conjunto de parámetros o variables dentro de un problema específico. Estos algoritmos buscan minimizar o maximizar una función objetivo, que representa el rendimiento del sistema o modelo a optimizar. En el contexto de entrenamiento de redes neuronales, la función objetivo comúnmente se refiere a una función de pérdida, que cuantifica la diferencia entre las predicciones del modelo y las etiquetas reales de los datos. Los algoritmos de optimización ajustan los parámetros, pesos y sesgos, de la red neuronal durante el proceso de entrenamiento con el fin de reducir dicha diferencia [34].

El funcionamiento de un algoritmo de optimización se basa en un proceso iterativo en el que se ajustan los pesos de la red neuronal utilizando información acerca del gradiente de la función de pérdida con respecto a estos parámetros. En este contexto, la retropropagación juega un papel crucial, ya que es el método utilizado para calcular esos gradientes. A través de la retropropagación, el error de la red se propaga hacia atrás desde la capa de salida hasta las capas anteriores, permitiendo calcular cómo cada parámetro contribuye al error total. Posteriormente, tal y como se muestra en la Figura 6.3, el algoritmo de optimización utiliza estos gradientes calculados por la retropropagación para actualizar los parámetros de la red en la dirección opuesta al gradiente, con el objetivo de reducir progresivamente la pérdida. Generalmente, se utilizan técnicas de descenso de gradiente para realizar estos ajustes. Durante cada iteración, el algoritmo calcula el gradiente de la función de pérdida a través de la retropropagación y lo utiliza para modificar los parámetros de manera eficiente, buscando alcanzar el mínimo global o local de la función de pérdida [49].

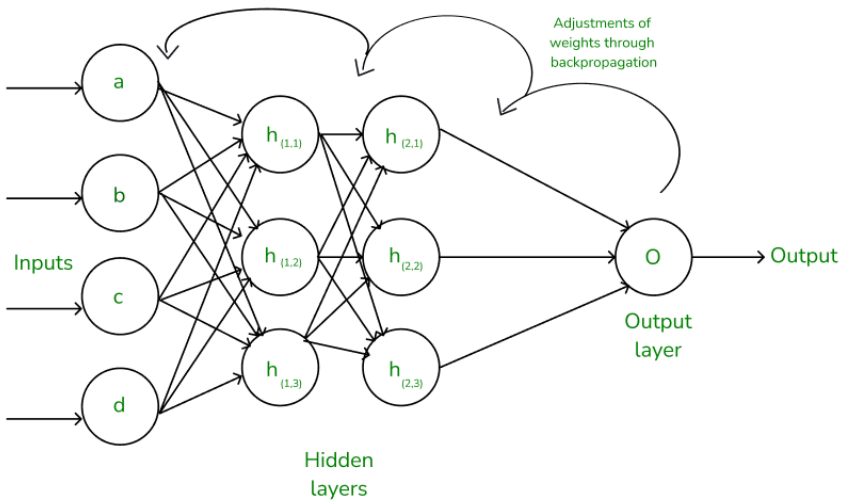


Figura 6.3: Esquema de algoritmo de optimización usando los gradientes calculados por la retropropagación [8].

Las características clave de los algoritmos de optimización incluyen la tasa de aprendizaje, la capacidad para escapar de mínimos locales, y la convergencia en un tiempo razonable. La tasa de aprendizaje (*learning rate* o *lr*) es un hiperparámetro crítico que determina la magnitud de los ajustes que se realizan en los parámetros del modelo durante cada iteración. Si la tasa de aprendizaje es muy alta, el algoritmo puede saltar el mínimo global, por otro lado, si es muy baja, el proceso puede volverse muy lento. Los métodos de optimización varían en cuanto a la forma en que gestionan estas características, algunos métodos mejoran la eficiencia utilizando técnicas como el momento, la normalización del gradiente, o el uso de segunda orden [50].

La importancia de los algoritmos de optimización en el entrenamiento de redes neuronales es fundamental, dado que la capacidad de un modelo para aprender de los datos y generalizar

de manera efectiva depende en gran medida de la optimización adecuada de los parámetros de la red. Un algoritmo de optimización correctamente elegido y configurado puede determinar la diferencia entre un modelo con un rendimiento subóptimo y uno que alcanza altos niveles de precisión y eficiencia. En consecuencia, la selección y ajuste apropiado de estos algoritmos se considera un aspecto crucial para el desarrollo exitoso de redes neuronales. [34].

Existen varios algoritmos de optimización ampliamente utilizados en el entrenamiento de redes neuronales. Entre los más comunes se encuentran el Descenso de Gradiente Estocástico (*Stochastic Gradient Descent*, SGD), *Momentum*, *Adagrad*, *RMSprop* y *Adam*. El SGD es uno de los más simples y fundacionales, pero sufre de inestabilidad debido a su naturaleza estocástica y la falta de adaptación de la tasa de aprendizaje. *Momentum* ayuda a mejorar la convergencia al incorporar la información de los gradientes pasados, lo que permite que el modelo escape de los mínimos locales más fácilmente. *Adagrad* adapta la tasa de aprendizaje para cada parámetro, mientras que *RMSprop* ajusta la tasa de aprendizaje utilizando una media móvil del gradiente, lo que mejora la estabilidad [49].

El sobreajuste, o *overfitting*, ocurre cuando un modelo de aprendizaje, como una red neuronal, ajusta excesivamente los parámetros a los datos de entrenamiento, capturando no solo la señal subyacente sino también el ruido o las fluctuaciones aleatorias. Esto resulta en un modelo que tiene un rendimiento excelente en el conjunto de entrenamiento pero que pierde capacidad de generalización, es decir, muestra un rendimiento deficiente en datos no vistos previamente, como los del conjunto de prueba. El sobreajuste es un desafío común en el entrenamiento de modelos complejos y puede mitigarse mediante técnicas como la regularización, el uso de datos adicionales o el ajuste adecuado de los hiperparámetros [41].

El sobreajuste puede ser especialmente problemático cuando se entrenan redes neuronales complejas, ya que estos modelos tienen una gran capacidad para ajustarse a las fluctuaciones específicas de los datos de entrenamiento, lo que lleva a una pérdida de generalización. Para mitigar este riesgo, es fundamental utilizar algoritmos de optimización y técnicas que favorezcan una buena generalización. Además de las estrategias mencionadas, la selección adecuada del algoritmo de optimización es crucial, ya que un algoritmo eficiente puede ayudar a encontrar un buen equilibrio entre el ajuste al conjunto de entrenamiento y la capacidad de generalización en datos no vistos.

El algoritmo más recomendado para entrenar un modelo de clasificación es *Adam* (*Adaptive Moment Estimation*). Adam combina las ventajas del descenso de gradiente estocástico y el momento, adaptando la tasa de aprendizaje de manera eficiente para cada parámetro del modelo. Esto lo hace particularmente adecuado para tareas de clasificación binaria, ya que garantiza una convergencia más rápida y estable, minimizando el riesgo de sobreajuste (*overfitting*) mientras se optimizan los parámetros del modelo. Sin embargo, una variante reciente de *Adam*, conocida como *AdamW*, introduce un cambio importante en la forma en que se maneja la regularización de los parámetros. *AdamW* separa explícitamente la actualización de los pesos y la regularización (típicamente L2 o *weight decay*), lo que mejora la eficacia de la regularización sin interferir con la adaptación de la tasa de aprendizaje.

La regularización es una técnica utilizada en el entrenamiento de redes neuronales, para prevenir el sobreajuste. Consiste en agregar un término adicional a la función de pérdida que penaliza los valores de los parámetros del modelo, de manera que se favorezcan soluciones

más simples y generalizables. Entre las formas más comunes de regularización se encuentran la regularización L1 (Lasso) y la regularización L2 (Ridge). La regularización L1 añade a la función de pérdida una penalización proporcional a la suma de los valores absolutos de los pesos. Esto tiende a hacer que muchos pesos se vuelvan exactamente cero, lo que induce *sparsity* (modelo más simple y con menos conexiones relevantes). Por su parte, la regularización L2, añade una penalización proporcional a la suma de los cuadrados de los pesos. No anula completamente los pesos, pero tiende a hacerlos más pequeños, distribuyendo la carga entre varias conexiones. Por lo que la regularización en general, actúa como un control de complejidad, ayudando al modelo a generalizar mejor a datos no vistos, lo que mejora su rendimiento en tareas de predicción y clasificación [41].

Aunque ambos algoritmos, *Adam* y *AdamW*, tienen un rendimiento destacado en tareas de clasificación binaria, *AdamW* se ha demostrado más eficaz en la prevención de sobreajuste en redes neuronales complejas. Esto se debe a su tratamiento explícito del *weight decay*, lo que permite una mejor generalización en problemas con grandes volúmenes de datos como el que se trata en este proyecto. Sin embargo, *Adam* sigue siendo una opción popular y efectiva, especialmente cuando se trabaja con redes neuronales menos complejas o cuando se desea una mayor estabilidad y rapidez de convergencia. En general, *AdamW* es preferido en situaciones donde la regularización adecuada es crucial, pero *Adam* sigue siendo una opción sólida y eficiente [50].

6.1.3. ¿Qué tipos de modelos neuronales existen?

Los modelos neuronales pueden clasificarse según su estructura, la forma en que procesan la información y la aplicación específica a la que están destinados. Estos modelos neuronales son fundamentales para resolver una variedad de problemas en áreas relacionadas con la inteligencia artificial como la visión por computadora, el procesamiento del lenguaje natural y el reconocimiento de patrones [41].

Uno de los tipos más comunes de modelos neuronales es el perceptrón multicapa (*Multilayer Perceptron*, MLP), que está formado por varias capas de neuronas organizadas en una estructura jerárquica. Cada capa en un MLP, como el que se muestra en la Figura 6.2, recibe la salida de la capa anterior y la procesa mediante una función de activación antes de pasar el resultado a la siguiente capa. Este tipo de red se utiliza principalmente para tareas de clasificación y regresión, y su entrenamiento se realiza utilizando algoritmos como el de retropropagación. Es el tipo de red que se utiliza en este proyecto para obtener un modelo capaz de detectar intrusiones en una red informática [34].

Otro tipo importante de modelo neuronal es la red neuronal convolucional (*Convolutional Neural Network*, CNN), que está diseñada específicamente para procesar datos con una estructura de cuadrícula, como las imágenes. En una CNN, las neuronas están organizadas en capas convolucionales que aplican filtros a los datos de entrada para extraer características relevantes, como bordes, texturas o formas. Esta estructura permite que las CNN sean altamente eficaces para tareas como el reconocimiento de imágenes y la visión por computadora [34].

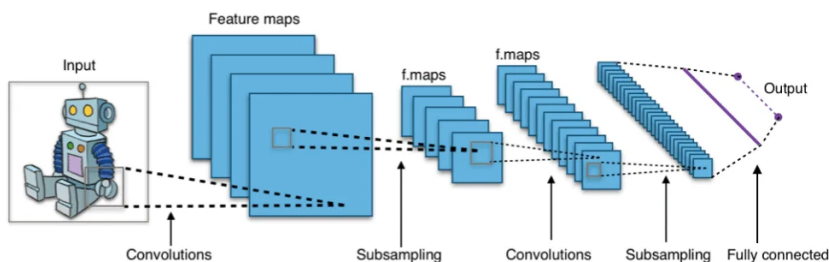


Figura 6.4: Esquema de redes neuronal convolucional [9].

Por otra parte, las redes neuronales recurrentes (*Recurrent Neural Network*, RNN), por otro lado, son adecuadas para procesar secuencias de datos, como el texto o el audio. Las neuronas de las RNN tienen conexiones que permiten que la información fluya hacia atrás a lo largo del tiempo, lo que las hace útiles para modelar dependencias temporales en los datos. Este tipo de red es comúnmente utilizado en tareas de procesamiento de lenguaje natural y en modelos de predicción de series temporales [42].

En un enfoque más avanzado, existen las redes generativas antagónicas (GAN), que se componen de dos redes neuronales: un generador y un discriminador. El generador crea datos falsos a partir de ruido, mientras que el discriminador intenta diferenciar entre los datos reales y los generados. A través de un proceso de entrenamiento competitivo, ambas redes mejoran en sus respectivas tareas. Las GANs se utilizan principalmente en la generación de imágenes, música y otros tipos de contenido artístico [43].

6.1.4. Técnicas útiles para obtener modelos que converjan correctamente

Las técnicas para obtener modelos que converjan correctamente son fundamentales en el proceso de entrenamiento de las redes neuronales. Cuando los modelos tratan de resolver problemas complejos como el que se trata en este proyecto, la optimización eficiente y la prevención de problemas como el sobreajuste o la falta de convergencia se vuelven desafíos clave. En esta sección, se comentan diversas estrategias y herramientas que facilitan el ajuste adecuado de los parámetros, garantizando que los modelos no solo alcancen una buena precisión en el conjunto de entrenamiento, sino que también generalicen de manera correcta para los datos no vistos durante el entrenamiento.

Validación cruzada

La validación cruzada consiste en dividir los datos de entrenamiento en n *folds* que se utilizan para entrenar al modelo con $n-1$ *folds* y después validar el modelo entrenado con el *fold* que no se ha utilizado para entrenarlo. Este proceso se repite para los n *folds* y en cada

iteración es un *fold* distinto el que no se utiliza para el entrenamiento. Una vez completadas las iteraciones, en este trabajo, se extraen las medias de las métricas sobre los datos que no se han usado para entrenar, obtenidas con cada *fold* durante la validación [51].

ReLU *Rectified Linear Unit*

La función de activación ReLU (*Rectified Linear Unit*) ampliamente utilizada en redes neuronales profundas. Su definición matemática es sencilla: $f(x) = \max(0, x)$, lo que significa que devuelve el valor de entrada si este es positivo y cero en caso contrario. La principal utilidad de la ReLU radica en su capacidad para introducir no linealidades en los modelos, lo cual permite que las redes neuronales aprendan relaciones complejas entre los datos. A diferencia de funciones de activación desarrolladas históricamente antes de la ReLU, como la sigmoide o la tangente hiperbólica, ReLU es computacionalmente eficiente y ayuda a mitigar el problema del *vanishing gradient* (suceso por el que los gradientes que se propagan hacia atrás a través de las capas se vuelven cada vez más pequeños a medida que se avanza hacia las primeras capas de la red), facilitando el entrenamiento de redes profundas. Es una de las funciones de activación más simples y con mejor rendimiento en la actualidad [52].

Batch Normalization

Batch Normalization es una técnica que normaliza las activaciones de una capa para que tengan media cercana a 0 y varianza cercana a 1, calculadas a partir del *batch* actual del entrenamiento del modelo. Aplicar *Batch Normalization* tras la primera capa en un modelo de clasificación multiclase ayuda a estabilizar, acelerar y regularizar el entrenamiento, mejorando la capacidad del modelo para aprender buenas representaciones desde las primeras fases del entrenamiento [53].

Dropout

Dropout es una técnica de regularización utilizada en redes neuronales para reducir el sobreajuste durante el entrenamiento. La función `Dropout()`, implementada en bibliotecas como `TensorFlow` y `PyTorch`, actúa introduciendo aleatoriamente una probabilidad de desactivación sobre las neuronas de una capa determinada. Durante cada iteración de entrenamiento, un subconjunto aleatorio de neuronas es temporalmente desactivado (es decir, sus salidas se anulan) con una probabilidad definida, denominada tasa de *dropout* (*dropout rate*), sin afectar la arquitectura del modelo ni sus pesos de forma permanente [54]. Los principales beneficios de utilizar *dropout* en un modelo de clasificación multiclase con unos datos de entrenamiento muy desbalanceados son:

- Reduce el sobreajuste hacia las clases mayoritarias, evitando que el modelo memorice solo los patrones frecuentes y mejorando su comportamiento general.
- Fomenta la generalización en clases minoritarias, al impedir dependencias excesivas entre neuronas y promover representaciones más robustas y diversas.

- Mejora métricas sensibles al desbalance como el weighted F1-score o macro recall, al favorecer que el modelo aprenda también de clases con baja representación.

SMOTE

SMOTE (*Synthetic Minority Over-sampling Technique*) es una técnica de sobremuestreo utilizada para abordar el problema del desbalance de clases en conjuntos de datos de clasificación. Su funcionamiento se basa en generar ejemplos sintéticos de las clases minoritarias, en lugar de replicar instancias existentes. En tareas de clasificación multiclase, **SMOTE** puede aplicarse a cada clase minoritaria de forma independiente, generando ejemplos sintéticos para aquellas clases con menor representación [55]. La aplicación de esta técnica en modelos de clasificación multiclase muy desbalanceados:

- Contribuye a reducir el sesgo del modelo hacia las clases mayoritarias.
- Mejora el rendimiento en métricas como *recall* y F1-score macro, explicadas en la Sección 6.2.4, al equilibrar el conjunto de entrenamiento.
- Favorece una mejor generalización en la predicción de clases minoritarias, al ampliar su representación en el espacio de características.

6.1.5. Conjunto de datos de entrenamiento

El conjunto de entrenamiento es fundamental en el proceso de desarrollo de un modelo de aprendizaje automático, ya que es sobre este conjunto de datos donde el modelo aprende a reconocer patrones y relaciones. Durante el entrenamiento, el modelo ajusta sus parámetros con el objetivo de minimizar el error en las predicciones, basándose en los ejemplos proporcionados en este conjunto. Un conjunto de entrenamiento bien diseñado y representativo de los datos reales es crucial para que el modelo sea capaz de generalizar correctamente a nuevos datos. Si el conjunto de entrenamiento es pequeño o no es representativo de la diversidad de situaciones que el modelo enfrentará, el rendimiento del modelo puede ser peor de lo esperado, incluso si tiene un buen desempeño en los datos de entrenamiento [41].

Para evitar el sobreajuste del modelo y obtener valores de las predicciones más confiables que se utilizarán para la evaluación del rendimiento del modelo, se utiliza validación cruzada *k-fold*. Para entrenar el modelo de clasificación binaria, se ha utilizado `StratifiedKFold()` que como su nombre indica, estratifica los *folds* para que las proporciones de las clases de los datos utilizados para el entrenamiento se mantengan similares en cada *fold* y no se creen *folds* con un único tipo de clase. Estratificar los datos es especialmente crítico para aquellos *datasets* cuyos datos están muy desbalanceados como es el caso de los datos utilizados para entrenar los modelos de este proyecto. Para el entrenamiento del modelo de clasificación binaria, se ha optado por utilizar validación cruzada `StratifiedKFold()` con 5 *folds* o divisiones.

6.1.6. Parámetros e hiperparámetros

Como se ha comentado previamente, los parámetros de un modelo neuronal son los valores ajustables que el modelo aprende a partir de los datos durante el proceso de entrenamiento. Estos parámetros incluyen los pesos y sesgos de las conexiones entre las neuronas de las diferentes capas de la red neuronal. Los pesos determinan la importancia de las entradas de cada neurona, mientras que los sesgos permiten ajustar la salida de cada neurona antes de la activación. Como se explica en los apartados anteriores, el aprendizaje de estos parámetros se realiza mediante algoritmos de optimización como el gradiente descendente, los cuales buscan minimizar la función de pérdida [34].

Los hiperparámetros son los valores que se configuran antes del entrenamiento y no se ajustan directamente durante el proceso de aprendizaje. Este tipo de parámetros controlan el comportamiento del proceso de entrenamiento y afectan la capacidad del modelo para aprender patrones complejos de los datos. A diferencia de los parámetros, los hiperparámetros deben ser establecidos manualmente o mediante técnicas de optimización específicas, como la búsqueda en cuadrícula o la búsqueda aleatoria, técnicas que se aplican en las siguientes secciones de este documento. Los hiperparámetros influyen en aspectos clave como la velocidad de aprendizaje, la regularización y la estructura de la red [56]. Los más importantes en el MLP son:

- **Tasa de aprendizaje (*Learning rate*):** Controla la magnitud de los ajustes realizados en los pesos durante cada iteración del entrenamiento. Como se comenta en la Sección 6.1.2 Algoritmo de optimización, una tasa demasiado alta puede hacer que el modelo no converja, mientras que una tasa demasiado baja puede llevar a una convergencia muy lenta.
- **Número de épocas (*Epochs*):** El número de épocas se refiere a cuántas veces el algoritmo de entrenamiento recorre todo el conjunto de datos. Un número adecuado de épocas permite que el modelo aprenda de manera efectiva, sin embargo, demasiadas épocas pueden llevar al sobreajuste.
- **Tamaño del lote (*Batch size*):** El tamaño del lote determina la cantidad de ejemplos utilizados en cada actualización de los pesos. Un tamaño de lote pequeño puede hacer que el modelo sea más ruidoso y difícil de entrenar, mientras que un tamaño de lote grande puede hacer que el entrenamiento sea más estable, pero menos eficiente.
- **Número de capas ocultas (*Hidden layers*):** Este hiperparámetro define la profundidad de la red neuronal, es decir, el número de capas intermedias entre la capa de entrada y la capa de salida. Un mayor número de capas permite modelar funciones más complejas, pero también aumenta el riesgo de sobreajuste.
- **Número de neuronas por capa (*Hidden factor*):** Este parámetro establece la cantidad de neuronas en cada capa oculta. Un mayor número de neuronas puede mejorar la capacidad del modelo para capturar patrones complejos, aunque también aumenta el tiempo de entrenamiento y la complejidad computacional.

6.1.7. PyTorch

PyTorch es una biblioteca de código abierto especializada en el desarrollo de modelos de aprendizaje automático. Proporciona una interfaz flexible y eficiente para la construcción de redes neuronales, apoyada por una arquitectura de tensores multidimensionales similar a la utilizada por **NumPy**, pero optimizada para operaciones en unidades de procesamiento gráfico (GPU). Fue desarrollada y la mantenida Meta AI y cuenta con una comunidad activa de desarrolladores y usuarios que contribuyen a su evolución [48].

En el contexto del aprendizaje automático, **PyTorch** se utiliza principalmente para el diseño, entrenamiento y evaluación de modelos de redes neuronales profundas. Su enfoque dinámico de construcción de grafos computacionales facilita el desarrollo de arquitecturas complejas, permite un control granular sobre los flujos de datos y resulta especialmente útil en tareas de investigación y prototipado rápido [57].

Entre sus principales características se destacan la compatibilidad con aceleración por GPU, la construcción dinámica de modelos mediante el mecanismo de *define-by-run*, la integración con bibliotecas como **TorchVision** para visión por computadora y **TorchText** para procesamiento de lenguaje natural, así como herramientas para la depuración y trazabilidad del entrenamiento. Además, **PyTorch** se integra con plataformas como **TensorBoard** y **Weights & Biases** para monitorización avanzada del rendimiento de los modelos [58].

En el contexto del aprendizaje automático y el procesamiento de datos numéricos, un tensor es una estructura de datos multidimensional que generaliza los conceptos de escalares (tensores de orden 0), vectores (orden 1) y matrices (orden 2) a dimensiones superiores. Los tensores permiten representar datos complejos de forma compacta y eficiente, siendo fundamentales para operaciones algebraicas utilizadas en redes neuronales. Un tensor puede considerarse como una colección de elementos numéricos organizados en un espacio de n dimensiones, lo cual posibilita la ejecución de operaciones paralelas en lotes de datos durante el entrenamiento de modelos [34].

Desde la perspectiva de la programación orientada a objetos (POO), un tensor se implementa típicamente como una instancia de una clase que encapsula tanto los datos como los métodos que operan sobre ellos. Esta abstracción permite que las bibliotecas como **PyTorch** y **TensorFlow** ofrezcan una interfaz coherente para manipular tensores, incluyendo funciones como transposición, difusión, operaciones elementales y derivación automática. Gracias a este enfoque, los tensores no solo representan estructuras matemáticas, sino también objetos programables que responden a métodos y se integran con otros componentes del sistema de manera modular y extensible [56].

6.2. Clasificación con redes neuronales

La clasificación con redes neuronales es una técnica clave en el aprendizaje supervisado, utilizada para asignar etiquetas a nuevas instancias a partir de ejemplos previamente etiquetados. En esta sección, se exploran los métodos para evaluar la efectividad de los modelos

de clasificación, incluyendo el uso de matrices de confusión, y se abordan las métricas más comunes para medir el rendimiento de modelos en tareas de clasificación binaria y multiclase.

6.2.1. Matriz de confusión para clasificación binaria

En esta sección se explica en que consiste una matriz de confusión y las variaciones que existen de las matrices de confusión en función del tipo de modelo neuronal de clasificación que se entrene. Son una herramienta fundamental utilizada en el campo del aprendizaje automático y la clasificación, especialmente cuando se evalúan modelos de clasificación como el que se propone en este proyecto.

La matriz de confusión es una representación tabular que permite evaluar el rendimiento de un modelo de clasificación. Esta matriz compara las predicciones del modelo con los valores reales (verdaderos) y proporciona una visión detallada sobre los errores cometidos. Esta matriz permite calcular diversas métricas de evaluación del modelo, que son esenciales para entender la efectividad del modelo en tareas de clasificación [59].

En el caso de los modelos neuronales de clasificación binaria, la matriz tiene una estructura 2x2, donde cada celda en la matriz representa la cantidad de veces que una combinación específica de etiqueta y clase predicha ocurrió. Los valores posibles para la clasificación binaria son:

- Verdaderos positivos (VP): son las instancias que pertenecen a la clase positiva y que el modelo ha clasificado correctamente como positivas.
- Falsos positivos (FP): corresponden a las instancias que no pertenecen a la clase positiva, pero que el modelo ha etiquetado incorrectamente como positivas.
- Falsos negativos (FN): se refieren a las instancias que deberían ser clasificadas como positivas, pero que el modelo ha predicho como negativas.
- Verdaderos negativos (VN): son las instancias que pertenecen a la clase negativa y que el modelo ha clasificado correctamente como negativas.

	Predicción Positiva	Predicción Negativa
Real Positivo	Verdaderos Positivos (VP)	Falsos Negativos (FN)
Real Negativo	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Tabla 6.1: Matriz de confusión para clasificación binaria.

6.2.2. Métricas útiles en clasificación binaria

Las métricas en el contexto de los modelos de aprendizaje automático son herramientas cuantitativas utilizadas para evaluar el rendimiento de un modelo entrenado. Estas métricas

permiten medir si el modelo está realizando la tarea para la cual fue diseñado de manera correcta, comparando sus predicciones con los valores reales de los datos de prueba. Son fundamentales para comprender la efectividad del modelo y para identificar áreas de mejora, ya que proporcionan una evaluación objetiva y reproducible del comportamiento del modelo [51].

El uso adecuado de las métricas es crucial, ya que guían las decisiones sobre el ajuste y la optimización del modelo. Sin métricas claras, el proceso de evaluación se vuelve subjetivo y difícil de manejar. Además, las métricas permiten realizar comparaciones entre diferentes modelos o configuraciones, facilitando la selección del modelo más adecuado para una tarea específica. Una interpretación correcta de las métricas ayuda a evitar problemas como el sobreajuste o el subajuste, y garantiza que el modelo generalice bien a nuevos datos [34].

Para evaluar el desempeño del modelo de detección y clasificación de ataques, se utilizan las siguientes métricas derivadas de la matriz de confusión que se explican en el contexto del principal objetivo de este proyecto, es decir, la detección de intrusiones en una red.

■ **Exactitud (*Accuracy*):**

$$\text{Accuracy} = \frac{VP + VN}{VP + FP + VN + FN} \quad (6.1)$$

En el entrenamiento de modelos neuronales para la detección de intrusiones, esta métrica representa la proporción del total de las clasificaciones realizadas correctamente. En este trabajo, indica la capacidad general del modelo para distinguir entre tráfico normal e intrusivo. Si bien ofrece una visión global del rendimiento, su valor disminuye en escenarios donde la cantidad de tráfico normal supera significativamente al tráfico malicioso, ya que el modelo puede obtener una alta exactitud simplemente clasificando la mayoría de las instancias como normales.

■ **Precisión (*Precision*):**

$$\text{Precision} = \frac{VP}{VP + FP} \quad (6.2)$$

Esta métrica evalúa la capacidad del modelo neuronal para evitar la identificación incorrecta de tráfico normal como intrusivo. En la detección de intrusiones, una alta precisión es crucial para minimizar las falsas alarmas, las cuales pueden generar una sobrecarga operativa en los equipos de seguridad, requiriendo la revisión de eventos benignos y distrayendo la atención de amenazas reales. Un modelo preciso reduce la fatiga de alertas y permite una respuesta más eficiente a incidentes genuinos.

■ **Sensibilidad (*Recall*):**

$$\text{Recall} = \frac{VP}{VP + FN} \quad (6.3)$$

La sensibilidad mide la habilidad del modelo neuronal para detectar todas las instancias de intrusión presentes en el tráfico de red. En el contexto de la seguridad, un alto

recall es de suma importancia, ya que implica una menor probabilidad de que ataques reales pasen desapercibidos. Un modelo con baja sensibilidad puede tener consecuencias graves, permitiendo que actividades maliciosas se infiltren y comprometan la integridad y la confidencialidad de los sistemas.

■ **Puntuación F1 (*F1-Score*):**

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (6.4)$$

Esta métrica proporciona una evaluación equilibrada del rendimiento del modelo neuronal al calcular la media entre la precisión y el *recall*. En la detección de intrusiones, donde a menudo existe un desequilibrio entre el tráfico normal y el malicioso, el F1-score ofrece una métrica más robusta que la exactitud, ya que considera tanto la capacidad de evitar falsas alarmas como la de detectar todas las intrusiones. Un valor alto de F1-score indica un buen compromiso entre ambas capacidades.

■ **Puntuación F2 (*F2-Score*):**

$$F2 = 5 \cdot \frac{Precision \cdot Recall}{4 \cdot Precision + Recall} \quad (6.5)$$

Esta variante de la puntuación F pondera la sensibilidad más que la precisión. En el ámbito de la detección de intrusiones, el *F2-score* resulta útil cuando las consecuencias de no detectar un ataque (FN) se consideran significativamente más perjudiciales que generar una falsa alarma (FP). Al asignar un mayor peso al *recall*, se prioriza la identificación de la mayor cantidad posible de actividades maliciosas, incluso a expensas de un posible aumento en las falsas alertas.

■ **Área bajo la curva (*ROC AUC*):**

$$FPR = \frac{FP}{FP + VN} \quad (6.6)$$

$$Recall(FPR(t)) = Recall(t) \quad (6.7)$$

$$ROC\ AUC = \int_0^1 Recall(FPR) d(FPR) \quad (6.8)$$

El área bajo la curva ROC (Receiver Operating Characteristic) mide la capacidad del modelo neuronal para discriminar entre clases positivas y negativas a lo largo de todos los posibles umbrales de decisión. En la curva ROC, la FPR (False Positive Rate) actúa como la variable del eje X , mientras que el *Recall* (o TPR, True Positive Rate) corresponde a la variable del eje Y . En el contexto de la seguridad, un alto valor de AUC indica que el modelo asigna puntuaciones más altas a las intrusiones que al tráfico normal de manera consistente, lo que sugiere una buena capacidad de priorización de amenazas. Esta métrica es especialmente útil cuando se desea evaluar el rendimiento global del modelo sin depender de un umbral fijo. Un modelo con bajo AUC podría fallar al separar el tráfico malicioso, lo que aumentaría la probabilidad de errores en la clasificación y podría comprometer la eficacia general del sistema de detección [60].

Aplicación en Seguridad

En el contexto de detección de intrusiones:

- Un *Recall* alto ($> 95\%$) asegura que pocos ataques pasan desapercibidos.
- La precisión debe optimizarse para reducir la carga operativa de analistas (falsos positivos $< 10\%$).
- El *F2-Score* es preferible al *F1* cuando la prioridad es minimizar riesgos de ataques no detectados.

6.2.3. Matriz de confusión para clasificación multiclase

Como se explica en la Sección [6.2.1 Matriz de confusión para clasificación binaria](#), la matriz de confusión es una representación tabular que permite evaluar el rendimiento de un modelo de clasificación. Esta matriz compara las predicciones del modelo con los valores reales (verdaderos) y proporciona una visión detallada sobre los errores cometidos. Esta matriz permite calcular diversas métricas de evaluación del modelo, que son esenciales para entender la efectividad del modelo en tareas de clasificación.

Para un modelo con múltiples clases, como es el caso del modelo de que se desarrolla en este documento, se debe tener en cuenta que en un modelo neuronal que clasifica nueve clases diferentes, la matriz tiene las siguientes interpretaciones:

- Diagonal principal: Cada celda de la diagonal principal de la matriz representa cuántas veces la clase real fue correctamente predicha. Este es el caso de las instancias que fueron correctamente clasificadas, y es lo más cercano a un «verdadero positivo» para esa clase específica. Sin embargo, en clasificación multiclase, se suele hablar de aciertos o instancias correctamente clasificadas para cada clase.
- Fuera de la diagonal: Las celdas fuera de la diagonal representan falsas clasificaciones. Es decir, representan cuántas veces una instancia de la clase real fue predicha incorrectamente como otra clase.

Rellenando una matriz con la información dada, se obtiene la matriz de confusión para un modelo de clasificación que distingue entre 9 clases que se muestra en la Figura [6.2](#).

	Predicción Clase 1	Predicción Clase 2	Predicción Clase 3	Predicción Clase 4	Predicción Clase 5	Predicción Clase 6	Predicción Clase 7	Predicción Clase 8	Predicción Clase 9
Real Clase 1	VP ₁	FP ₁₂	FP ₁₃	FP ₁₄	FP ₁₅	FP ₁₆	FP ₁₇	FP ₁₈	FP ₁₉
Real Clase 2	FP ₂₁	VP ₂	FP ₂₃	FP ₂₄	FP ₂₅	FP ₂₆	FP ₂₇	FP ₂₈	FP ₂₉
Real Clase 3	FP ₃₁	FP ₃₂	VP ₃	FP ₃₄	FP ₃₅	FP ₃₆	FP ₃₇	FP ₃₈	FP ₃₉
Real Clase 4	FP ₄₁	FP ₄₂	FP ₄₃	VP ₄	FP ₄₅	FP ₄₆	FP ₄₇	FP ₄₈	FP ₄₉
Real Clase 5	FP ₅₁	FP ₅₂	FP ₅₃	FP ₅₄	VP ₅	FP ₅₆	FP ₅₇	FP ₅₈	FP ₅₉
Real Clase 6	FP ₆₁	FP ₆₂	FP ₆₃	FP ₆₄	FP ₆₅	VP ₆	FP ₆₇	FP ₆₈	FP ₆₉
Real Clase 7	FP ₇₁	FP ₇₂	FP ₇₃	FP ₇₄	FP ₇₅	FP ₇₆	VP ₇	FP ₇₈	FP ₇₉
Real Clase 8	FP ₈₁	FP ₈₂	FP ₈₃	FP ₈₄	FP ₈₅	FP ₈₆	FP ₈₇	VP ₈	FP ₈₉
Real Clase 9	FP ₉₁	FP ₉₂	FP ₉₃	FP ₉₄	FP ₉₅	FP ₉₆	FP ₉₇	FP ₉₈	VP ₉

Tabla 6.2: Matriz de confusión para clasificación con 9 clases.

Para facilitar la comprensión de los datos, en las siguientes secciones se utilizará la matriz de confusión con los datos obtenidos. En estas representaciones, las clases del modelo, que se corresponden con las clases del *dataset* utilizado y que se mencionan en la Sección 5.2, se encuentra codificadas de la siguiente manera:

- Clase 0: *Analysis*.
- Clase 1: *Backdoor*.
- Clase 2: *DoS*.
- Clase 3: *Exploits*.
- Clase 4: *Fuzzers*.
- Clase 5: *Generic*.
- Clase 6: *Reconnaissance*.
- Clase 7: *Shellcode*.
- Clase 8: *Worms*.

6.2.4. Métricas útiles en clasificación multiclase

En el desarrollo de modelos neuronales para clasificación multiclase, es fundamental contar con métricas que permitan evaluar el desempeño general y específico del modelo. A continuación, se presentan las métricas más relevantes, agrupadas según su naturaleza: generales, macro-promediadas y ponderadas.

Métricas Generales

■ **Exactitud (*Accuracy*):**

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}_{\{y_i = \hat{y}_i\}} \quad (6.9)$$

La exactitud cuantifica la proporción de predicciones correctas sobre el total de instancias evaluadas. En contextos de clasificación multiclase, proporciona una medida global del rendimiento del modelo. Sin embargo, su uso puede resultar limitado en escenarios donde las clases están desbalanceadas, ya que tiende a favorecer aquellas clases con mayor representación. En estos casos, una alta exactitud puede enmascarar un bajo rendimiento en clases minoritarias, lo que podría comprometer la robustez del modelo. Se considera una métrica de referencia general, pero insuficiente para evaluar en profundidad el comportamiento del modelo en todos los grupos de clase [61].

Métricas Macro-promediadas

Las métricas macro-promediadas se calculan evaluando individualmente el rendimiento del modelo para cada clase y luego promediando estos resultados. Esta metodología asigna igual peso a todas las clases, independientemente de su frecuencia, lo que permite detectar deficiencias de desempeño en clases poco representadas. Son particularmente útiles en problemas de clasificación multiclase con desequilibrios significativos [62].

■ **Precisión Macro (*Macro Precision*):**

$$\text{Precision}_{\text{macro}} = \frac{1}{C} \sum_{c=1}^C \text{Precision}_c \quad (6.10)$$

Donde:

- C es el número total de clases en el problema de clasificación.
- Precision_c es la precisión calculada para la clase c .

La precisión macro evalúa la proporción de verdaderos positivos entre todas las predicciones positivas, calculada individualmente para cada clase y promediada de forma equitativa. Esta métrica permite identificar si el modelo tiende a emitir predicciones incorrectas en ciertas clases, lo cual puede ser crítico en aplicaciones donde los falsos positivos resultan costosos. En entornos multiclase, su utilidad radica en evaluar la calidad de las predicciones para todas las clases de forma uniforme [63].

■ **Sensibilidad Macro (*Macro Recall*):**

$$\text{Recall}_{\text{macro}} = \frac{1}{C} \sum_{c=1}^C \text{Recall}_c \quad (6.11)$$

Donde:

- C es el número total de clases en el problema de clasificación.
- Recall_c es el recall calculado para la clase c , que se define como:

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c}$$

donde TP_c son los verdaderos positivos y FN_c son los falsos negativos para la clase c .

La sensibilidad macro (o exhaustividad) mide la proporción de verdaderos positivos sobre el total de instancias reales de cada clase. Al promediarla sin ponderar por la frecuencia de clase, esta métrica refleja la capacidad del modelo de detectar correctamente todas las clases, incluidas aquellas menos frecuentes. Resulta esencial en situaciones donde los falsos negativos tienen consecuencias graves o donde se busca una cobertura completa del espacio de clases [64].

- **F1 Macro (*Macro F1 Score*):**

$$F1_{\text{macro}} = \frac{1}{C} \sum_{c=1}^C F1_c \quad (6.12)$$

Donde:

- C es el número total de clases en el problema de clasificación.
- $F1_c$ es la puntuación F1 calculada para la clase c , que es la media armónica entre la precisión y el recall para la clase c :

$$F1_c = 2 \cdot \frac{\text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

donde Precision_c y Recall_c son la precisión y el recall de la clase c , respectivamente.

El F1 macro se obtiene calculando el F1 score individual por clase (media armónica entre precisión y sensibilidad), y promediándolo uniformemente. Esta métrica proporciona una visión equilibrada del desempeño del modelo en todas las clases, y es especialmente útil cuando se busca un rendimiento homogéneo en un entorno multiclase, independientemente de la distribución de las muestras [61].

Métricas Ponderadas (*Weighted*)

Las métricas ponderadas (*weighted*) también se calculan por clase, pero se ajustan según la frecuencia de aparición de cada clase en el conjunto de datos. Este enfoque permite obtener una representación más realista del rendimiento global del modelo, otorgando mayor peso a las clases con mayor número de instancias. Son adecuadas para obtener una evaluación que respete la distribución natural del conjunto de datos, sin perder información relevante de las clases menos representadas [65].

■ **Precisión Ponderada (*Weighted Precision*):**

$$\text{Precision}_{\text{weighted}} = \sum_{c=1}^C \frac{N_c}{N} \cdot \text{Precision}_c \quad (6.13)$$

Donde:

- C es el número total de clases.
- N_c es el número de muestras de la clase c .
- N es el número total de muestras.
- Precision_c es la precisión de la clase c , calculada como:

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}$$

donde TP_c son los verdaderos positivos y FP_c son los falsos positivos para la clase c .

La precisión ponderada evalúa la calidad de las predicciones positivas, ajustada por la proporción de instancias de cada clase. Esta métrica resulta útil para capturar el rendimiento general del modelo en conjuntos de datos con distribución desigual, sin que las clases poco frecuentes dominen el resultado final [66].

■ **Sensibilidad Ponderada (*Weighted Recall*):**

$$\text{Recall}_{\text{weighted}} = \sum_{c=1}^C \frac{N_c}{N} \cdot \text{Recall}_c \quad (6.14)$$

Donde:

- C es el número total de clases.
- N_c es el número de muestras de la clase c .
- N es el número total de muestras.
- Recall_c es el recall de la clase c , calculado como:

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c}$$

donde TP_c son los verdaderos positivos y FN_c son los falsos negativos para la clase c .

La sensibilidad ponderada proporciona una medida del porcentaje de instancias reales correctamente identificadas, teniendo en cuenta la frecuencia de cada clase. En modelos de clasificación multiclase, permite analizar cómo se comporta el modelo con respecto a la cobertura general de los datos, favoreciendo una evaluación proporcional al tamaño de las clases [67].

■ **F1 Ponderado (*Weighted F1 Score*):**

$$F1_{\text{weighted}} = \sum_{c=1}^C \frac{N_c}{N} \cdot F1_c \quad (6.15)$$

Donde:

- C es el número total de clases.
- N_c es el número de muestras de la clase c .
- N es el número total de muestras.
- $F1_c$ es la puntuación F1 de la clase c , calculada como:

$$F1_c = 2 \cdot \frac{\text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

El F1 ponderado combina precisión y sensibilidad ajustadas por la proporción de cada clase. Es una de las métricas más importantes cuando se desea evaluar un rendimiento global equilibrado que refleje el impacto relativo de cada clase en el conjunto de datos, resultando especialmente útil para comparar modelos entrenados sobre *datasets* desbalanceados [68].

Área Bajo la Curva ROC (AUC)

El área bajo la curva ROC (AUC) mide la capacidad del modelo para distinguir entre clases. En clasificación multiclase, se emplean dos métodos principales para su cálculo:

■ **AUC *One-vs-One* (OVO):**

$$\text{AUC}_{\text{ovo}} = \frac{1}{C(C-1)/2} \sum_{i < j} \text{AUC}_{i,j} \quad (6.16)$$

Donde:

- C es el número total de clases en el problema de clasificación.
- $\text{AUC}_{i,j}$ es el área bajo la curva (AUC) para la comparación entre las clases i y j (*One-vs-One*). Esta métrica calcula la capacidad del modelo para distinguir entre todas las posibles parejas de clases en un problema multiclase.

El enfoque *One-vs-One* calcula el AUC para cada par de clases, evaluando la capacidad del modelo para distinguir una clase frente a otra. El resultado final se obtiene promediando los valores obtenidos para todos los pares posibles. Esta estrategia proporciona una evaluación detallada de la separabilidad entre clases específicas. Es útil cuando se quiere analizar el comportamiento del modelo en decisiones binarias dentro del espacio multiclase [69].

■ **AUC *One-vs-Rest* (OVR):**

$$\text{AUC}_{\text{ovr}} = \frac{1}{C} \sum_{c=1}^C \text{AUC}_c \quad (6.17)$$

Donde:

- C es el número total de clases en el problema de clasificación.
- AUC_c es el área bajo la curva (AUC) para la clase c , que mide la capacidad del modelo para distinguir entre la clase c y el resto de las clases (*One-vs-Rest*). Es un valor entre 0 y 1, donde 1 indica una clasificación perfecta y 0.5 indica una clasificación aleatoria.

El enfoque *One-vs-Rest* evalúa, para cada clase, la capacidad del modelo de distinguir dicha clase frente a todas las demás combinadas. Esta métrica permite examinar el rendimiento del modelo desde la perspectiva de cada clase individual y es adecuada para diagnósticos detallados en entornos multiclase [70].

6.3. Arquitectura de los modelos de clasificación binaria desarrollados

En esta sección se abordan cuales son las arquitecturas a desarrollar para el modelo de clasificación binaria. Como se comenta en la Sección [6.5.2 Diseño del modelo e hiperparámetros seleccionados](#), la diferencia entre las arquitecturas radica en el tamaño de la capa oculta del modelo.

Para el desarrollo de las diferentes arquitecturas, la red neuronal elegida es el MLP. Además, se ha optado por realizar experimentos de entrenamiento del modelo de clasificación binaria con un tamaño de capa oculta igual a:

- **MCB25:** La mitad del número de atributos o entradas que recibe el modelo
- **MCB49:** El mismo número que atributos o entradas que tiene el modelo.
- **MCB98:** El doble del número de atributos o entradas.

Esta configuraciones mencionadas, equivalen a un tamaño de la capa oculta de 25, 49 y 98 respectivamente.

6.3.1. Capa oculta con la mitad de neuronas que atributos de entrada

En este apartado se comentan cuales son las principales ventajas e inconvenientes, a priori, de un modelo de clasificación binaria basado en MLP con un número de neuronas en

la capa oculta igual a la mitad del número de parámetros que recibe el modelo, como la que se muestra en la Figura 6.5.

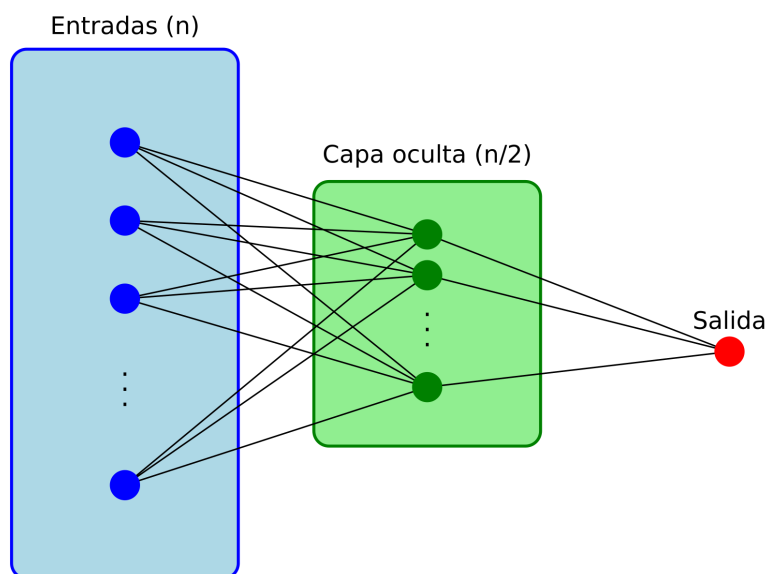


Figura 6.5: Arquitectura del modelo con $n/2$ neuronas en la capa oculta siendo n el número de parámetros de entrada.

Ventajas ($n/2$)

- El riesgo de sobreajuste es menor, esto es especialmente importante si no se utiliza validación cruzada o el conjunto de datos de entrenamiento es pequeño o simple [71].
- Al tener un número de neuronas menor, tanto la función de pérdida como el algoritmo de optimización, tardan menos en realizar los cálculos de los ajustes de los pesos y sesgos del modelo. Este menor coste de cálculo se refleja directamente en el tiempo necesario para entrenar el modelo y en el tiempo de respuesta del mismo una vez entrenado [34].
- Un número bajo de neuronas es especialmente útil cuando los datos son linealmente separables o poco complejos [72].

Inconvenientes ($n/2$)

- Si las relaciones entre los datos son demasiado complejas, el modelo puede sufrir *underfitting* o subajuste [34].

- Sensibilidad alta a la inicialización de los pesos, lo que puede provocar que un porcentaje alto de neuronas no se utilicen por tener pesos iguales o muy cercanos a 0. Esto implica que parte de la capacidad de la red quede inutilizada, empeorando el rendimiento incluso en problemas simples [73].

Tras comentar cuales son las principales ventajas e inconvenientes de utilizar una capa oculta con un número de neuronas igual a la mitad de los atributos del modelo, aparentemente, las propiedades del conjunto de datos que se utiliza para entrenar al modelo no son las idóneas para esta arquitectura dado que la complejidad de la red puede ser algo limitada para resolver un problema complejo como el de detectar intrusiones en sistemas informáticos. Por este motivo, no se esperan resultados especialmente destacables de esta arquitectura.

6.3.2. Capa oculta con el mismo número de neuronas que atributos de entrada

En este apartado se comentan cuales son las principales ventajas e inconvenientes, a priori, de un modelo de clasificación binaria con un número de neuronas en la capa oculta igual al número de parámetros que recibe el modelo, como la que se muestra en la Figura 6.6.

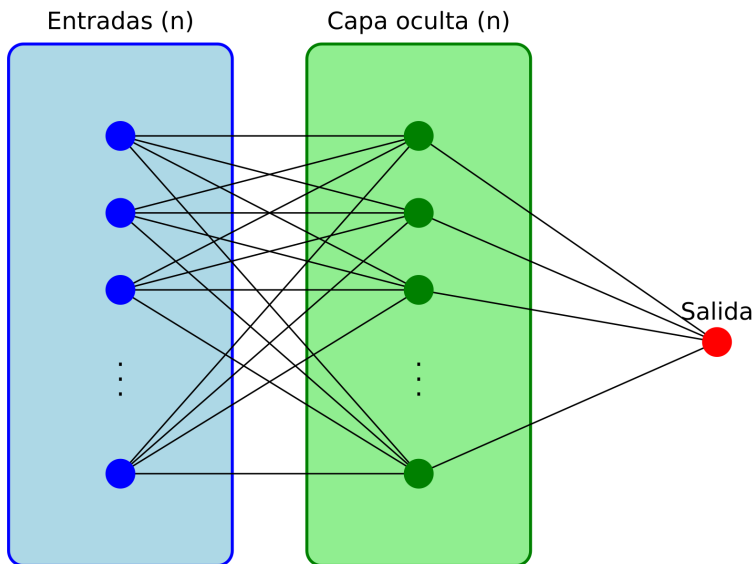


Figura 6.6: Arquitectura del modelo con n neuronas en la capa oculta siendo n el número de parámetros de entrada.

Ventajas (n)

- La capacidad y la generalización están balanceadas, esto significa que el modelo evita tanto el sobreajuste como el subajuste [74].
- Es capaz de modelar relaciones más complejas entre conjuntos de datos que arquitecturas con un número de neuronas menor, lo que es especialmente útil en problemas con relaciones no lineales entre sus datos [34].
- Son computacionalmente más eficientes que arquitecturas con un número elevado de neuronas en la capa oculta [75].

Inconvenientes(n)

- Puede resultar insuficiente en problemas demasiado complejos o con un gran volumen de datos [76].
- Si el conjunto de datos utilizado para entrenar el modelo es pequeño puede sobreajustar [77].

Tras comentar cuales son las principales ventajas e inconvenientes de utilizar una capa oculta con el mismo número de neuronas que entradas recibe el modelo, aparentemente, esta arquitectura ofrecerá resultados significativamente positivos gracias a su equilibrio y balanceo.

6.3.3. Capa oculta con el doble de neuronas que atributos de entrada

En este apartado se comentan cuales son las principales ventajas e inconvenientes, a priori, de un modelo de clasificación binaria con el doble de neuronas en la capa oculta que parámetros recibe el modelo, como la que se muestra en la Figura 6.7.

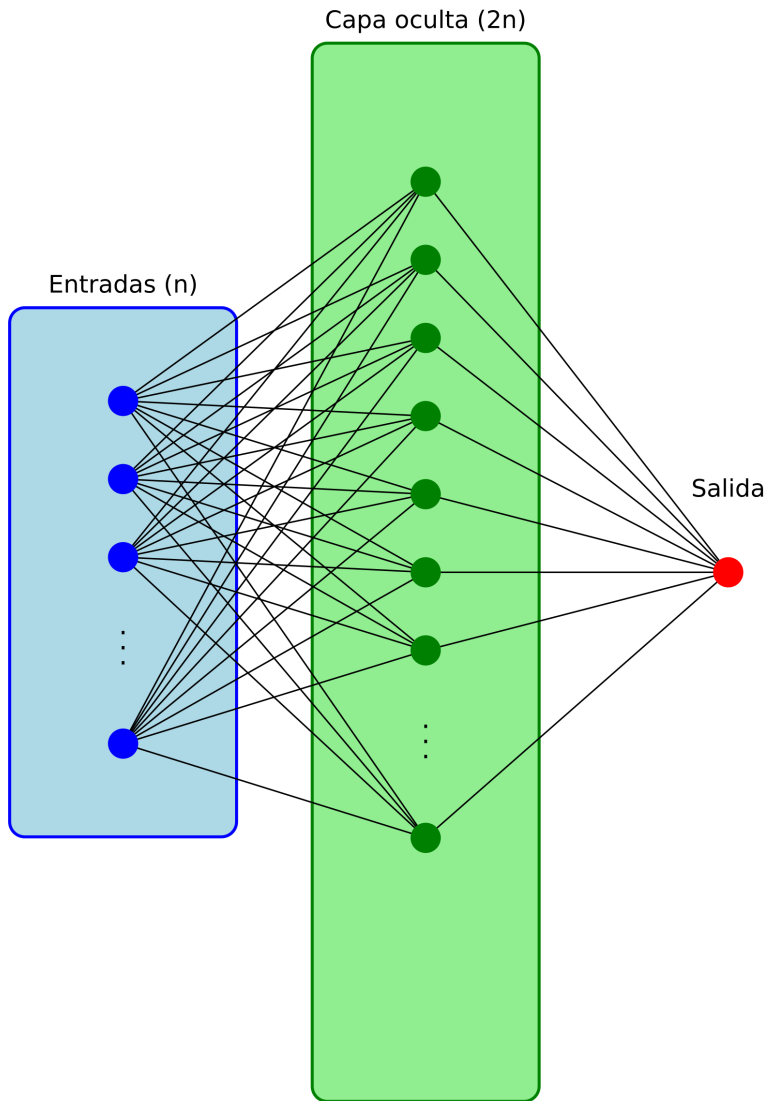


Figura 6.7: Arquitectura del modelo con $2n$ neuronas en la capa oculta siendo n el número de parámetros de entrada.

Ventajas ($2n$)

- Alta capacidad de aprendizaje, lo que se traduce como la posibilidad de capturar patrones de datos muy complejos que no son lineales ni perceptibles en un análisis superficial del problema [34].

- Cuanto mayor es el conjunto de datos mejor se ajustan los pesos [78].
- Es menos propenso a subajustar que otras arquitecturas [41].

Inconvenientes (2n)

- El riesgo de sobreajuste es elevado, memorizando de esta manera ruido en vez de aprender los patrones generalizables [41].
- Nivel computacional más elevado, lo que se refleja en el tiempo necesario de entrenamiento de esta arquitectura y en su posterior tiempo de respuesta [34].
- Si el número de muestras del conjunto de datos es bajo tiende a sobreajustar [79].

Tras comentar cuales son las principales ventajas e inconvenientes de utilizar una capa oculta con un número de neuronas igual al doble de los parámetros del modelo, aparentemente, esta arquitectura obtendrá mejores resultados si los parámetros no tiene relaciones lineales. En cambio, si los datos presentan relaciones lineales tiene más probabilidades de sobreajustar los pesos del modelo, lo que implicaría que el modelo no ha sido capaz de converger a una solución genérica.

6.4. Arquitecturas de los modelos de clasificación multiclase desarrollados

En esta sección se abordan cuales son las arquitecturas a desarrollar para el modelo de clasificación multiclase. Como se comenta en la Sección [6.6.2 Diseño del modelo e hiperparámetros seleccionados](#), la diferencia entre las arquitecturas radica en el tamaño de la capa oculta del modelo.

Para el desarrollo de las diferentes arquitecturas, se ha optado por realizar experimentos de entrenamiento del modelo de clasificación multiclase con un tamaño de ventana oculta igual a:

- **MCM25:** La mitad del número de atributos que recibe el modelo
- **MCM49:** El mismo número de atributos que tiene el modelo.
- **MCM98:** El doble del número de atributos.

Esto equivale a un tamaño de la capa de ventana oculta de 25, 49 y 98 respectivamente, de forma análoga a como sucede con las arquitecturas del modelo de clasificación binaria.

A continuación, se comentan las posibles ventajas e inconvenientes de cada arquitectura que se esperan antes de realizar los experimentos.

6.4.1. Capa oculta con la mitad de neuronas que atributos de entrada

En este apartado se comentan cuales son las principales ventajas e inconvenientes, a priori, de un modelo de clasificación multiclase con un número de neuronas en la capa oculta igual a la mitad del número de parámetros que recibe el modelo, como la que se muestra en la Figura 6.8.

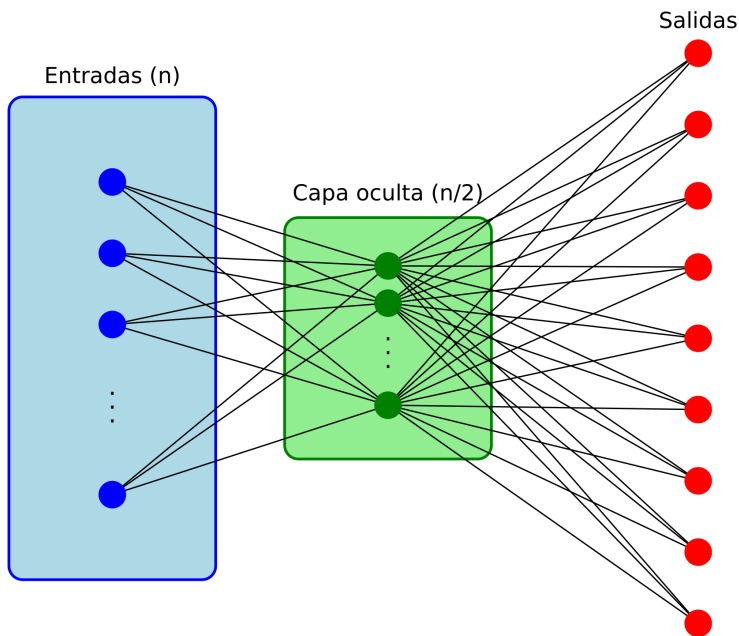


Figura 6.8: Arquitectura del modelo con $n/2$ neuronas en la capa oculta siendo n el número de parámetros de entrada.

Ventajas ($n/2$)

- Un número reducido de neuronas en la capa oculta limita la capacidad del modelo para memorizar patrones dominantes, lo cual puede ayudar a mitigar el sobreajuste hacia las clases mayoritarias del conjunto de entrenamiento [74].
- Al reducirse la complejidad del modelo, se disminuye el tiempo necesario para calcular los gradientes durante el entrenamiento y, en consecuencia, también el tiempo de respuesta durante la inferencia [80].
- En presencia de clases desbalanceadas, un modelo menos expresivo puede favorecer una mayor estabilidad en el aprendizaje, evitando que la red se especialice únicamente en las clases más frecuentes [81].

Inconvenientes ($n/2$)

- Un número bajo de neuronas puede limitar la capacidad del modelo para aprender representaciones suficientes de las clases minoritarias, especialmente si estas presentan patrones complejos o poco diferenciados [82].
- La capacidad reducida del modelo puede dificultar la separación entre clases en problemas multiclase, afectando negativamente al rendimiento en métricas sensibles al desbalance como el *macro Recall* o el *macro F1-score* [81].
- Si el número de parámetros de entrada es muy alto, una capa oculta con solo la mitad de neuronas podría actuar como un cuello de botella, perdiéndose información útil para distinguir correctamente entre clases poco representadas [34].

Tras comentar cuales son las principales ventajas e inconvenientes de utilizar una capa oculta con un número de neuronas igual a la mitad de los parámetros del modelo, aparentemente, las propiedades del conjunto de datos que se utiliza para entrenar al modelo no son las idóneas para esta arquitectura y no se esperan resultados especialmente destacables. Debido al alto desbalanceo entre clases, se espera que esta arquitectura proporcione resultados peores que su equivalente del modelo de clasificación binaria.

6.4.2. Capa oculta con el mismo número de neuronas que atributos de entrada

En este apartado se comentan cuales son las principales ventajas e inconvenientes, a priori, de un modelo de clasificación multiclase con un número de neuronas en la capa oculta igual al número de parámetros que recibe el modelo, como la que se muestra en la Figura 6.9.

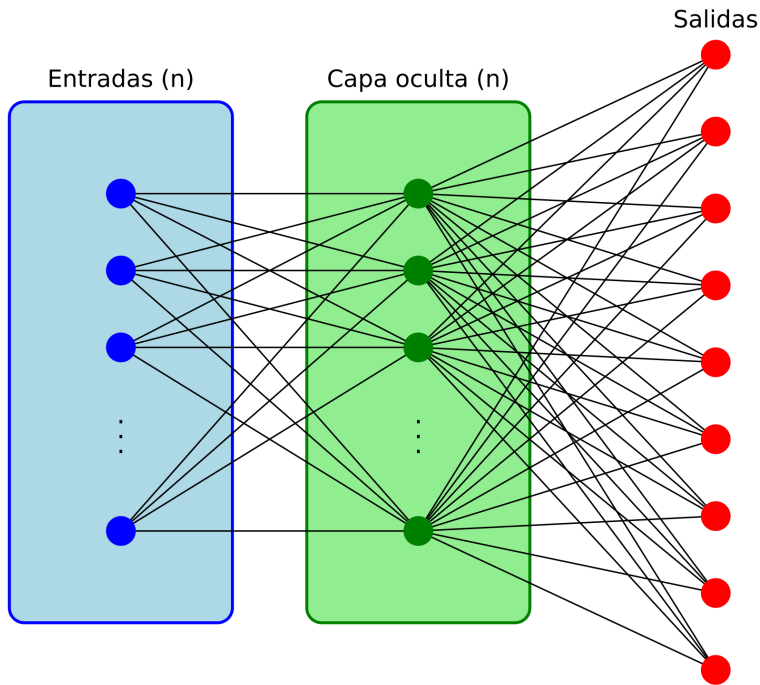


Figura 6.9: Arquitectura del modelo con n neuronas en la capa oculta siendo n el número de parámetros de entrada.

Ventajas (n)

- Un número de neuronas igual al de parámetros de entrada proporciona al modelo una capacidad representativa suficiente para capturar patrones relevantes tanto en clases frecuentes como en las minoritarias [83].
- Esta configuración ofrece un equilibrio razonable entre complejidad y eficiencia computacional, manteniendo un coste de entrenamiento y de inferencia moderado sin comprometer la expresividad del modelo [34].
- Permite al modelo adaptar sus pesos de manera más flexible, facilitando el aprendizaje de límites de decisión más precisos entre clases desbalanceadas sin riesgo inmediato de sobreajuste [81].

Inconvenientes (n)

- Aun sin ser excesivo, este tamaño puede generar cierta sobreadaptación a las clases mayoritarias si no se aplican mecanismos de regularización adecuados [34].

- En conjuntos de datos muy desbalanceados, esta configuración puede no ser suficiente para compensar la escasa representación de clases minoritarias, especialmente si estas requieren una mayor capacidad para ser correctamente diferenciadas [81].
- Si las características de entrada tienen una alta correlación entre sí, mantener el mismo número de neuronas puede introducir redundancia, afectando la eficiencia del aprendizaje y aumentando el riesgo de converger a soluciones deficientes [84].

Tras comentar cuales son las principales ventajas e inconvenientes de utilizar una capa oculta con el mismo número de neuronas que parámetros recibe el modelo, aparentemente, esta arquitectura ofrecera resultados significativamente positivos gracias a su equilibrio. Aunque, si las clases que tienen un número inferior de muestras, tienen relaciones complejas entre sus datos, esta arquitectura puede ser insuficiente para obtener un modelo capaz de distinguir entre algunas de las clases más minoritarias.

6.4.3. Capa oculta con el doble de neuronas que atributos de entrada

En este apartado se comentan cuales son las principales ventajas e inconvenientes, a priori, de un modelo de clasificación multiclase con el doble de neuronas en la capa oculta que parámetros recibe el modelo, como la que se muestra en la Figura 6.10.

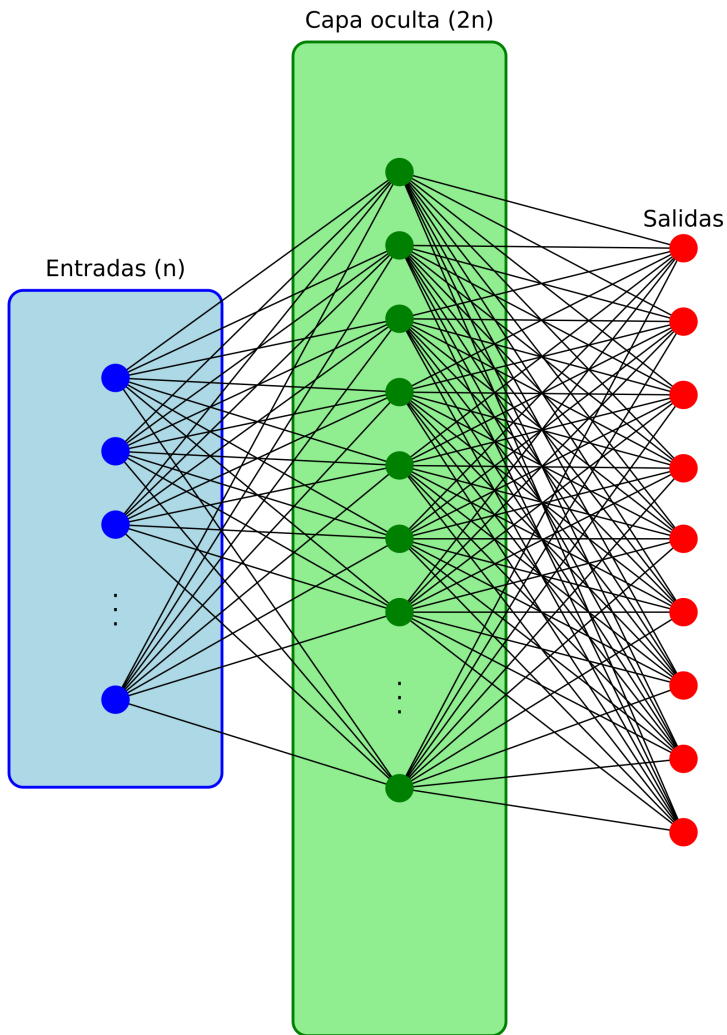


Figura 6.10: Arquitectura del modelo con $2n$ neuronas en la capa oculta siendo n el número de parámetros de entrada.

Ventajas ($2n$)

- Una capa oculta más amplia incrementa la capacidad del modelo para aprender representaciones complejas, lo que puede ser especialmente útil para distinguir correctamente clases minoritarias con patrones sutiles o poco definidos [54].
- Esta configuración ofrece una mayor flexibilidad al modelo para aproximar funciones no lineales, favoreciendo una mejor separación entre clases en escenarios multiclasa con alta variabilidad entre categorías [41].

- Al disponer de más neuronas, el modelo tiene mayor margen para explorar combinaciones de características relevantes durante el entrenamiento, lo cual puede traducirse en una mejora de métricas como macro F1 o recall por clase [85].

Inconvenientes (2n)

- El aumento en el número de parámetros incrementa el riesgo de sobreajuste, especialmente en presencia de clases mayoritarias dominantes o cuando el conjunto de entrenamiento es reducido [54].
- Un modelo con mayor complejidad requiere más recursos computacionales y tiempos de entrenamiento más prolongados, lo que puede dificultar su implementación en entornos con restricciones de capacidad o temporales [86].
- Si no se utilizan mecanismos de control adecuados (como regularización o balanceo de clases), es posible que el modelo aprenda a ajustar principalmente las clases frecuentes, sin mejorar sustancialmente el rendimiento en las minoritarias [55].

Tras comentar cuales son las principales ventajas e inconvenientes de utilizar una capa oculta con un número de neuronas igual al doble de los parámetros del modelo, aparentemente, esta arquitectura será la que mejores resultados obtenga en la detección de clases minoritarias al tener la capacidad de aprender representaciones más complejas. Sin embargo, esta arquitectura es más propensa a sobreajustes debido la presencia de las clases mayoritarias.

6.4.4. Diseños del modelo de clasificación multiclase descartados

Debido a la complejidad del problema al resolver, se probaron otros diseños diferentes a los finalmente utilizados que fueron descartados por no obtener los resultados esperados. A continuación, se detalla cuales fueron estos diseños descartados, la motivación que llevo a probarlo y cuales fueron sus resultados.

Durante las pruebas realizadas antes de ejecutar los experimentos para encontrar la combinación de hiperparámetros que mejor convergían para este modelo, se detectó que el desbalanceo de las clases afectaba de manera significativa a los resultados que se obtenían. Con el objetivo de reducir el impacto del desbalanceo en el entrenamiento del modelo, se optó por ejecutar experimentos para sondear el desempeño de diferentes técnicas. Los resultados de estos experimentos se muestran en la Tabla 6.3, junto con los resultados que obtuvieron estas combinaciones de técnicas en la métrica *F1 weighted*.

Experimentos	Capa bn1 e hiperparámetros con valores altos	Con capa bn1	Con capa bn1 y dropout	Con SMOTE y capa bn1	Con SMOTE
batch_size	256	32	32	32	32
epochs	80	30	30	30	30
hidden_size	512	128	128	128	128
learning_rate	10^{-2}	10^{-5}	10^{-5}	10^{-5}	10^{-5}
avg_f1_weighted	0.544441	0.467310	0.461414	0.369414	0.282888

Tabla 6.3: Resultados de los diferentes diseños probados para el modelo de clasificación multiclase.

Sin embargo, tras realizar una serie de pruebas, el modelo diseñado que implementaba SMOTE fue el que peores resultados dió. Por eso motivo, se combinó con la función **Batch Normalization** explicada en la Sección 6.1.4 [Batch Normalization](#).

Los resultados de **Batch Normalization** (bn1) y SMOTE mostrados en la Tabla 6.3 fueron algo mejores que los valores obtenidos de las pruebas realizadas solo con SMOTE. Como se noto una mejora en el desempeño del modelo al utilizar bn1, pero los resultados obtenidos seguían sin ser los esperados, se probó con combinaciones de diseño utilizando bn1 con otras técnicas.

Con la implementación del **Dropout**, los resultados mejoraron significativamente. Sin embargo, el diseño del modelo que solo utilizaba bn1 obtuvo mejores resultados de media que aquel que utilizaba bn1 y *dropout*. Por este motivo, el diseño utilizado en los experimentos solo implementa **Batch Normalization**. Además, tras realizar pruebas con valores altos para los hiperparámetros, se obtuvieron unos resultados con una convergencia mayor que la de las pruebas realizadas con valores menores, tal y como puede observarse en la Tabla 6.3, lo que incentivó a aumentar los valores de los hiperparámetros probados, tal y como se puede observar en la sección anterior.

6.5. Implementación del modelo neuronal de clasificación binaria

En esta sección se describe tanto el proposito como los procedimientos empleados para el desarrollo del modelo de clasificación binaria. Se detalla el proceso de selección de hiperparámetros, así como las distintas arquitecturas diseñadas con el fin de identificar la combinación que optimiza el rendimiento del modelo. Además, se presenta una comparación entre las arquitecturas evaluadas para determinar la más adecuada.

6.5.1. Proposito del modelo de clasificación binaria

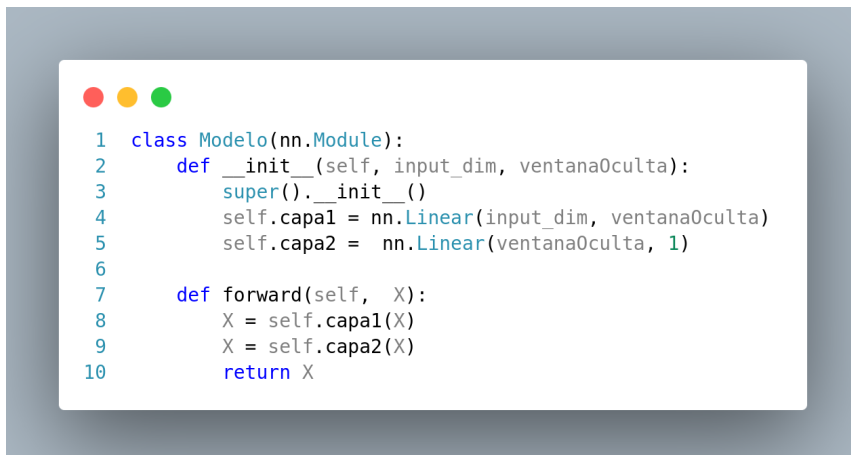
El modelo de clasificación binaria se ha diseñado con el objetivo de identificar de manera precisa y eficiente, si una instancia de tráfico de red corresponde a una actividad legítima o a un comportamiento malicioso. Su función principal consiste en distinguir entre accesos normales y posibles intentos de intrusión, permitiendo así la detección temprana de ataques y contribuyendo a la protección proactiva de los sistemas informáticos.

6.5.2. Diseño del modelo e hiperparámetros seleccionados

Para diseñar la arquitectura del modelo, se ha desarrollado la clase `Modelo`. Esta clase está conformada por dos métodos, la primera corresponde con la inicialización de la clase, y la segunda con la arquitectura utilizada para este modelo.

Como se puede observar en la Figura 6.11, el modelo está compuesto por dos capas. La `capa1` tiene un número de entradas igual al número de parámetros que posee el `dataset`, como se comenta en la Sección 5.5, el número de entradas del modelo es 49. El número de salidas depende del parámetro `ventanaOculta`, con el que se llama a la función de inicialización del modelo. El número de salidas de la `capa1` será precisamente el que determine la arquitectura específica que se utiliza. Por su parte, la `capa2` recibe como entrada el número de salidas de la `capa1` y al tratarse de la capa final de un modelo de clasificación binaria, esta capa tendrá una única salida.

Debido a que la función de pérdida utilizada para entrenar el modelo es `BCEWithLogitsLoss()`, que ya aplica la función sigmoidea, sería contraproducente aplicar la función sigmoidea sobre la salida del modelo para su entrenamiento.



```
1 class Modelo(nn.Module):
2     def __init__(self, input_dim, ventanaOculta):
3         super().__init__()
4         self.capa1 = nn.Linear(input_dim, ventanaOculta)
5         self.capa2 = nn.Linear(ventanaOculta, 1)
6
7     def forward(self, X):
8         X = self.capa1(X)
9         X = self.capa2(X)
10        return X
```

Figura 6.11: Definición de la clase del modelo de clasificación binaria.

Como algoritmo de optimización para el entrenamiento del modelo de clasificación bina-

ria, se ha optado por utilizar `AdamW()`. Como se comenta en la Sección 6.1.2 [Algoritmo de optimización](#), se ha demostrado que AdamW es más eficaz en la prevención de sobreajuste en redes neuronales que otros algoritmos de optimización. Además, permite una mejor generalización en problemas con grandes volúmenes de datos como el que se trata en este proyecto.

Los hiperparámetros que han sido seleccionados para encontrar la mejor configuración para el modelo de clasificación binaria diseñado son: el batch size, la tasa de aprendizaje (*learning rate*) y el número de épocas. Estos hiperparámetros resultan imprescindibles en el entrenamiento de modelos de clasificación binaria, dado que influyen directamente en la eficiencia del proceso de optimización y en la calidad del modelo resultante. Los posibles valores de estos hiperparámetros con los que se van a realizar los experimentos son los que se muestran en la Tabla 6.4.

Como se comenta en la Sección 6.1.6 [Parámetros e hiperparámetros](#), los hiperparámetros controlan el comportamiento del proceso de entrenamiento y afectan la capacidad del modelo para aprender patrones complejos de los datos. Teniendo en cuenta las características de cada hiperparámetro, se han escogido los siguientes valores para los experimentos del entrenamiento del modelo binario con el objetivo de encontrar la mejor combinación de ellos.

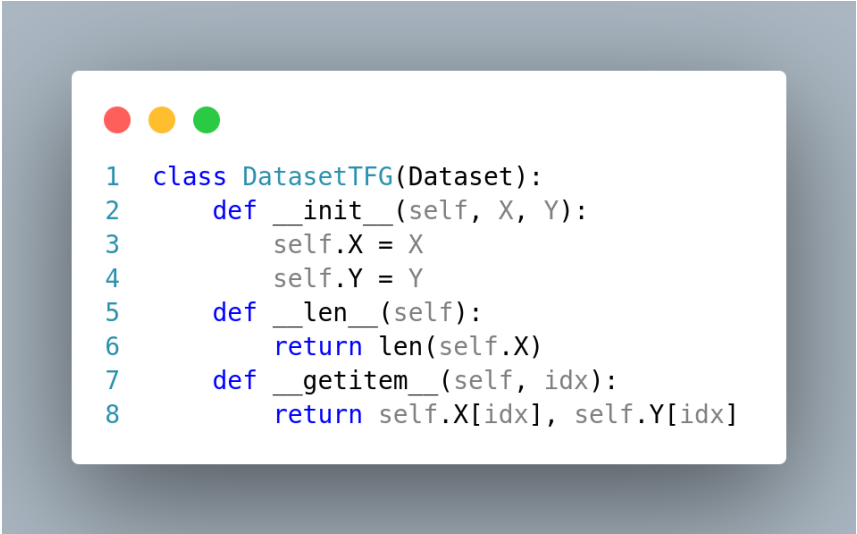
- **Batch size:** Teniendo en cuenta el tamaño de los datos que se utilizan para la fase de entrenamiento del modelo binario, se han considerado que los valores que se muestran en la Tabla 6.4, eran apropiados y suficientemente dispares como para notar diferencias significativas en los resultados de la ejecución de los experimentos.
- **Learning rate:** Siguiendo los avisos de la sección anterior en la que se explicaba el propósito y características de cada hiperparámetro, se ha optado por elegir unos valores con los que se espera que el modelo converja sin sobreajustarse ni se exceda en el tiempo de obtención de una solución válida.
- **Épocas:** De forma similar a como sucede con la tasa de aprendizaje, un valor excesivamente bajo en el número de épocas puede provocar que el modelo no converja. Sin embargo, un número demasiado alto de épocas provoca sobreajuste en el modelo.

Hiperparámetro	Posibles valores
<i>Batch size</i>	[2000, 10000, 15000, 20000]
<i>Learning rate</i>	[10^{-2} , 10^{-3} , 10^{-4}]
Épocas	[10, 20, 30]

Tabla 6.4: Valores de los hiperparámetros utilizados en los experimentos del modelo de clasificación binaria.

Una vez definidos los valores de los hiperparámetros con los que se probaran durante la búsqueda de la mejor configuración, se divide el conjunto de entrenamiento en los correspondientes *folds* que se utilizan en la validación cruzada, dentro del bucle que prueba con todas

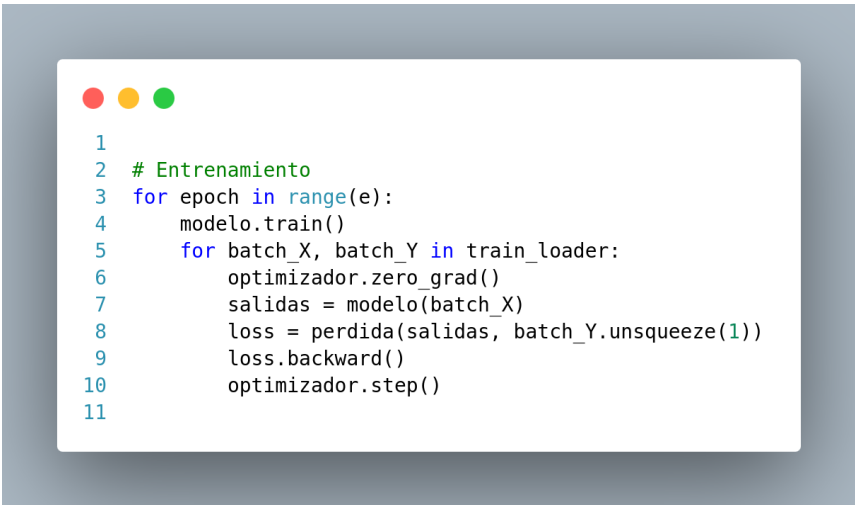
las configuraciones de hiperparámetros. Después, se inicializa **wandb** para registrar los resultados de cada experimento. A continuación, se crean los objetos de clase **DatasetTFG** que utilizarán los correspondientes **DataLoaders** durante el entrenamiento y la validación de los modelos. En la Figura 6.12 se muestra el código utilizado para describir la clase **DatasetTFG**.



```
1 class DatasetTFG(Dataset):
2     def __init__(self, X, Y):
3         self.X = X
4         self.Y = Y
5     def __len__(self):
6         return len(self.X)
7     def __getitem__(self, idx):
8         return self.X[idx], self.Y[idx]
```

Figura 6.12: Definición de la clase **DatasetTFG**.

Una vez inicializados todos los objetos necesarios se entrena al modelo dentro del bucle de entrenamiento cuyo número de iteraciones depende del hiperparámetro *Epochs*, tal y como se puede observar en la Figura 6.13. Para comprobar los resultados del entrenamiento del modelo, se ejecuta el bucle de validación del modelo entrenado para obtener las métricas que se almacenan en la herramienta **wandb**.



```
1
2 # Entrenamiento
3 for epoch in range(e):
4     modelo.train()
5     for batch_X, batch_Y in train_loader:
6         optimizador.zero_grad()
7         salidas = modelo(batch_X)
8         loss = perdida(salidas, batch_Y.unsqueeze(1))
9         loss.backward()
10        optimizador.step()
11
```

Figura 6.13: Bucle de entrenamiento del modelo de clasificación binaria.

6.6. Implementación del modelo neuronal de clasificación multiclase

Esta sección aborda los objetivos y procedimientos aplicados en el desarrollo del modelo de clasificación multiclase. Se describe el proceso llevado a cabo para la selección de hiperparámetros, con el propósito de identificar la configuración que maximiza el desempeño del modelo.

6.6.1. Proposito del modelo de clasificación multiclase

El modelo de clasificación multiclase se ha diseñado con el objetivo de identificar de manera precisa y eficiente, a que tipo de ataque corresponde una conexión maliciosa que el modelo de clasificación binaria ha clasificado previamente como intrusiva. Se trata de un modelo de clasificación jerárquica, ya que el problema se divide en varios niveles o capas (binaria y multiclase) y a medida que avanzamos por la jerarquía, las decisiones se vuelven más específica. La función principal de este modelo de clasificación multiclase consiste en distinguir entre 9 clases diferentes de ataques que se detallan en la Sección [5.2 Tipos de ataques registrados en los datos de NF-UNSW- NB15-v3](#), contribuyendo a la protección proactiva de los sistemas informáticos.

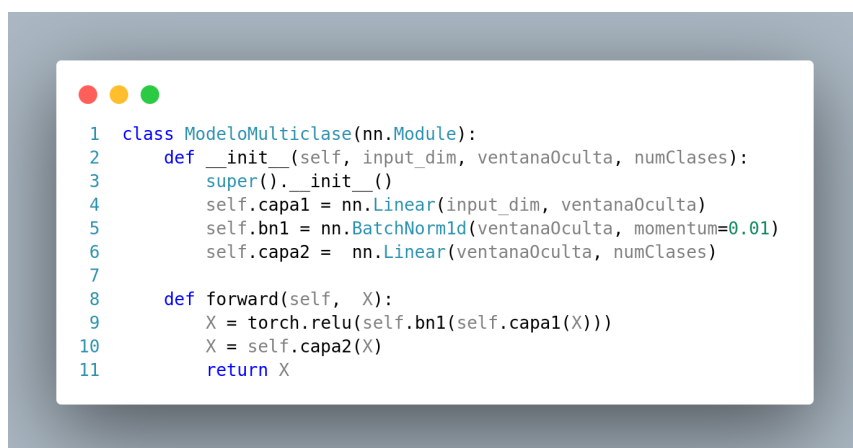
6.6.2. Diseño del modelo e hiperparámetros seleccionados

Para diseñar la arquitectura del modelo, se ha desarrollado la clase `ModeloMulticlase`. Esta clase está conformada por dos métodos, la primera corresponde con la inicialización de la clase, y la segunda con la arquitectura utilizada para este modelo.

Como se puede observar en la Figura 6.14, el modelo está compuesto por dos capas. La `capa1` tiene un número de entradas igual al número de atributos que poseen los datos utilizados para entrenar al modelo y un número de salidas que depende del parámetro *ventanaOculta*, con el que se llama a la función de inicialización del modelo. El número de salidas de la `capa1` será precisamente el que determine la arquitectura específica que se utiliza. Una vez que los datos pasan por la `capa1`, se utiliza la técnica *Batch Normalization*.

Como se explica anteriormente, las redes neuronales necesitan funciones no lineales para aprender relaciones complejas entre las entradas y salidas, por este motivo, se aplica la función de activación ReLU después del *Batch Normalization*. Sin activaciones como ReLU, toda la red sería equivalente a una simple transformación lineal, sin capacidad de aprendizaje automático. Que el modelo presente una alta capacidad de aprendizaje automático es esencial para los modelos de clasificación multiclase cuya complejidad es mucho mayor que la de los modelos de clasificación binaria. Para finalizar, la `capa2` recibe como entrada las salidas de la `capa1` y al tratarse de la capa final de un modelo de clasificación multiclase, esta capa tendrá una salida por cada clase que es capaz de identificar el modelo.

Por lo general, en clasificación multiclase, la salida del modelo suele ser un vector de *logits* (valores sin escalar), uno por cada clase. La función de activación Softmax transforma esos *logits* en probabilidades, es decir, cada valor representa lo probable que es que los datos que han entrado al modelo, pertenezca a esa clase. Debido a que durante el entrenamiento, se utiliza la biblioteca PyTorch, no es necesario aplicar Softmax a la salida del modelo cuando se utiliza la función de pérdida `CrossEntropyLoss`. La razón es que función `CrossEntropyLoss`, ya incluye `Softmax` internamente por razones de eficiencia y estabilidad numérica.



```
1 class ModeloMulticlase(nn.Module):
2     def __init__(self, input_dim, ventanaOculta, numClases):
3         super().__init__()
4         self.capa1 = nn.Linear(input_dim, ventanaOculta)
5         self.bn1 = nn.BatchNorm1d(ventanaOculta, momentum=0.01)
6         self.capa2 = nn.Linear(ventanaOculta, numClases)
7
8     def forward(self, X):
9         X = torch.relu(self.bn1(self.capa1(X)))
10        X = self.capa2(X)
11        return X
```

Figura 6.14: Estructura del modelo de clasificación multiclase diseñada.

6.6. IMPLEMENTACIÓN DEL MODELO NEURONAL DE CLASIFICACIÓN MULTICLASE

Al igual que en el caso de la clasificación binaria para evitar el sobreajuste del modelo y obtener valores de las predicciones más confiables que se utilizarán para la evaluación del rendimiento del modelo, se utiliza validación cruzada **k-fold** explicada en la Sección 6.1.4. Para entrenar el modelo de clasificación multiclase, se ha utilizado **StratifiedKFold** que como su nombre indica, estratifica los *folds* para que las proporciones de las clases de los datos utilizados para el entrenamiento se mantengan similares en cada *fold* y no se creen *folds* con un único tipo de clase. Estratificar los datos es especialmente crítico para aquellos datasets cuyos datos están muy desbalanceados como es el caso de los datos utilizados para entrenar los modelos de este proyecto. Concretamente, tras el tratamiento de los datos, existen 284 veces más muestras de la clase con mayor presencia en el *dataset*, que de la clase con menor presencias. Para el entrenamiento del modelo de clasificación multiclase, se ha optado por utilizar validación cruzada **StratifiedKFold** con 5 *folds* o divisiones.

Como algoritmo de optimización para el entrenamiento del modelo de clasificación multiclase, se ha optado por utilizar *AdamW*. Como se comenta en la Sección 6.1.2 [Algoritmo de optimización](#), se ha demostrado que *AdamW* es más eficaz en la prevención de sobreajuste en redes neuronales que otros algoritmos de optimización.

Los hiperparámetros que han sido seleccionados para encontrar la mejor configuración para el modelo de clasificación multiclase diseñado son: el *batch size*, la tasa de aprendizaje (*learning rate*) y el número de épocas. Estos hiperparámetros resultan imprescindibles en el entrenamiento de modelos de clasificación multiclase, dado que influyen directamente en la eficiencia del proceso de optimización y en la calidad del modelo resultante.

Como se comenta en la Sección 6.1.6 [Parámetros e hiperparámetros](#), los hiperparámetros controlan el comportamiento del proceso de entrenamiento y afectan la capacidad del modelo para aprender patrones complejos de los datos. Teniendo en cuenta las características de cada hiperparámetro, se han escogido los siguientes valores para los experimentos del entrenamiento del modelo multiclase con el objetivo de encontrar la mejor combinación de ellos.

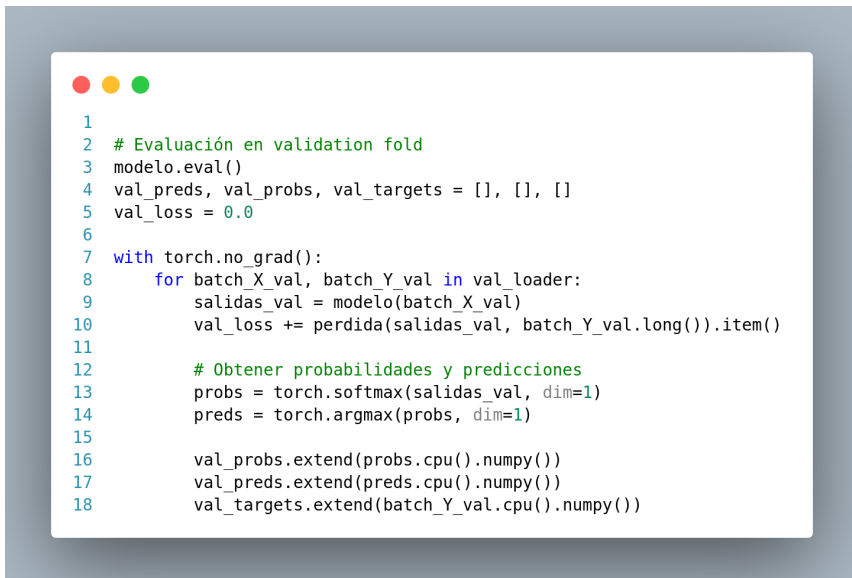
- **Batch size:** Teniendo en cuenta el tamaño de los datos que se utilizan para la fase de entrenamiento del modelo de clasificación multiclase, se han considerado que los valores que se muestran en la Tabla 6.5, eran apropiados y suficientemente dispares como para notar diferencias significativas en los resultados de la ejecución de los experimentos.
- **Learning rate:** Siguiendo los avisos de la sección anterior en la que se explicaba el propósito y características de cada hiperparámetro, se ha optado por elegir unos valores con los que se espera que el modelo converja sin sobreajustarse ni se exceda en el tiempo de obtención de una solución válida.
- **Épocas:** De forma similar a como sucede con la tasa de aprendizaje, un valor excesivamente bajo en el número de épocas puede provocar que el modelo no converja. Sin embargo, un número demasiado alto de épocas provoca sobreajuste en el modelo.

Hiperparámetro	Posibles valores
<i>Batch size</i>	[32, 64, 128, 256, 512]
<i>Learning rate</i>	[10^{-2} , 10^{-3} , 10^{-4} , 10^{-5}]
Épocas	[30, 50, 80, 100]

Tabla 6.5: Valores de los hiperparámetros utilizados en los experimentos del modelo de clasificación multiclase.

Una vez definidos los valores de los hiperparámetros que se evaluarán durante el proceso de búsqueda de la mejor configuración, se procede a dividir el conjunto de entrenamiento en los correspondientes *folds*, los cuales se emplean en la validación cruzada dentro del ciclo que recorre todas las combinaciones posibles de hiperparámetros. Posteriormente, se inicializa la herramienta **wandb** con el fin de registrar automáticamente los resultados de cada experimento ejecutado. A continuación, se instancian los objetos de la clase **DatasetTFG**, que serán utilizados por los respectivos **DataLoaders** tanto en la fase de entrenamiento como en la de validación del modelo multiclase.

Una vez inicializados todos los componentes necesarios, se puede entrenar el modelo dentro del ciclo correspondiente, cuyo número de iteraciones está determinado por el hiperparámetro *Epochs*. Para obtener las métricas que se almacenaran en la herramienta **wandb**, finalmente se ejecuta el bucle de validación con el *fold* correspondiente tal y como se muestra en la Figura 6.15.



```

1
2 # Evaluación en validation fold
3 modelo.eval()
4 val_preds, val_probs, val_targets = [], [], []
5 val_loss = 0.0
6
7 with torch.no_grad():
8     for batch_X_val, batch_Y_val in val_loader:
9         salidas_val = modelo(batch_X_val)
10        val_loss += perdida(salidas_val, batch_Y_val.long()).item()
11
12    # Obtener probabilidades y predicciones
13    probs = torch.softmax(salidas_val, dim=1)
14    preds = torch.argmax(probs, dim=1)
15
16    val_probs.extend(probs.cpu().numpy())
17    val_preds.extend(preds.cpu().numpy())
18    val_targets.extend(batch_Y_val.cpu().numpy())

```

Figura 6.15: Bucle de validación del modelo de clasificación multiclase.

La convergencia de los modelos de clasificación multiclase es mucho más complicada que en los modelos de clasificación binaria. Esto se debe a que en los modelos de clasificación binaria hay 4 posibles respuestas (VP, VN, FP, FN), mientras que en el modelo de clasificación multiclase que se propone en este proyecto hay 9 clases por 9 posibilidades de respuesta por clase, lo que suma un total de 81 posibles respuestas. Para encontrar una configuración de los hiperparámetros que se ajuste mejor a la complejidad de este problema, se han probado más combinaciones de valores de los hiperparámetros en el modelo de clasificación multiclase que en el modelo de clasificación binaria. El aumento del número de experimentos se ha realizado teniendo en cuenta que el número de muestras de conexiones maliciosas es mucho menor que el número total de conexiones que se ha utilizado para entrenar el modelo de clasificación binaria. Esto se traduce en que el tiempo de ejecución de los experimentos de ambos modelos ha sido similar compensando el número de muestras con el número de experimentos.

6.7. Selección de modelos de clasificación binaria

En esta sección se recogen los resultados de los experimentos realizados con los posibles valores de los hiperparámetros comentados en la Sección [6.5.2 Diseño del modelo e hiperparámetros seleccionados](#).

La plataforma *Weights & Biases* (wandb) se emplea para el seguimiento de experimentos, registro de hiperparámetros, métricas y artefactos de los modelos desarrollados en este trabajo. Esta herramienta ofrece una interfaz visual e interconectada que permite comparar ejecuciones de entrenamiento en proyectos de clasificación binaria y multiclase. También permite iniciar ejecuciones, almacenar la configuración del experimento, registrar métricas como *Recall* y *F1-weighted*. Además, permite gestionar versiones del modelo y datasets desde su API, lo cual mejora la transparencia, reproducibilidad y eficiencia en el desarrollo de sistemas de detección de conexiones malintencionadas [87].

Wandb se diseñó con el objetivo de ser la pizarra digital del patrón pizarra. El patrón arquitectónico pizarra (*blackboard*) se basa en un espacio de trabajo compartido que es accesible por componentes especializados o agentes, los cuales escriben y leen información de forma coordinada y colaborativa. Este enfoque es ideal para integrar etapas de preprocesamiento, inferencia y monitorización, donde cada módulo contribuye con su aporte al cruce de datos y resultados sin acoplamientos rígidos, favoreciendo la modularidad y la adaptabilidad ante cambios en el flujo de trabajo [88].

Para medir la eficiencia y eficacia del modelo, se han obtenido las métricas derivadas de la matriz de confusión comentadas en la Sección [6.2.2 Métricas útiles en clasificación binaria](#). El objetivo del modelo es detectar intrusiones en redes informáticas, por este motivo, los falsos negativos, es decir cuando se produce un ataque o intrusión y no se detecta, pueden tener consecuencias muy catastróficas para el sistema. Teniendo esto en cuenta, la métrica más importante es *Recall*, puesto que es la que se ve más afectada por la presencia de falsos negativos y por este motivo será la que se utilice para crear los *rankings* de esta sección .

Otras métricas relevantes que se muestran en las siguientes figuras son:

- **F1 score**
- **Precisión**
- **ROC AUC**

En las siguientes figuras se hace referencia a los posibles combinaciones de la matriz de confusión en inglés, es decir:

- **tp**: Verdaderos positivos (VP), intrusiones que se han clasificado como intrusiones.
- **tn**: Verdaderos negativos (VN), conexiones legítimas identificadas como tal.
- **fp**: Falsos positivos (FP), conexiones legítimas clasificadas como intrusiones.
- **fn**: Falsos negativos (FN), intrusiones que se han clasificado como conexiones legítimas.

En la Tabla 6.6, se encuentran representadas las cinco mejores configuraciones de hiperparámetros para el modelo de clasificación binaria con una arquitectura de 25 neuronas ($n/2$) en la capa oculta. Los resultados obtenidos superan lo esperado tras el análisis de la Sección 6.3.1 para esta arquitectura. La mejor configuración para la arquitectura con la mitad de neuronas en la capa oculta que de parámetros de entrada tiene el modelo, ha sido:

- **Batch size**: 15 000
- **Epochs**: 10
- **Learning rate**: 10^{-2}

Posicion_EXP	1º-MCB25	2º-MCB25	3º-MCB25	4º-MCB25	5º-MCB25
batch_size	15000	10000	20000	20000	2000
epochs	10	30	10	20	10
learning_rate	10^{-2}	10^{-3}	10^{-2}	10^{-2}	10^{-3}
avg_f1	0.997914	0.997717	0.997791	0.998022	0.997791
avg_fn	20.4	21.8	22	22.6	23
avg_fp	41	45.4	43	35.6	42
avg_precision	0.997216	0.996918	0.99708	0.997581	0.997147
avg_recall	0.998613	0.998517	0.998504	0.998463	0.998436
avg_roc_auc	0.999767	0.999779	0.999753	0.999776	0.999786
avg_tn	344123.2	344118.8	344121.2	344128.6	344122.2
avg_tp	14684.2	14682.8	14682.6	14682	14681.6

Tabla 6.6: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria con una capa oculta de 25 neuronas ($n/2$).

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el Modelo de Clasificación Binaria de 25 neuronas (MCB25), es la que se muestra en la Figura 6.16. Se observa que la cantidad de predicciones incorrectas es significativamente menor en comparación con las predicciones correctas. Además, el menor número de errores se da en el peor escenario, cuando el modelo clasifica una conexión maligna como benigna. Esto sugiere que el modelo ha logrado obtener buenos resultados.

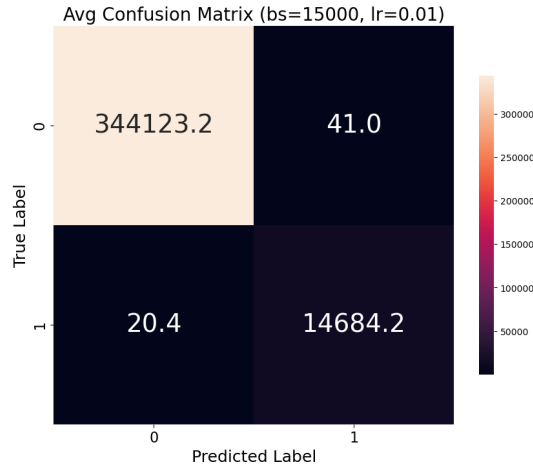


Figura 6.16: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el MCB25 durante la fase de búsqueda.

La Tabla 6.7, muestra la representación de las cinco mejores configuraciones de hiperparámetros para el modelo de clasificación binaria con una arquitectura de 49 neuronas (n) en la capa oculta. Los resultados medios obtenidos concuerdan con lo comentado en el análisis de la Sección 6.3.2 para esta arquitectura. La mejor configuración para la arquitectura con la el mismo número de neuronas en la capa oculta que de parámetros de entrada tiene el modelo, ha sido:

- *Batch size*: 20 000
- *Epochs*: 10
- *Learning rate*: 10^{-2}

Posicion_EXP	1º-MCB49	2º-MCB49	3º-MCB49	4º-MCB49	5º-MCB49
batch_size	20000	10000	15000	2000	2000
epochs	10	30	10	10	30
learning_rate	10^{-2}	10^{-3}	10^{-2}	10^{-3}	10^{-4}
avg_f1	0.997900	0.997907	0.998036	0.998022	0.997649
avg_fn	19.6	19.8	21	21.2	23.2
avg_fp	42.2	41.8	36.8	37	46
avg_precision	0.997135	0.997162	0.997500	0.997486	0.996877
avg_recall	0.998667	0.998653	0.998572	0.998558	0.998422
avg_roc_auc	0.999765	0.999780	0.999782	0.999769	0.999829
avg_tn	344123	344122.4	344127.4	344127.2	344118.2
avg_tp	14685	14684.8	14683.6	14683.4	14681.4

Tabla 6.7: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria con una capa oculta de 49 neuronas (n).

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el Modelo de Clasificación Binaria de 49 neuronas (MCB49), es la que se muestra en la Figura 6.17. Se puede notar que las predicciones incorrectas son considerablemente menos frecuentes que las correctas. Además, el menor número de errores ocurre en el caso más crítico, cuando el modelo clasifica incorrectamente una conexión maligna como benigna. Este comportamiento sugiere que el modelo ha obtenido un buen rendimiento.

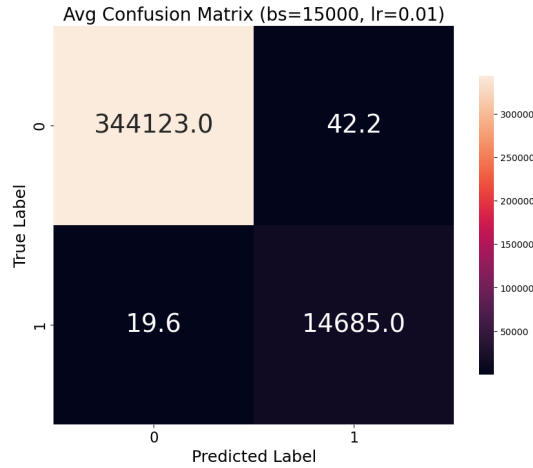


Figura 6.17: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el MCB49 durante la fase de búsqueda.

En la Tabla 6.8, se encuentran representadas las cinco mejores configuraciones de hiperparámetros para el modelo de clasificación binaria con una arquitectura de 98 neuronas ($2n$) en la capa oculta. Los resultados obtenidos coinciden con lo esperado tras el análisis de la Sección 6.3.3 para esta arquitectura. La mejor configuración para la arquitectura con el doble de neuronas en la capa oculta que de parámetros de entrada tiene el modelo, ha sido:

- **Batch size:** 20 000
- **Epochs:** 10
- **Learning rate:** 10^{-2}

Posicion_EXP	1º-MCB98	2º-MCB98	3º-MCB98	4º-MCB98	5º-MCB98
batch_size	20000	10000	15000	15000	10000
epochs	10	30	30	10	20
learning_rate	10^{-2}	10^{-3}	10^{-3}	10^{-2}	10^{-3}
avg_f1	0.998124	0.997941	0.997771	0.998015	0.997730
avg_fn	18.4	21.6	22	22.4	22.6
avg_fp	36.8	39	43.6	36	44.2
avg_precision	0.997501	0.997351	0.997039	0.997554	0.996999
avg_recall	0.998749	0.998531	0.998504	0.998477	0.998463
avg_roc_auc	0.999781	0.999777	0.999776	0.999773	0.999777
avg_tn	344127.4	344125.2	344120.6	344128.2	344120
avg_tp	14686.2	14683	14682.6	14682.2	14682

Tabla 6.8: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria con una capa oculta de 98 neuronas ($2n$).

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el Modelo de Clasificación Binaria de 98 neuronas (MCB98), es la que se muestra en la Figura 6.18. Las predicciones erróneas son mucho menos frecuentes que las correctas. Además, los errores son más escasos en el peor de los casos, cuando el modelo clasifica una conexión maligna como benigna. Esto indica que el modelo ha alcanzado un buen nivel de desempeño.

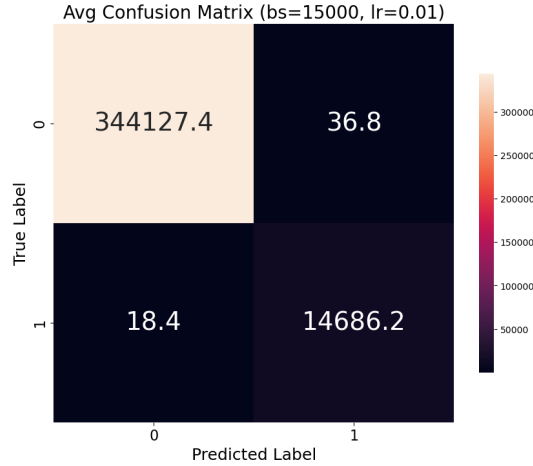


Figura 6.18: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el MCB98 durante la fase de búsqueda.

Estos resultados no son definitivos, puesto que no se está evaluando la posibilidad de que el modelo haya sufrido sobreajuste incluso utilizando validación cruzada. Para obtener resultados reales del desempeño del modelo, se realizan pruebas, también conocidas como test, en las que se obtienen las métricas del modelo al recibir datos que no ha visto nunca. Este proceso y sus resultados se comentan en el Capítulo 7 como parte de la fase de evaluación de CRISP-DM.

6.7.1. Comparación de las arquitecturas seleccionadas de los modelos de clasificación binaria

En esta sección se comparan los resultados obtenidos entre las tres arquitecturas propuestas. Una vez comparadas, se desarrolla una posible explicación para comprender los resultados de los experimentos.

Para comprender mejor los resultados globales de los experimentos, se han recogido en la Figura 6.19, las 5 configuraciones de hiperparámetros con mayor *Recall* independientemente de su arquitectura.

batch_size	epochs	hidden_size	learning_rate	avg_recall
20000	10	98	0.01	0.9987486972
20000	10	49	0.01	0.9986670886
10000	30	49	0.001	0.9986534868
15000	10	25	0.01	0.9986126862
15000	10	49	0.01	0.9985718792

Figura 6.19: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación binaria.

El problema para el que ha sido diseñado el modelo no presenta una relación lineal tras transformar los datos como se explica en el Capítulo 5 [Entendimiennto de los datos](#). Por este motivo, se esperaba que el modelo con un tamaño de capa oculta mayor fuese capaz de converger mejor, tal y como se comenta en la Sección 6.3.3.

Al observar los datos de la Figura 6.19, se puede apreciar que puntualmente, la mejor arquitectura ha sido la que tenía un tamaño de ventana oculta igual a 98. Cabe destacar que, la diferencia entre los valores de *recall* de los cinco mejores experimentos es muy pequeña y puede que se deba a factores externos, como el estado de la máquina en el momento en el que se entreno al modelo, o la inicialización aleatoria de los pesos. Por este motivo, los resultados pueden variar si se realizan los mismos experimentos.

Ente los mejores resultados de los experimentos, destaca la presencia de la arquitectura con un tamaño de capa oculta igual al número de parámetros de entrada del modelo. Tal y como se comenta en las secciones anteriores, esta es la arquitectura más equilibrada. Gracias a este equilibrio ha sido capaz de ajustar sus pesos mejor que el rseto de arquitecturas para configuraciones de hiperparámetros diferentes.

Al aumentar el número de épocas del entrenamiento de los modelos, estos tienden a sobreajustar los pesos. Este sobreajuste implica que el modelo obtiene unas métricas excepcionalmente positivas durante los experimentos. Sin embargo, aunque los valores de las métricas son muy positivos en los experimentos realizados, sorprende que el número de épocas del 80 % de las 5 mejores configuraciones, es el menor de los valores con los que se han realizado los experimentos para el hiperparámetro *epochs*. Estos resultados reflejan que la probabilidad de que el modelo haya sobreajustado sus pesos es muy baja. Posiblemente, el uso de la validación cruzada que penaliza el sobreajuste de los pesos de los modelos, como se explica en secciones anteriores, sea el responsable de que los experimentos con un número de épocas más elevado hayan obtenido unos resultados peores.

Finalmente, es destacable que el *learning rate* de la mayoría de los 5 mejores experimentos del global de las arquitecturas tenga el valor más alto de los que se han probado. Esto puede interpretarse de manera positiva o negativa. En el mejor de los casos, un *learning rate* alto muestra que los datos están bien estructurados y que el optimizador y las arquitecturas son robustos frente a valores altos de *learning rate*. En el peor de los casos, la convergencia se realiza rápidamente pero de manera inestable, los resultados dependan en gran medida de los

valores iniciales de los pesos o se este sobreajustando el modelo. Este último caso es el más improbable debido a las técnicas utilizadas que se han descrito y a los resultados analizados en el párrafo anterior.

Para conocer el verdadero desempeño del modelo y determinar si ha sufrido sobreajuste de los pesos durante el entrenamiento, se realiza un test con datos que no ha visto el modelo durante el entrenamiento. Estas pruebas y análisis se describen en el Capítulo 7 [Evaluación](#). Si los resultados obtenidos de estos test son similares a los resultados obtenidos de los experimentos, significa que el modelo converge correctamente hacia una solución real. En caso de que los resultados obtenidos de los test sean significativamente peores que los obtenidos durante los experimentos, el modelo habrá sobreajustado sus pesos.

6.8. Selección de modelos de clasificación multiclase

En esta sección se pueden observar los resultados de los experimentos realizados con los posibles valores de los hiperparámetros comentados en la Sección 6.6.2 [Diseño del modelo e hiperparámetros seleccionados](#).

Para medir la eficiencia y eficacia del modelo, se han obtenido las métricas derivadas de la matriz de confusión comentadas en la Sección 6.2.4 [Métricas útiles en clasificación multiclase](#). El objetivo de este modelo es clasificar intrusiones en redes, ya detectadas en diferentes tipos específicos de ataque enumerados en la Sección 5.2 para facilitar una respuesta adecuada. Aunque una clasificación incorrecta puede afectar la eficiencia de la mitigación, las consecuencias no son tan catastróficas como en el caso de falso negativo del modelo de clasificación binaria. Sin embargo, la precisión en la identificación del tipo de ataque sigue siendo clave para optimizar los recursos y la estrategia de defensa. Teniendo esto en cuenta, la métrica más importante es *Weighted F1 score*, ya que como se comenta en el apartado anterior, refleja el impacto relativo de cada clase en el conjunto de datos, resultando especialmente útil para comparar modelos entrenados con datasets desbalanceados.

Otras métricas relevantes que se muestran en las siguientes figuras son:

- **Macro F1 Score**
- **Macro Recall**
- **Macro Precision**

En la Tabla 6.9, se encuentran representadas las cinco mejores configuraciones de hiperparámetros para el modelo de clasificación multiclase con una arquitectura de 25 ($n/2$) neuronas (MCM25) en la capa oculta. Coinciden con lo esperado tras el análisis de la Sección 6.4.1 para esta arquitectura. La mejor configuración para la arquitectura con la mitad de neuronas en la capa oculta que de parámetros de entrada tiene el modelo, ha sido:

- **Batch size:** 64

- *Epochs*: 100
- *Learning rate*: 10^{-3}

Posicion_EXP	1º-MCM25	2º-MCM25	3º-MCM25	4º-MCM25	5º-MCM25
batch_size	64	512	128	256	128
epochs	100	100	80	80	50
learning_rate	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
avg_accuracy	0.541490	0.545803	0.543123	0.537111	0.536840
avg_f1_macro	0.376131	0.374716	0.376293	0.379411	0.368504
avg_f1_weighted	0.554636	0.552154	0.551329	0.550053	0.547048
avg_precision_macro	0.357461	0.371904	0.367818	0.364547	0.368754
avg_precision_weighted	0.652805	0.656511	0.659467	0.654297	0.656738
avg_recall_macro	0.550876	0.540895	0.551185	0.553603	0.535375
avg_recall_weighted	0.541490	0.545803	0.543123	0.537111	0.536840
avg_roc_auc_ovo	0.813081	0.795706	0.805615	0.801453	0.798093
avg_roc_auc_ovr	0.781124	0.774298	0.775449	0.780476	0.771334

Tabla 6.9: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación multiclase con una capa oculta de 25 neuronas.

Gracias a la matriz de confusión mostrada en la Figura 6.20, es posible visualizar fácilmente los resultados obtenidos durante la fase de búsqueda de las mejores configuraciones de los hiperparámetros para el MCM25, ofreciendo una visión aproximada del rendimiento alcanzado.

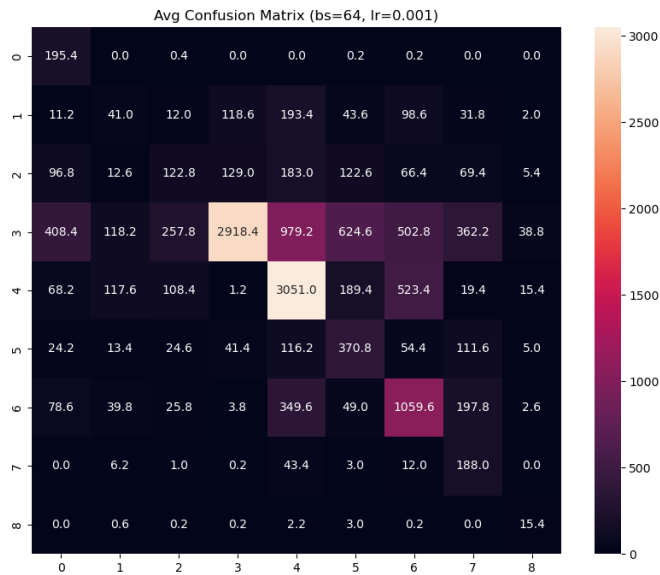


Figura 6.20: Matriz de confusión de la mejor configuración de hiperparámetros encontrada en la fase de búsqueda para el MCM25.

La Tabla 6.10, muestra la representación de las cinco mejores configuraciones de hiperparámetros para el modelo de clasificación multiclase con una arquitectura de 49 (n) neuronas (MCM49) en la capa oculta. Los resultados medios obtenidos son algo peores de lo comentado en el análisis de la Sección 6.4.2 para esta arquitectura. La mejor configuración para la arquitectura con la el mismo número de neuronas en la capa oculta que de parámetros de entrada tiene el modelo, ha sido:

- *Batch size*: 256
- *Epochs*: 50
- *Learning rate*: 10^{-3}

Posicion_EXP	1º-MCM49	2º-MCM49	3º-MCM49	4º-MCM49	5º-MCM49
batch_size	256	128	128	64	256
epochs	50	100	80	50	80
learning_rate	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
avg_accuracy	0.561552	0.552671	0.552875	0.552154	0.535329
avg_f1_macro	0.389355	0.393880	0.388328	0.382577	0.390220
avg_f1_weighted	0.569816	0.565581	0.565403	0.561813	0.558702
avg_precision_macro	0.369135	0.376647	0.375087	0.366740	0.378177
avg_precision_weighted	0.656464	0.670506	0.664273	0.658986	0.675658
avg_recall_macro	0.554809	0.562095	0.548626	0.550733	0.565014
avg_recall_weighted	0.561552	0.552671	0.552875	0.552154	0.535329
avg_roc_auc_ovo	0.800611	0.811428	0.803724	0.803414	0.809392
avg_roc_auc_ovr	0.780406	0.779326	0.776840	0.773186	0.782339

Tabla 6.10: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación multiclase con una capa oculta de 49 neuronas.

Gracias a la matriz de confusión mostrada en la Figura 6.21, es posible visualizar fácilmente los resultados obtenidos durante la fase de búsqueda de las mejores configuraciones de los hiperparámetros para el MCM49, ofreciendo una visión aproximada del rendimiento alcanzado.

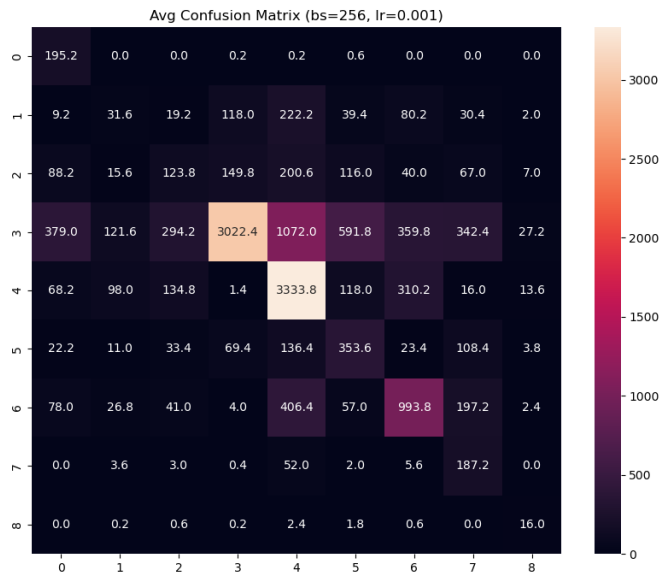


Figura 6.21: Matriz de confusión de la mejor configuración de hiperparámetros encontrada en la fase de búsqueda para el MCM49.

En la Tabla 6.11, se encuentran representadas las cinco mejores configuraciones de hiperparámetros para el modelo de clasificación multiclase con una arquitectura de 98 ($2n$) neuronas (MCM98) en la capa oculta. Los resultados obtenidos en esta arquitectura son los que más coinciden con lo explicado en el análisis de la Sección 6.4.3 para esta arquitectura. La mejor configuración para la arquitectura con el doble de neuronas en la capa oculta que de parámetros de entrada tiene el modelo, ha sido:

- *Batch size*: 256
- *Epochs*: 100
- *Learning rate*: 10^{-3}

Posicion_EXP	1º-MCM98	2º-MCM98	3º-MCM98	4º-MCM98	5º-MCM98
batch_size	256	256	512	64	256
epochs	100	80	100	80	50
learning_rate	10^{-3}	10^{-3}	10^{-2}	10^{-3}	10^{-3}
avg_accuracy	0.569414	0.556057	0.556915	0.556969	0.562083
avg_f1_macro	0.413180	0.406773	0.388808	0.398720	0.397308
avg_f1_weighted	0.583123	0.577553	0.574200	0.571020	0.569831
avg_precision_macro	0.394898	0.385322	0.372836	0.377543	0.387029
avg_precision_weighted	0.681971	0.675326	0.674534	0.669836	0.669688
avg_recall_macro	0.577069	0.564962	0.573083	0.566046	0.550391
avg_recall_weighted	0.569414	0.556057	0.556915	0.556969	0.562083
avg_roc_auc_ovo	0.813320	0.806782	0.809286	0.812789	0.800316
avg_roc_auc_ovr	0.788041	0.784984	0.781135	0.782758	0.777422

Tabla 6.11: Mejores cinco configuraciones de hiperparámetros del modelo de clasificación multiclase con una capa oculta de 98 neuronas.

Gracias a la matriz de confusión mostrada en la Figura 6.22, es posible visualizar fácilmente los resultados obtenidos durante la fase de búsqueda de las mejores configuraciones de los hiperparámetros para el MCM98, ofreciendo una visión aproximada del rendimiento alcanzado.

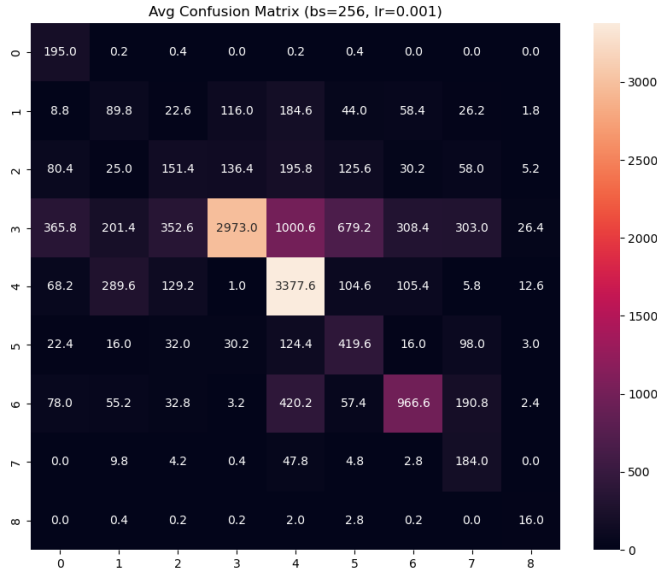


Figura 6.22: Matriz de confusión de la mejor configuración de hiperparámetros encontrada en la fase de búsqueda para el MCM98.

Los resultados obtenidos no son definitivos, puesto que no se está evaluando la posibilidad de que el modelo haya sufrido sobreajuste incluso utilizando validación cruzada. Para obtener resultados reales del desempeño del modelo, se realizan pruebas, también conocidas como test, en las que se obtienen las métricas del modelo al recibir datos que no ha visto nunca. Este proceso y sus resultados se comentan en el Capítulo 7.

6.8.1. Comparación de las arquitecturas seleccionadas de los modelos de clasificación multiclase

En esta sección se comparan los resultados obtenidos entre las tres arquitecturas propuestas. Una vez comparadas, se desarrolla una posible explicación para comprender los resultados de los experimentos.

Para comprender mejor los resultados globales de los experimentos, se han recogido en la Figura 6.23, las 10 configuraciones de hiperparámetros con mayor weighted f1 score independientemente de su arquitectura.

batch_size	epochs	hidden_size	learning_rate	avg_f1_weighted
256	100	98	0.001	0.5831225815
256	80	98	0.001	0.5775525405
512	100	98	0.01	0.5741997027
64	80	98	0.001	0.5710195986
256	50	98	0.001	0.5698308505
256	50	49	0.001	0.5698160035
128	80	98	0.001	0.5669125716
128	100	98	0.001	0.5657792633
128	100	49	0.001	0.5655814612
128	80	49	0.001	0.5654025284

Figura 6.23: Mejores diez configuraciones de hiperparámetros del modelo de clasificación multiclase.

Para poder comparar los resultados de los experimentos del modelo de clasificación multiclase entre varias arquitecturas, es necesario recoger más combinaciones de hiperparámetros independientemente de las arquitecturas que en la Sección 6.7.1. Esto denota una gran diferencia ente los resultados de los experimentos del modelo de clasificación multiclase y los resultados del modelo de clasificación binaria. Como se puede observar en la Figura 6.19, los resultados del modelo de clasificación binaria, demuestran que la arquitectura no supone una gran diferencia a la hora de que el modelo converja hacia una solución. En cambio, en los resultados de los experimentos del modelo de clasificación multiclase, la arquitectura con un tamaño de la capa oculta igual al doble del número de entradas del modelo ha sido la dominante.

Los resultados de la Figura 6.23 reflejan la complejidad del problema. La hegemonía que presenta la arquitectura con un mayor número de neuronas implica que la relación entre los datos es muy compleja.

La dificultad del problema que el modelo debe resolver reside en tres características del mismo:

1. **Clases muy desbalanceadas:** Al igual que en el mundo real, los datos utilizados para entrenar el modelo representan varios tipos de intrusiones informáticas a sistemas. Estos tipos de ataques no tienen los mismos objetivos, tal y como se comenta en el Capítulo 4 [Entendimiento del problema](#). Como la finalidad de cada tipo de ataque es diferente, el número de ataques de cada clase que se realizan depende de lo que los usuarios que los lanzan deseen conseguir. Esta característica del problema se ve reflejada en la proporción de muestras de cada clase que contiene el *dataset* utilizado. Para comprender la dimensión del problema, de las muestras utilizadas para el entrenamiento del modelo,

31 052 pertenecen a ataques de tipo *Exploits*, mientras que solo 109 de las muestras son de tipo *Worms*. Esto implica que el número de muestras que son *Exploits* es 284 veces mayor que el número de muestras de ataques *Worms*.

2. **Complejidad para relacionar los datos:** Como se comenta en varias de las secciones anteriores de este capítulo, las relaciones entre los datos que se utilizan para el entrenamiento del modelo no son lineales. La complejidad de los datos implica un mayor número de ajustes por retropropagación en cada época de entrenamiento. Esta característica además de aumentar el coste computacional del entrenamiento y el uso del modelo, requiere de poder ajustar los pesos de la red con mayor precisión que en modelos con una complejidad menor, como es el caso del modelo de clasificación binaria propuesto en este proyecto.
3. **Número de clases elevado:** Cuanto mayor es el número de clases entre las que debe diferenciar un modelo de clasificación multiclase, mayor número de muestras de cada clase son necesarias para entrenar el modelo y mayor número de situaciones de clasificación erróneas existe.

El modelo que se ha diseñado debe identificar nueve clases distintas de intrusiones pero debido a la complejidad de obtener datos representativos de cada intrusión, el número de muestras utilizado para entrenar al modelo es de aproximadamente 72 000. De ese número de muestras, 60 000 pertenecen solo a tres tipos de clases.

Si se toma de punto de partida la matriz de confusión del modelo de clasificación binaria, existen 4 posibles respuestas del modelo, de las cuales dos de ellas son correctas. Esto implica que, en el hipotético caso, de que el modelo simplemente diese respuestas aleatorias, este daría la respuesta correcta dos de cada cuatro veces, o lo que es lo mismo el 50 % de las ocasiones. En el caso del modelo de clasificación multiclase, la matriz de confusión tiene 81 posibilidades diferente. De estas posibilidades, solo las que pertenecen a la diagonal principal son respuestas correctas, lo que equivale a nueve respuestas correctas. Aplicando la misma lógica que se ha aplicado para el modelo de clasificación binaria, si el modelo de clasificación multiclase solo diese respuestas de manera aleatoria, acertaría en 9 de cada 81 ocasiones, lo que se traduce en una tasa de acierto del 11,1 %. Estas tasas de acierto corresponden con la métrica de precisión de cada modelo. Sin embargo, las métricas utilizadas para evaluar que combinación de hiperparámetros es mejor para cada modelo, se ven especialmente afectadas por los falsos negativos y por la clasificación errónea de clases minoritarias, correspondientemente.

Todas estas características comentadas provocan que la matriz de confusión con los datos de entrenamiento del modelo de clasificación multiclase sea la que se muestra en la Figura 6.22. La figura muestra que de las nueve clases que el modelo es capaz de identificar, solo tres de ellas se clasifican con exactitud en la mayoría de las ocasiones. Destaca el número de predicciones erróneas de las clases más minoritarias que han sido clasificadas como clase tres (*Exploits*). Esto se debe al desbalanceo presente en los datos comentado a lo largo de este capítulo.

Capítulo 7

Evaluación

Este capítulo se centra en la fase de evaluación de los modelos entrenados y correspondiente con la fase de evaluación de la metodología CRISP-DM. Se describe el conjunto de datos de prueba, el proceso de evaluación, la presentación de los resultados de las métricas y el análisis de las fortalezas y debilidades de los modelos.

7.1. Conjunto de datos de prueba

Para obtener el conjunto de datos para el entrenamiento de los modelos y el conjunto de datos de prueba, se ha de dividir el conjunto total de los datos, una vez que estos han sido preparados como siguiendo los pasos que se comentan en la Sección [5.5 Preparación de los datos](#).

Al dividir el conjunto de datos, es crucial aplicar estratificación. Como se menciona en capítulos previos, esta técnica implica segmentar los datos de forma que se mantenga la misma distribución de clases en cada subconjunto. Esto resulta especialmente útil cuando las clases están desbalanceadas, ya que garantiza que todos los subconjuntos cuenten con una representación proporcional de cada clase, lo que mejora la precisión del modelo y minimiza los sesgos. Dado el desbalanceo observado en los datos de este trabajo, es esencial realizar una estratificación en cada división de los datos para evitar que algunas particiones contengan únicamente datos de una clase.

La división del conjunto de datos suele ser de un 80 % para entrenamiento y un 20 % para evaluación, esta división se fundamenta en la necesidad de proporcionar suficiente cantidad de datos para que el modelo aprenda de manera efectiva, mientras se mantiene un conjunto de datos no utilizado en el entrenamiento para evaluar su rendimiento generalizando. La proporción del 80 % se considera adecuada para capturar patrones significativos en el entrenamiento sin comprometer la capacidad de evaluar el modelo en datos no vistos, lo que permite estimar su desempeño en situaciones realistas. Esta práctica también ayuda a evitar

el sobreajuste, garantizando que los resultados del modelo no dependa en exclusiva de los datos utilizados en el entrenamiento [41].

Para realizar la división, se utiliza la función `train_test_split` de la biblioteca `sklearn.model_selection`. Esta función recibe el conjunto de los atributos y de las etiquetas de todo el *dataset* utilizado, una vez que sus datos han sido preparados. Además, esta función recibe el porcentaje de los datos que se desea reservar para la evaluación o la fase de test, como se comenta en el párrafo anterior, en este trabajo el 0,2 sobre 1 de los datos se destinan a evaluar los modelos entrenados. Finalmente, se especifica una semilla para que los datos se organicen de manera aleatoria y se especifica que se desea estratificar siguiendo el conjunto de las etiquetas.

Una vez divididos los datos, se normalizan los datos de evaluación, se convierten a tensores el conjunto de los atributos de pruebas y el conjunto de las etiquetas y se crea con estos tensores la instancia de la clase `DatasetTFG` que se utilizará para evaluar los modelos.

7.2. Proceso de evaluación

En el proceso de búsqueda de las configuraciones de hiperparámetros de los modelos con las que estos convergen mejor hacia una solución más generalizada, se ha utilizado la validación cruzada como se comenta en la Sección 6.1.5. La validación cruzada se utiliza para encontrar los mejores hiperparámetros de un modelo debido a su capacidad para proporcionar una evaluación más robusta y confiable del rendimiento del modelo. Este método divide el conjunto de datos en varios subconjuntos, realizando múltiples entrenamientos y evaluaciones, lo que permite reducir el riesgo de sobreajuste a un subconjunto específico. Además, la validación cruzada maximiza el uso de los datos disponibles, lo que resulta especialmente útil cuando los datos son limitados.

Una vez que se han encontrado los hiperparámetros óptimos, se utiliza todo el conjunto de datos de entrenamiento, es decir, el 80 % del conjunto de datos para entrenar los modelos finales. Esto se debe a que, al haber ajustado previamente los hiperparámetros, se considera que los modelos se encuentran en sus configuraciones más eficientes, lo cual maximiza sus capacidades de generalización al aprender de la mayor cantidad de datos posible. De este modo, se obtienen unos modelos más robustos antes de realizar la evaluación final sobre un conjunto de prueba independiente que no ha sido utilizado en el proceso de entrenamiento [51].

El proceso de evaluación consiste en proporcionar a los modelos entrenados con las configuraciones comentadas en los párrafos anteriores, los datos de evaluación que no han visto durante la fase de entrenamiento. Durante este proceso se recogen los resultados que se obtienen en función de la matriz de confusión correspondiente a cada modelo para posteriormente obtener las métricas con las que comparar el desempeño de los modelos.

7.3. Resultados de las métricas en el proceso de evaluación

Para comparar los resultados obtenidos en la fase de evaluación de las cinco mejores configuraciones de hiperparámetros de cada arquitectura de cada modelo, se utilizan las mismas métricas comentadas en las Secciones [6.2.2 Métricas útiles en clasificación binaria](#) y [6.2.4 Métricas útiles en clasificación multiclase](#) correspondientemente.

Con el objetivo de que la interpretación de los datos sea más sencilla, se ha añadido a las siguientes figuras una columna con la posición del *ranking* que ocupó cada configuración en la fase anterior.

7.3.1. Resultados del modelo de clasificación binaria

La métrica utilizada para establecer el orden en los siguientes *rankings* de configuraciones para los modelos de clasificación binaria es la misma empleada durante la fase previa de búsqueda de las mejores configuraciones de hiperparámetros: el *Recall*.

MCB25: Modelo de clasificación binaria con un tamaño de capa oculta igual a la mitad del número de atributos de entrada que recibe el modelo

En la Tabla [7.1](#), se muestran los resultados obtenidos durante las evaluaciones de las cinco configuraciones de hiperparámetros con mejor desempeño durante la fase de búsqueda mencionados en la Figura [6.6](#). Corresponden con las mejores configuraciones para la arquitectura del modelo de clasificación binaria que posee un tamaño de capa oculta de 25 neuronas.

Posicion_EXP	3º-MCB25	2º-MCB25	1º-MCB25	5º-MCB25	4º-MCB25
batch_size	20000	10000	15000	2000	20000
learning_rate	10^{-2}	10^{-3}	10^{-2}	10^{-3}	10^{-2}
epochs	10	30	10	10	20
test_recall	0.998749	0.998640	0.998586	0.998422	0.998368
test_accuracy	0.999831	0.999813	0.999824	0.999826	0.999826
test_f1	0.997934	0.997717	0.997853	0.997879	0.997879
test_f2	0.998423	0.998271	0.998292	0.998205	0.998172
test_precision	0.997121	0.996796	0.997121	0.997337	0.997391
test_roc_auc	0.999765	0.999772	0.999841	0.999818	0.999899
test_specificity	0.999877	0.999863	0.999877	0.999886	0.999888

Tabla 7.1: Resultados de la evaluación del modelo de clasificación binaria con $n/2$ neuronas en la capa oculta siendo n el número de atributos de entrada.

Los valores de la métrica *Recall* obtenidos en las evaluaciones del modelo MCB25, son muy altos para todas las configuraciones y la diferencia entre sus valores es del orden de 10^{-4} . Estas diferencias pueden deberse a circunstancias no controlables como el estado en el que se encontraba la máquina cuando se realizó la evaluación de los modelos o los pesos aleatorios iniciales que se eligieron en esa ejecución de los test.

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 3º MCB25 durante su evaluación es la que se muestra en la Figura 7.1.

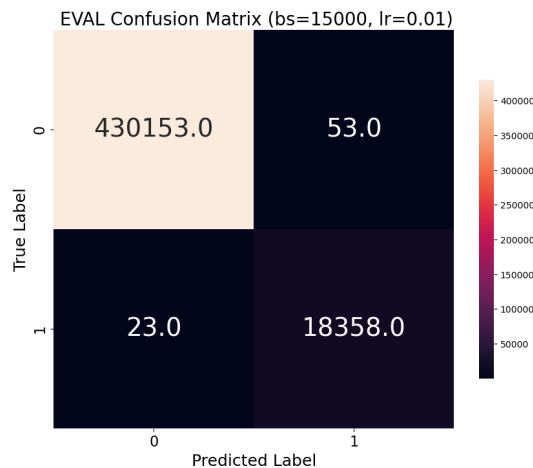


Figura 7.1: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 3º MCB25 durante la fase de evaluación.

MCB49: Modelo de clasificación binaria con un tamaño de capa oculta igual al número de atributos de entrada que recibe el modelo

En la Tabla 7.2 se presentan los resultados obtenidos durante las evaluaciones de las cinco configuraciones de hiperparámetros con mejor desempeño, seleccionadas en la fase de búsqueda en la Figura 6.7 para la arquitectura del modelo de clasificación binaria que cuenta con una capa oculta de 49 neuronas.

Posicion_EXP	2º-MCB49	1º-MCB49	5º-MCB49	3º-MCB49	4º-MCB49
batch_size	10000	20000	2000	15000	2000
learning_rate	10^{-3}	10^{-2}	10^{-4}	10^{-2}	10^{-3}
epochs	30	10	30	10	10
test_recall	0.998640	0.998640	0.998477	0.998477	0.998259
test_accuracy	0.999826	0.999835	0.999799	0.999835	0.999824
test_f1	0.997880	0.997988	0.997554	0.997988	0.997852
test_f2	0.998336	0.998379	0.998107	0.998281	0.998096
test_precision	0.997121	0.997338	0.996633	0.997500	0.997445
test_roc_auc	0.999807	0.999843	0.999809	0.999898	0.999863
test_specificity	0.999877	0.999886	0.999856	0.999893	0.999891

Tabla 7.2: Resultados de la evaluación del modelo de clasificación binaria con n neuronas en la capa oculta siendo n el número de atributos de entrada.

Los valores de la métrica *Recall* obtenidos durante las evaluaciones del modelo MCB49 son consistentemente altos para todas las configuraciones, y las diferencias entre ellos se encuentran en el orden de magnitud de 10^{-4} . Estas variaciones podrían atribuirse a factores no controlables, como el estado del sistema al momento de la evaluación o la aleatoriedad en la inicialización de los pesos durante la ejecución de las pruebas.

La matriz de confusión correspondiente a la mejor configuración de hiperparámetros obtenida en el 2º MCB49 durante su evaluación se presenta en la Figura 7.2.

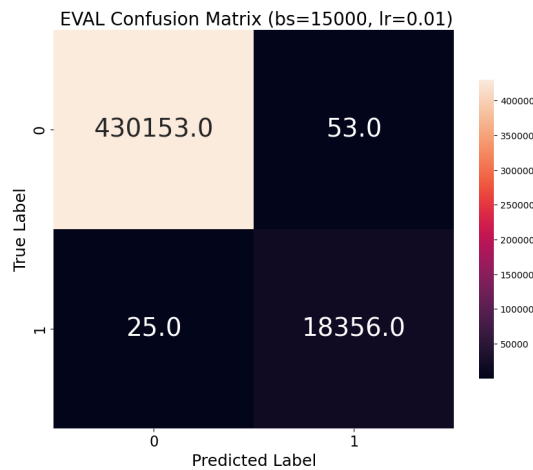


Figura 7.2: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 2º MCB49 durante la fase de evaluación.

MCB98: Modelo de clasificación binaria con un tamaño de capa oculta igual al doble del número de atributos de entrada que recibe el modelo

La Tabla 7.3 muestra los resultados obtenidos en las evaluaciones correspondientes a las cinco configuraciones de hiperparámetros que presentaron el mejor desempeño durante la fase de búsqueda mostradas en al Figura 6.8, aplicadas a la arquitectura del modelo de clasificación binaria con una capa oculta de 98 neuronas.

Posicion_EXP	3º-MCB98	5º-MCB98	1º-MCB98	4º-MCB98	2º-MCB98
batch_size	15000	10000	20000	15000	10000
learning_rate	10^{-3}	10^{-3}	10^{-2}	10^{-2}	10^{-3}
epochs	30	20	10	10	30
test_recall	0.998749	0.998749	0.998694	0.998477	0.998096
test_accuracy	0.999828	0.999822	0.999842	0.999833	0.999817
test_f1	0.997907	0.997826	0.998070	0.997961	0.997770
test_f2	0.998412	0.998379	0.998444	0.998270	0.997966
test_precision	0.997067	0.996905	0.997446	0.997446	0.997445
test_roc_auc	0.999783	0.999770	0.999899	0.999899	0.999868
test_specificity	0.999874	0.999868	0.999891	0.999891	0.999891

Tabla 7.3: Resultados de la evaluación del modelo de clasificación binaria con $2n$ neuronas en la capa oculta siendo n el número de atributos de entrada.

En las evaluaciones del modelo MCB98, los valores obtenidos para la métrica *Recall* fueron elevados en todas las configuraciones, con diferencias del orden de 10^{-4} . Estas pequeñas variaciones pueden explicarse por factores no controlables, como el estado del sistema en el momento de la evaluación o la inicialización aleatoria de los pesos durante la ejecución de las pruebas.

La matriz de confusión correspondiente a la mejor configuración de hiperparámetros obtenida en el 3º MCB98 durante su evaluación se presenta en la Figura 7.3.

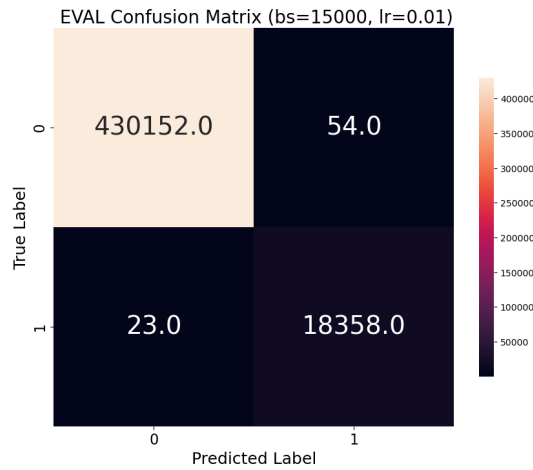


Figura 7.3: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 3º MCB98 durante la fase de evaluación.

7.3.2. Resultados del modelo de clasificación multiclase

Para determinar el orden de los *rankings* presentados a continuación, se emplea la misma métrica utilizada en la fase de búsqueda de configuraciones de hiperparámetros para los modelos de clasificación multiclase: el *F1-weighted*.

MCM25: Modelo de clasifcación multiclase con un tamaño de capa oculta igual a la mitad del número de atributos de entrada que recibe el modelo

En la Tabla 7.4 se presentan los resultados obtenidos durante las evaluaciones de las cinco configuraciones de hiperparámetros con mejores métricas, identificadas durante la fase de búsqueda de hiperparámetros para la arquitectura del modelo de clasificación multiclase que cuenta con una capa oculta de 25 neuronas.

7.3. RESULTADOS DE LAS MÉTRICAS EN EL PROCESO DE EVALUACIÓN

Posicion_EXP	3º-MCM25	2º-MCM25	1º-MCM25	4º-MCM25	5º-MCM25
batch_size	128	512	64	256	128
learning_rate	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
epochs	80	100	100	80	50
test_f1_weighted	0.538465	0.529706	0.491948	0.490342	0.435692
test_accuracy	0.519667	0.498341	0.463631	0.458354	0.406536
test_f1_macro	0.371639	0.340072	0.339593	0.333974	0.305804
test_precision_macro	0.415200	0.384902	0.376401	0.374790	0.354471
test_precision_weighted	0.681380	0.670107	0.642648	0.646698	0.640704
test_recall_macro	0.356826	0.325361	0.327492	0.320251	0.293147
test_recall_weighted	0.568975	0.553186	0.550136	0.550342	0.534025
test_roc_auc_ovo	0.878231	0.871589	0.871532	0.877010	0.870398
test_roc_auc_ovr	0.875697	0.868443	0.870934	0.874640	0.867136

Tabla 7.4: Resultados de la evaluación del modelo de clasificación multiclase con $n/2$ neuronas en la capa oculta siendo n el número de atributos de entrada.

La Tabla 7.4 anterior evidencia una diferencia en los valores de la métrica de comparación considerablemente mayor que la observada para las mismas configuraciones durante la fase de búsqueda de hiperparámetros. En los cinco mejores modelos representados en la figura, las diferencias en la métrica *F1-weighted* durante la fase de búsqueda se encuentran en el orden de 10^{-2} , mientras que en la fase de evaluación alcanzan un orden de 10^{-1} , lo que indica una discrepancias con respecto a los resultados observados en la búsqueda de hiperparámetros de la fase de modelado.

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 3º MCM25 durante su evaluación, es la que se muestra en la Figura 7.4.

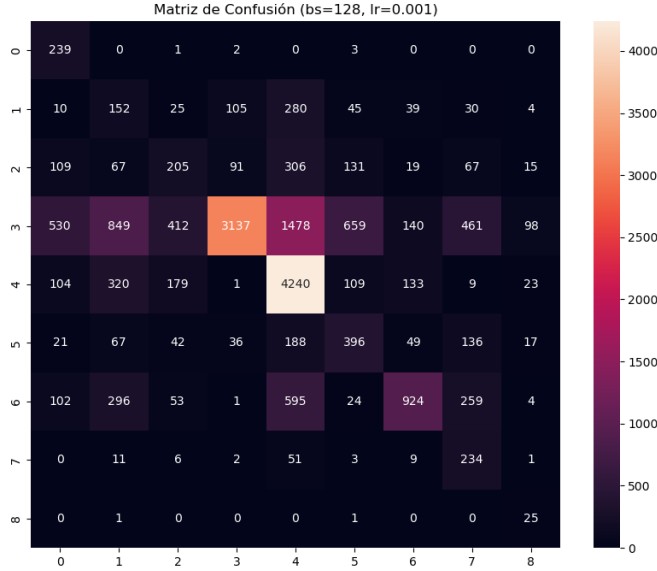


Figura 7.4: Matriz de confusión de la mejor configuración de hiperparámetros obtenida con en el 3º MCM25 durante la fase de evaluación.

Dado que los datos empleados para entrenar el modelo de clasificación multiclase presentan un elevado desbalance entre clases, se recurre a la funcionalidad de normalización de matrices de confusión de **wandb** para generar una representación visual más informativa que la mostrada en la Figura 7.4. Esta técnica permite observar de forma más clara el comportamiento del modelo en cada clase, independientemente de su frecuencia en los datos, tal y como se puede apreciar en la Figura 7.5. Al comparar ambas figuras, destaca la alto porcentaje de aciertos en la clase 0 y 8 en la Figura 7.5 y los resultados menos precisos para la clase 3.

La normalización en **wandb** convierte los conteos absolutos de la matriz de confusión en proporciones por clase real, lo que facilita la comparación del rendimiento del modelo entre clases desbalanceadas.

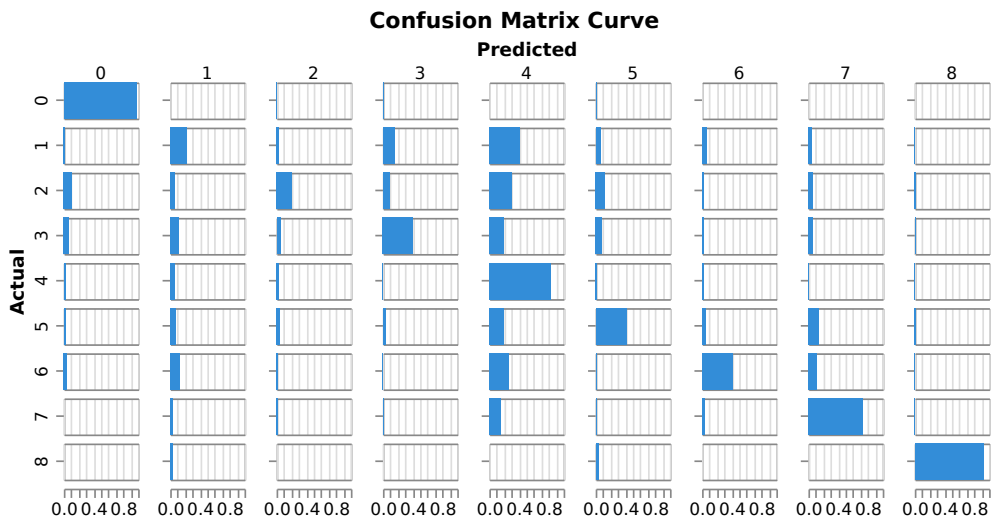


Figura 7.5: Matriz de confusión normalizada de la mejor configuración de hiperparámetros obtenida en el 3º MCM25 durante la fase de evaluación.

MCB49: Modelo de clasificación multiclase con un tamaño de capa oculta igual al número de atributos de entrada que recibe el modelo

La Tabla 7.5 presenta los resultados de las evaluaciones realizadas sobre las cinco configuraciones de hiperparámetros con mejores métricas, seleccionadas durante la fase de búsqueda aplicada a la arquitectura del modelo de clasificación multiclase con una capa oculta compuesta por 49 neuronas.

Posicion_EXP	5º-MCM49	4º-MCM49	3º-MCM49	2º-MCM49	1º-MCM49
batch_size	256	64	128	128	256
learning_rate	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
epochs	80	50	80	100	50
test_f1_weighted	0.516647	0.501065	0.498233	0.467576	0.436878
test_accuracy	0.486589	0.461617	0.452423	0.429247	0.416081
test_f1_macro	0.368803	0.325784	0.339188	0.316954	0.327882
test_precision_macro	0.348930	0.326530	0.338391	0.327875	0.331214
test_precision_weighted	0.657655	0.648645	0.671855	0.656385	0.651054
test_recall_macro	0.560934	0.555375	0.572878	0.552602	0.534408
test_recall_weighted	0.486589	0.461617	0.452423	0.429247	0.416081
test_roc_auc_ovo	0.883130	0.877907	0.887773	0.883284	0.875257
test_roc_auc_ovr	0.850867	0.848678	0.859088	0.847522	0.841436

Tabla 7.5: Resultados de la evaluación del modelo de clasificación multiclase con n neuronas en la capa oculta siendo n el número de atributos de entrada.

Al analizar los resultados obtenidos para esta arquitectura, destaca más allá del bajo rendimiento reflejado por la métrica *F1-weighted*, el hecho de que el orden de las configuraciones se ha invertido respecto a lo observado en la fase de búsqueda. Las posibles causas de este comportamiento se detallan en la Sección 7.4.

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 5º MCM49 durante su evaluación, es la que se muestra en la Figura 7.6.

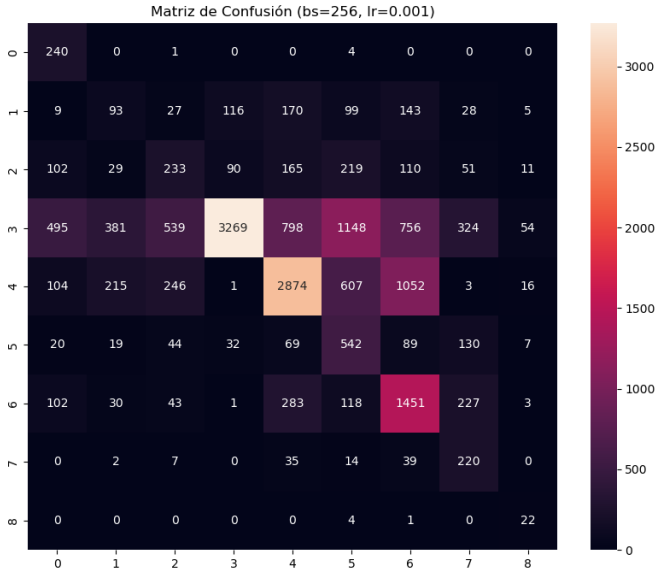


Figura 7.6: Matriz de confusión de la mejor configuración de hiperparámetros obtenida con en el 5º MCM49 durante la fase de evaluación.

Dado el elevado desbalance entre clases en los datos utilizados para entrenar el modelo de clasificación multiclase, se emplea la funcionalidad de normalización de matrices de confusión de **wandb** para generar una representación visual más informativa que la presentada en la Figura 7.6. Esta técnica facilita la observación más clara del comportamiento del modelo en cada clase, independientemente de su frecuencia en los datos, como se puede observar en la Figura 7.7. Comparando la Figura 7.5 con la figura de los datos normalizados del MCM49, se observa que como consecuencia del cambio de arquitectura, las prácticamente las predicciones para todas las clases han empeorado, a excepción de la clase 5 que ha obtenido mejores resultados que en el MCM25.

La normalización en **wandb** convierte los conteos absolutos de la matriz de confusión en proporciones por clase real, lo que facilita la comparación del rendimiento del modelo entre clases desbalanceadas.

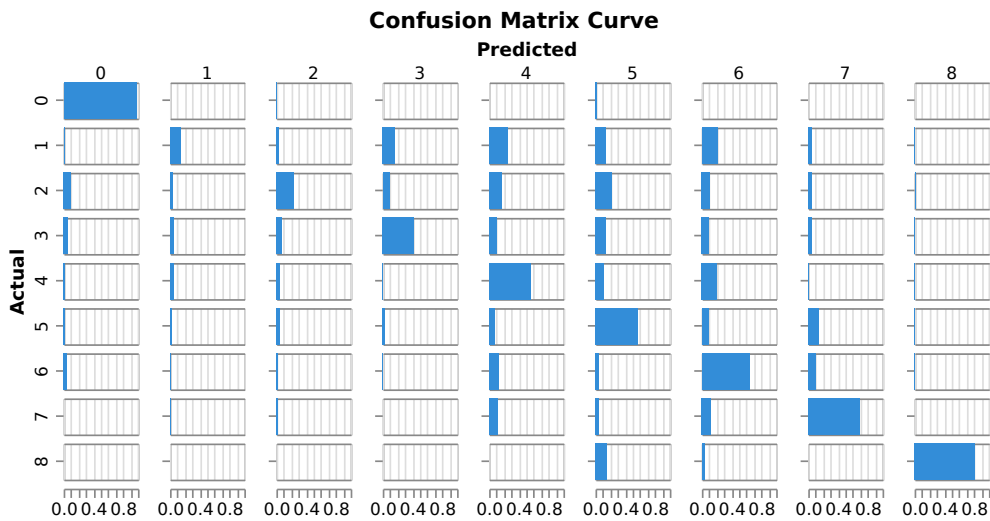


Figura 7.7: Matriz de confusión normalizada de la mejor configuración de hiperparámetros obtenida en el 5º MCM49 durante la fase de evaluación.

MCB98: Modelo de clasificación multiclase con un tamaño de capa oculta igual al doble del número de atributos de entrada que recibe el modelo

En la Tabla 7.6 se ilustran los resultados obtenidos durante las evaluaciones correspondientes a las cinco configuraciones de hiperparámetros con mejores métricas, seleccionadas en la etapa de búsqueda de configuraciones para la arquitectura del modelo de clasificación multiclase con una capa oculta de 98 neuronas..

Posicion_EXP	4º-MCM98	2º-MCM98	1º-MCM98	3º-MCM98	5º-MCM98
batch_size	64	256	256	512	256
learning_rate	10^{-3}	10^{-3}	10^{-3}	10^{-2}	10^{-3}
epochs	80	80	100	100	50
test_f1_weighted	0.568500	0.509088	0.506429	0.492392	0.465032
test_accuracy	0.545944	0.456177	0.447037	0.460094	0.424786
test_f1_macro	0.375845	0.345703	0.339473	0.359955	0.315588
test_precision_macro	0.365704	0.355874	0.368615	0.348133	0.324259
test_precision_weighted	0.663088	0.690381	0.700087	0.646679	0.658504
test_recall_macro	0.574354	0.583812	0.578126	0.573286	0.567694
test_recall_weighted	0.545944	0.456177	0.447037	0.460094	0.424786
test_roc_auc_ovo	0.888441	0.890550	0.888079	0.879829	0.884775
test_roc_auc_ovr	0.865108	0.857470	0.859871	0.858953	0.847066

Tabla 7.6: Resultados de la evaluación del modelo de clasificación multiclase con $2n$ neuronas en la capa oculta siendo n el número de atributos de entrada.

Los resultados obtenidos durante la fase de evaluación de esta arquitectura fueron significativamente inferiores a los alcanzados en la fase de búsqueda. Por ejemplo, en la quinta mejor configuración de hiperparámetros, la diferencia entre los resultados experimentales y los obtenidos en la evaluación es del orden de 10^{-1} . Esta diferencia implica una reducción del 10 % en la precisión, o bien, un mayor número de errores en las clases con menor cantidad de muestras en comparación con la fase anterior.

La matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 4º MCM98 durante su evaluación, es la que se muestra en la Figura 7.8.

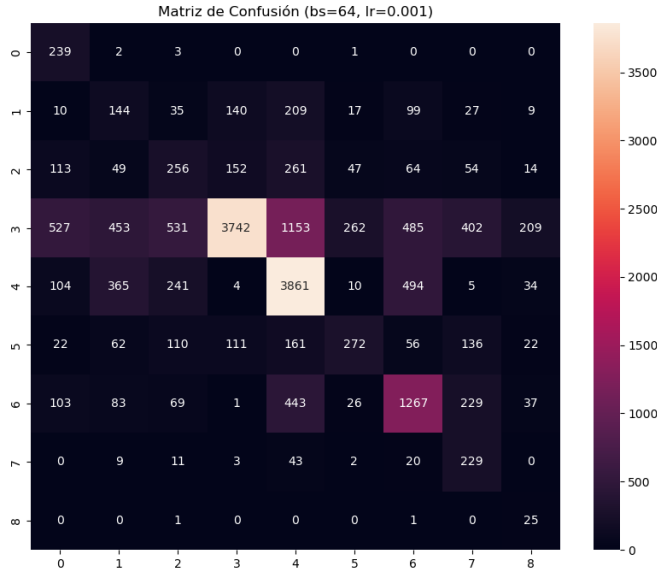


Figura 7.8: Matriz de confusión de la mejor configuración de hiperparámetros obtenida en el 4º MCM98 durante la fase de evaluación.

Debido al desbalanceo significativo entre las clases del conjunto de datos utilizados para entrenar el modelo de clasificación multiclase, se opta por utilizar la funcionalidad de normalización de matrices de confusión de **wandb**. Esta técnica permite generar una representación visual más clara que la mostrada en la Figura 7.8, facilitando así un análisis más preciso del comportamiento del modelo en cada clase, sin que este dependa de la frecuencia de las clases en los datos. Al comparar la Figura 7.9, que corresponde con la versión normalizada del MCM98, con las Figuras 7.7 y 7.5, se percibe una mejora generalizada en la predicción de todas las clases. La única excepción es la clase 5, donde la arquitectura del MCM98 presenta peores resultados en comparación con el MCM25.

La normalización en **wandb** transforma los conteos absolutos de la matriz de confusión en proporciones por clase real, lo que simplifica la comparación del rendimiento del modelo, especialmente en situaciones con clases muy desbalanceadas.

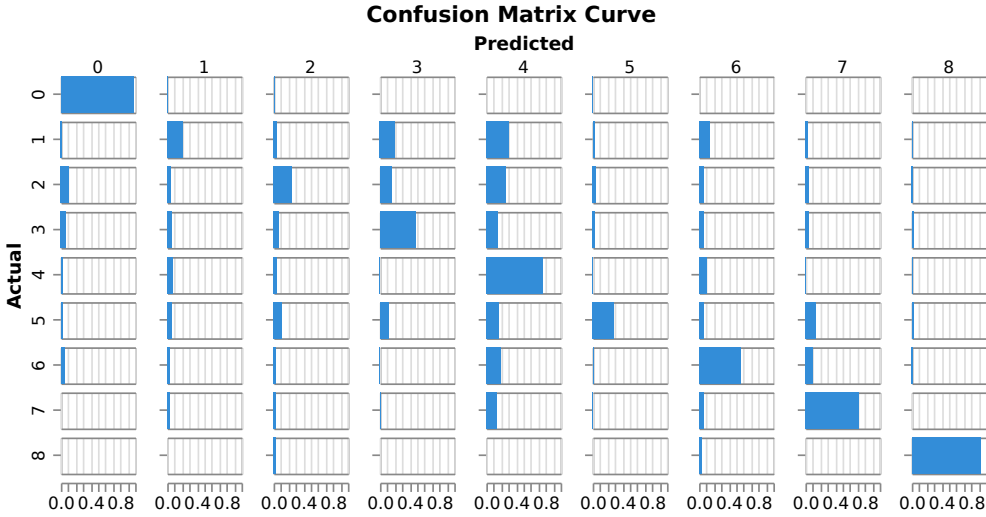


Figura 7.9: Matriz de confusión normalizada de la mejor configuración de hiperparámetros obtenida en el MCM98 durante la fase de evaluación.

7.4. Análisis de los resultados obtenidos

7.4.1. Modelo de clasificación binaria

Los resultados obtenidos durante la fase de búsqueda de las mejores configuraciones de hiperparámetros empleando validación cruzada *k-fold* con estratificación muestran valores de *Recall* consistentemente altos, todos por encima de 0.9985, como se detalla en la Figura 6.19. Este rendimiento indica una capacidad del modelo para identificar correctamente la clase mayoritaria de forma fiable y robusta en distintos subconjuntos de datos, lo cual respalda la estabilidad de las configuraciones seleccionadas durante esta fase de optimización [89].

Sin embargo, al aplicar estas mismas configuraciones en la fase de evaluación sobre un conjunto independiente (Figura 7.10), se observa una inversión parcial en el orden del rendimiento de las configuraciones. Algunas de las configuraciones con las mejores métricas en la búsqueda no mantienen su posición, mientras que otras ascienden en el *ranking*. Este comportamiento puede atribuirse a pequeñas variaciones aleatorias en los datos o a la sensibilidad del modelo frente a distribuciones ligeramente diferentes, aun cuando estas diferencias se hayan mitigado mediante estratificación [90].

A pesar de que las diferencias absolutas entre los valores de *Recall* en ambas fases son reducidas (del orden de 10^{-4} a 10^{-3}), su impacto puede ser relevante en tareas de clasificación binaria crítica, como la detección de conexiones maliciosas, donde incluso mínimas variaciones pueden traducirse en decisiones erróneas. El empeoramiento de algunas de las métricas puede deberse a factores como la inicialización aleatoria de pesos o el estado del sistema durante la inferencia, más que a un verdadero sobreajuste, dado que el proceso de validación cruzada

ya controla adecuadamente este riesgo [34].

Asimismo, el análisis muestra que distintas combinaciones de `batch_size`, `epochs`, `hidden_size` y `learning_rate` conducen a valores de *Recall* muy próximos en evaluación. Esta convergencia en el rendimiento es coherente con estudios que sugieren que múltiples configuraciones en regiones planas del espacio de pérdida pueden llevar a modelos igualmente eficaces, especialmente en arquitecturas con un número de neuronal elevado como las utilizadas [91].

La observación de que varias configuraciones producen valores prácticamente idénticos de *Recall* en la evaluación sugiere la presencia de una región de estabilidad en el espacio de soluciones. Sin embargo, el hecho de que estas configuraciones no coincidan con las mejor posicionadas en la búsqueda indica que, aunque la validación cruzada estratificada proporciona una estimación robusta, puede no capturar completamente la variabilidad inherente del conjunto de evaluación, especialmente si este presenta ligeras desviaciones de la distribución global [92].

Los resultados reflejan un comportamiento coherente con lo esperado para modelos que han sido correctamente validados. Las diferencias encontradas entre fases no parecen ser estadísticamente significativas, pero sí lo suficientemente marcadas como para sugerir la conveniencia de utilizar estrategias complementarias de evaluación en futuros experimentos, como pruebas de sensibilidad o análisis de varianza.

Posicion_EXP	batch_size	learning_rate	epochs	test_recall
3º- MCB98	15000	0.001	30	0.9987487079
3º-MCB25	20000	0.01	10	0.9987487079
5º-MCB98	10000	0.001	20	0.9987487079
1º-MCB98	20000	0.01	10	0.9986943039
2º-MCB25	10000	0.001	30	0.9986398999

Figura 7.10: Mejores cinco modelos en la fase de evaluación del modelo de clasificación binaria.

7.4.2. Modelo de clasificación multiclase

Los resultados obtenidos durante la fase de búsqueda de hiperparámetros del modelo de clasificación multiclase (MCM), reflejados en la Figura 6.23, indican valores moderados en las métricas utilizadas. En particular, el valor promedio de *F1-weighted* se sitúa en torno a 0.57, con ligeras variaciones entre las distintas configuraciones probadas. Este nivel de desempeño es coherente con la naturaleza desbalanceada del conjunto de datos, ya que el predominio de algunas clases afecta negativamente al promedio ponderado de precisión y *Recall* [81].

Las métricas *F1-macro* y *Precision-macro* presentan valores sustancialmente más bajos que sus contrapartes ponderadas, lo que confirma que el modelo no logra un rendimiento equilibrado entre clases minoritarias y mayoritarias. Esta brecha sugiere que las clases menos representadas no son detectadas con suficiente eficacia, un fenómeno frecuente en escenarios

de desbalance como el que se presenta en este trabajo, incluso al aplicar validación cruzada estratificada [93].

Al observar los resultados de la fase de evaluación (Figura 7.11), se constata que la métrica *F1-weighted* no mejora significativamente respecto a la fase de búsqueda. De hecho, se observan valores incluso más bajos en algunas configuraciones, con un máximo de aproximadamente 0.568 y un mínimo cercano a 0.509. Este comportamiento sugiere que las configuraciones seleccionadas no generalizan adecuadamente al conjunto completo, a pesar de haber sido validadas mediante *k-folds* [90].

El análisis también muestra que el orden de las configuraciones en términos de rendimiento cambia entre ambas fases. Configuraciones que ocupaban posiciones intermedias o bajas durante la búsqueda escalan posiciones en la evaluación y viceversa. Esto podría estar relacionado con la distribución específica de las clases en el conjunto completo usado para la evaluación, el cual puede no coincidir exactamente con la distribución estratificada de los pliegues empleados durante la validación por circunstancias no controlables [94].

La pérdida de rendimiento observada puede explicarse también por el hecho de que, al entrenar con el conjunto completo en la evaluación, se pierde la ventaja del promedio sobre múltiples particiones que ofrece la validación cruzada. Esta situación puede inducir mayor varianza en los resultados y resaltar la sensibilidad del modelo a configuraciones particulares del entrenamiento [95].

Por otra parte, el hecho de que el mejor resultado de *F1-weighted* en evaluación (0.568) sea comparable al mejor valor obtenido en la búsqueda sugiere que el modelo alcanza un rendimiento máximo limitado por la naturaleza del problema. La estructura altamente desbalanceada del conjunto impone un techo al rendimiento global del modelo [96].

A pesar de que el modelo 4^o-MCM98 presenta una precisión global moderada (`test_accuracy` $\approx 54.6\%$), sus valores de AUC, tanto en el enfoque *One-vs-Rest* (0.8651) como en *One-vs-One* (0.8884), indican una buena capacidad para distinguir entre las diferentes clases. Esto sugiere que el modelo, aunque no siempre acierta en la clase exacta, logra asignar puntuaciones de probabilidad adecuadas que reflejan correctamente la proximidad a la clase verdadera.

El modelo de clasificación multiclase presenta un rendimiento aceptable dadas las condiciones desbalanceadas del problema, pero revela claras limitaciones en la detección equilibrada de clases. Las discrepancias entre búsqueda y evaluación reflejan la dificultad del modelo para mantener un rendimiento robusto al exponerse a la totalidad del conjunto, y sugieren la necesidad de introducir nuevas estrategias no probadas en la Sección 6.4.4 Diseños del modelo de clasificación multiclase descartados orientadas al tratamiento explícito del desbalanceo de clases en futuras fases del desarrollo.

Posicion_EXP	batch_size	epochs	learning_rate	test_f1_weighted
4º-MCM98	64	80	0.001	0.5685004671
3º-MCM25	128	80	0.001	0.5384648868
2º-MCM25	512	100	0.001	0.5297063107
5º-MCM49	256	80	0.001	0.5166473814
2º-MCM98	256	80	0.001	0.5090883811

Figura 7.11: Mejores cinco modelos en la fase de evaluación del modelo de clasificación multiclase.

7.5. Comparación de los resultados obtenidos con trabajos similares

A continuación, se realiza una comparación entre los modelos propuestos en el presente trabajo y el desarrollado por *Sarhan et al.* [97], ambos aplicados a la detección de ataques en redes informáticas utilizando versiones del conjunto de datos NF-UNSW-NB15-v3.

- Para el modelo de clasificación binaria, el representante de este trabajo será el que mejores resultados ha obtenido durante la fase de evaluación, el 3º-MCB98.
- Para el modelo de clasificación multiclase, el representante de este trabajo será el que mejores resultados ha obtenido durante la fase de evaluación, el 4º-MCM98.

7.5.1. Comparación de los modelos de clasificación binaria

En la Tabla 7.7, el modelo 3º-MCB98 evidencia una mejora en todas las métricas comparado con el modelo de *Sarhan et al.*, destacando un aumento de 0,1479 en el *F1-Score* y más de cinco puntos porcentuales en el *AUC*. Estas mejoras pueden atribuirse al uso de técnicas avanzadas de entrenamiento, ajuste de hiperparámetros y optimización de características.

Modelo	<i>Accuracy</i>	<i>AUC</i>	<i>F1-Score</i>	<i>Recall</i>	<i>Precision</i>	<i>Specificity</i>
<i>Sarhan et al.</i>	98.62 %	0.9485	0.85	–	–	–
3º-MCB98	99.98 %	0.9998	0.9979	0.9987	0.9971	0.9999

Tabla 7.7: Comparación de modelos de clasificación binaria.

7.5.2. Comparación de los resultados por clase de los modelos de clasificación multiclase

A pesar de que el modelo 4^o-MCM98 no supera al de *Sarhan et al.* en clases como *Exploits* y *Generic*, presenta un desempeño significativamente superior en *Analysis* y *Worms*. Estas diferencias que se pueden observar en la Tabla 7.8 pueden estar relacionadas con el preprocesamiento, la selección de características y la estrategia de balanceo utilizada mencionadas en el Capítulo 6.

Clase	<i>F1-Score</i> (<i>Sarhan et al.</i>)	<i>F1-Score</i> (4 ^o -MCM98)
Analysis (0)	0.15	0.97
Backdoor (1)	0.17	0.20
DoS (2)	0.41	0.25
Exploits (3)	0.82	0.48
Fuzzers (4)	0.55	0.75
Generic (5)	0.66	0.28
Reconnaissance (6)	0.82	0.56
Shellcode (7)	0.75	0.72
Worms (8)	0.55	0.92

Tabla 7.8: Comparación de los valores obtenidos en *F1-Score* por clase.

7.5.3. Evaluación comparativa del rendimiento

Tras observar las diferencias entre los valores de las métricas obtenidas en ambos trabajos, se puede asumir que:

- El modelo 3^o-MCB98 supera de forma consistente al modelo propuesto por *Sarhan et al.* en todas las métricas evaluadas para la tarea de clasificación binaria.
- El modelo 4^o-MCM98 destaca en clases con un menor número de muestras en el conjunto de datos utilizado para el entrenamiento como *Analysis* y *Worms*, mientras que el modelo de *Sarhan et al.* muestra mayor estabilidad en clases con una mayor población de muestras. Esta diferencia puede deberse al uso de la función de pérdida ajustada utilizada en los modelos desarrollados en este proyecto, que penaliza con mayor severidad los errores en clases poco representadas, tal como se discute a lo largo del presente trabajo.

Los modelos desarrollados en este trabajo presentan mejoras notables en la clasificación binaria y un rendimiento competitivo en la clasificación multiclase. Aunque persisten desafíos en clases como *Generic* y *Exploits*, los resultados alcanzados en clases con relaciones de datos más complejas sugieren una mayor capacidad de generalización frente a los modelos publicados previamente.

7.5. COMPARACIÓN DE LOS RESULTADOS OBTENIDOS CON TRABAJOS SIMILARES

Capítulo 8

Despliegue

Esta sección aborda la fase de despliegue de los modelos de clasificación binaria y multiclase desarrollados para la detección de conexiones malignas. La fase de despliegue en la metodología CRISP-DM implica la integración de los modelos entrenados en entornos operativos donde sus predicciones contribuyen a la toma de decisiones en tiempo real o en análisis periódicos, facilitando la detección automatizada y escalable de amenazas en redes como se comenta en capítulos anteriores [98].

8.1. Integración en entornos operativos

Los modelos de clasificación binaria, orientados a distinguir conexiones malignas de benignas, y los modelos multiclase, diseñados para identificar diferentes categorías de conexiones malignas, deben ser implementados en plataformas que permitan la recepción continua de datos y procesamiento eficiente [99].

Para ello, es recomendable desplegarlos mediante APIs o microservicios, asegurando compatibilidad con sistemas de monitorización y respuesta automatizada. En el caso del modelo multiclase, la alta desproporción entre las clases y la naturaleza específica de los datos requieren especial atención en la validación en entorno real, dado que durante la fase de evaluación correspondiente con el Capítulo 7 se utiliza el conjunto completo de datos y puede observarse degradación en desempeño frente a la fase de búsqueda [100].

8.2. Consideraciones técnicas

El uso de validación cruzada estratificada durante la fase de búsqueda garantiza la estabilidad en la estimación de rendimiento, pero el despliegue debe contemplar la variabilidad

inherente a datos nuevos y no vistos, especialmente en contextos dinámicos como la detección de conexiones malignas [90].

El ajuste adecuado de hiperparámetros, demostrado durante el Capítulo 6, debe ser preservado en producción, considerando limitaciones de hardware, latencia y volumen de datos. El control de versiones de los modelos es crucial para realizar actualizaciones y retrocesos en sus configuraciones de pesos, sin afectar a los resultados que devuelva el modelo [101].

8.2.1. Patrones arquitectónicos recomendados: Pipeline y Observador

Para garantizar un despliegue robusto y adaptable de los modelos de clasificación en entornos operativos, lo recomendable es estructurar el sistema empleando patrones arquitectónicos que favorezcan la modularidad, la escalabilidad y la capacidad de respuesta ante eventos. Los patrones *Pipeline* o Filtro Tubería y Observador destacan como enfoques adecuados para satisfacer los requerimientos funcionales y no funcionales del sistema.

El patrón *Pipeline* permite organizar el procesamiento de datos en una secuencia de etapas independientes, donde cada etapa representa una transformación o acción específica sobre el flujo de datos. Este enfoque favorece la separación de responsabilidades, facilita el mantenimiento, y permite escalar individualmente cada componente. Es particularmente útil para el procesamiento continuo de datos en tiempo real, como en entornos de detección de conexiones malignas [102].

Por otro lado, el patrón Observador resulta útil para implementar una arquitectura reactiva, en la cual distintos componentes del sistema (por ejemplo, sistemas de alerta, interfaces de usuario, módulos de registro) pueden suscribirse a los eventos generados por el modelo como la detección de una amenaza. Este patrón facilita una integración flexible entre los modelos de predicción y otros servicios que requieren actuar en función de los resultados generados, sin acoplamiento directo entre ellos [103].

La combinación de ambos patrones permite un diseño desacoplado, extensible y más mantenible, características fundamentales para sistemas de monitorización y respuesta en entornos dinámicos y críticos como la ciberseguridad.

8.3. Monitorización y mantenimiento

Una vez desplegados, los modelos requieren monitorización continua para detectar posibles cambios en la distribución de los datos o en el comportamiento de las conexiones, que puedan afectar el rendimiento del modelo (*data drift* y *concept drift*) [100]. En particular, para el modelo multiclase que ha sido entrenado con clases muy desbalanceadas, la monitorización de métricas como el *F1-Weighted* es esencial para asegurar la calidad del diagnóstico en todas las categorías.

El mantenimiento de los modelos, incluye la recopilación de datos reales etiquetados, la reevaluación periódica del modelo y la posible reentrenamiento o ajuste de hiperparámetros para mantener la eficacia en la detección [104].

8.4. Impacto y aplicación en el mundo real

El despliegue efectivo de estos modelos contribuye a mejorar la seguridad en redes mediante la automatización de la detección de conexiones malinas, permitiendo respuestas rápidas y reducción de falsos positivos y negativos. La diferenciación entre múltiples tipos de conexiones malinas aporta un valor añadido para estrategias de mitigación específicas y optimización de recursos [105].

La Figura 8.1 muestra como podría implementarse de manera básica los modelos desarrollados en un entorno real de un sistema informático.

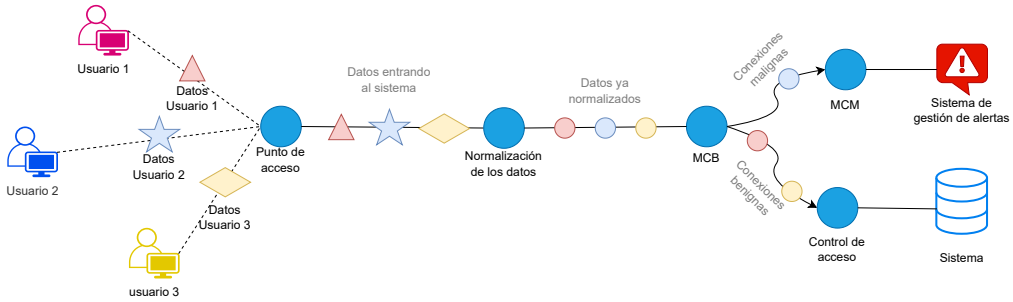


Figura 8.1: Ejemplo de despliegue de los modelos en un sistema informático.

Capítulo 9

Conclusiones

En este capítulo se exponen las conclusiones derivadas del desarrollo y evaluación del modelo neuronal diseñado para la clasificación de conexiones benignas y malignas. El enfoque propuesto consiste en un modelo de clasificación binaria que, una vez realiza la distinción entre conexiones benignas y malignas, redirige aquellas conexiones clasificadas como malignas hacia un modelo de clasificación multiclase. Este último modelo es capaz de diferenciar las conexiones malignas en nueve clases distintas, permitiendo una identificación más detallada y precisa de los diversos tipos de intrusiones. Los resultados obtenidos destacan la eficacia de ambos modelos, evidenciando su capacidad para abordar de manera jerárquica y especializada la clasificación. Finalmente, se presentan las limitaciones del estudio y se plantean posibles líneas de trabajo futuro.

9.1. Conclusiones

El primer objetivo del proyecto, consistente en diseñar e implementar un modelo capaz de detectar intrusiones en redes informáticas y proporcionar una clasificación preliminar de las mismas, se ha cubierto de forma satisfactoria. Este primer objetivo del proyecto se ha alcanzado al obtener un modelo de clasificación binaria capaz de distinguir entre conexiones benignas y malignas con un *Recall* superior a 0,99. Con respecto a la clasificación de intrusiones o conexiones malignas, se ha desarrollado un modelo de clasificación multiclase, que recibe las conexiones clasificadas por el modelo binario como malignas y clasifica estas conexiones en nueve clases diferentes de intrusiones en sistemas informáticos con un *F1-weighted* cercano a 0,6.

El segundo objetivo del proyecto, según el cual los modelos de detección debían ser modelos neuronales, se ha alcanzado satisfactoriamente. Este objetivo se logró mediante el diseño de modelos de clasificación binaria (MsCB) y multiclase (MsCM), ambos con una arquitectura compuesta por dos capas. La primera de las capas en todos los modelos corresponde con la capa oculta, es la capa que recibe los atributos de entrada del modelo y la que permite que

se pueda resolver el problema no lineal que se propone en este trabajo. Esta primera capa cuenta con un número diferente de neuronas en función de la arquitectura del modelo que se observe. La segunda capa por su parte, es la que corresponde con la salida del modelo. Esta segunda capa cuenta con una única neurona en el caso de los MsCB y con nueve neuronas en el caso de los MsCM. La utilización de un Perceptrón Multicapa (MLP) en ambos modelos de clasificación permite aprovechar la capacidad de las redes neuronales para aprender representaciones complejas y no lineales, mejorando la capacidad de clasificación

El tercer objetivo del proyecto, consistente en evaluar y comparar los modelos generados utilizando un conjunto de datos real y complejo, se ha alcanzado de manera adecuada. Tal y como se expone en el Capítulo 5, correspondiente a la fase de entendimiento de los datos de la metodología CRISP-DM, el conjunto de datos empleado para dicha evaluación ha sido el NF-UNSW-NB15-v3 [33]. Este conjunto de datos es semi-sintético, esto se debe a que los flujos benignos son registros reales de la conexión entre dos sistemas, mientras que los flujos correspondientes a los ataques se generaron en un entorno controlado. Para evitar que los datos sintéticos de los flujos maliciosos perjudicasen el entrenamiento y la evaluación de los modelos, se transformaron los datos, eliminando aquellos atributos que presentaban valores que diferían con la realidad, como fue el caso de las direcciones IP.

Evaluar el grado en el que se han alcanzado los objetivos académicos es más complejo que evaluar si los objetivos del proyecto se han conseguido cumplir. Esta diferencia se debe a que mientras que los objetivos del proyecto son medibles cuantitativamente, los objetivos académicos no son medibles objetivamente. Sin embargo, teniendo en cuenta el trabajo desarrollado se considera que si se han alcanzado todos los objetivos académicos fijados.

El primer objetivo académico, centrado en comprender el funcionamiento de los modelos neuronales mediante PyTorch y las métricas de evaluación, se ha abordado a lo largo del desarrollo del proyecto. Al haber desarrollado varios modelos con difertes arquitecturas y calcular las métricas tanto de la fase de validación como de la fase de evaluación utilizando la biblioteca PyTorch, se puede afirmar que se ha comprendido el funcionamiento de los modelos neuronales, alcanzando de esta manera el primer objetivo académico.

El segundo objetivo académico, consiste en aprender las características de varios de los tipo de modelos neruonales que existen, se ha alcanzado de manera satisfactoria. En la Sección 6.1.3 del Capítulo 6, se explican algunos de los modelos neuronales que más se utilizan en la actualidad y se comentan las caracterísitcas de cada tipo de modelo, proporcionando una visión de las tareas en las que mejor desempeño muestra cada tipo. Tras analizar las características de los diferentes tipos de modelos neuronales, se determinó que el Perceptrón Multicapa (MLP) era la opción más adecuada para este problema, dado que los datos presentan una naturaleza no lineal y el MLP tiene la capacidad de aprender representaciones complejas y no lineales, lo cual resulta fundamental para resolver este tipo de clasificación. Para redactar la sección mencionada fue necesario entender cuales eran las caraterísticas de cada tipo de modelo neuronal, por lo que el segundo objetivo académico se ha cumplido correctamente durante el desarrollo de este trabajo.

El tercer objetivo académico, se centra en descubrir el potencial de las redes neuronales para optimizar y mejorar las tecnologías de la información, incluyendo la ciberseguridad de los sistemas, se ha cubiertod de manera satisfactoria . Tras el desarrollo de este trabajo se ha

descubierto que los modelos neuronales tienen una gran capacidad para resolver problemas de clasificación con una complejidad elevada. Además, resulta fascinante el hecho de que los pesos de las neuronas se puedan ajustar en cualquier momento para converger hacia una solución más generalizada o precisa del problema. Esta capacidad de adaptación es esencial en las tecnologías de la información, y en especial en la ciberseguridad ya que es un campo que se encuentra en constante evolución. Teniendo esto en cuenta, no solo se ha descubierto el potencial de las redes neuronales para optimizar y mejorar las tecnologías de la información, incluyendo la ciberseguridad de los sistemas, si no que además se ha descubierto el potencial que tienen las redes neuronales para resolver problemas complejos y su alta capacidad de adaptación a cambios en el problema que resuelven.

9.2. Trabajo futuro

En un posible trabajo futuro se podría explorar como afecta el uso de las técnicas que se comentan en esta sección en la eficacia y en la eficiencia de los modelos desarrollados, con el objetivo de obtener modelos que convejan hacia una solución que generalice mejor, especialmente en el modelo de clasificación multiclase.

- Modificar la arquitectura del modelo de clasificación multiclase. Durante el desarrollo del trabajo se han comparado los resultados que se obtenían al modificar el tamaño de la capa oculta. Estas comparaciones han demostrado que en función de la complejidad del problema aumentar el número de neuronas puede ser beneficioso o contraproducente. También se ha probado a añadir una tercera capa en el modelo de clasificación multiclase para comprobar si aplicando esta técnica, el modelo conseguía compensar el desbalanceo en el número de muestras de algunas clases del conjunto de datos con la complejidad de la red neuronal. Desafortunadamente, el valor más alto obtenido en la métrica *F1-weighted* utilizando una arquitectura de tres capas, no superó los resultados obtenidos por los modelos con dos capas.

Sin embargo, al modificar la arquitectura del modelo de clasificación multiclase, sería posible lograr que el modelo converja hacia una solución que sea capaz de identificar con una mayor exactitud aquellas clases con un menor número de muestras, mejorando de esta manera la probabilidad de identificar tanto las clases minoritarias como las mayoritarias.

- Balancear las clases que identifica el modelo de clasificación multiclase añadiendo nuevos datos al conjunto de datos utilizado durante la fase de entrenamiento. Como se ha comentado en varias ocasiones, el conjunto de datos utilizado durante el entrenamiento de los modelos tiene una proporción realista entre los datos de las conexiones que lo conforman. Que el conjunto de datos presente esta característica es fundamental para comprobar cual sería el comportamiento del modelo en un entorno de producción real. El aspecto negativo, es que en la realidad no existe un equilibrio entre los tipos de intrusiones que sufre un sistema informático, por lo que los datos se encuentran extremadamente desbalanceados.

Al utilizar datos balanceados durante la fase de entrenamiento del modelo y datos realistas durante la fase de evaluación del mismo, se obtendría un modelo capaz de

distinguir mejor entre las nueve clases de intrusiones que se tratan en este trabajo y se podría comprobar la eficacia real del modelo de clasificación multiclase al evaluarlo con un conjunto de datos que representan de manera más fidedigna las intrusiones que puede sufrir un sistema informático.

- Desplegar los modelos para integrarlos en entornos operativos. Tal y como se comenta en el Capítulo 8, desplegar los modelos contribuiría a mejorar la seguridad en redes mediante la automatización de la detección de conexiones malignas, como sucede con los IDS. Una vez detectada la actividad maliciosa, se podría dar respuestas rápidas a las intrusiones para mitigar sus efectos, como hacen los IPS.
- Entrenar los modelos con los datos que estos reciben una vez desplegados. En las secciones anteriores se comenta que una de las ventajas de utilizar modelos neuronales para la detección de intrusiones en vez de algoritmos, es la capacidad de los modelos de aprender de las conexiones que reciben mientras están en producción y su potencial para detectar patrones no lineales que pueden presentar nuevos tipos de ataques desconocidos como los *zero day*.

Capítulo 10

Apéndice

El código desarrollado durante el desarrollo de este Trabajo Fin de Grado, así como los resultados de los experimentos y de la fase de evaluación, pueden descargarse desde el siguiente repositorio: <https://github.com/huglope/TFG>.

Bibliografía

- [1] Pablo Haya. La metodología CRISP-DM en ciencia de datos. <https://www.iic.uam.es/innovacion/metodologia-crisp-dm-ciencia-de-datos/>, noviembre 2021. Consultado el 18 de mayo de 2025.
- [2] Ken Schwaber and Jeff Sutherland. The scrum guide: The definitive guide to scrum – the rules of the game. <https://www.scrum.org/resources/scrum-guide>, 2020. Accedido: 2025-05-29.
- [3] IONOS España. Tcp protocol: así funciona el protocolo de transmisión. <https://www.ionos.es/digitalguide/servidores/know-how/que-es-tcp-transport-control-protocol/>, 2020. Accedido: 2025-06-02.
- [4] Vicente González Ruiz. TCP (transmission control protocol). <https://w3.ual.es/~vruiz/Docencia/Apuntes/Networking/Protocols/Level-4/05-TCP/index.html>, December 2014. Accedido: 2025-04-15.
- [5] Vicente González Ruiz. El IP (internet protocol). <https://w3.ual.es/~vruiz/Docencia/Apuntes/Networking/Protocols/Level-3/01-IP/index.html>, December 2014. Accedido: 2025-06-02.
- [6] Tomorrow Bio. Peso a peso: cómo se fortalecen las redes neuronales. <https://www.tomorrow.bio/es/post/peso-a-peso-como-se-fortalecen-las-redes-neuronales>, 2023. Consultado: 2025-06-15.
- [7] AprendeIA. ¿qué es deep learning? <https://aprendeia.com/2021/03/02/que-es-deep-learning/>, 2021. Accedido: 2025-04-04.
- [8] MSMK University. Red neuronal de retropropagación (backpropagation neural network). <https://msmk.university/red-neuronal-de-retropropagacion-backpropagation-neural-network/>, 2023.
- [9] Daniel Nielson Unite. AI. ¿qué son las redes neuronales convolucionales (CNN)? <https://www.unite.ai/es/what-are-convolutional-neural-networks/>, 2020. Accedido: 2025-05-14.
- [10] James Rundle. The ai effect: Amazon sees nearly 1 billion cyber threats a day. *The Wall Street Journal*, December 2024.

- [11] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. *CRISP-DM 1.0: Step-by-step data mining guide*. SPSS Inc., 2000.
- [12] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. <https://agilemanifesto.org/>, 2001. Accedido: 2025-05-29.
- [13] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, 2021.
- [14] Diego Carmona Fernández and Silvia Román Suero. *Gestión de Riesgos: Fundamentos sobre la gestión de riesgos en los proyectos*. Universidad de Cádiz, 2021.
- [15] Miguel Ángel Oliveros Villegas and Haydeé Cecilia Rincón de Parra. Gestión de costos en los proyectos: un abordaje teórico desde las mejores prácticas del project management institute. *Visión Gerencial*, 10(1):85–94, 2011.
- [16] David Puche ACISSI, Marion Agé. *Seguridad informática - Ethical Hacking: Conocer el ataque para una mejor defensa*. Ediciones ENI, 2022.
- [17] ServerNet. Consecuencias de ataques informáticos: reputación, daños y pérdidas. *ServerNet Blog*, 2025.
- [18] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 5th edition, 2009.
- [19] J. Scott. A comprehensive study on denial of service attacks. *International Journal of Computer Science*, 12(4):45–60, 2015.
- [20] International Information System Security Certification Consortium. Phishing attacks and how to avoid them. <https://www.isc2.org>, 2018. Accedido: 2025-04-16.
- [21] Francisco Santos and María García. Impacto de los ciberataques en la continuidad operativa de las empresas. *Revista de Ciberseguridad Empresarial*, 5:45–58, 2020.
- [22] Ponemon Institute. *The Cost of a Data Breach Report 2019*. Ponemon Institute, Michigan, USA, 2019.
- [23] Akin Bada and M. Angela Sasse. The impact of data breaches on consumer trust and company reputation. *Journal of Cybersecurity and Digital Trust*, 1(3):123–134, 2017.
- [24] Ross Anderson. *Security engineering: A guide to building dependable distributed systems*. Wiley, 2020.
- [25] Parlamento Europeo y Consejo de la Unión Europea. Reglamento (ue) 2016/679 del parlamento europeo y del consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos. <https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32016R0679>, 2016. Accedido: 2025-04-16.

- [26] Lars Sommestad, Mattias Ekstedt, and Per Johnson. The impact of proactive cybersecurity measures on organizational reputation and trust. *Journal of Information Security*, 8(4):115–127, 2019.
- [27] Cosmikal. ¿qué es y cómo funciona un firewall? guía básica 2025. <https://www.cosmikal.es/que-es-y-como-funciona-un-firewall/>, december 2024. Accedido: 2025-05-25.
- [28] Geekflare. Ids vs ips: A comprehensive guide to network security solutions. <https://geekflare.com/es/ids-vs-ips-network-security-solutions/>, december 2024. Accedido: 2025-05-30.
- [29] Palo Alto Networks. ¿qué es la microsegmentación? <https://www.paloaltonetworks.es/cyberpedia/what-is-microsegmentation>. Accedido: 2025-05-19.
- [30] ISO/IEC/IEEE. Systems and software engineering — life cycle processes — requirements engineering. <https://www.iso.org/standard/72089.html>, 2018.
- [31] National Institute of Standards and Technology (NIST). Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-2e2023.pdf>, 2021. Accedido: 2025-05-10.
- [32] John Brooke. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194), 1996.
- [33] Majed Luay, Siamak Layeghy, Seyedehfaezeh Hosseininoorbin, Mohanad Sarhan, Nour Moustafa, and Marius Portmann. Temporal analysis of netflow datasets for network intrusion detection systems. <https://arxiv.org/abs/2503.04404>, 2025. Accedido: 2025-04-22.
- [34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [35] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [36] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
- [37] Frank Rosenblatt. *The Perceptron: A Perceiving and Recognizing Automaton*. Cornell Aeronautical Laboratory, Buffalo, NY, 1958.
- [38] Johannes Lederer. Activation functions in artificial neural networks: A systematic overview. *arXiv preprint arXiv:2101.09957*, 2021.
- [39] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *arXiv preprint arXiv:2109.14545*, 2021.
- [40] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
- [41] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

- [42] Simon Haykin. *Neural Networks and Learning Machines*. Pearson, 2009.
- [43] Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.
- [44] EITCA Academy. ¿cuál es el papel de la función de pérdida en el aprendizaje automático? <https://es.eitca.org/inteligencia-artificial/eitc-ai-tff-tensorflow-fundamentos/introducci%C3%B3n-a-tensorflow/fundamentos-del-aprendizaje-autom%C3%A1tico/examen-revisar-los-fundamentos-del-aprendizaje-autom%C3%A1tico/%C2%BFCu%C3%A1l-es-el-papel-de-la-funci%C3%B3n-de-p%C3%A9rdida-en-el-aprendizaje-autom%C3%A1tico%3F/>. Accedido: 2025-05-22.
- [45] Ultralytics. Función de pérdida. <https://www.ultralytics.com/es/glossary/loss-function>. Accedido: 2025-05-22.
- [46] DataCamp. Explicación de las funciones de pérdida en el machine learning. <https://www.datacamp.com/es/tutorial/loss-function-in-machine-learning>. Accedido: 2025-05-22.
- [47] BigDataFran. 7. introducción problemas de clasificación — trabajando con pytorch. https://bigdatafran.github.io/pytorch/jupyter/Tema_3_clasificacion.html. Accedido: 2025-05-22.
- [48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019.
- [49] Léon Bottou. Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT 2010*, pages 177–186, 2010.
- [50] D.P. Kingma and J.B. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [51] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2nd edition, 2009.
- [52] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.
- [53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456. PMLR, 2015.
- [54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. In *Journal of Machine Learning Research*, volume 15, pages 1929–1958, 2014.
- [55] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

- [56] François Chollet. *Deep learning with Python*. Manning Publications Co., 2018.
- [57] Adam Paszke, Adam Lerer, Sam Gross, and Soumith Chintala. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- [58] Eli Steven Gross, Brandon Lengerich, and Luca Dey. *Deep Learning with PyTorch*. Manning Publications, 2021.
- [59] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 3rd edition, 2011.
- [60] Tom Fawcett. An introduction to roc analysis. In *Pattern Recognition Letters*, volume 27, pages 861–874, 2006.
- [61] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing And Management*, 42(1):245–275, 2006.
- [62] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.
- [63] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.
- [64] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.
- [65] N. V. Chawla and W. P. Kegelmeyer. Random under-sampling approaches for imbalanced datasets. *International Journal of Artificial Intelligence*, 8(6):63–72, 2003.
- [66] F. Girosi, M. Jones, and T. Poggio. Support vector machines and their application to image classification. *IEEE Transactions on Neural Networks*, 15(1):58–68, 2004.
- [67] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pages 253–260, 2000.
- [68] X. Liu, L. Wang, and J. Yang. Class imbalance problem in multi-class classification: A review. *Pattern Recognition*, 44(7):1575–1587, 2011.
- [69] C. W. Hsu and C. J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.
- [70] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.
- [71] François Chollet. *Deep learning with python*. Manning Publications, 2017.
- [72] J. R. Ruck, S. J. Rogers, R. K. C. Yeung, and M. M. Naylor. *The Effect of the Number of Hidden Neurons on the Performance of Multilayer Networks*. IEEE, New York, 1996.
- [73] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

- [74] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ICLR 2017*, 2017.
- [75] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 2006.
- [76] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [77] Takeshi Yamada and Masashi Sugiyama. Understanding overfitting and generalization in deep learning. *Proceedings of the 35th International Conference on Machine Learning*, pages 1617–1626, 2018.
- [78] D. Sun, T. Liao, and Y. Wang. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Access*, 5:4917–4930, 2017.
- [79] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
- [80] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28, pages 1135–1143. Curran Associates, Inc., 2015.
- [81] Haibo He and Eduardo A Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.
- [82] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: Special issue on learning from imbalanced data sets. In *ACM SIGKDD Explorations Newsletter*, volume 6, pages 1–6. ACM, 2004.
- [83] Jeff Heaton, Nicholas G Polson, and Jan Witte. An empirical analysis of deep learning for credit card fraud detection. *arXiv preprint arXiv:1703.05703*, 2017. Discute la influencia de la arquitectura, incluyendo la relación entre entrada y tamaño de capas ocultas, en tareas con clases desbalanceadas.
- [84] Bengio. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1798–1828, 2013.
- [85] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Deep learning for anomaly detection in cybersecurity: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1542–1559, 2020.
- [86] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.
- [87] Machine Learning Experiment Tracking with Weights & Biases. <https://wandb.ai/site>, 2025. Accedido: 2025-06-03.
- [88] Blackboard Architecture. <https://www.geeksforgeeks.org/blackboard-architecture-in-ai/>, 2023. Accedido: 2025-06-03.

- [89] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- [90] Nils Reimers and Iryna Gurevych. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1432–1437, 2017.
- [91] Xavier Bouthillier, Pascal Vincent, and Laurent Dinh. Sloppy models, flat minima, and generalization in deep learning. *arXiv preprint arXiv:2106.10176*, 2021.
- [92] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do image-net classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.
- [93] Justin M Johnson and Taghi M Khoshgoftaar. A survey of data augmentation approaches for imbalanced classification. *Journal of Big Data*, 6(1):1–54, 2019.
- [94] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.
- [95] Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.
- [96] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: Significance and strategies. *Proceedings of the 2002 International Conference on Artificial Intelligence*, 56:111–117, 2002.
- [97] Mohanad Sarhan, Rami Alsaqour, Mohammad Abdelhaq, and Shukor Abd Razak. Netflow datasets for machine learning-based network intrusion detection systems. *arXiv preprint arXiv:2011.09144*, 2020.
- [98] R. Wirth and J. Hipp. Crisp-dm: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, pages 29–39, 2000.
- [99] T. Baylor et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 265–283. USENIX Association, 2017.
- [100] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.
- [101] M. Peters, S.J. Pan, and P.S. Yu. Machine learning model management. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2245–2257, 2017.
- [102] Neptune.ai Team. ML pipeline architecture design patterns (with examples). *Neptune.ai Blog*, 2023.
- [103] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Elements of reusable object-oriented software. 1994.

- [104] A. Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 2004.
- [105] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, B. Nushi, A. Selvin, J. Suh, and T. Zimmermann. Software engineering for machine learning: A case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, pages 291–300, 2019.