UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"DESARROLLO DE UN SISTEMA PARA DETECCIÓN DE INTRUSOS QUE CLASIFIQUE TRÁFICO DE RED NORMAL O MALICIOSO USANDO UN MODELO CNN."

Trabajo de titulación previo a la obtención del título de Ingeniero en Tecnologías de la Información

Línea de investigación: Desarrollo, aplicación de software y cyber security (seguridad cibernética)

AUTOR:

Merly Leonela Rivera Elizalde

DIRECTOR:

MSc. Marco Remigio Pusdá Chulde

Ibarra – Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	RIVERA ELIZALDE MERLY LEONELA

DATOS DE LA OBRA	
TÍTULO:	DESARROLLO DE UN SISTEMA PARA DETECCIÓN DE INTRUSOS QUE CLASIFIQUE TRÁFICO DE RED NORMAL O MALICIOSO USANDO UN MODELO CNN.
AUTOR (ES):	RIVERA ELIZALDE MERLY LEONELA
FECHA: DD/MM/AAAA	31/08/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	[x] **PREGRADO** [ ] **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN
ASESOR /DIRECTOR:	ING. MARCO R. PUSDÁ, PHD

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 31 días del mes de agosto de 2026

EL AUTOR:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
MERLY LEONELA
RIVERA ELIZALDE

(Firma)……
Nombre: Merly Leonela Rivera Elizalde

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 31 de agosto de 2026.

Ing. Marco Pusdá, PhD
DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:
Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
MARCO REMIGIO PUSDA CHULDE

Ing. Marco Pusdá, PhD

C.C: 0401200951

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificado del trabajo de Integración Curricular “ Desarrollo de un sistema para detección de intrusos que clasifique tráfico de red normal o malicioso usando un modelo CNN ” elaborado por Merly Leonela Rivera Elizalde , previo a la obtención del título del Ingeniero en Tecnologias de la Información , aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
MARCO REMIGIO PUSDA CHULDE

Ing. Marco Pusdá, PhD

C.C: 0401200951

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
IVAN DANILO GARCIA SANTILLAN

Ing. Ivan Garcia, PhD

C.C: 1002292603

DEDICATORIA

A mis padres, Lisandro Rivera y Rocío Elizalde, quienes han sido el pilar más importante de mi vida y el regalo más valioso que Dios me ha dado. Gracias por acompañarme durante toda esta etapa. Todo lo que soy y este logro se los debo a ustedes, y espero siempre hacerlos sentir orgullosos.

Al amor de mi vida, Melinton Prado, por su amor, paciencia y apoyo incondicional. Gracias por hacer de mis estudios una prioridad, asumir muchas responsabilidades para que pudiera dedicarme a esta meta, acompañarme en los momentos más difíciles y creer en mí incluso cuando yo misma dudaba. Este logro también es tuyo.

A mi pequeño hijo, Andrés, quien llegó a mi vida a mitad de este camino y se convirtió en una de mis mayores motivaciones. Espero que algún día este logro sea un ejemplo para ti de que, con perseverancia y el apoyo de quienes nos aman, todo es posible.

A mis cinco hermanas: Sandra, Karen, Teresa, Raquel y Rubí; y a mi hermano Brandon, por estar siempre presentes y apoyarme cuando más lo necesité para que pudiera culminar esta etapa. Su apoyo fue fundamental para no rendirme. A mis sobrinos Jared, Erick, Petita y Nelson, por llenar mi vida de alegría y ser una inspiración constante.

Al señor Edgar Vera, mi cuñado y compadre, por ser una de las primeras personas en escuchar mi deseo de estudiar una carrera universitaria. Gracias por ayudarme a dar ese primer paso, motivarme cuando dudaba de mí y recordarme siempre que todo esfuerzo tiene su recompensa.

Finalmente, a quienes ya no están con nosotros, pero permanecen por siempre en mi corazón. Este logro también es por ustedes.

Merly Leonela Rivera Elizalde

AGRADECIMIENTO

A Dios, por rodearme de amor, darme vida, salud y poner en mi camino a personas maravillosas que hicieron posible este logro. Hoy culmino una etapa que representa cinco años de esfuerzo, aprendizaje y crecimiento personal y profesional, por los cuales me siento profundamente agradecida.

Agradezco al Msc. Marco Pusdá, quien desde el primer día me acompañó como director de este trabajo, guiándome con paciencia, profesionalismo y dedicación hasta alcanzar esta meta; y al Msc. Iván García, tutor de este proyecto, por su disposición, apoyo y acompañamiento constante durante el desarrollo de esta investigación.

Al Magíster Víctor Caranqui, por enseñarnos con disciplina, ética y rectitud, demostrándonos que los desafíos forman parte del camino y que cada experiencia deja un aprendizaje. Al Msc. Pablo Landeta, al Msc. Carlos Dávila y al Msc. Diego Terán, por su paciencia, calidad humana y dedicación, brindándonos siempre su apoyo y motivándonos a aprender cada día más. Asimismo, al Magíster Fausto Salazar, al Msc. Diego Trejo y a todos los docentes que formaron parte de esta etapa académica, por compartir sus conocimientos, inculcarnos valores y prepararnos como profesionales y para saber afrontar los retos fuera de las aulas.

A mis compañeros de clase, con quienes compartí conocimientos y grandes experiencias a lo largo de esta carrera. En especial a mis amigos Jairo y Belén, gracias por su amistad, apoyo y por hacer de la universidad una etapa muy bonita. Les deseo el mayor de los éxitos en su vida profesional.

A mi familia, a mis amigos y a la Universidad Técnica del Norte.

Gracias por todo.

Merly Leonela Rivera Elizalde

ÍNDICE DE CONTENIDOS

DEDICATORIA VII

AGRADECIMIENTO VIII

ÍNDICE DE FIGURAS XV

ÍNDICE DE TABLAS XV

RESUMEN XVII

ABSTRACT XVIII

CAPÍTULO I

1.1 Planteamiento del problema 1

1.2 Objetivos 2

1.2.1 Objetivo General 2

1.2.2 Objetivos Específicos 2

1.3 Alcance 2

1.4 Justificación 3

1.4.1 Contribución académica 4

1.4.2 Contribución tecnológica 4

1.4.3 Contribución práctica 5

CAPÍTULO II

2.1 Antecedentes 6

2.2 Seguridad informática y detección de intrusos 7
2.2.1 Conceptos fundamentales de seguridad 8
2.2.2 Amenazas y ataques cibernéticos en redes 10
2.2.3 Sistemas de Detección de Intrusos (IDS) 13
2.2.4 Tipos de IDS: basados en red (NIDS) vs. basados en host (HIDS) 14
2.2.5 Metodologías de detección: por firmas (signature-based) vs. por anomalías (anomaly-based) 16
2.2.6 Redes y tráfico de datos 16
2.2.7 Arquitectura TCP/IP y modelo OSI 18
2.2.8 Características del tráfico de red 19
2.2.9 Flujos de red (NetFlow) vs. paquetes crudos 21
2.2.10 Conjuntos de datos (Datasets) públicos para IDS 22
2.3 Inteligencia artificial en ciberseguridad 24
2.3.1 Aplicaciones de la IA en ciberseguridad 26
2.3.2 Ventajas del uso de IA para IDS 28
2.3.3 Desafíos y limitaciones de la IA aplicada a la ciberseguridad 29
2.4 Redes neuronales 30
2.4.1 Concepto básico de neurona artificial, función de activación, capas 33
2.4.2 Aprendizaje supervisado y proceso de entrenamiento 34
2.4.3 Redes Neuronales Convolucionales (CNN) 35
2.4.4 Preprocesamiento de datos para redes neuronales 37
2.5 Trabajos relacionados 38

CAPÍTULO III

3.1 Enfoque metodológico 40
3.1.1 Enfoque de la investigación 40
3.1.2 Tipo de investigación 41
3.2 Fuente de datos 43
3.2.1 Descripción del dataset CICIDS2017 43
3.2.2 Estructura del dataset 45
3.2.3 Conversión de etiquetas 46
3.3 Diseño metodológico 47
3.3.1 Preprocesamiento de datos 50
3.3.2 Limpieza de datos 50
3.3.3 Reducción del dataset 51
3.3.4 Generación de subconjuntos de datos 52
3.3.5 División del dataset 52
3.3.6 Balanceo de datos mediante SMOTE 53
3.3.7 Normalización de datos 54
3.3.8 Transformación de datos para redes neuronales convolucionales. 55
3.4 Diseño del modelo 56

3.4.1 Arquitectura del modelo CNN 57
3.4.2 Compilación del modelo 60
3.4.3 Entrenamiento del modelo 60
3.4.4 Guardado del modelo entrenado 62
3.4.5 Preparación de recursos auxiliares 63
3.5 Desarrollo del aplicativo 64
3.5.1 Autenticación de usuarios 66
3.5.2 Pantalla principal de monitoreo en tiempo real. 67
3.5.3 Visualización de clasificación de tráfico. 68
3.5.4 Registro de intrusiones detectadas 69
3.5.5 Pantalla de análisis histórico 69
3.5.6 Tendencia de puertos atacados 70
3.5.7 Evolución de la precisión del modelo 71
3.5.8 Generación de reportes 72
3.6 Validación del modelo 73
3.6.1 Precisión global (Accuracy) 74
3.6.2 Precisión de clasificación (Precision) 75
3.6.3 Sensibilidad del modelo (Recall) 75
3.6.4 Medida F1-score 76
3.6.5 Matriz de confusión. 76

CAPÍTULO IV

4.1 Métricas de clasificación 79
4.1.1 Exactitud (*Accuracy*) 80
4.1.2 Precisión (*Precision*) 81
4.1.3 Sensibilidad (*Recall*) 82
4.1.4 Puntuación F1 (*F1-score*) 82
4.2 Métricas estadísticas 83
4.2.1 Función de pérdida (*Binary Cross-Entropy*) 84
4.2.2 Validación Cruzada (Cross-Validation) 85
4.2.3 Análisis de las Matrices de Confusión 86
4.2.3.1 Ataques mayoritarios 86
4.2.3.2 Ataques minoritarios 87
4.2.4 Curva ROC y área bajo la curva (AUC) 88
4.3 Comparativa del modelo 89
4.3.1 Desempeño frente a modelos base 90
4.3.2 Posicionamiento frente a arquitecturas complejas 90
4.3.3 Tabla comparativa 91
4.4 Limitaciones 92
4.5 Impactos 94
4.5.1 Impacto tecnológico 94
4.5.2 Impacto económico 94
4.5.3 Impacto social 95
CONCLUSIONES 96

RECOMENDACIONES 97
BIBLIOGRAFÍA 98
ANEXOS 107

ÍNDICE DE FIGURAS

Figura 1 Diagrama de la Tríada CIA 9
Figura 2 Resumen de tipos de ataques cibernéticos 12
Figura 3 Tráfico de red normal y malicioso en un NIDS 18
Figura 4 Distribución de características del tráfico de red 20
Figura 5 Aplicaciones de inteligencia artificial en ciberseguridad 27
Figura 6 Estructuras de redes neuronales CNN y RNN 32
Figura 7 Arquitecturas de modelos LSTM y GRU 33
Figura 8 Entrenamiento supervisado de una red neuronal 35

Figura 9 Archivos del dataset CICIDS2017 45
Figura 10 Registros y atributos del dataset 46
Figura 11 Flujo metodológico del sistema propuesto 48
Figura 12 División del dataset en entrenamiento y prueba 53
Figura 13 Balanceo de clases mediante SMOTE 54
Figura 14 Arquitectura de la red neuronal convolucional 58
Figura 15 Distribución de clases antes de SMOTE 61
Figura 16 Resultados iniciales del entrenamiento del modelo 62
Figura 17 Resultados finales del entrenamiento del modelo 62
Figura 18 Archivos generados tras el entrenamiento del modelo 63
Figura 19 Arquitectura del aplicativo desarrollado 65
Figura 20 Pantalla de inicio de sesión del sistema 67
Figura 21 Pantalla principal de monitoreo en tiempo real 68
Figura 22 Simulación de clasificación de tráfico en tiempo real 68
Figura 23 Registro de intrusiones y resumen del sistema 69
Figura 24 Pantalla de análisis histórico 70
Figura 25 Tendencia de puertos atacados 71
Figura 26 Evolución de la precisión del modelo 72
Figura 27 Generación de reportes por intervalos de tiempo 73
Figura 28 Generación de reportes generales del sistema 73
Figura 29 Visualización de métricas del modelo 74
Figura 30 Matriz de confusión 77

Figura 31 Comparación de tráfico normal y malicioso 79
Figura 32 Comparación de métricas en ataques mayoritarios y minoritarios 80
Figura 33 Resultados de Accuracy 81
Figura 34 Resultados de precisión del modelo 82
Figura 35 Resultados de Recall 82
Figura 36 Resultados de F1-score del modelo 83
Figura 37 Evolución de la función de pérdida 84
Figura 38 Resultados de validación del modelo 85
Figura 39 Matriz de confusión para ataques mayoritarios 86
Figura 40 Matriz de confusión para ataques minoritarios 87
Figura 41 Curva ROC del modelo CNN 1D 89

ÍNDICE DE TABLAS

Tabla I Amenazas cibernéticas y su impacto en la tríada CIA 11
Tabla II Comparación entre NIDS y HIDS 15
Tabla III Distribución de clases en datasets utilizados para IDS 23
Tabla IV Arquitecturas de redes neuronales y su uso en IDS 31

Tabla V Proceso de limpieza y eliminación de valores inválidos en el dataset . . . 51
Tabla VI Distribución de clases con SMOTE 56

Tabla VII Comparación del modelo propuesto con trabajos relacionados 92

RESUMEN

El aumento en la frecuencia y sofisticación de los ciberataques ha superado la capacidad de los sistemas tradicionales de detección de intrusiones (IDS) basados en firmas, lo que motiva el desarrollo de soluciones basadas en inteligencia artificial. Esta tesis presenta el diseño e implementación de un sistema IDS basado en una red neuronal convolucional unidimensional (1D-CNN) para clasificar el tráfico de red como normal o malicioso. Se utilizó el conjunto de datos público CIC-IDS2017, que contiene tráfico etiquetado con múltiples tipos de ataques modernos (DDoS, PortScan, DoS, ataques web, infiltración, fuerza bruta y botnet). El preprocesamiento de los datos incluyó limpieza de valores nulos e infinitos, normalización con StandardScaler y balanceo de clases mediante la técnica SMOTE aplicada sobre las etiquetas multiclase, lo que permitió manejar el fuerte desbalance entre el tráfico normal y los ataques minoritarios. La arquitectura del modelo consta de dos capas convolucionales con 64 y 128 filtros, capas de pooling, dropout (0,3) y una capa de salida con activación sigmoide. El entrenamiento se realizó en la plataforma Kaggle con una GPU Tesla T4 durante 12 épocas, alcanzando una exactitud de validación del 98,7 %, un recall del 99,5 % y una precisión del 97,8 %. En las pruebas con ataques mayoritarios (DDoS, PortScan, DoS) se obtuvo una exactitud del 98,65 % y un recall del 99,98 %; en ataques minoritarios (web, infiltración, fuerza bruta) el recall se mantuvo en el 100 %, aunque la precisión descendió al 21,7 % debido a la escasez de ejemplos. El área bajo la curva ROC (AUC) alcanzó un valor de 0,953, confirmando la excelente capacidad discriminatoria del modelo. Además, se implementó una aplicación web interactiva con Streamlit que permite simular el análisis de tráfico en tiempo real, visualizar métricas, matrices de confusión y generar reportes descargables. Las principales limitaciones son la baja precisión en ataques poco frecuentes y la dependencia de un dataset estático. Se concluye que la arquitectura 1D-CNN propuesta es eficiente, ligera y competitiva, superando a los métodos tradicionales y alcanzando un rendimiento comparable con modelos más complejos de la literatura.

Palabras clave: Detección de intrusiones, red neuronal convolucional 1D, CIC-IDS2017, clasificación binaria, aprendizaje profundo, Streamlit

ABSTRACT

The increasing frequency and sophistication of cyberattacks have exceeded the capabilities of traditional signature-based intrusion detection systems (IDS), motivating the development of artificial intelligence-based solutions. This thesis presents the design and implementation of an IDS based on a one-dimensional convolutional neural network (1D-CNN) to classify network traffic as normal or malicious. The public CIC-IDS2017 dataset was used, containing labeled traffic with multiple modern attack types (DDoS, PortScan, DoS, web attacks, infiltration, brute force, and botnet). Preprocessing included cleaning of null and infinite values, normalization using StandardScaler, and class balancing through the SMOTE technique applied to multiclass labels, allowing us to handle the severe imbalance between normal traffic and minority attacks. The model architecture consists of two convolutional layers with 64 and 128 filters, pooling layers, dropout (0.3), and a sigmoid output layer. Training was performed on the Kaggle platform using a Tesla T4 GPU for 12 epochs, achieving validation accuracy of 98.7 %, recall of 99.5 %, and precision of 97.8 %. In tests with majority attacks (DDoS, PortScan, DoS), accuracy reached 98.65 % and recall 99.98 %; for minority attacks (web, infiltration, brute force), recall remained at 100 %, although precision dropped to 21.7 % due to the scarcity of examples. The area under the ROC curve (AUC) reached 0.953, confirming the excellent discriminatory capability of the model. Additionally, an interactive web application was implemented with Streamlit to simulate real-time traffic analysis, visualize metrics and confusion matrices, and generate downloadable reports. The main limitations are low precision on very rare attacks and dependence on a static dataset. It is concluded that the proposed 1D-CNN architecture is efficient, lightweight, and competitive, outperforming traditional methods and achieving performance comparable to more complex models reported in the literature.

Keywords: Intrusion detection, 1D convolutional neural network, CIC-IDS2017, binary classification, deep learning, Streamlit

CAPÍTULO I

INTRODUCCIÓN

1.1 Planteamiento del problema

La evolución constante de los ciberataques [1] ha desafiado la efectividad de los sistemas tradicionales de detección de intrusiones. Las amenazas informáticas han provocado que los atacantes empleen métodos cada vez más complejos, como la explotación de vulnerabilidades desconocidas y la evasión de firmas conocidas, lo cual dificulta la detección de patrones de ataque inesperados [2].

Esta creciente sofisticación de las amenazas informáticas ha superado las capacidades de los sistemas tradicionales de detección de intrusiones (IDS). Dichos métodos resultan ineficaces frente a ataques novedosos y adaptativos, dejando vulnerables a los sistemas informáticos críticos [3].

Los ataques cibernéticos maliciosos se han convertido en una amenaza crítica que afecta principalmente a organizaciones de alto valor, como instituciones financieras, hospitales y entidades gubernamentales [4]. Esta situación provoca graves problemas operativos, financieros y de reputación para las víctimas, además de poner en riesgo la confidencialidad y disponibilidad de información sensible [5].

En América Latina, durante 2023, los ciberataques tuvieron un impacto significativo en la seguridad y operatividad de las organizaciones[6]. El 30 % de las empresas sufrió al menos un incidente de seguridad en el último año, lo cual evidencia la alta vulnerabilidad en la región. Además, los ataques de ransomware afectaron a un 23 % de las organizaciones en los últimos dos años, siendo los sectores de Petróleo, Gas y Minería los más atacados (36 %), seguidos por

Telecomunicaciones (31 %) y Servicios Públicos (30 %). Esta exposición también se refleja en que el 62 % de las empresas considera insuficiente su presupuesto de ciberseguridad, lo que agrava la vulnerabilidad de sus sistemas [6].

1.2 Objetivos

1.2.1 Objetivo General

Diseñar un sistema de detección de intrusiones para clasificar el tráfico de red normal o malicioso, utilizando un modelo de red neuronal convolucional (CNN) que mejore la precisión y eficiencia en la identificación de ataques cibernéticos.

1.2.2 Objetivos Específicos

- Analizar información teórica sobre detección de intrusiones en redes, redes neuronales convolucionales (CNN) y herramientas tecnológicas para el desarrollo del tema propuesto.
- Desarrollar un modelo CNN para clasificar el tráfico de red como normal o malicioso.
- Evaluar el desempeño del modelo mediante métricas de eficiencia en la detección de ataques.

1.3 Alcance

El proyecto se centra en el diseño e implementación de un sistema de detección de intrusos en redes que integre y evalúe un modelo de red neuronal convolucional (CNN) preexistente para clasificar el tráfico de red en normal o malicioso en tiempo real. Para ello, primero se realizó una revisión de la información relevante sobre las técnicas de detección de intrusiones, aprendizaje profundo y por supuesto redes neuronales convolucionales con el objetivo de contar con una base teórica sólida que respalde todas las decisiones y procesos a seguir dentro del proyecto.

Se trabajó con datasets representativos y actualizados que permitieron entrenar y validar el modelo CNN seleccionado, asegurando que puedan reconocer patrones de tráfico de red normal y malicioso con precisión. Esto incluye la preparación y normalización de los datos, la adaptación del modelo al entorno del sistema y la integración de un módulo de monitoreo en tiempo real que permita evaluar su desempeño y confiabilidad.

El desarrollo se restringe a ciertas limitaciones que deben ser consideradas durante el proceso. Primero, los datasets utilizados provienen de fuentes públicas o son generados previamente, por lo que no reflejan completamente el tráfico de producción en redes reales. Segundo, el sistema fue validado únicamente en entornos simulados, locales o en la nube, sin acceso a redes institucionales reales. Otra restricción es el hardware disponible, ya que entrenar una CNN requiere alta capacidad de procesamiento, por ello se dependió de equipos propios o servicios externos de bajo costo.

Como resultado del proyecto, se generaron entregables concretos que evidencien el cumplimiento de los objetivos planteados en el proyecto. Además, un modelo CNN funcional capaz de clasificar tráfico de red normal o malicioso, junto con los datasets preprocesados utilizados para entrenamiento y prueba. También se generaron informes de métricas de desempeño, incluyendo precisión, exactitud, tasa de detección y falsos positivos, acompañados de análisis y visualizaciones que respalden los resultados obtenidos.

1.4 Justificación

Frente a la creciente sofisticación de los ataques cibernéticos [7], es necesario desarrollar sistemas que empleen técnicas avanzadas de inteligencia artificial, capaces de aprender y generalizar a partir de grandes volúmenes de datos de red.

Entre estas técnicas, las redes neuronales convolucionales (CNN) destacan por su capacidad para procesar datos complejos y extraer características relevantes sin necesidad de un preprocesamiento exhaustivo [8]. Esto permite superar las limitaciones de los métodos tradicionales

y mejorar la precisión en la clasificación del tráfico de red como normal o malicioso, contribuyendo así a fortalecer la seguridad de los sistemas informáticos [9].

En este sentido, las redes neuronales convolucionales (CNN) [8] se presentan como una herramienta eficaz para la clasificación del tráfico de red, debido a su capacidad para extraer características relevantes automáticamente y detectar anomalías con alta precisión. El diseño de un sistema de detección de intrusos que utiliza un modelo CNN para diferenciar entre tráfico normal y malicioso permitirá una identificación temprana de amenazas, mejorando la capacidad de respuesta y mitigación de incidentes. Esta solución contribuye a fortalecer la seguridad informática, reducir los riesgos operativos y proteger la integridad y confidencialidad de la información en entornos cada vez más complejos y dinámicos.

1.4.1 Contribución académica

El proyecto aporta un estudio actualizado sobre el uso de redes neuronales convolucionales en el campo de la ciberseguridad, enriqueciendo la literatura al mostrar cómo un modelo de inteligencia artificial puede aplicarse al análisis de tráfico de red, un área donde todavía existen brechas de investigación y que sirve como referencia para futuras investigaciones relacionadas con la aplicación de inteligencia artificial en la detección de intrusos, consolidando su aporte académico.

1.4.2 Contribución tecnológica

Desde el punto de vista tecnológico, se desarrolla un prototipo funcional de sistema IDS (Intrusion Detection System) basado en CNN para la detección de patrones maliciosos frente a los métodos tradicionales, optimizando el proceso de clasificación del tráfico al reducir los falsos positivos y aumentar la precisión, además de integrar técnicas de preprocesamiento y normalización de datos para mejorar la eficiencia y efectividad del modelo.

1.4.3 Contribución práctica

En el ámbito práctico, la validación del modelo en la institución proporciona una herramienta real para fortalecer la seguridad informática y ofrece una base que puede ser replicada o escalada en otros entornos empresariales, académicos o gubernamentales, contribuyendo a la toma de decisiones en la gestión de ciberseguridad y evidenciando la viabilidad de aplicar técnicas de inteligencia artificial en la protección de redes locales.

CAPÍTULO II

MARCO TEÓRICO

2.1 Antecedentes

A lo largo de los años, la ciberseguridad ha dejado atrás los modelos estáticos para adoptar enfoques más dinámicos. Antes, los sistemas de detección de intrusos (IDS) se limitaban a usar firmas de ataques conocidos, esto funcionaba, pero presentaban un carácter reactivo que no servía contra amenazas nuevas o ataques de día cero [10].

La búsqueda de alternativas frente a estas limitaciones impulsó el desarrollo de enfoques basados en la detección de anomalías. En lugar de depender exclusivamente de firmas conocidas, estos métodos emplean modelos estadísticos y algoritmos tradicionales de aprendizaje automático, como máquinas de vectores de soporte y los bosques aleatorios, para reconocer desviaciones respecto al comportamiento habitual de una red o sistema. Sin embargo, su desempeño seguía condicionado por la ingeniería de características, un proceso que exige la selección y construcción manual de variables relevantes, además de un alto nivel de conocimiento especializado [10], [11].

La etapa más reciente de desarrollo de los IDS, el aprendizaje profundo ha adquirido un papel relevante. Dentro de este enfoque, las redes neuronales convolucionales (CNN) se han utilizado por su capacidad para reconocer representaciones jerarquizadas y relaciones complejas en datos de tráfico de red, ya sean crudos o previamente procesados. Esta forma de aprendizaje disminuye la necesidad de definir características de manera manual y facilita la identificación de patrones no lineales asociados a ciberataques actuales [11], [12].

El avance de estas técnicas también ha sido posible gracias a la disponibilidad de conjun-

tos de datos públicos más completos y cercanos a escenarios reales, como CICIDS2017, CIC-IDS2018 y UNSW-NB15. Estos conjuntos permiten evaluar los modelos con tráfico normal y malicioso bajo condiciones más representativas. En varios estudios se han reportado precisiones cercanas o superiores al 99 % en tareas de clasificación. Desde esa línea, esta investigación analiza el uso de aprendizaje profundo como una alternativa para fortalecer los sistemas de detección de intrusos [13], [14].

2.2 Seguridad informática y detección de intrusos

La seguridad informática se centra en proteger redes y sistemas frente a accesos no autorizados, modificaciones maliciosas y ataques que puedan comprometer la confidencialidad, integridad o disponibilidad de la información. Visto así, los sistemas de detección de intrusos (IDS) se han convertido en herramientas esenciales, ya que permiten monitorear de manera continua el tráfico de red o las actividades internas de un sistema para identificar comportamientos anómalos o potencialmente peligrosos [15].

Según la revisión sistemática “Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges”, los sistemas de detección de intrusiones pueden agruparse principalmente en dos enfoques: detección basada en firmas y detección basada en anomalías [15].

En los IDS basados en firmas, el tráfico de red se compara con patrones registrados previamente. Este método resulta útil cuando se trata de amenazas conocidas, ya que permite identificarlas con un alto nivel de precisión. Su limitación aparece frente a ataques nuevos o variantes que aún no cuentan con una firma documentada, situación que reduce su capacidad de respuesta ante escenarios cambiantes [16].

A diferencia del enfoque basado en firmas, los IDS orientados a la detección de anomalías construyen un modelo del comportamiento habitual de la red y generan alertas cuando identifican desviaciones significativas respecto a ese patrón. Estas características les permiten detectar amenazas desconocidas, incluidos ataques de día cero. Sin embargo, su eficacia depende de

la calidad de los datos utilizados para el entrenamiento y de la actualización continua de los modelos, además de que pueden presentar tasas elevadas de falsos positivos [17].

En las redes actuales, la presencia de dispositivos IoT, sistemas distribuidos y grandes volúmenes de tráfico ha hecho que el monitoreo sea más complejo. La variedad de equipos conectados, los cambios constantes en el comportamiento del tráfico y la existencia de múltiples puntos vulnerables exigen IDS capaces de adaptarse y procesar información en tiempo real [16].

Frente a este escenario, el aprendizaje automático (ML) y el aprendizaje profundo (DL) han permitido ampliar las capacidades de los sistemas de detección de intrusiones. Estas técnicas ayudan a reconocer patrones más complejos, identificar ataques sofisticados y detectar comportamientos maliciosos que no siempre pueden ser reconocidos mediante métodos tradicionales [18].

Aunque el uso de ML y DL ha ampliado las capacidades de los IDS, su implementación no está libre de dificultades. El rendimiento del modelo depende de la calidad y vigencia de los datos empleados durante el entrenamiento. Cuando los conjuntos de datos no representan adecuadamente el entorno real o contienen registros desactualizados, el rendimiento puede verse afectado. A esto se suma el costo computacional del análisis en tiempo real, que exige arquitecturas optimizadas, distribuidas o apoyadas en Edge computing [15].

De este modo, los sistemas de detección de intrusos se mantienen como una capa relevante dentro de la seguridad informática. Su desempeño no depende únicamente del enfoque utilizado, ya sean firmas, anomalías o modelos basados en ML/DL, sino también de la actualización continua de sus reglas o modelos, de su adaptación al entorno y de la disponibilidad de datos que reflejen las amenazas vigentes [16].

2.2.1 Conceptos fundamentales de seguridad

La seguridad de la información se basa en proteger el valor de los activos informáticos, centrándose en el mantenimiento de sus propiedades cardinales: la confidencialidad, la integridad

y la disponibilidad, colectivamente conocidas como la tríada CIA [19]. Estos atributos son el objetivo último de la protección y su menoscabo constituye el daño potencial resultante de un incidente de seguridad, el cual es iniciado por una amenaza [19].

La Figura 1 presenta la tríada CIA, conformada por la confidencialidad, la integridad y la disponibilidad.

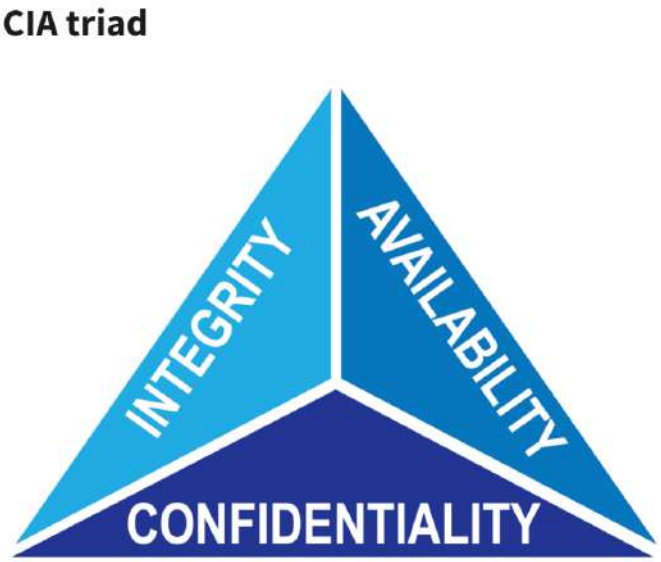

Figura 1.
Diagrama de la tríada de confidencialidad, integridad y disponibilidad [20]

Comprender cómo se materializa esta secuencia teórica de amenaza a incidente y daño requiere descomponer los ataques en componentes operativos. Lallie et al. [21] identifican estos componentes fundamentales como exploits (acciones que aprovechan fallos), vulnerabilidades (las debilidades en sí mismas) y precondiciones/postcondiciones (el estado del sistema antes y después del ataque). Las vulnerabilidades, entendidas como la ausencia o debilidad de un mecanismo de seguridad [19], son precisamente lo que los exploits buscan aprovechar.

El análisis de estas cadenas de ataque no siempre es sencillo. Tradicionalmente, se han utilizado modelos gráficos para explicar estas relaciones con fines educativos [19]. En un nivel más operativo, también se emplean técnicas de modelado de amenazas, como árboles y grafos de ataque, que permiten visualizar las posibles rutas que puede seguir un atacante dentro de un sistema [21]. Sin embargo, la complejidad y el volumen del tráfico de red moderno, unido al

elevado costo económico de las brechas de seguridad [21], excede la capacidad del análisis manual. Esto justifica la necesidad de sistemas automáticos de detección, como el propuesto en este trabajo, que, utilizando redes neuronales convolucionales, pueden aprender a identificar los patrones complejos que estos componentes de ataque generan en el tráfico de red, automatizando y escalando la percepción de la amenaza [21].

2.2.2 Amenazas y ataques cibernéticos en redes

El panorama actual de la ciberseguridad está definido por una evolución constante y una creciente sofisticación de las amenazas, lo que demanda sistemas de detección igualmente dinámicos y capaces. Según el informe más reciente de la Agencia de la Unión Europea para la Ciberseguridad (ENISA), entre julio de 2022 y junio de 2023 se consolidó un conjunto de ocho categorías principales de amenazas que destacaron por su frecuencia e impacto [22]. Estas categorías se presentan en la Tabla I.

Tabla I.
Amenazas cibernéticas y su impacto en la tríada CIA

Amenaza	Descripción	Problemas	C	I	D
Ransomware	Malware que cifra la información y exige rescate	Pérdida de acceso a datos críticos	Alta	Alta	Alta
Malware	Software malicioso que daña sistemas o roba información	Robo de datos, espionaje, sabotaje	Alta	Media	Media
Ingeniería social	Manipulación psicológica de usuarios	Accesos no autorizados	Alta	Media	Baja
Amenazas contra los datos	Alteración o destrucción de información	Corrupción de datos	Media	Alta	Media
DoS / DDoS	Saturación de servicios de red	Interrupción de servicios	Baja	Baja	Alta
Amenazas contra Internet	Ataques a infraestructura crítica	Caídas de conectividad	Baja	Media	Alta
Manipulación de información	Alteración intencional de contenidos	Desinformación	Media	Alta	Baja
Ataques a la cadena de suministro	Compromiso de proveedores	Inserción de código malicioso	Alta	Alta	Media

Fuente: Elaboración propia con base en ENISA [22].

La taxonomía de amenazas propuesta por la ENISA evidencia la diversidad de vectores de ataque que comprometen redes y sistemas. Para los sistemas de detección de intrusos basados en red (NIDS), muchas de estas amenazas, como el ransomware, el malware y los ataques DDoS, generan patrones de comunicación, tráfico anómalo o comportamientos de red característicos que, en teoría, pueden ser identificados mediante análisis automatizado [22].

Entre las amenazas que afectan la disponibilidad de los servicios, los ataques de denegación de servicios distribuidos (DDoS) ocupan un lugar relevante por su frecuencia e impacto [22]. Este tipo de ataque intenta saturar los recursos de red para impedir que los usuarios legítimos accedan a un servicio, lo que puede afectar de forma directa la operación de una organización. Por sus características, la detección de DDoS resulta adecuada para el uso de modelos de aprendizaje profundo, entre ellos las redes neuronales convolucionales (CNN), ya que estos modelos pueden diferenciar el tráfico normal de los patrones de inundación (flood) asociados a este tipo de ataque. Esta situación muestra las limitaciones de los sistemas que dependen solo de firmas estáticas y da sustento al uso de métodos de detección por anomalías apoyados en inteligencia artificial. En el caso de esta investigación, el modelo CNN propuesto no requiere firmas previamente registradas, sino que busca reconocer actividades maliciosas a partir de aprendizaje del comportamiento normal de la red. Esto puede mejorar su respuesta frente a variantes nuevas o ataques de día cero (zero day).

La Figura 2 presenta un resumen de los principales tipos de ataques cibernéticos considerados dentro del análisis.

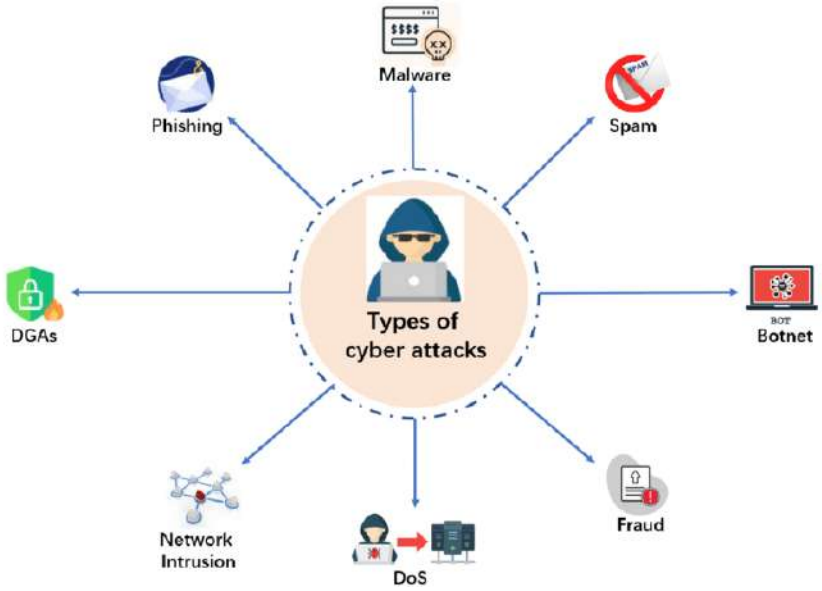

Figura 2.
Resumen de tipos de ataques cibernéticos [23].

El análisis de los sectores más afectados, siendo la administración pública el más atacado

($\sim$ 19 %), seguido por individuos ($\sim$ 11 %), salud ($\sim$ 8 %) e infraestructura digital ($\sim$ 7 %), confirma que la amenaza es transversal y de alto impacto social. Esto refuerza la necesidad de desarrollar herramientas de detección accesibles y eficaces que puedan proteger tanto infraestructuras críticas como a usuarios finales, objetivo último del sistema de detección de intrusos que se pretende diseñar y evaluar en este trabajo [22].

2.2.3 Sistemas de Detección de Intrusos (IDS)

Un IDS permite supervisar de forma automática los eventos que ocurren dentro de una red o de un sistema, con el fin de reconocer señales asociadas a posibles incidentes de seguridad [24]. En términos operativos, su tarea principal consiste en clasificar la actividad observada y diferenciar el comportamiento legítimo de aquel que puede considerarse intrusivo [25].

Desde una perspectiva metodológica, los IDS se clasifican en dos enfoques principales:

- **Basados en firmas**: comparan la actividad registrada con patrones previamente conocidos. Este método ofrece buenos resultados frente a amenazas documentadas, aunque pierde efectividad cuando se enfrenta a ataques nuevos o variantes que todavía no cuentan con una firma registrada.
- **Basados en anomalías**: construyen una referencia del comportamiento normal de la red o del sistema y generan alertas cuando identifican desviaciones relevantes. Su ventaja está en la posibilidad de detectar ataques no conocidos; sin embargo, su aplicación exige definir con precisión qué se considera normal, ya que una caracterización deficiente puede incrementar las falsas alarmas [24].

La efectividad operativa de cualquier IDS se ve limitada por un problema estadístico fundamental conocido como la falacia de la tasa base [25]. Dado que las intrusiones reales son extremadamente infrecuentes en comparación con el volumen abrumador de tráfico normal, incluso una tasa de error aparentemente baja en la detección genera una cantidad inmanejable de

falsos positivos. Esto sobrecarga a los administradores, reduce la confianza en el sistema y puede ocultar alertas genuinas [25].

A partir de esta limitación, el diseño de un IDS basado en redes neuronales convolucionales no debería enfocarse únicamente en detectar la mayor cantidad de ataques. También debe controlar los falsos positivos y mantener un equilibrio entre sensibilidad y precisión. En este caso, el modelo aprende patrones del tráfico de red para mejorar la clasificación frente a métodos tradicionales y responder mejor ante comportamientos maliciosos.

2.2.4 Tipos de IDS: basados en red (NIDS) vs. basados en host (HIDS)

Los sistemas de detección de intrusos también pueden diferenciarse según el tipo de información que analizan. Bajo este criterio, se distinguen dos grupos principales: los IDS basados en red, conocidos como NIDS, y los IDS basados en host, denominados HIDS. Cada uno opera sobre una fuente de datos distinta y, por ello, ofrece capacidades de monitoreo diferentes dentro de una arquitectura de seguridad [26], [27]. La Tabla II presenta una comparación entre ambos enfoques.

Tabla II.
Comparación entre NIDS y HIDS

Característica	NIDS (Network IDS)	HIDS (Host IDS)
Ámbito de monitoreo	Analiza paquetes y flujos de tráfico en segmentos de red, normalmente ubicados en puntos estratégicos como firewalls o enlaces principales.	Supervisa eventos internos de un host específico, como registros del sistema operativo, procesos, archivos y tráfico local.
Tipo de ataques detectados	Identifica escaneos de puertos, ataques DoS/DDoS y amenazas que afectan a varios sistemas dentro de una red.	Detecta intrusiones locales, escalamiento de privilegios y cambios maliciosos en archivos o configuraciones.
Fortaleza principal	Permite observar el comportamiento general del tráfico y detectar ataques a escala de red.	Permite revisar con mayor detalle el comportamiento interno de un equipo o servidor.
Relevancia para esta investigación	Alta, porque el sistema trabaja con tráfico de red y lo clasifica como normal o malicioso.	Limitada, porque la investigación no analiza el monitoreo interno de equipos individuales.
Implementación típica	Se ubica en puntos estratégicos de la red para monitorear y analizar el tráfico.	Se instala en servidores o estaciones de trabajo que requieren supervisión local.

Fuente: Elaboración propia con base en Debar et al. [26] y Scarfone y Mell [27].

La elección entre NIDS y HIDS, o su implementación de manera complementaria, depende de los objetivos de seguridad de la organización. Mientras que un NIDS proporciona una defensa perimetral y una visión del tráfico agregado de la red, un HIDS ofrece una auditoría profunda de la actividad interna de sistemas críticos [27]. El modelo de detección propuesto en esta tesis, al estar diseñado para clasificar el tráfico de red, se enmarca explícitamente dentro de la

arquitectura y los objetivos de un NIDS basado en anomalías, buscando automatizar y mejorar la precisión de la supervisión de la red.

2.2.5 Metodologías de detección: por firmas (signature-based) vs. por anomalías (anomaly-based)

Las metodologías tradicionales de detección tienen limitaciones importantes. En los sistemas basados en firmas, el principal problema es que dependen de patrones registrados con anterioridad. Por otro lado, en los sistemas basados en anomalías, la dificultad está en definir con claridad qué comportamiento puede considerarse normal dentro de una red. Frente a estas limitaciones, la investigación en IDS ha incorporado técnicas de aprendizaje profundo para mejorar el análisis del tráfico de red [28].

A diferencia de los enfoques clásicos, que requieren ingeniería manual de características, los modelos basados en aprendizaje profundo pueden extraer representaciones complejas directamente desde datos de tráfico crudos o de bajo nivel. En este grupo se encuentran las redes neuronales convolucionales (CNN), cuyo uso permite reconocer relaciones no lineales y patrones sutiles que no siempre son evidentes mediante métodos estadísticos tradicionales [28].

Esta capacidad resulta relevante para la detección de ataques novedosos o de día cero (zero-day), ya que el modelo no depende únicamente de reglas o firmas definidas con anterioridad. Además, al identificar mejor las diferencias entre comportamiento normal y actividad sospechosa, puede contribuir a reducir falsos positivos en el sistema basado en anomalías [28]. En esta investigación, la CNN se utiliza con este propósito: clasificar el tráfico de red como normal o malicioso mediante un proceso automatizado de aprendizaje.

2.2.6 Redes y tráfico de datos

El diseño y la evaluación de un sistema de detección de intrusos basado en red (NIDS) requieren comprender cómo está organizada la red y qué tipo de tráfico circula por ella. En este

contexto, el análisis de flujos permite observar el comportamiento general de las comunicaciones sin depender únicamente del contenido interno de los paquetes [29].

Este enfoque resulta útil cuando se emplean modelos de aprendizaje automático o aprendizaje profundo, ya que el rendimiento del sistema depende en gran medida de la calidad de los datos, de su representatividad y del tratamiento previo aplicado antes del entrenamiento [30].

Tráfico de red normal:

El tráfico de red normal no quiere decir que todos los registros sean iguales. Más bien, sus variaciones se mantienen dentro de rangos esperados para el uso cotidiano de la red. En este tipo de tráfico, la duración del flujo, el volumen de datos transferidos y la frecuencia de comunicación suelen estar relacionados con actividades legítimas, como el acceso a servicios, el uso de aplicaciones o las conexiones habituales entre dispositivos.

Esta información sirve como referencia para que un NIDS pueda reconocer el comportamiento esperado de la red. A partir de ese punto de comparación, el sistema puede diferenciar con mayor claridad cuándo un flujo mantiene una actividad normal y cuándo empieza a mostrar cambios que requieren mayor atención [29], [30].

Tráfico de red malicioso:

El tráfico malicioso, en cambio, rompe ese comportamiento regular de la red. Puede aparecer mediante variaciones bruscas en la duración de los flujos, aumento inusual de paquetes, conexiones repetidas hacia determinados puertos o patrones que no corresponden con un uso normal del sistema. Estas señales no significan necesariamente que exista un ataque en todos los casos, pero sí ayudan a identificar comportamientos sospechosos dentro del proceso de detección.

En un sistema NIDS, este tipo de comportamiento resulta importante porque permite identificar posibles intrusiones sin depender únicamente del contenido de los paquetes. Por ello, los modelos de aprendizaje automático analizan estas diferencias para clasificar los registros como tráfico normal o como tráfico potencialmente malicioso [29], [30].

La Figura 3 presenta las diferencias generales entre el tráfico de red normal y malicioso dentro de un NIDS basado en análisis de flujos IP.

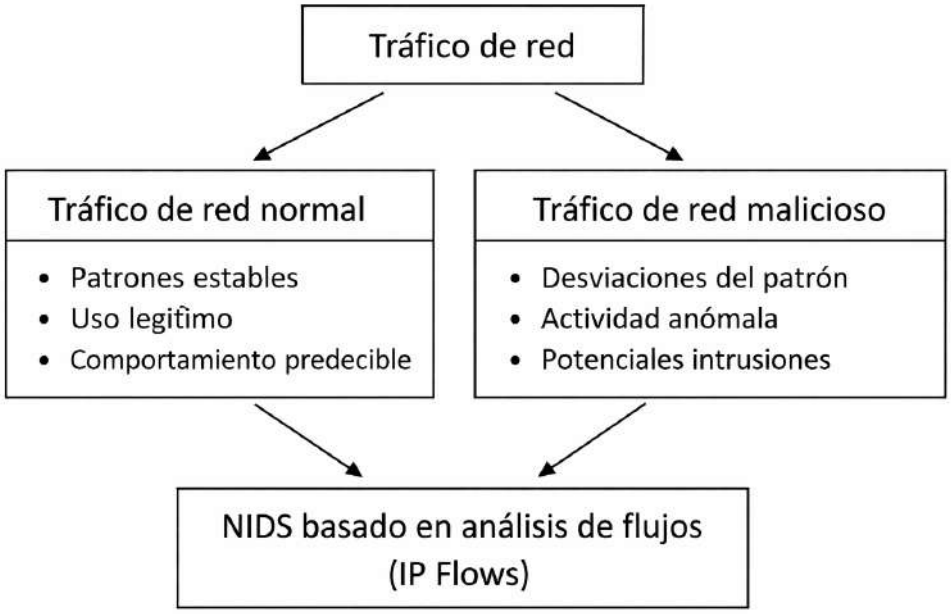

Figura 3.
Diferencias generales entre el tráfico de red normal y malicioso en un NIDS basado en análisis de flujos IP [29].

2.2.7 Arquitectura TCP/IP y modelo OSI

Un NIDS analiza el tráfico que circula por una red a partir de protocolos de comunicación estandarizados. En las redes modernas, la suite TCP/IP constituye el marco de referencia más utilizado; por ello, su comprensión permite identificar qué tipo de datos puede monitorear el sistema y cómo estos son interpretados durante el proceso de detección [31].

De acuerdo con guías técnicas especializadas, un NIDS puede revisar la actividad en distintas capas de esta arquitectura. En la capa de red, por ejemplo, es posible analizar cabeceras IP para reconocer escaneos o intentos de suplantación (spoofing). En la capa de transporte, en cambio, el análisis se centra en segmentos TCP o datagramas UDP, lo que permite identificar conexiones anómalas o ataques de denegación de servicio (DoS) [31].

El análisis por capas permite relacionar eventos que ocurren en distintos niveles del proto-

colo. Por ello, la arquitectura TCP/IP no solo actúa como un medio de transporte de datos, sino también como una referencia para delimitar el alcance del análisis que realiza un NIDS. En esta investigación, este marco es importante porque el sistema propuesto clasifica tráfico de red y necesita interpretar sus características a partir de una base protocolar definida.

2.2.8 Características del tráfico de red

La detección de intrusiones con modelos de aprendizaje automático y profundo depende de las características que se extraen del tráfico de red para representar su comportamiento [11]. Estas características corresponden a datos obtenidos de los flujos o paquetes de red, y son las que el algoritmo utiliza para diferenciar entre actividad normal y maliciosa.

La técnica t-SNE, del inglés *t-distributed Stochastic Neighbor Embedding*, reduce datos de alta dimensión a una representación bidimensional. Esto facilita observar agrupamientos o separaciones entre clases. En este caso, se utiliza para representar de forma gráfica la distribución de características del tráfico normal y malicioso.

La Figura 4 presenta una proyección t-SNE de las características del tráfico de red normal y malicioso.

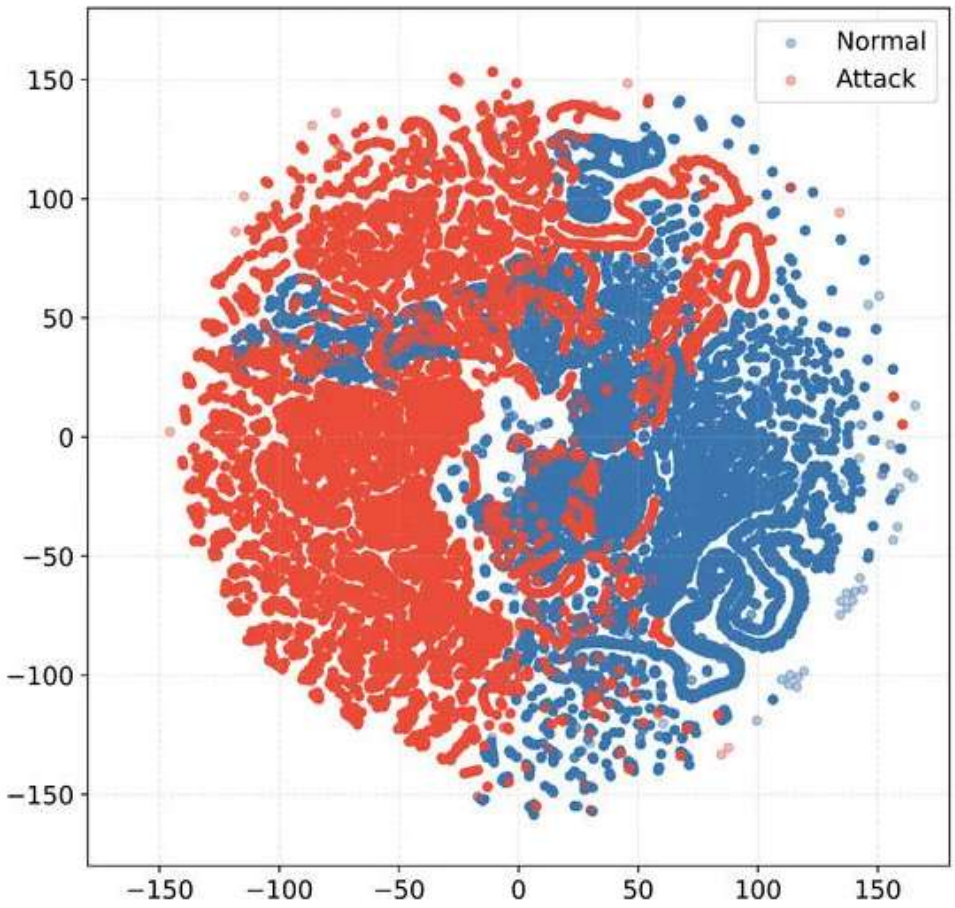

Figura 4.
Proyección t-SNE de las características del tráfico de red normal y malicioso [32].

Una práctica ampliamente aceptada en la literatura consiste en categorizar las características según la capa del protocolo y el tipo de información que capturan. Ferrag et al. destacan que las características básicas de flujo, como direcciones IP, puertos, protocolo y duración de la conexión, conforman la base de cualquier análisis. Por su parte, las características estadísticas temporales, como la tasa de paquetes, el volumen de bytes transmitidos, la frecuencia de flags TCP o la relación entre paquetes entrantes y salientes dentro de una ventana de tiempo, resultan esenciales para identificar patrones de comportamiento anómalo, característicos de ataques como la denegación de servicio distribuida (DDoS) o los escaneos de red [11].

En algunos casos, las características relacionadas con el contenido de los paquetes también pueden ser útiles para detectar exploits específicos a nivel de aplicación.

La selección de características es una etapa importante en el diseño de un sistema de de-

tección de intrusos. Cuando se incluyen demasiadas variables irrelevantes, el modelo puede volverse más complejo, incorporar ruido y presentar problemas de sobreajuste. En ese caso, el sistema obtiene buenos resultados con los datos de entrenamiento, pero pierde capacidad para responder correctamente ante tráfico no conocido [33]. Por esta razón, la selección de características y la reducción de dimensionalidad forman parte del preprocesamiento necesario antes de entrenar un modelo IDS.

En el aprendizaje profundo, una de las ventajas principales es que se reduce parte del trabajo manual asociado a la ingeniería de características. Estos modelos pueden aprender representaciones directamente desde los datos; sin embargo, esto no significa que se pueda dejar de analizar la entrada del modelo o interpretar sus resultados. Sigue siendo necesario conocer qué representa cada característica y cómo se relaciona con el comportamiento de la red. En esta tesis, las características estadísticas obtenidas de los flujos de red se utilizan como base para representar los datos de entrada del modelo.

2.2.9 Flujos de red (NetFlow) vs. paquetes crudos

En un sistema de detección de intrusos basado en red, la forma en que se representa el tráfico es una decisión importante. De manera general, se puede trabajar con paquetes crudos o con flujos de red. Ambos enfoques permiten analizar la comunicación dentro de una red, pero no ofrecen las mismas ventajas cuando se busca entrenar un modelo de aprendizaje automático o aprendizaje profundo [34].

El análisis de paquetes crudos trabaja con la información completa que circula por la red. Esto puede incluir las cabeceras y también la carga útil, conocida como *payload*. Al revisar este nivel de detalle, es posible observar elementos específicos de la comunicación y reconocer ciertos ataques que se ocultan dentro del contenido de aplicaciones, solicitudes web o archivos transmitidos.

No obstante, este tipo de análisis también presenta dificultades. Procesar paquetes comple-

tos requiere mayor capacidad de almacenamiento, más recursos de cómputo y un tratamiento cuidadoso de la información, ya que puede involucrar datos sensibles de los usuarios. Por esta razón, su aplicación puede volverse compleja en redes con grandes volúmenes de tráfico o en escenarios donde se necesita una respuesta cercana al tiempo real [34].

El análisis mediante flujos de red trabaja con una representación más resumida del tráfico. En lugar de inspeccionar todo el contenido de cada paquete, agrupa comunicaciones que comparten elementos comunes, como dirección IP de origen, dirección IP de destino, puertos, protocolo y una ventana temporal. Tecnologías y formatos como NetFlow, IPFIX o sFlow utilizan esta lógica para describir el comportamiento general de una comunicación sin acceder directamente al contenido transportado.

A partir de estos flujos se obtienen variables estadísticas como la duración de la conexión, el número de paquetes, la cantidad de bytes transferidos y las tasas de transmisión. Estos datos permiten observar cómo se comporta una comunicación dentro de la red y detectar variaciones que podrían estar asociadas con actividad maliciosa [34]. En esta investigación se trabaja con flujos de red porque este enfoque se adapta al modelo desarrollado. La CNN 1D no revisa el contenido completo de los paquetes; utiliza una entrada numérica organizada que describe el comportamiento del tráfico. Por eso, las características estadísticas de los flujos sirven como entrada para el modelo y permiten clasificar cada registro como tráfico normal o malicioso.

Este enfoque también ayuda a reducir la carga computacional frente al análisis de paquetes crudos y disminuye la exposición de información sensible. Además, al trabajar con datos organizados en columnas, se facilitan las etapas de limpieza, normalización, transformación y entrenamiento del modelo.

2.2.10 Conjuntos de datos (Datasets) públicos para IDS

La investigación y el desarrollo de sistemas de detección de intrusos basados en aprendizaje automático requieren conjuntos de datos (datasets) públicos, representativos y correctamente

etiquetados para entrenar, validar y comparar de manera objetiva el rendimiento de distintos modelos [35]. La calidad del dataset empleado es fundamental en la validez, utilidad y capacidad de generalización de los resultados obtenidos.

La Tabla III presenta la distribución de clases en algunos datasets utilizados para la evaluación de sistemas IDS.

Tabla III.
Distribución de clases en datasets utilizados para IDS

Dataset	Class-1	Class-2	Class-3	Class-4	Class-5	Class-6
KDD Cup 99	4,113,223	553,301	45,268	18,599	112	–
NSL-KDD	77,054	53,387	14,077	4,833	119	–
CIC-IDS2017	2,358,036	453,438	15,967	1,966	36	21
CSE-CIC-IDS2018	2,856,035	1,289,544	286,191	93,063	513	53

Fuente: Adaptado de Karatas et al. [36].

Los primeros conjuntos de datos utilizados en la investigación de IDS, como KDD Cup 1999 y su versión mejorada NSL-KDD, marcaron un precedente importante para el desarrollo de modelos de detección. Sin embargo, presentan limitaciones frente a los escenarios actuales, debido a que se basan en tráfico de red antiguo, incluyen poca diversidad de ataques modernos y contienen problemas como desbalance de clases y registros duplicados. Estas condiciones pueden generar evaluaciones demasiado favorables y poco realistas sobre el desempeño de los modelos de detección [37].

Para responder a las limitaciones de los conjuntos de datos tradicionales, se han desarrollado datasets más recientes y con mayor nivel de complejidad, orientados a representar escenarios de red más cercanos a condiciones reales. Entre los más utilizados se encuentran CIC-IDS2017 y CSE-CIC-IDS2018, desarrollados por el Canadian Institute for Cybersecurity (CIC), los cuales

han tenido una presencia importante en investigaciones sobre sistemas de detección de intrusiones [35]. Estos conjuntos fueron generados en entornos de laboratorio controlados, con perfiles de comportamiento que buscan aproximarse al uso de usuarios reales. Además, incorporan distintas categorías de ataques, entre ellas infiltración, fuerza bruta, denegación de servicio distribuido (DDoS) y ataques dirigidos a aplicaciones web.

Los datasets de la serie CIC-IDS también incluyen características del flujo de red obtenidas mediante la herramienta CICFlowMeter. Esto permite trabajar con variables previamente calculadas, lo que resulta útil para el entrenamiento y la evaluación de modelos de aprendizaje automático y aprendizaje profundo [35].

Una revisión sistemática sobre conjuntos de datos para IDS indica que datasets como CIC-IDS2018 representan un avance frente a conjuntos anteriores. Sin embargo, todavía existen dificultades relacionadas con la rápida evolución de las amenazas, la estandarización de formatos y la claridad en la metodología usada para generar los datos [37]. Aun con estas limitaciones, los datasets de la serie CIC-IDS siguen siendo útiles para evaluar IDS basados en aprendizaje automático, ya que incluyen ataques variados, características precalculadas y escenarios relativamente actuales.

En esta investigación, la selección del conjunto de datos tomó en cuenta criterios como actualidad, completitud y relación con el problema de estudio. A partir de estos aspectos, se priorizaron datasets que permitan entrenar y evaluar de forma consistente el modelo de red neuronal convolucional propuesto.

2.3 Inteligencia artificial en ciberseguridad

La creciente complejidad, volumen y sofisticación de las amenazas cibernéticas ha llevado a la comunidad de seguridad a buscar soluciones que superen las capacidades de los sistemas tradicionales, basados en reglas estáticas o firmas conocidas. En este contexto, la inteligencia artificial (IA), y en particular los subcampos del aprendizaje automático (ML) y el aprendiza-

je profundo (DL), han pasado a ser paradigmas transformadores para la ciberseguridad [28]. Estos enfoques permiten a los sistemas de seguridad aprender automáticamente patrones y comportamientos complejos a partir de grandes volúmenes de datos, adaptándose así a amenazas evolutivas y mejorando la capacidad de detección proactiva.

Las aplicaciones de la IA en ciberseguridad son diversas e incluyen la detección de malware, el filtrado de phishing, el análisis de comportamiento de usuarios y entidades (UEBA) y, de manera central para esta investigación, la detección de intrusiones en redes [28]. Los métodos de inteligencia artificial ofrecen ventajas importantes para la detección de intrusiones. Entre ellas se encuentran la capacidad de procesar datos de red con muchas variables, la automatización del análisis y la posibilidad de reducir la dependencia de una revisión completamente manual. Además, estos métodos pueden apoyar la identificación de ataques novedosos o de día cero, ya que permiten modelar desviaciones respecto a un comportamiento normal aprendido, en lugar de depender únicamente de coincidencias con patrones predefinidos [11].

Dentro de las técnicas de inteligencia artificial aplicadas a la ciberseguridad, el aprendizaje profundo ha ganado relevancia en la detección de intrusiones en redes. A diferencia de varios algoritmos tradicionales de aprendizaje automático, que dependen de un proceso manual de ingeniería de características, las arquitecturas de aprendizaje profundo, entre ellas las redes neuronales convolucionales (CNN), pueden extraer representaciones en distintos niveles a partir de los datos de entrada [11].

Esta capacidad resulta importante en entornos de red, donde las relaciones entre paquetes, las secuencias temporales y ciertos patrones presentes en los flujos de datos pueden contener indicios de actividad maliciosa. Cuando se entrenan con conjuntos de datos amplios y representativos, los modelos basados en DL tienen mejores condiciones para reconocer estas relaciones complejas. Esto puede reflejarse en una mayor precisión de detección y en una reducción de falsos positivos frente a enfoques convencionales [11].

Por lo tanto, la elección de una red neuronal convolucional para clasificar el tráfico de red se apoya en los avances reportados sobre el uso de aprendizaje profundo en sistemas de detección

de intrusos. El modelo propuesto busca aprovechar esa capacidad de extracción automática de características para mejorar la identificación de tráfico normal y malicioso.

2.3.1 Aplicaciones de la IA en ciberseguridad

El aumento de las ciberamenazas y del tráfico digital ha impulsado el uso de técnicas de inteligencia artificial (IA) y aprendizaje automático (ML) en soluciones de seguridad. Estas tecnologías se emplean en diferentes áreas de la ciberseguridad, sobre todo cuando se requiere analizar grandes volúmenes de información y detectar comportamientos que no siempre son reconocidos mediante reglas tradicionales [38].

Una de sus aplicaciones principales está en la detección de intrusiones y anomalías en redes. En este ámbito, los modelos de IA, especialmente los basados en aprendizaje profundo, analizan patrones de tráfico y ayudan a identificar comportamientos maliciosos que podrían pasar desapercibidos para los sistemas basados en firmas. Esto resulta útil frente a amenazas desconocidas, ataques modificados o nuevas variantes [38].

La Figura 5 presenta algunas aplicaciones de la inteligencia artificial y el aprendizaje automático en ciberseguridad.

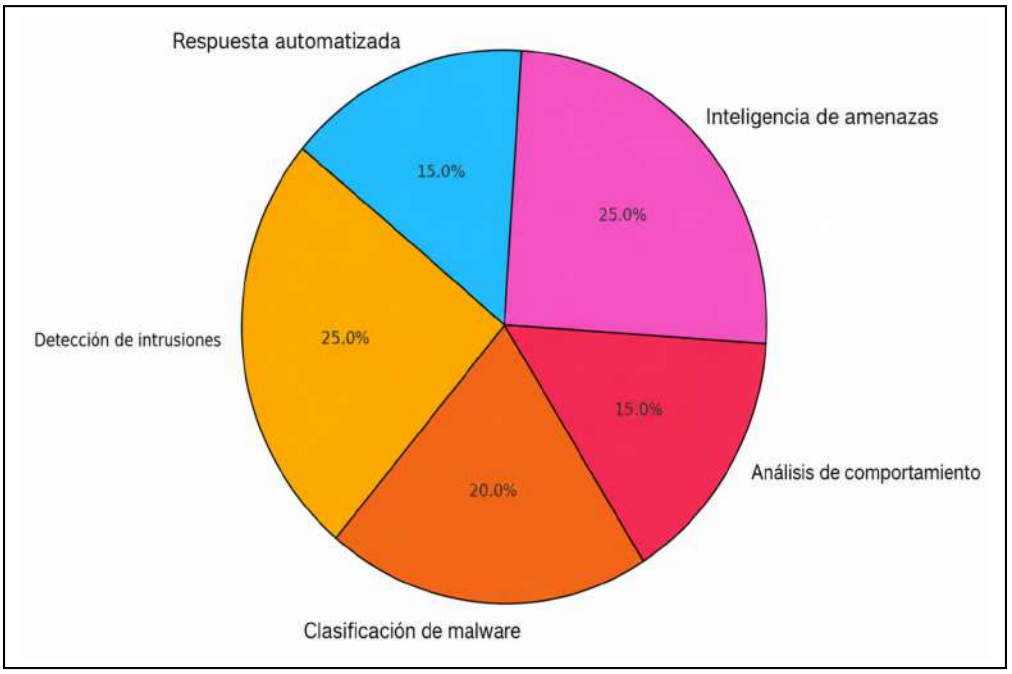

Figura 5.
Aplicaciones de inteligencia artificial y aprendizaje automático en ciberseguridad [39].

Otro campo de aplicación corresponde a la seguridad de *endpoints* y a la detección de malware. En este caso, los algoritmos de IA analizan características estáticas y dinámicas del software con el fin de identificar familias de malware, ransomware y variantes ofuscadas. Por esta razón, suelen integrarse en plataformas avanzadas de detección y respuesta, como EDR y XDR [38].

La inteligencia artificial también se utiliza en el análisis de comportamiento y en la detección de fraudes asociados a entornos web. En este ámbito, los modelos de aprendizaje automático pueden analizar transacciones en línea, patrones de comportamiento de los usuarios y contenido de aplicaciones web para reconocer actividades fraudulentas, bots maliciosos e intentos de suplantación de identidad. Esto permite automatizar parte del proceso de detección y responder mejor ante grandes volúmenes de información [38].

Además, la IA participa en la respuesta a incidentes y en la gestión de eventos de seguridad. Al integrarse con plataformas SOAR y sistemas SIEM, puede ayudar a relacionar alertas, ordenar incidentes según su nivel de criticidad y ejecutar procedimientos de respuesta automatizada [38].

En conjunto, la revisión citada evidencia que la IA y el aprendizaje automático no cumplen

únicamente una función de apoyo dentro de la ciberseguridad. Su aplicación puede mejorar la precisión, la eficiencia y la capacidad de respuesta de las soluciones de protección, especialmente en escenarios donde el volumen de datos y la velocidad de los ataques superan las capacidades del análisis manual [38].

2.3.2 Ventajas del uso de IA para IDS

La incorporación de técnicas de inteligencia artificial, en especial el aprendizaje profundo, ha permitido ampliar las capacidades de los sistemas de detección de intrusos. Su aporte se observa principalmente en la forma en que estos modelos procesan grandes volúmenes de tráfico, reconocen patrones complejos y reducen algunas limitaciones presentes en los enfoques tradicionales de detección [11].

Una ventaja significativa es la capacidad de apoyar la detección de amenazas novedosas y de día cero (zero-day). A diferencia de los sistemas basados en firmas, que son inherentemente reactivos, los modelos de IA basados en detección de anomalías pueden identificar comportamientos previamente no observados al aprender una representación del comportamiento normal del tráfico o del sistema, alertando sobre desviaciones sustanciales [11].

Una de sus principales ventajas es la reducción del trabajo manual en la ingeniería de características En los métodos tradicionales de aprendizaje automático (ML), la elección de variables relevantes del tráfico de red suele depender del criterio de expertos. En cambio, los modelos de aprendizaje profundo (DL), como las redes neuronales convolucionales, pueden aprender patrones directamente desde datos crudos o de bajo nivel, como cabeceras de paquetes o secuencias de bytes. Esto ayuda a identificar relaciones no lineales y comportamientos difíciles de definir manualmente [11].

Otra ventaja se relaciona con el desempeño de los IDS en escenarios de red más complejos. Al analizar relaciones entre distintas variables del tráfico y patrones temporales o espaciales en los datos, los modelos basados en IA pueden mejorar la detección y reducir falsas alarmas frente

a métodos estadísticos tradicionales o reglas simples. Esta capacidad es útil frente a ataques más elaborados o diseñados para evadir mecanismos de detección tradicionales [11].

Finalmente, la IA también ayuda a procesar grandes volúmenes de tráfico. Una vez entrenados, los modelos de DL pueden analizar datos de red en tiempo real con un costo computacional manejable. Esto permite aplicarlos en redes de alta velocidad, donde el análisis manual o basado solo en firmas resulta poco práctico [11]. Estas ventajas posicionan a la IA, y en particular al DL, como una base tecnológica relevante para la próxima generación de sistemas de detección de intrusos.

2.3.3 Desafíos y limitaciones de la IA aplicada a la ciberseguridad

El uso de soluciones basadas en inteligencia artificial y aprendizaje profundo en ciberseguridad todavía presenta varias limitaciones. Estas no dependen solo del modelo utilizado, sino también de la calidad de los datos, la capacidad computacional disponible y la forma en que el sistema se integra en un entorno real de seguridad [40]. Por eso, estos aspectos deben tomarse en cuenta al diseñar sistemas de detección que sean confiables, comprensibles y aplicables en un entorno real.

Uno de los principales desafíos es la disponibilidad y calidad de los datos. Los modelos de IA necesitan conjuntos de datos amplios, actualizados, representativos y bien etiquetados. Sin embargo, en ciberseguridad no siempre es fácil acceder a datos reales de ataques, sobre todo cuando se trata de amenazas nuevas, avanzadas o poco frecuentes. Además, muchos datasets presentan un fuerte desbalance entre clases, con una mayor cantidad de registros normales frente a ejemplos maliciosos. Esta condición puede generar modelos orientados hacia la clase mayoritaria y reducir su capacidad para detectar intrusiones menos frecuentes, pero críticas [40].

Otro problema importante es la falta de explicabilidad, conocida como inteligencia artificial explicable o XAI. Esta limitación se presenta con mayor fuerza en modelos complejos, como las redes neuronales profundas. En ciberseguridad, una alerta no solo debe ser precisa, sino que

también debe poder interpretarse. Los analistas necesitan entender por qué el sistema generó una alerta, validar el nivel de riesgo y definir una respuesta adecuada. Cuando el modelo funciona como una "caja negra", disminuye la confianza en sus resultados y se dificulta su auditoría, ajuste y mejora continua [40].

También deben considerarse los ataques adversarios (*adversarial attacks*). En este tipo de amenaza, el atacante modifica los datos de entrada mediante pequeñas perturbaciones que pueden ser difíciles de percibir para una persona, pero suficientes para alterar la decisión del modelo. Como resultado, una actividad maliciosa podría ser clasificada como benigna. Por ello, fortalecer la resistencia de los IDS basados en IA frente a este tipo de manipulación sigue siendo un tema abierto dentro de la investigación en ciberseguridad [40].

Finalmente, también hay dificultades al implementar y usar estos sistemas. El entrenamiento y la ejecución de modelos complejos pueden exigir una capacidad computacional alta, lo que limita su uso en organizaciones con recursos técnicos reducidos. A esto se suma la dificultad de integrarlos con arquitecturas de seguridad ya existentes y la necesidad de contar con personal que entienda tanto de ciberseguridad como de ciencia de datos. Estos aspectos deben considerarse antes de implementar soluciones basadas en IA, ya que pueden afectar su funcionamiento en un entorno real [40].

2.4 Redes neuronales

Las redes neuronales artificiales (ANN) forman parte de la base conceptual del aprendizaje profundo (*Deep Learning*) y constituyen el tipo de modelo adoptado en esta investigación para el sistema de detección propuesto. Su utilidad radica en que pueden aprender representaciones jerárquicas directamente a partir de los datos, lo que resulta conveniente en tareas de clasificación y reconocimiento de patrones complejos. En el contexto de este estudio, esa capacidad es relevante para diferenciar entre tráfico de red normal y tráfico malicioso [41]. En esta sección se revisan sus fundamentos teóricos, desde la estructura básica de una neurona artificial hasta arquitecturas más especializadas, como las redes neuronales convolucionales, junto con los

procesos de preprocesamiento requeridos para su entrenamiento y funcionamiento.

La Tabla IV presenta algunas arquitecturas de redes neuronales y su uso en sistemas de detección de intrusos.

Tabla IV.
Arquitecturas de redes neuronales y su uso en IDS

Arquitectura	Funcionamiento esencial	Uso en IDS
MLP	Red multicapa con propagación hacia adelante y activaciones no lineales.	Clasificación de tráfico normal y malicioso a partir de características derivadas de flujos de red.
CNN	Emplea operaciones de convolución y *pooling* para extraer patrones jerárquicos de manera automática.	Detección de intrusiones mediante el análisis de patrones estructurales presentes en el tráfico de red.
RNN	Modela relaciones temporales a través de conexiones recurrentes entre secuencias de entrada.	Análisis secuencial del tráfico y detección de ataques dependientes del tiempo.
LSTM	Variante de RNN con compuertas que permiten conservar información relevante durante intervalos prolongados.	Identificación de ataques persistentes y de comportamientos anómalos sostenidos en el tiempo.
Autoencoder	Aprende representaciones compactas de los datos y reconstruye patrones considerados normales.	Detección de anomalías a partir de desviaciones respecto al tráfico legítimo.

Fuente: Elaboración propia con base en Pouyanfar et al. [41] y Mortezapour et al. [42].

Las redes neuronales profundas no responden a una sola estructura, sino a varias arquitecturas que difieren en la forma de procesar la información y en los patrones que pueden aprender. Estas diferencias condicionan su desempeño y su utilidad en tareas de análisis y clasificación,

sobre todo cuando se trabaja con datos de alta dimensionalidad o con secuencias que presentan dependencia temporal [42].

En este grupo, las redes neuronales convolucionales (CNN) destacan por el uso de operaciones de convolución, las cuales permiten extraer características locales y jerárquicas directamente desde los datos de entrada. Gracias a ello, disminuye la dependencia de una selección manual de características y se facilita su aplicación en tareas de clasificación más complejas.

Las redes neuronales recurrentes (RNN), en cambio, están orientadas al análisis de secuencias. Sus conexiones recurrentes permiten modelar relaciones temporales entre observaciones consecutivas. No obstante, las RNN tradicionales presentan dificultades cuando deben conservar información durante periodos largos, en gran medida por el problema del desvanecimiento del gradiente durante el entrenamiento.

La Figura 6 presenta las estructuras generales de las redes neuronales convolucionales y recurrentes.

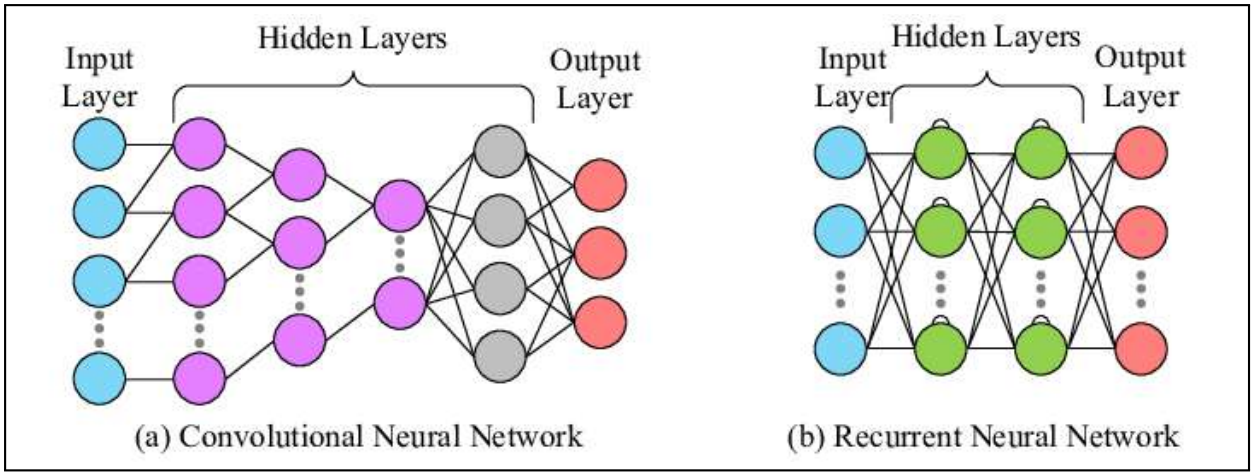

Figura 6.
Estructuras generales de redes neuronales convolucionales y redes neuronales recurrentes [43].

Para reducir esta limitación surgieron variantes más avanzadas, entre ellas Long Short-Term Memory (LSTM) y Gated Recurrent Units (GRU). Ambas incorporan compuertas que regulan el flujo de información y permiten retener datos relevantes durante más tiempo. Las redes LSTM suelen emplearse cuando se requiere modelar secuencias largas, mientras que las GRU ofre-

cen una estructura más simple y con menor costo computacional, aunque con un rendimiento comparable en varias tareas secuenciales [42].

La Figura 7 presenta las arquitecturas generales de los modelos LSTM y GRU.

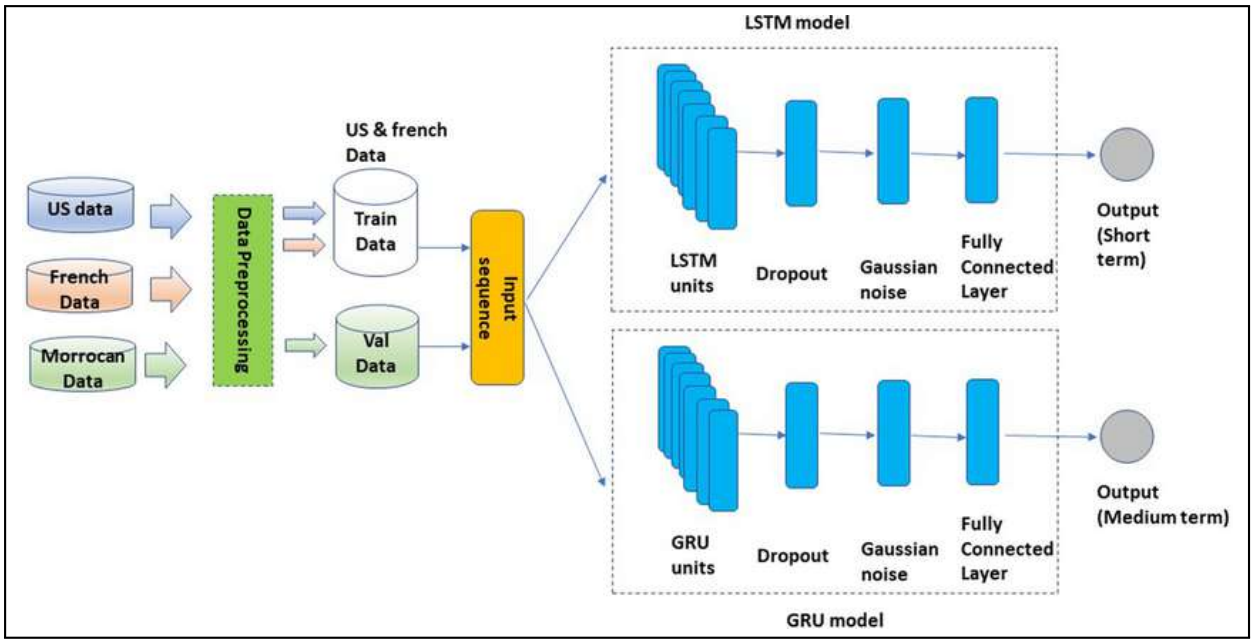

Figura 7.
Arquitecturas generales de los modelos Long Short-Term Memory y Gated Recurrent Units [44].

2.4.1 Concepto básico de neurona artificial, función de activación, capas

Las redes neuronales artificiales (ANN) son modelos computacionales inspirados, de manera general, en ciertos principios del funcionamiento neuronal biológico. Su propósito es aprender patrones presentes en los datos mediante unidades de procesamiento llamadas neuronas artificiales o nodos. Cada neurona recibe una o varias señales de entrada, les asigna un peso, calcula una suma ponderada y luego aplica una función de activación para generar una salida [41].

La función de activación permite incorporar no linealidad al modelo. Esto es importante porque muchos problemas de clasificación no pueden resolverse únicamente mediante relaciones lineales. Entre las funciones más utilizadas se encuentran ReLU (*Rectified Linear Unit*), la función sigmoide y la tangente hiperbólica [41].

En una red neuronal, las neuronas se organizan por capas. La capa de entrada recibe los datos iniciales; las capas ocultas realizan el procesamiento interno y aprenden representaciones intermedias; finalmente, la capa de salida entrega el resultado del modelo, por ejemplo, una clase o categoría. En las arquitecturas totalmente conectadas (*fully connected*), cada neurona de una capa se conecta con todas las neuronas de la capa siguiente [41]. Esta estructura permite abordar tareas de clasificación, como la diferenciación entre tráfico de red normal y tráfico malicioso.

2.4.2 Aprendizaje supervisado y proceso de entrenamiento

El aprendizaje supervisado se utiliza con frecuencia para entrenar redes neuronales en tareas de clasificación, como la diferenciación entre tráfico de red normal y tráfico malicioso. En este enfoque, el modelo aprende a partir de ejemplos previamente etiquetados, donde cada registro de entrada se relaciona con una salida conocida. A partir de esa información, la red ajusta su comportamiento para reconocer patrones y realizar predicciones sobre datos que no fueron utilizados durante el entrenamiento [41].

El entrenamiento de una red neuronal puede entenderse como un proceso de ajuste progresivo. Durante este proceso, los parámetros internos del modelo, principalmente los pesos y sesgos, se modifican con el objetivo de reducir el error entre la predicción generada y la etiqueta real. Este error se calcula mediante una función de pérdida. En problemas de clasificación binaria, una de las funciones más empleadas es la entropía cruzada binaria, conocida como *binary cross-entropy*, debido a que permite penalizar las predicciones incorrectas y orientar el aprendizaje del modelo [41].

El ajuste de los parámetros se realiza mediante el algoritmo de retropropagación, conocido como *backpropagation*. Este procedimiento calcula cómo cambia la función de pérdida respecto a cada parámetro de la red y permite determinar la dirección en la que deben actualizarse los pesos para disminuir el error. Posteriormente, un optimizador, como Adam, utiliza esta información para realizar las actualizaciones de forma eficiente durante el entrenamiento [41].

La Figura 8 presenta el proceso de entrenamiento supervisado de una red neuronal para tareas de clasificación.

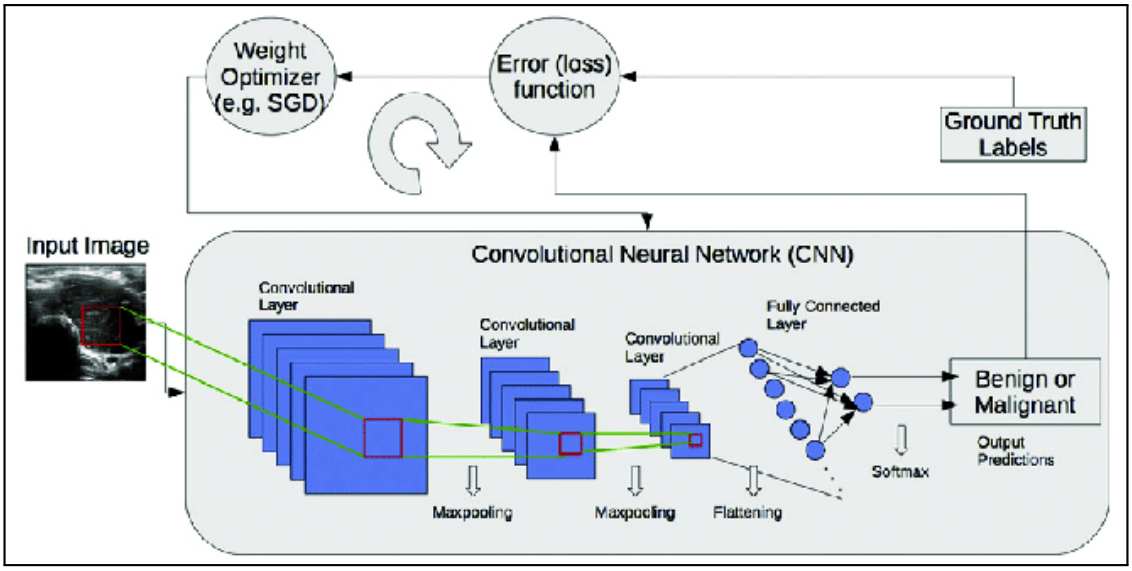

Figura 8.
Proceso de entrenamiento supervisado de una red neuronal para clasificación, incluyendo el cálculo de la pérdida [45].

El entrenamiento se desarrolla a lo largo de varias épocas. En cada época, el modelo procesa los datos de entrenamiento, genera una salida, calcula el error y actualiza sus parámetros internos. Una época corresponde a una revisión completa del conjunto de datos utilizado para entrenar.

Durante este proceso puede aparecer sobreajuste, también denominado *overfitting*. Esto ocurre cuando el modelo se adapta demasiado a los datos de entrenamiento y pierde capacidad para responder correctamente ante información nueva. Para reducir este riesgo, se pueden emplear estrategias como la separación de un conjunto de prueba, la validación cruzada y técnicas de regularización, con el fin de evaluar mejor la capacidad de generalización del modelo final [41].

2.4.3 Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN) surgieron inicialmente en el campo de la visión por computadora; sin embargo, su uso se ha extendido a otros problemas de clasificación,

entre ellos la detección de intrusiones en redes (NIDS). En este ámbito, se han empleado para diferenciar tráfico normal y malicioso debido a su capacidad de extraer patrones relevantes a partir de los datos de entrada [12], [46], [47]. Estas características pueden provenir de distintas representaciones del tráfico, como imágenes 2D, secuencias 1D o matrices de flujos. De esta manera, las CNN reducen la dependencia de una ingeniería manual de características, común en métodos clásicos de aprendizaje automático (*Machine Learning*) [12], [46].

El funcionamiento de las CNN se basa en operaciones de convolución. Mediante filtros que recorren la entrada, el modelo puede identificar patrones locales relacionados con combinaciones de puertos, protocolos, tamaños de paquetes o intervalos de tiempo asociados a posibles comportamientos de ataque [12]. Luego, las capas posteriores integran estos patrones iniciales para formar representaciones de mayor complejidad. Esto permite analizar relaciones entre características del tráfico que podrían ser difíciles de reconocer mediante procedimientos manuales o métodos estadísticos convencionales.

Diversos estudios recientes han evaluado el uso de CNN en NIDS. Por ejemplo, investigaciones basadas en datasets como CIC-IDS2017 muestran que las CNN, especialmente cuando se combinan con capas recurrentes como LSTM, pueden alcanzar altas tasas de detección y reducir falsas alarmas en distintos tipos de ataques, entre ellos DoS, DDoS, infiltración y ataques web [46]. Otros enfoques convierten los flujos de red en imágenes 2D para aplicar arquitecturas CNN estándar en la clasificación del tráfico. Esta estrategia aprovecha la capacidad de estas redes para reconocer patrones espaciales, adaptándola al análisis de seguridad de red [47].

Además de su desempeño en clasificación, las CNN presentan ventajas computacionales asociadas a la compartición de parámetros (*parameter sharing*) [12]. Por ello, en esta tesis se adopta una arquitectura CNN para el sistema de clasificación de tráfico de red, con el propósito de identificar diferencias entre el tráfico de red normal y malicioso de forma precisa y eficiente.

2.4.4 Preprocesamiento de datos para redes neuronales

El preprocesamiento de datos es una etapa necesaria antes de entrenar modelos de aprendizaje automático o aprendizaje profundo. Los datos crudos no siempre se encuentran en una forma adecuada para ser procesados por el algoritmo, por lo que deben revisarse, limpiarse y transformarse en características que puedan ser interpretadas durante el entrenamiento. Esta preparación influye en la precisión del modelo, en la velocidad con la que aprende y en su capacidad para responder ante datos que no fueron utilizados previamente [14], [48].

En los sistemas de detección de intrusos, esta etapa adquiere mayor importancia porque el tráfico de red puede incluir ruido, registros incompletos, clases desbalanceadas y variables con escalas muy diferentes. Si estos aspectos no se tratan antes del entrenamiento, el modelo puede aprender patrones poco representativos o mostrar un desempeño artificialmente alto sin detectar correctamente los ataques.

Una de las tareas más importantes dentro del preprocesamiento es la normalización o estandarización de los datos. Las características del tráfico de red, como la duración de un flujo, el número de paquetes o la cantidad de bytes transferidos, pueden tener rangos muy diferentes. Si estas variables no se escalan, las que tienen valores más altos pueden influir más durante el aprendizaje del modelo. Para evitar este problema, se usan métodos como la normalización Min-Max, que ajusta los valores a un intervalo definido, por ejemplo [0,1], o la estandarización Z-score, que transforma los datos para obtener una media cercana a 0 y una desviación estándar igual a 1 [48].

También es necesario revisar las variables categóricas presentes en el tráfico de red. Algunas características, como el tipo de protocolo TCP, UDP o ICMP, así como ciertas banderas de conexión, no se representan de forma numérica directa. Sin embargo, modelos como las CNN requieren entradas expresadas mediante valores numéricos. Por esta razón, estas categorías deben codificarse antes del entrenamiento. Una técnica utilizada para este propósito es la codificación *one-hot*, que representa cada categoría mediante columnas binarias, donde el valor 1 indica

presencia y el valor 0 indica ausencia en una muestra determinada [14].

Otro aspecto que debe considerarse es el desbalance de clases, un problema frecuente en los conjuntos de datos utilizados para IDS. En muchos casos, el tráfico malicioso aparece en menor proporción que el tráfico normal. Si el modelo se entrena directamente con esta distribución, puede favorecer a la clase mayoritaria y mostrar una exactitud general aparentemente alta, aunque con poca capacidad para detectar ataques reales. Para disminuir este sesgo, se aplican técnicas de muestreo, como el sobremuestreo de la clase minoritaria mediante copias o ejemplos sintéticos, o el submuestreo de la clase mayoritaria [48].

En esta investigación, el preprocesamiento permitió preparar los datos antes del entrenamiento de la red neuronal convolucional. Para ello, se consideraron procesos de limpieza, transformación numérica, normalización y tratamiento del desbalance de clases, con el fin de obtener una entrada más consistente para la clasificación de tráfico normal y malicioso.

2.5 Trabajos relacionados

En [49] se presentó un modelo de detección de intrusiones basado en redes neuronales convolucionales (CNN). El trabajo buscó responder a una limitación de los IDS tradicionales: su dependencia de patrones definidos previamente. Esto puede dificultar la detección de ataques nuevos o variantes que aún no han sido registradas. Para el entrenamiento se utilizó el conjunto de datos CICIDS2017, que contiene tráfico benigno y diferentes clases de ataques. Con ello, los autores evaluaron el modelo frente a amenazas conocidas y también ante patrones no vistos durante el entrenamiento.

Para la clasificación multiclase, los autores aplicaron técnicas de preprocesamiento que facilitaron el aprendizaje de patrones en el tráfico de red. Los resultados alcanzaron una precisión global de hasta 99,78 %. Además, el modelo logró reconocer ataques que no formaban parte de los datos de entrenamiento, lo que muestra una respuesta favorable ante escenarios cambiantes. Este estudio apoya el uso de CNN en sistemas IDS para detectar intrusiones emergentes [49].

En [50] se desarrolló un modelo CNN-IDS enfocado en automatizar el aprendizaje de características del tráfico de red. La propuesta parte de una limitación frecuente de los métodos tradicionales: la selección manual de variables relevantes. Este proceso puede afectar el rendimiento del sistema cuando se trabaja con grandes volúmenes de datos o con ataques más complejos. La arquitectura utilizada está compuesta por tres capas convolucionales 2D con activación ReLU, una capa *Flatten* y una capa totalmente conectada, entrenadas con el conjunto de datos CICIDS2017.

Los resultados reportan una precisión de 99,78 %, con valores de *recall* y *F1-score* cercanos al 100 %, además de una baja pérdida durante el entrenamiento. Esto muestra un buen desempeño en la clasificación de eventos de red. Los autores también indican que el modelo tuvo tiempos adecuados de entrenamiento e inferencia, lo que puede ser útil cuando se requiere análisis en tiempo real. En general, el estudio muestra que las CNN pueden apoyar la detección de ataques sofisticados y mejorar la clasificación automática del tráfico [50].

Por su parte, en [51] se propuso un marco de detección de intrusiones basado en redes neuronales convolucionales profundas (DCNN). El estudio se enfoca en escenarios donde el tráfico de red debe analizarse de forma continua y con tiempos de respuesta reducidos. En estos entornos, los IDS tradicionales pueden generar varias falsas alarmas o responder con menor precisión frente a ataques avanzados. Para comprobar el desempeño del modelo, los autores lo evaluaron con distintos conjuntos de datos, entre ellos ISCX2012, DDoS de Kaggle, CICIDS2017 y CICIDS2018. Esta evaluación permitió revisar cómo respondió el modelo ante distintas condiciones de tráfico y categorías de ataque.

La arquitectura DCNN se diseñó para reconocer patrones complejos relacionados con actividades intrusivas y clasificar eventos maliciosos. Según los resultados reportados, el modelo alcanzó precisiones entre 99,79 % y 100 %, además de bajas tasas de falsos positivos. También presentó tiempos adecuados de entrenamiento e inferencia, lo que puede facilitar su uso en sistemas que necesitan respuesta en tiempo real. Con estos resultados, el estudio plantea que los modelos basados en DCNN pueden ser una alternativa frente a métodos tradicionales, especialmente cuando se busca mejorar la detección de ataques sofisticados o adaptativos [51].

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1 Enfoque metodológico

3.1.1 Enfoque de la investigación

Este trabajo de investigación se desarrolló bajo un enfoque cuantitativo, debido a que se basa en los análisis sistemáticos de datos extraídos del tráfico de red. La elección de este enfoque responde a la necesidad de procesar grandes volúmenes de información y desarrollar un sistema automatizado de detección de anomalías mediante técnicas de aprendizaje profundo.

El enfoque cuantitativo permite trabajar con datos medibles y revisar los resultados mediante procedimientos estadísticos y matemáticos. En este estudio, ayudó a analizar el comportamiento del tráfico de red e identificar patrones útiles para diferenciar entre tráfico normal y tráfico malicioso.

Los datos obtenidos del tráfico de red son importantes para la investigación porque permiten representar, de forma estructurada, varias características asociadas a la comunicación entre dispositivos dentro de una red informática. Entre estas características se consideran la duración de las conexiones, el número de paquetes transmitidos, el tamaño de los paquetes, los protocolos utilizados, los puertos de comunicación y otros indicadores relacionados con el comportamiento de los flujos de datos.

La adopción de este enfoque hizo posible aplicar técnicas computacionales orientadas al análisis de datos, particularmente algoritmos de aprendizaje profundo basados en redes neuronales convolucionales (CNN). Este tipo de modelo requiere datos estructurados para su entrenamien-

to, por lo que la organización y preparación previa de la información resultan necesarias dentro del proceso metodológico.

La orientación cuantitativa también permitió medir el desempeño del modelo mediante métricas estadísticas. Estas métricas facilitan la evaluación del sistema al cuantificar su precisión y su capacidad de clasificar el tráfico de red e identificar eventos maliciosos.

Otro aspecto importante de este enfoque es que permite repetir los experimentos realizados. En este estudio, la preparación de los datos, el entrenamiento del modelo y la evaluación del sistema se hicieron con herramientas tecnológicas que permiten ejecutar nuevamente las mismas operaciones bajo condiciones controladas. Esto ayuda a mantener la consistencia de los resultados y facilita su verificación posterior.

Además, el enfoque cuantitativo permitió trabajar con grandes volúmenes de información provenientes del dataset seleccionado. Esto es relevante en seguridad informática, porque el análisis manual del tráfico de red resulta poco práctico frente a la cantidad de datos que se generan continuamente. Por ello, el uso de modelos basados en inteligencia artificial permite automatizar parte del proceso de detección y evaluar el comportamiento del sistema frente a diferentes volúmenes y condiciones de datos.

Finalmente, esta orientación metodológica facilitó la comparación entre distintas configuraciones del modelo, lo que permitió seleccionar aquella con mejor desempeño en términos de precisión y eficacia. De esta manera, el enfoque cuantitativo aportó una base adecuada para desarrollar y evaluar un sistema de detección de intrusiones en redes informáticas.

3.1.2 Tipo de investigación

Este proyecto se desarrolla como una investigación aplicada y experimental. Es aplicada porque parte de un problema concreto de la seguridad informática: la detección de intrusiones en redes. A partir de este problema, se propone un sistema apoyado en técnicas de inteligencia artificial, específicamente en una red neuronal convolucional (CNN).

También se considera experimental, porque el trabajo incluyó la construcción, configuración, entrenamiento y evaluación de un modelo computacional.

El carácter aplicado de la investigación se observa en la manera en que los conceptos teóricos fueron llevados a una solución funcional. En este caso, se utilizaron fundamentos relacionados con redes neuronales convolucionales, procesamiento de datos y aprendizaje automático para construir un sistema capaz de diferenciar entre tráfico normal y tráfico malicioso.

De esta forma, el estudio no se queda solo en una revisión conceptual sobre los sistemas de detección de intrusiones. También incluye una implementación técnica que analiza registros de tráfico de red y los clasifica en dos grupos: tráfico normal y tráfico malicioso. Este enfoque práctico busca aportar una herramienta de apoyo para identificar amenazas en entornos de red.

El desarrollo del sistema requirió herramientas del área computacional. Se usaron lenguajes de programación, bibliotecas de aprendizaje automático y plataformas para el análisis y visualización de datos. Estos recursos permitieron preparar la información, entrenar el modelo y revisar su desempeño mediante métricas cuantificables.

Por otra parte, la investigación tiene un componente experimental, ya que el modelo CNN fue evaluado bajo condiciones controladas. Su arquitectura fue definida y configurada a partir de datos previamente procesados. Durante esta etapa se realizaron pruebas con distintas configuraciones, lo que permitió observar el comportamiento del modelo y su capacidad para reconocer patrones relacionados con tráfico normal y malicioso.

El proceso experimental incluyó el entrenamiento, ajuste y validación del modelo. Para esta etapa se usó un conjunto de datos para el aprendizaje de la red neuronal y luego se evaluó el sistema con datos independientes. Esta separación permitió revisar si el modelo respondía bien ante registros que no fueron usados durante el entrenamiento.

Además, se aplicaron técnicas de normalización de características y balanceo de datos para mejorar el entrenamiento. Estas acciones ayudaron a reducir posibles sesgos derivados del desbalance entre tráfico normal y malicioso, y permitieron generar una entrada más consistente

para el modelo.

También se trabajó con subconjuntos de datos generados bajo criterios definidos, con el fin de revisar el comportamiento del sistema frente a diferentes volúmenes de información. Esto permitió identificar ajustes necesarios dentro del proceso de clasificación y analizar la estabilidad del modelo frente a distintas condiciones de prueba.

En conjunto, la combinación de investigación aplicada y experimental permitió desarrollar un sistema funcional orientado a la detección de intrusiones en redes. Asimismo, hizo posible evaluar su desempeño mediante métricas objetivas, lo que aporta una base cuantitativa para analizar la eficacia del modelo propuesto.

3.2 Fuente de datos

3.2.1 Descripción del dataset CICIDS2017

La selección del conjunto de datos fue un aspecto importante para el desarrollo del sistema basado en aprendizaje automático. En este trabajo de tesis se empleó el dataset CICIDS2017, desarrollado por el Canadian Institute for Cybersecurity [52]. Este conjunto de datos ha sido utilizado en diversos estudios sobre sistemas de detección de intrusiones, debido a que incorpora tráfico benigno y diferentes tipos de ataques dentro de escenarios controlados de red.

Algunos trabajos han usado CICIDS2017 como base para comparar el rendimiento de modelos de Machine Learning y Deep Learning. Xu y Liu [53] lo emplearon para evaluar modelos de clasificación, mientras que Tripathy et al. [54] lo incluyeron dentro de una revisión sistemática de datasets utilizados en el área. En estos estudios, CICIDS2017 aparece junto a otros conjuntos de datos como NSL-KDD y UNSW-NB15, lo que muestra su presencia dentro de la investigación reciente sobre IDS.

Aunque existen otros datasets, como KDD Cup 1999, NSL-KDD, UNSW-NB15 y CSE-CIC-IDS2018, CICIDS2017 resulta pertinente para los objetivos de esta tesis. Su valor principal

está en que ofrece escenarios de tráfico más actuales que los conjuntos de datos clásicos y permite trabajar con distintas categorías de comportamiento en red. La evaluación de IDS modernos depende en gran medida del tipo de dataset utilizado, ya que el rendimiento del sistema puede variar según el entorno, las clases de ataque y las características disponibles [55].

Además, CICIDS2017 facilita la validación y comparación de resultados, ya que ha sido utilizado en investigaciones previas como conjunto de referencia. La vigencia y representatividad del dataset influyen en la evaluación de soluciones de inteligencia artificial aplicadas a ciberseguridad [38]. En este sentido, su uso permite analizar el comportamiento del modelo propuesto frente a registros etiquetados de tráfico normal y malicioso.

Entre los criterios considerados para seleccionar este dataset se encuentran la disponibilidad de datos etiquetados, la diversidad de escenarios incluidos, la cantidad de características disponibles y la calidad de la información contenida. La presencia de etiquetas que identifican el tipo de tráfico permite entrenar modelos supervisados y evaluar su capacidad para distinguir entre comportamiento normal y actividad maliciosa.

La Figura 9 presenta los archivos CSV del dataset CICIDS2017 utilizados para el desarrollo del sistema.

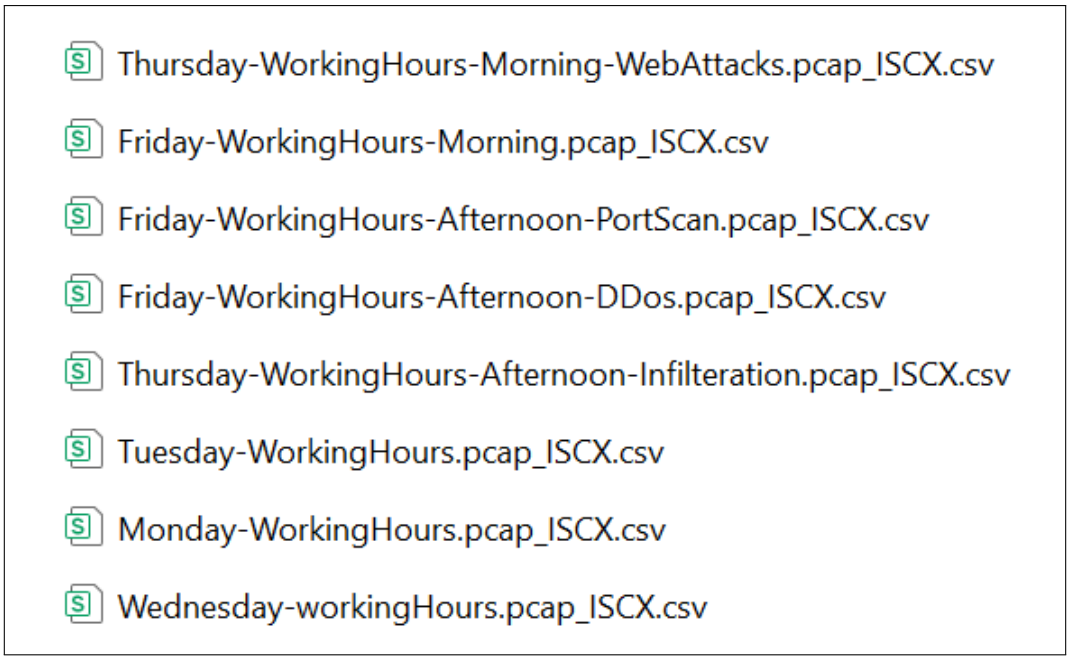

Figura 9.
Dataset de prueba [52].

El dataset presenta una estructura organizada en múltiples archivos correspondientes a diferentes días de monitoreo, lo cual facilita su procesamiento progresivo y permite analizar distintos escenarios de tráfico dentro de la red.

3.2.2 Estructura del dataset

Cada archivo del dataset contiene registros correspondientes a conexiones realizadas dentro de la red. Estos registros incluyen características que describen el comportamiento de cada flujo de datos y permiten analizar distintos aspectos de la comunicación.

En este estudio se utilizaron 79 características numéricas extraídas del dataset. Las variables seleccionadas corresponden a atributos cuantificables del tráfico de red, por lo que pueden ser procesadas directamente durante el entrenamiento del modelo de red neuronal convolucional. El uso de este tipo de características facilita el tratamiento matemático de los datos y permite representar el comportamiento de los flujos de manera estructurada.

Las características consideradas describen elementos como el tiempo de conexión, el vo-

lumen de datos transmitidos, la frecuencia de paquetes, el tamaño de los paquetes y otros indicadores asociados al comportamiento del flujo de red. A partir de estas variables, el modelo puede reconocer diferencias dentro de los registros correspondientes a tráfico normal y registros asociados a actividad maliciosa.

De igual manera, cada registro del dataset contiene una etiqueta que identifica el tipo de tráfico analizado. Esta etiqueta se utilizó como referencia durante el proceso de entrenamiento del modelo, ya que permitió clasificar los datos en dos grupos: tráfico normal y tráfico malicioso.

La Figura 10 presenta los registros y atributos del dataset en formato tabular.

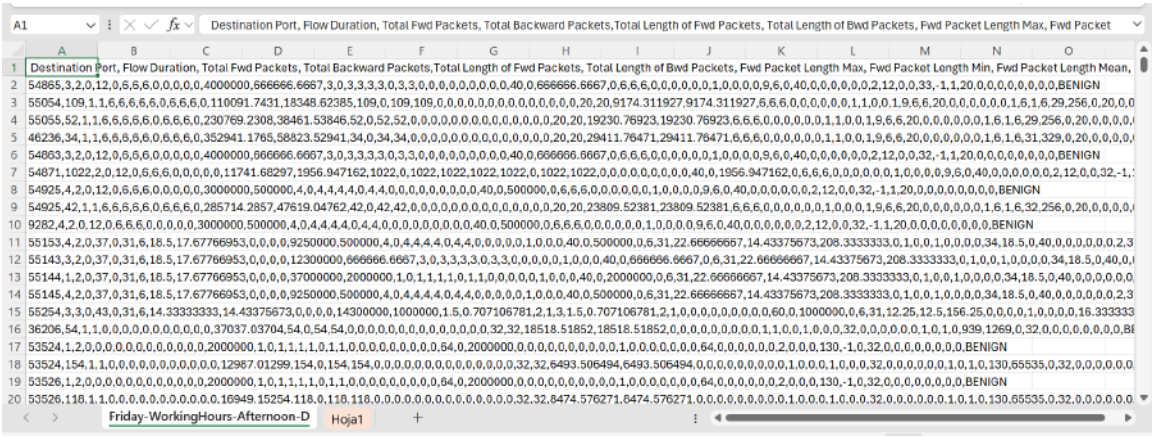

Figura 10.
Registros y atributos del dataset en formato tabular [52].

3.2.3 Conversión de etiquetas

Una vez cargado el dataset, se realizó la conversión de las etiquetas originales del conjunto de datos. Este paso fue necesario para adaptar las clases categóricas a un formato compatible con el modelo de clasificación binaria.

El dataset CICIDS2017 contiene etiquetas asociadas al tráfico registrado, tanto para tráfico normal como para distintos tipos de ataques informáticos. Para esta investigación, el problema se simplificó a una clasificación binaria, diferenciando únicamente entre tráfico normal y tráfico malicioso.

La conversión consistió en asignar el valor numérico 0 a los registros identificados como tráfico normal, correspondientes a la etiqueta “BENIGN” dentro del dataset. Por otro lado, todos los registros asociados a cualquier tipo de ataque fueron agrupados en una sola categoría y se les asignó el valor numérico de 1.

Esta transformación permitió utilizar las etiquetas como variables de salida durante el entrenamiento del modelo de red neuronal convolucional. Al reducir las clases originales a dos categorías, el modelo se centró en diferenciar el comportamiento normal de la actividad maliciosa dentro del tráfico de red.

3.3 Diseño metodológico

El diseño metodológico ayudó a organizar el desarrollo del sistema, desde la selección del conjunto de datos hasta la evaluación final del modelo. El proceso se dividió en fases consecutivas: preparación de la información, entrenamiento de la red neuronal convolucional y revisión del desempeño del sistema de detección.

La Figura 11 resume el flujo seguido durante el desarrollo del sistema. Este esquema permite visualizar el orden de las actividades realizadas y la relación entre las etapas de preparación de datos, entrenamiento del modelo y evaluación de resultados.

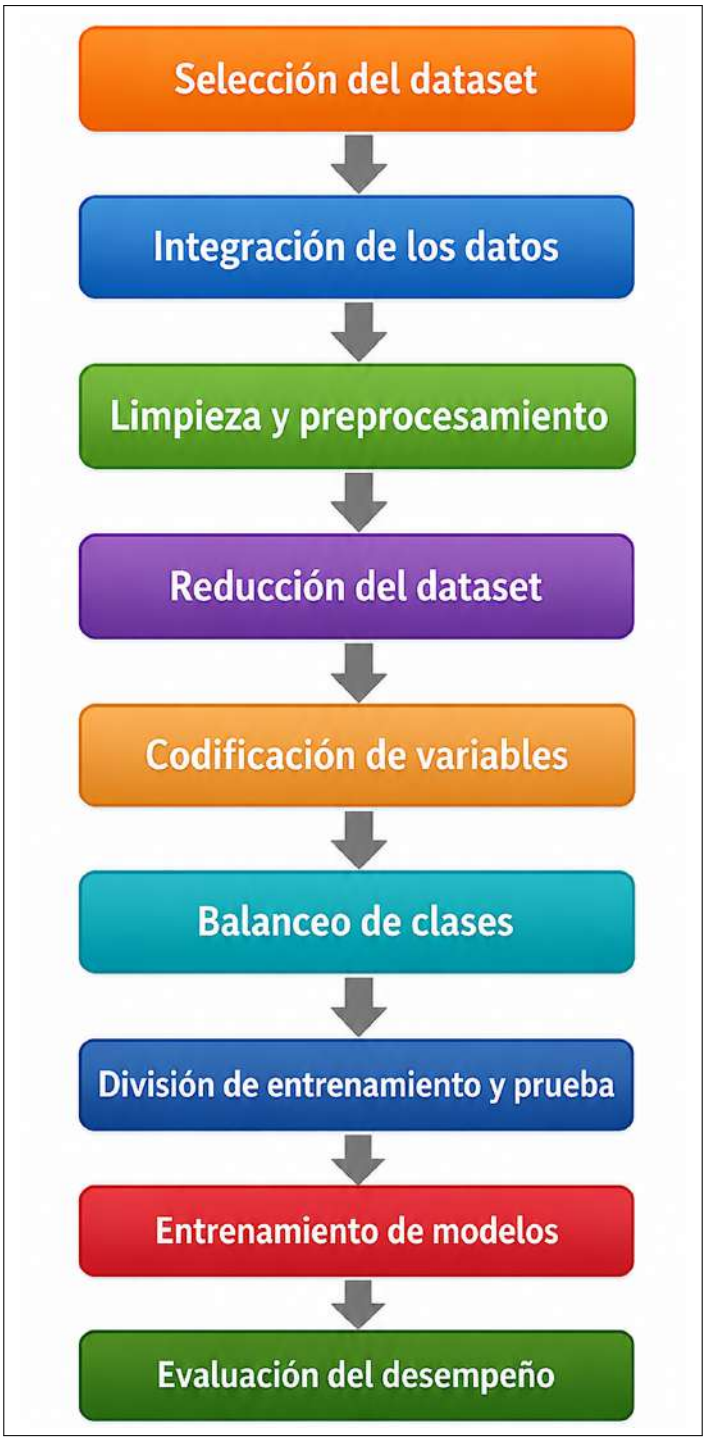

Figura 11.
Flujo metodológico general del sistema propuesto. Fuente: elaboración propia.

- **Selección del dataset**

  Se seleccionó el conjunto de datos CICIDS2017 porque incluye tráfico benigno y diferentes tipos de ataques. Esto permitió trabajar con información útil para evaluar la detección de intrusiones en redes.

- **Integración de los datos**

  Los archivos CSV del dataset se unificaron en una sola base de trabajo. También se revisó que los formatos fueran consistentes para evitar errores en las siguientes etapas de procesamiento.

- **Limpieza y preprocesamiento**

  Se eliminaron valores nulos, registros duplicados e inconsistencias presentes en los datos. Con esta depuración se obtuvo una base más limpia para entrenar el modelo.

- **Reducción del dataset**

  Se descartaron características irrelevantes o redundantes, conservando únicamente las variables útiles para la clasificación. Esta etapa ayudó a disminuir información innecesaria y a mejorar las condiciones de trabajo del modelo.

- **Codificación de variables**

  Las variables categóricas se transformaron en valores numéricos, debido a que los modelos de aprendizaje automático y aprendizaje profundo requieren entradas procesables de forma matemática.

- **Balanceo de clases**

  Se aplicó balanceo para reducir la diferencia entre el tráfico benigno y el tráfico malicioso. Con esto se evitó que el modelo aprendiera principalmente de la clase con más registros.

- **División de entrenamiento y prueba**

  El dataset se dividió en dos partes: una para entrenamiento y otra para prueba. La primera se usó para ajustar el modelo, mientras que la segunda permitió revisar su desempeño con datos que no fueron usados durante el entrenamiento.

- **Entrenamiento de modelos**

  Se entrenó el modelo y se ajustaron sus principales parámetros. En esta etapa se trabajó con una arquitectura basada en redes neuronales convolucionales, por su capacidad para reconocer patrones en los datos de tráfico de red.

- **Evaluación del desempeño**

  El rendimiento del modelo se revisó mediante métricas como exactitud, precisión, *recall* y *F1-score*. Estos indicadores ayudaron a comprobar si el sistema clasificaba correctamente el tráfico normal y malicioso.

3.3.1 Preprocesamiento de datos

La fase de preprocesamiento permitió preparar la información antes de utilizarla en el entrenamiento del modelo. Esta etapa fue necesaria porque los datos originales del dataset requerían una revisión previa para asegurar que pudieran ser procesados correctamente por la red neuronal convolucional.

Durante esta fase se realizaron varias operaciones sobre el conjunto de datos. Primero, se aplicó la limpieza de registros para eliminar valores nulos, duplicados o inconsistencias. Después, se redujo el dataset mediante la selección de características útiles para el modelo. También se generaron subconjuntos de datos, se dividió la información en conjuntos de entrenamiento y prueba, se aplicó el balanceo de clases mediante la técnica SMOTE y se normalizaron las características numéricas.

Estas actividades permitieron transformar los datos originales en una estructura más ordenada para el entrenamiento. Con ello se redujo la presencia de errores o inconsistencias y se mejoraron las condiciones de entrada para el modelo propuesto.

3.3.2 Limpieza de datos

La limpieza de datos constituye una etapa crítica dentro del proceso de preparación de la información, ya que permite eliminar inconsistencias que puedan afectar negativamente el desempeño del modelo durante su entrenamiento.

Se realizó la carga de los archivos CSV mediante el uso del lenguaje de programación Python

y la biblioteca Pandas, la cual permitió manipular grandes volúmenes de datos de manera eficiente e integrar múltiples archivos en un único conjunto de datos.

Posteriormente, se ejecutaron procesos destinados a eliminar registros que contienen valores nulos dentro de sus características. Los valores nulos pueden generar errores durante el entrenamiento del modelo, ya que impiden la realización de cálculos matemáticos necesarios para la actualización de los parámetros internos de la red neuronal.

Se realizó también la eliminación de valores infinitos positivos y negativos que pudieran estar presentes dentro del dataset, garantizando la integridad de la información utilizada.

La Tabla V presenta el proceso de limpieza y eliminación de valores inválidos en el dataset.

Tabla V.
Proceso de limpieza y eliminación de valores inválidos en el dataset

Tipo de valor	Cantidad antes	Cantidad después
Valores NaN	1358	0
Valores infinitos	Detectados	0
Registros totales	2,830,743	2,827,876

Fuente: Elaboración propia a partir del procesamiento del dataset CICIDS2017.

Una vez completados los procesos de limpieza, el dataset resultante fue almacenado para continuar con las siguientes etapas del preprocesamiento.

3.3.3 Reducción del dataset

El dataset CICIDS2017 contiene una cantidad elevada de registros, lo que puede aumentar el consumo de recursos computacionales durante el entrenamiento del modelo. Debido a esto, se aplicó una reducción del tamaño del conjunto de datos, con el propósito de trabajar con una

muestra más manejable sin perder la representación general de las clases presentes.

Para esta fase se seleccionó un subconjunto del dataset original, estableciendo previamente un límite máximo de registros para el entrenamiento. Con ello se pudo controlar el volumen de información procesada y disminuir el tiempo necesario para realizar las pruebas del modelo.

La selección de registros se realizó mediante muestreo aleatorio. Este procedimiento permitió mantener una distribución similar a la del conjunto original, incluyendo registros de las distintas clases de tráfico. Así, el modelo pudo entrenarse con una cantidad menor de datos, pero todavía representativa del comportamiento normal y malicioso presente en el conjunto inicial.

3.3.4 Generación de subconjuntos de datos

Para realizar pruebas adicionales del sistema, se generaron subconjuntos de datos a partir del dataset original. Esta división permitió trabajar con muestras controladas y observar el comportamiento del modelo bajo diferentes condiciones de carga.

La generación de los subconjuntos se realizó mediante muestreo estratificado. Con este método se buscó conservar la proporción original entre las clases de tráfico normal y tráfico malicioso, evitando que alguna categoría quedara sobrerrepresentada o reducida en exceso durante las pruebas.

Estos subconjuntos fueron utilizados posteriormente en la fase de evaluación del sistema. Su uso permitió simular distintos escenarios de procesamiento y analizar la respuesta del modelo frente a variaciones en la cantidad de datos disponibles.

3.3.5 División del dataset

Una vez preparado el dataset, se dividió en dos conjuntos principales: entrenamiento y prueba. El conjunto de entrenamiento se utilizó para ajustar los parámetros del modelo, mientras que el conjunto de prueba permitió evaluar su comportamiento con datos que no participaron en la

etapa de aprendizaje.

La división se realizó con una proporción estándar de 80 % para entrenamiento y 20 % para prueba. Esta distribución permitió contar con una cantidad suficiente de datos para entrenar el modelo y, al mismo tiempo, reservar un grupo independiente para medir su desempeño.

La Figura 12 presenta la división del dataset en conjuntos de entrenamiento y prueba.



Figura 12.
División del datase. Fuente: Elaboración propia.

Durante esta etapa también se aplicó una división estratificada, con el fin de mantener la proporción original de las clases presentes en el dataset. De esta manera, tanto el conjunto de entrenamiento como el de prueba conservaron registros representativos de tráfico normal y tráfico malicioso.

3.3.6 Balanceo de datos mediante SMOTE

Uno de los problemas frecuentes en el entrenamiento de modelos de clasificación es el desbalance entre las clases del dataset. En este caso, la cantidad de registros correspondientes a tráfico normal y tráfico malicioso no presenta la misma proporción, lo que puede afectar el aprendizaje del modelo.

Para tratar este problema se aplicó la técnica Synthetic Minority Over-sampling Technique (SMOTE). Este método permite generar registros sintéticos de la clase minoritaria a partir de ejemplos existentes, con el fin de equilibrar la distribución de clases dentro del conjunto de entrenamiento.

La Figura 13 muestra la distribución de clases antes y después del balanceo mediante SMOTE.

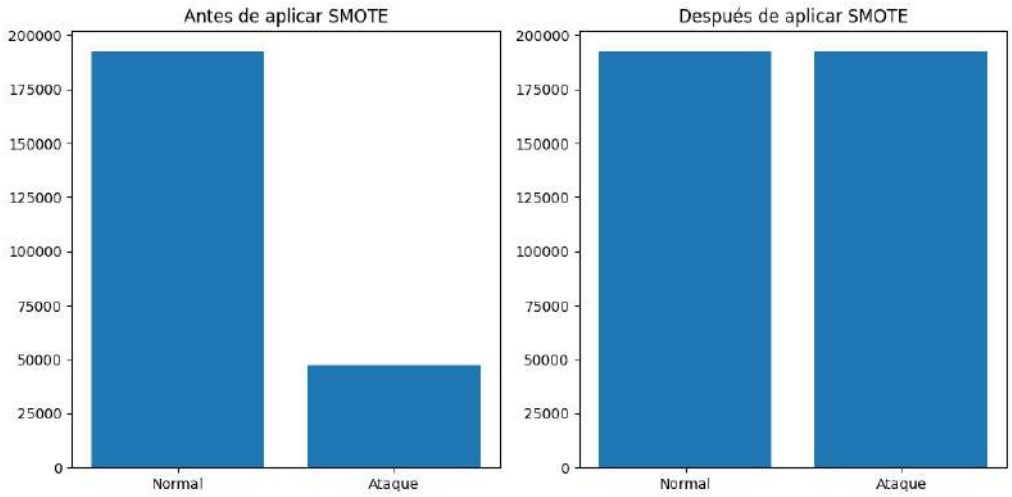

Figura 13.
Distribución de clases mediante SMOTE. Fuente: Elaboración propia.

La aplicación de SMOTE permitió reducir el desbalance entre las clases utilizadas durante el entrenamiento. Con una distribución más equilibrada, el modelo contó con mejores condiciones para reconocer patrones asociados tanto al tráfico normal como a los ataques informáticos.

3.3.7 Normalización de datos

Una vez balanceado el dataset, se normalizaron las características numéricas mediante técnicas de escalamiento. Esta etapa fue necesaria porque las variables del tráfico de red pueden manejar rangos muy distintos.

Para ello, se aplicó una estandarización que ajusta los valores de cada característica a una escala común, con media cercana a cero y desviación estándar igual a uno. Así, las variables

quedan en condiciones similares antes de ingresar al modelo.

La normalización ayudó a evitar que las características con valores numéricos más altos tuvieran mayor peso durante el entrenamiento. Así, el modelo pudo procesar las variables de manera más equilibrada y mejorar las condiciones de aprendizaje.

3.3.8 Transformación de datos para redes neuronales convolucionales

Una vez normalizados los datos, se realizó su transformación al formato requerido por el modelo de red neuronal convolucional. Este paso fue necesario porque las CNN no procesan los datos tabulares de la misma forma que otros modelos tradicionales de clasificación.

Las redes neuronales convolucionales requieren que los datos de entrada tengan una estructura compatible con las operaciones de convolución. Por ello, las matrices obtenidas después del preprocesamiento fueron reorganizadas de acuerdo con las dimensiones necesarias para el modelo propuesto.

Esta adaptación permitió que los datos pudieran ingresar correctamente a las capas convolucionales, donde se realiza la extracción automática de patrones presentes en el tráfico de red. De esta manera, las características normalizadas quedaron listas para ser utilizadas durante el entrenamiento del modelo.

Para mostrar el efecto del balanceo mediante SMOTE, se comparó la distribución de clases antes y después de aplicar la técnica. Antes del balanceo, el conjunto de entrenamiento presentaba 7.332.419 registros normales y 610.532 registros maliciosos, lo que evidenciaba una relación aproximada de 12:1. Esta diferencia podía afectar el aprendizaje del modelo, ya que existía el riesgo de que la red favoreciera la clase mayoritaria.

Posteriormente, se aplicó SMOTE utilizando los parámetros k_neighbors = 3 y sampling_strategy = auto. Con esta configuración se generaron nuevos registros sintéticos para la clase minoritaria hasta equilibrar ambas clases. Como resultado, el conjunto de entrenamiento quedó conformado

por 7.332.419 registros normales y 7.332.419 registros maliciosos.

La Tabla VI presenta la distribución de clases antes y después del balanceo mediante SMOTE.

Tabla VI.
Distribución de clases con SMOTE

Estado	Registros normales	Registros maliciosos
Antes del balanceo (entrenamiento)	7.332.419	610.532
Después del balanceo (entrenamiento)	7.332.419	7.332.419

Fuente: Elaboración propia a partir del procesamiento del dataset CICIDS2017.

3.4 Diseño del modelo

El diseño del modelo correspondió a una de las etapas principales del sistema propuesto, ya que permitió definir la arquitectura encargada de procesar los datos de entrada y generar la clasificación del tráfico de red.

En esta investigación se implementó un modelo basado en una red neuronal convolucional unidimensional (CNN 1D). Esta arquitectura se eligió porque trabaja bien con datos tabulares provenientes del tráfico de red y permite extraer patrones mediante filtros convolucionales aplicados sobre secuencias numéricas. Así, el modelo puede analizar relaciones entre características asociadas al tráfico normal y al tráfico malicioso.

El modelo se desarrolló con TensorFlow y la interfaz Keras. Su arquitectura siguió una estructura secuencial formada por capas convolucionales unidimensionales (Conv1D), capas de reducción dimensional (MaxPooling1D), una capa de aplanamiento (Flatten) y capas densas (Dense). Además, se incorporó *Dropout* como mecanismo de regularización para disminuir el riesgo de sobreajuste durante el entrenamiento.

La estructura definida permitió procesar los datos previamente normalizados y transformados. De esta forma, la red pudo aprender patrones útiles para la clasificación binaria del tráfico de red, diferenciando entre registros normales y registros maliciosos.

3.4.1 Arquitectura del modelo CNN

La arquitectura implementada corresponde a una red neuronal convolucional unidimensional (CNN 1D), diseñada para trabajar con datos numéricos provenientes del tráfico de red. Este tipo de modelo permite analizar secuencias de características y reconocer patrones asociados al comportamiento normal o malicioso de los registros.

El modelo se organizó mediante una secuencia de bloques funcionales. Cada bloque cumple una tarea específica dentro del procesamiento: extracción de características, reducción dimensional, transformación de la información y clasificación final. La Figura 14 muestra la arquitectura general utilizada en esta investigación.

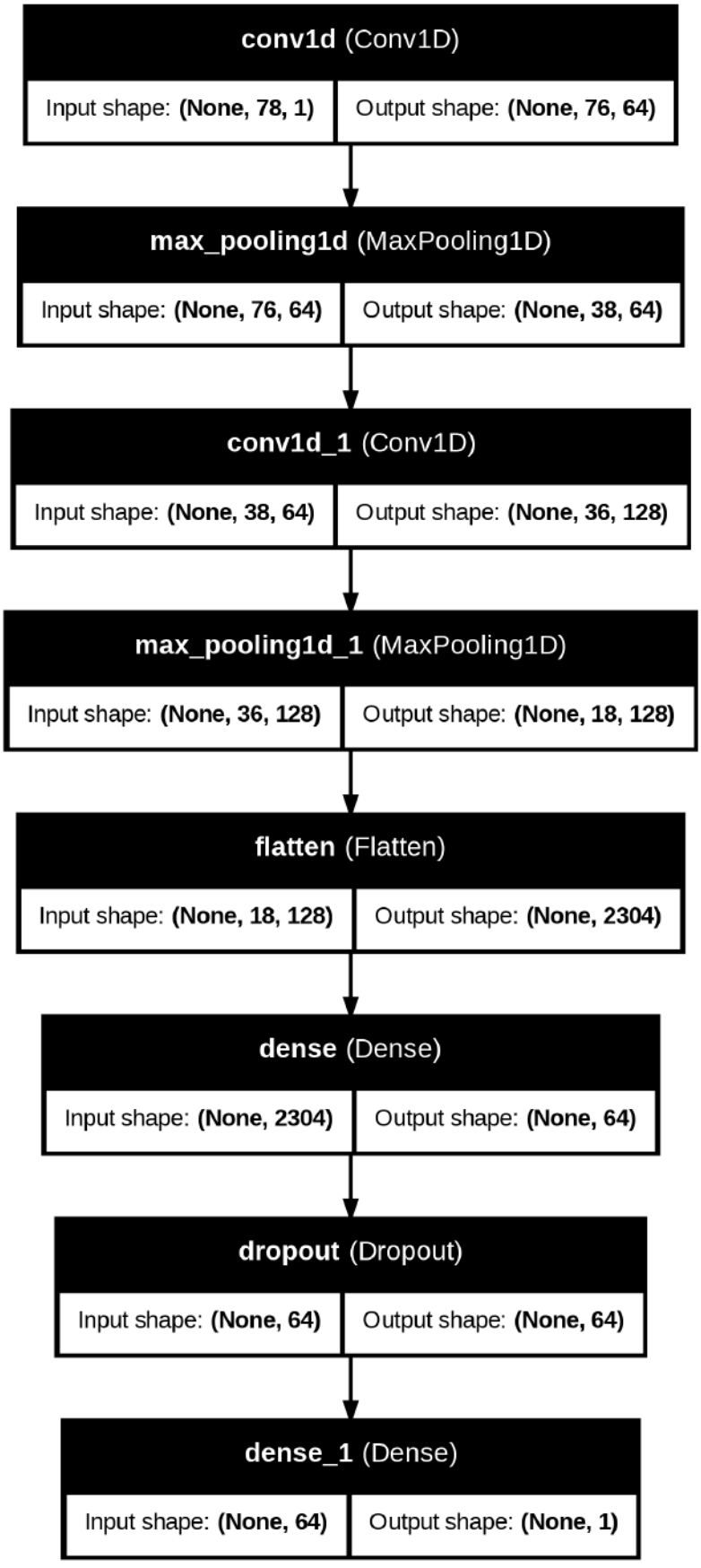

Figura 14.

Arquitectura de la red neuronal convolucional unidimensional utilizada para la clasificación del tráfico de red [12].

La arquitectura desarrollada se compone de los siguientes bloques principales:

- **Capas convolucionales (Conv1D):**

  Las capas convolucionales se encargan de extraer patrones presentes en los datos de entrada. En este modelo se utilizaron varias capas Conv1D organizadas de forma secuencial. La primera capa analiza pequeñas regiones del conjunto de características para identificar relaciones locales entre variables del tráfico de red. Las capas posteriores procesan esas representaciones iniciales y permiten reconocer patrones más elaborados. Esta estructura ayuda al modelo a distinguir comportamientos asociados al tráfico normal y al tráfico malicioso.

- **Capas de reducción dimensional (MaxPooling1D):**

  Después de las capas convolucionales, se incorporaron capas MaxPooling1D para reducir el tamaño de las representaciones generadas. Esta reducción permite conservar la información más relevante y disminuir la carga computacional durante el procesamiento. Además, facilita que las siguientes capas trabajen con una representación más compacta de los datos.

- **Capa de aplanamiento (Flatten):**

  La capa *Flatten* convierte las matrices generadas por las capas anteriores en un vector unidimensional. Este paso permite que la información pase a las capas densas, donde se realiza la clasificación.

- **Capas densas (Dense):**

  Después de convertir los datos en vectores, se incorporaron capas densas para clasificar los registros. También se utilizó una capa densa intermedia para combinar las características extraídas y fortalecer el aprendizaje del modelo antes de la salida final.

- **Capa de regularización (Dropout):**

  Se añadió una capa *Dropout* como mecanismo de regularización. Durante el entrenamiento, esta técnica desactiva de forma aleatoria un porcentaje de neuronas. Con esto se busca

reducir el riesgo de sobreajuste y ayudar al modelo a responder mejor frente a datos que no fueron usados durante el entrenamiento.

- **Capa de salida (Sigmoid):**

  La capa de salida utilizó una función de activación sigmoide. Esta función entrega una probabilidad para cada registro, lo que permite clasificarlo como tráfico normal o tráfico malicioso dentro del esquema de clasificación binaria propuesto.

3.4.2 Compilación del modelo

Una vez definida la arquitectura, se realizó la compilación del modelo. Esta etapa permitió establecer los elementos necesarios para iniciar el entrenamiento, entre ellos el optimizador, la función de pérdida y la métrica de evaluación.

Como optimizador se seleccionó Adam (*Adaptive Moment Estimation*), debido a que ajusta la tasa de aprendizaje durante el entrenamiento y suele ofrecer un comportamiento estable en modelos de aprendizaje profundo.

Para la función de pérdida se utilizó entropía cruzada binaria (*Binary Cross-Entropy*), adecuada para problemas de clasificación binaria. En este caso, la función permitió medir la diferencia entre la clase real de cada registro y la probabilidad estimada por el modelo.

Como métrica principal se configuró la exactitud (*Accuracy*), con el fin de medir el porcentaje de predicciones correctas realizadas durante el entrenamiento y la evaluación del modelo.

3.4.3 Entrenamiento del modelo

El entrenamiento del modelo tuvo como propósito ajustar los parámetros internos de la red neuronal a partir de los datos preparados en la etapa de preprocesamiento. Para ello, se utilizó el conjunto de entrenamiento obtenido después de la limpieza, transformación y balanceo de los datos.

Durante esta fase, el modelo procesó la información en varios ciclos de aprendizaje, conocidos como épocas. En cada época, la red recibió los datos de entrada, generó una predicción, comparó el resultado con la etiqueta real y ajustó sus parámetros internos para reducir el error de clasificación.

El modelo fue aprendiendo de manera gradual la relación entre las características del tráfico de red y las clases de salida. Además, se definió un tamaño de lote para procesar los datos por grupos, sin cargar toda la información al mismo tiempo. También se usó un conjunto de validación para revisar el comportamiento del modelo durante el entrenamiento y detectar posibles señales de sobreajuste.

La Figura 15 presenta la distribución de clases antes de aplicar SMOTE. Esta distribución muestra el desbalance inicial entre el tráfico normal y el tráfico malicioso, lo que justificó el uso de una técnica de balanceo antes del entrenamiento final del modelo.

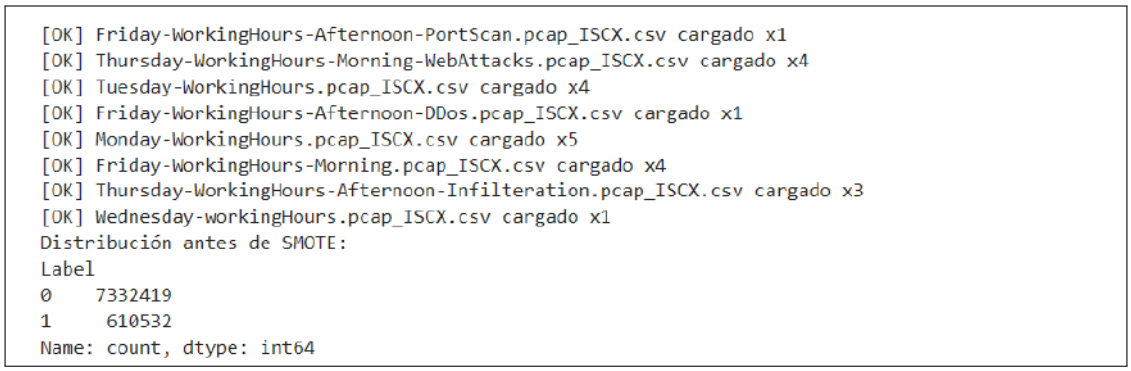

Figura 15.
Distribución de clases antes de la aplicación de SMOTE. Fuente: Elaboración propia.

Las Figuras 16 y 17 muestran el comportamiento del modelo durante las épocas iniciales y finales del entrenamiento. En los primeros ciclos se realiza el ajuste progresivo de la red, mientras que en las épocas finales las métricas tienden a estabilizarse. Los valores de *accuracy*, precisión y *recall* alcanzaron niveles cercanos a 0,98 tanto en entrenamiento como en validación. De igual manera, la función de pérdida, conocida como *loss*, presentó una tendencia descendente, lo que indica una convergencia adecuada durante el aprendizaje.

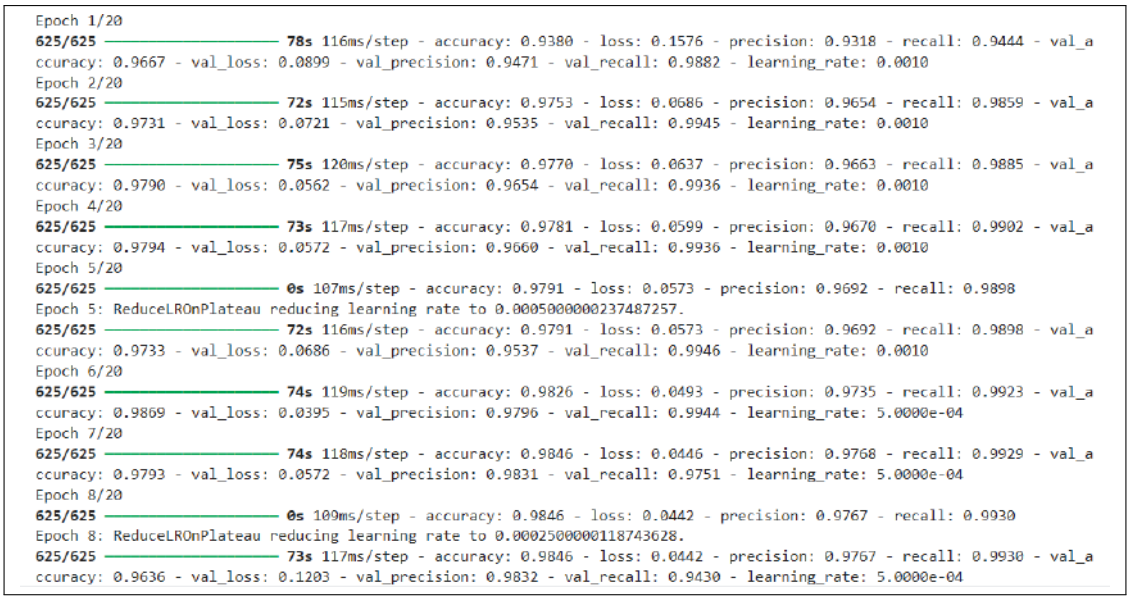

Figura 16.
Resultados del entrenamiento del modelo durante las épocas iniciales. Fuente: Elaboración propia.

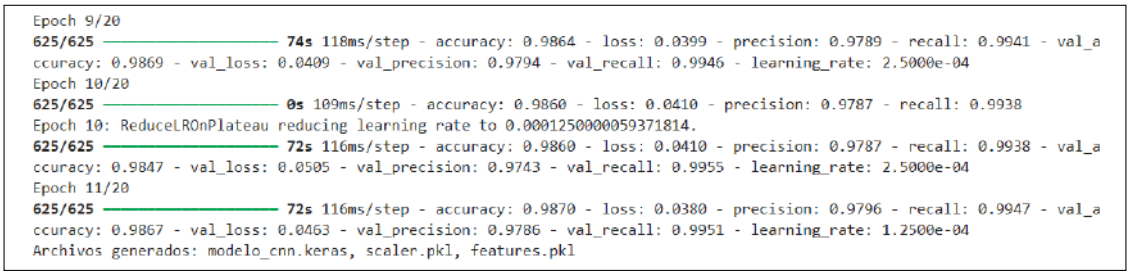

Figura 17.
Resultados del entrenamiento del modelo durante las épocas finales. Fuente: Elaboración propia.

3.4.4 Guardado del modelo entrenado

Una vez finalizado el entrenamiento, se guardó el modelo generado para conservar tanto su arquitectura como los parámetros aprendidos durante el proceso. Esto permitió reutilizarlo en pruebas posteriores sin tener que repetir todo el entrenamiento.

En esta etapa también se generaron varios archivos necesarios para el funcionamiento del sistema. El archivo `modelo_cnn.keras` contiene la arquitectura de la red y los pesos entrenados. Además, se guardaron los archivos `scaler.pkl` y `features.pkl`, necesarios para aplicar el mismo preprocesamiento en nuevas ejecuciones.

La Figura 18 presenta los archivos generados tras el entrenamiento del modelo.

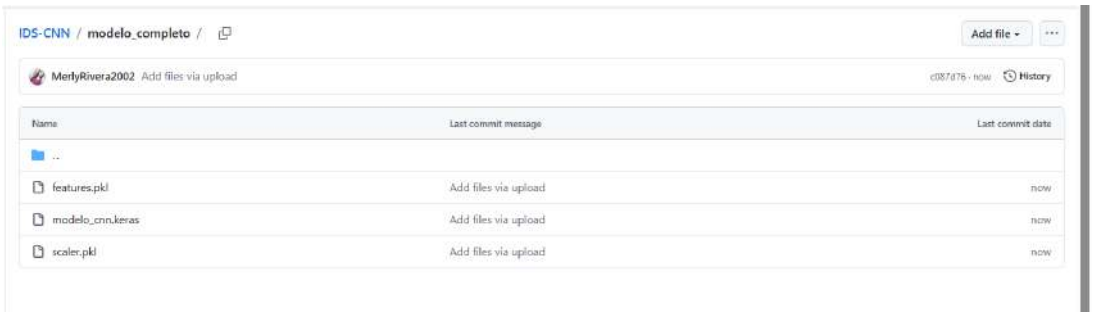

Figura 18.
Archivos generados tras el entrenamiento del modelo. Fuente: Elaboración propia.

Estos archivos se organizaron para que el aplicativo pueda cargarlos de forma directa. Con ello, el sistema recupera el modelo entrenado, aplica las transformaciones a los datos de entrada y ejecuta nuevas clasificaciones de tráfico.

Guardar el modelo también facilita su uso en entornos de prueba y reduce el tiempo necesario para nuevas evaluaciones, ya que evita repetir procesos de entrenamiento que requieren mayor carga computacional.

3.4.5 Preparación de recursos auxiliares

Además del almacenamiento del modelo principal, se generaron recursos auxiliares necesarios para garantizar el correcto funcionamiento del sistema durante su ejecución.

Entre estos recursos se incluyen los archivos correspondientes al escalador utilizado durante la normalización de los datos y la lista de características empleadas durante el entrenamiento del modelo.

Estos elementos mantienen la coherencia del sistema, porque hacen que los datos ingresados en la predicción se procesen igual que los datos usados durante el entrenamiento.

3.5 Desarrollo del aplicativo

Para facilitar el uso del sistema de detección de intrusiones, se desarrolló un aplicativo web interactivo en Streamlit. Desde la interfaz, el usuario puede cargar archivos de tráfico de red, ejecutar el modelo entrenado y revisar los resultados.

El aplicativo integra el modelo 1D-CNN en un entorno funcional. El usuario puede ingresar archivos de datos, aplicar el mismo preprocesamiento usado durante el entrenamiento, ejecutar la predicción y consultar la clasificación generada. Además, se incorporó un registro para conservar el historial de las simulaciones realizadas.

La estructura del aplicativo se organizó de forma modular. Esto permitió separar funciones como la carga de datos, el preprocesamiento, la ejecución del modelo, la visualización de resultados y el almacenamiento de información. Con esta organización, el sistema es más fácil de mantener y puede ampliarse en futuras versiones.

El flujo de uso inicia con la autenticación del usuario mediante credenciales previamente configuradas. Después, el usuario puede cargar archivos con registros de tráfico de red. Una vez ingresados los datos, el sistema aplica las transformaciones necesarias, entre ellas la normalización de características y la selección de variables relevantes.

Luego del preprocesamiento, el modelo 1D-CNN analiza los datos de entrada y genera una predicción para cada registro. Con estos resultados, el sistema clasifica el tráfico como normal o malicioso. Finalmente, la información se presenta mediante tablas, gráficos estadísticos y registros históricos, lo que facilita la revisión del comportamiento del tráfico analizado.

La Figura 19 presenta la arquitectura modular y el flujo general del aplicativo desarrollado para la detección de intrusiones.

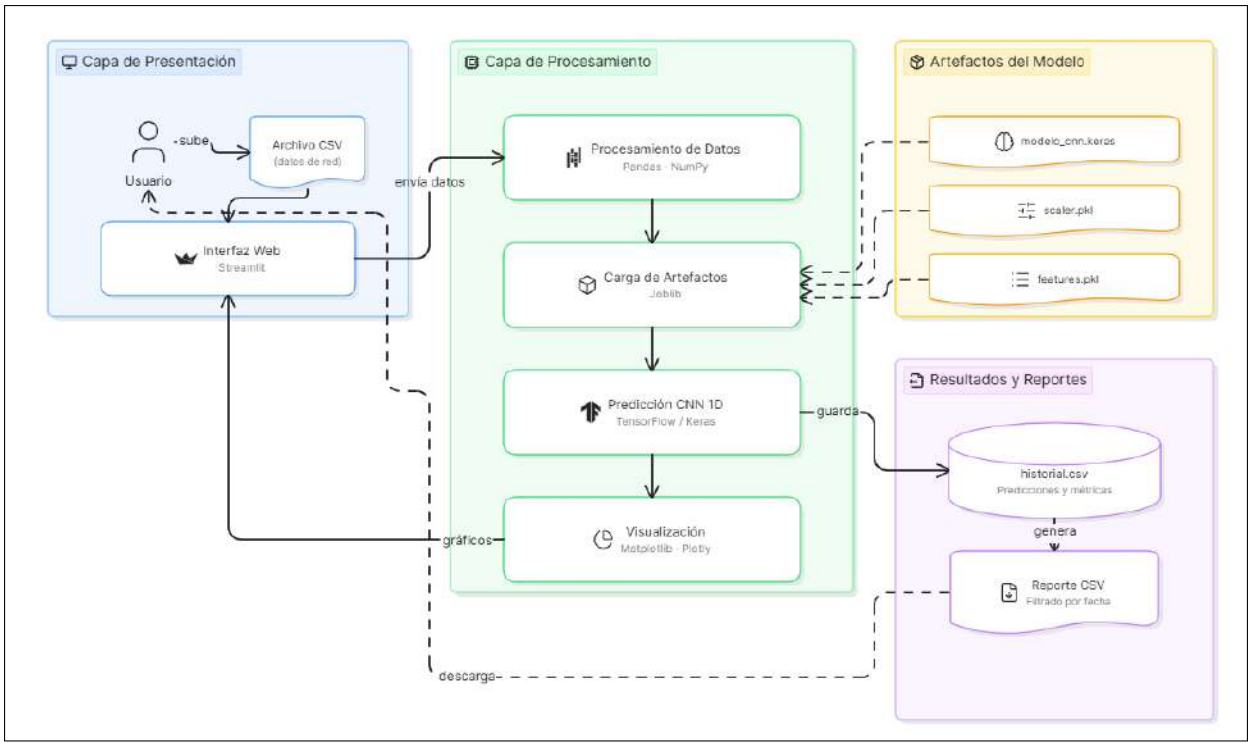

Figura 19.
Arquitectura modular y flujo general del aplicativo desarrollado para la detección de intrusiones. Elaboración propia.

Además del modelo entrenado, el aplicativo integra varias herramientas tecnológicas necesarias para su funcionamiento. Estas herramientas permitieron desarrollar la interfaz, procesar los datos, cargar los recursos generados durante el entrenamiento y presentar los resultados de forma visual. Las principales tecnologías utilizadas fueron las siguientes:

- **Python 3.10**: utilizado como lenguaje principal para el desarrollo del sistema y el procesamiento de datos.
- **Streamlit 1.25.0**: empleado para construir la interfaz web interactiva del aplicativo.
- **TensorFlow 2.12.0 y Keras 2.12.0**: utilizados para cargar y ejecutar el modelo CNN previamente entrenado.
- **Pandas 1.5.3 y NumPy 1.24.2**: empleados para la manipulación, limpieza y procesamiento de los conjuntos de datos.

- **Joblib 1.2.0**: utilizado para cargar el escalador y la lista de características generadas durante la etapa de entrenamiento.
- **Matplotlib 3.7.1 y Plotly 5.14.0**: empleados para generar gráficos y visualizaciones estadísticas dentro del aplicativo.

3.5.1 Autenticación de usuarios

El acceso al sistema inicia con una pantalla de autenticación, donde el usuario debe ingresar sus credenciales antes de utilizar la aplicación. Este mecanismo permite restringir el acceso a las funciones del sistema y mantener un control básico sobre la información generada durante las simulaciones.

La pantalla de inicio de sesión contempla perfiles previamente definidos, como administrador y espectador. Esta diferenciación permite asignar niveles de acceso según el tipo de usuario y el rol que cumple dentro del sistema.

La Figura 20 presenta la pantalla de inicio de sesión del sistema.

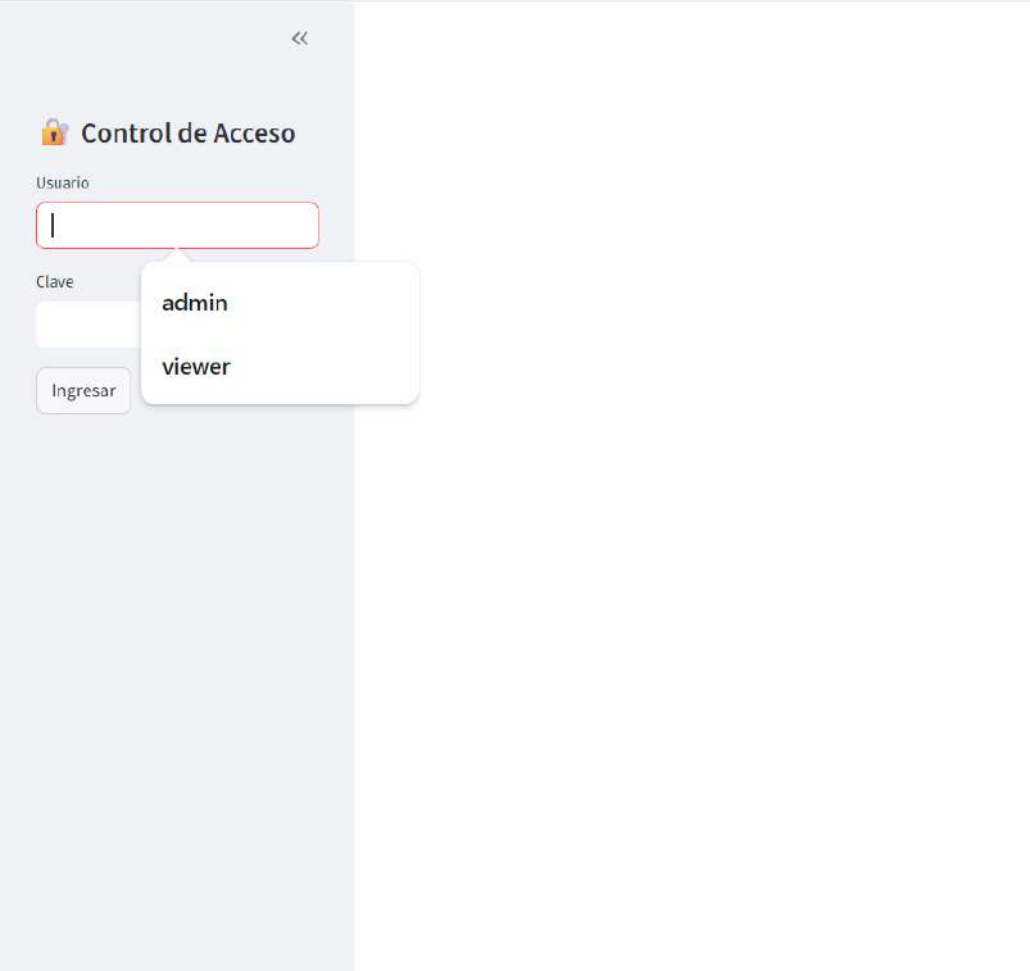

Figura 20.
Pantalla de inicio de sesión del sistema.

3.5.2 Pantalla principal de monitoreo en tiempo real

Una vez autenticado el usuario, el sistema muestra la pantalla principal del aplicativo. Desde esta interfaz, el usuario puede cargar archivos de tráfico de red y ejecutar la detección de intrusiones con el modelo entrenado. Una vez cargados los datos, el sistema aplica el preprocesamiento definido en la metodología y genera las predicciones. Con esa información, el usuario puede revisar si los registros analizados corresponden a tráfico normal o a actividad maliciosa.

La Figura 21 presenta la pantalla principal de monitoreo en tiempo real del aplicativo.

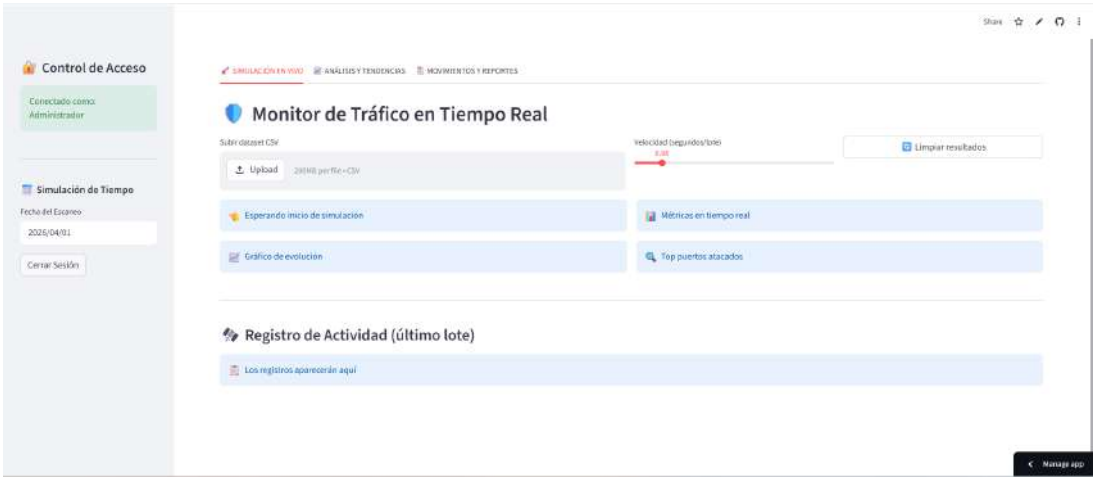

Figura 21.
Pantalla principal de monitoreo en tiempo real.

3.5.3 Visualización de clasificación de tráfico

Durante la ejecución del sistema, se generan representaciones gráficas que permiten visualizar la distribución del tráfico clasificado como normal y malicioso. Estas visualizaciones incluyen gráficos circulares y conteos estadísticos que facilitan la interpretación del estado del tráfico analizado en tiempo real.

La Figura 22 presenta la simulación del proceso de clasificación de tráfico en tiempo real.



Figura 22.
Simulación del proceso de clasificación de tráfico en tiempo real.

3.5.4 Registro de intrusiones detectadas

El sistema presenta una tabla con los registros analizados y la clasificación asignada por el modelo. En esta vista se muestra información específica de cada evento, indicando si corresponde a tráfico normal o malicioso. Esta funcionalidad permite revisar los resultados obtenidos durante la ejecución del sistema y observar el comportamiento del tráfico procesado. De esta manera, el usuario cuenta con una base organizada para identificar registros que requieren mayor atención dentro del análisis.

La Figura 23 presenta el registro de intrusiones detectadas y el resumen general del sistema.

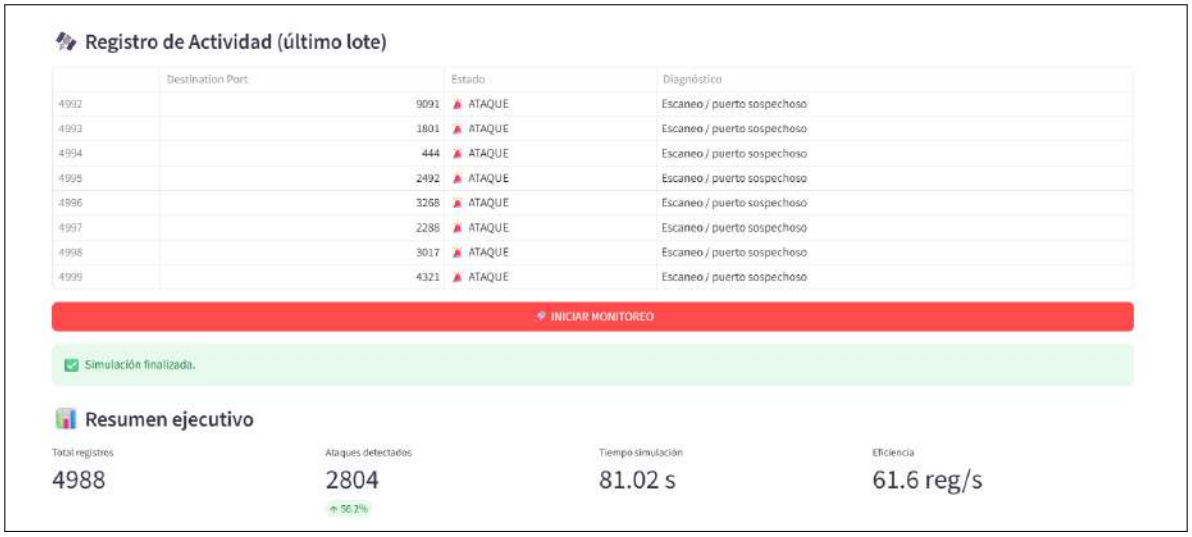

Figura 23.
Registro de intrusiones detectadas y resumen general del sistema.

3.5.5 Pantalla de análisis histórico

El aplicativo incorpora una pestaña destinada al análisis histórico de las simulaciones realizadas. Esta sección reúne información acumulada de ejecuciones anteriores, lo que permite revisar el comportamiento del tráfico analizado en distintos momentos.

Dentro de esta pantalla se presentan indicadores y visualizaciones que ayudan a reconocer tendencias, variaciones y patrones asociados a los eventos detectados. Esta información sirve como apoyo para comparar resultados entre simulaciones y observar la evolución del compor-

tamiento registrado por el sistema.

La Figura 24 presenta la pantalla de análisis histórico de las simulaciones realizadas.

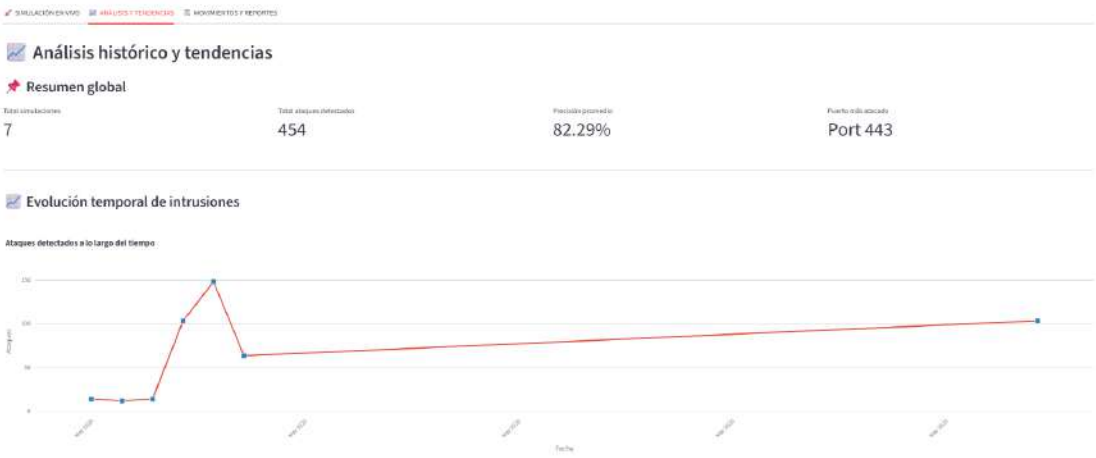

Figura 24.
Pantalla de análisis histórico de simulaciones realizadas.

3.5.6 Tendencia de puertos atacados

El sistema permite visualizar gráficos estadísticos que muestran los puertos que presentan mayor frecuencia de ataques durante las simulaciones realizadas.

Esta funcionalidad facilita la identificación de patrones relacionados con posibles vulnerabilidades dentro del sistema analizado.

La Figura 25 presenta la tendencia de puertos atacados detectados durante las simulaciones.

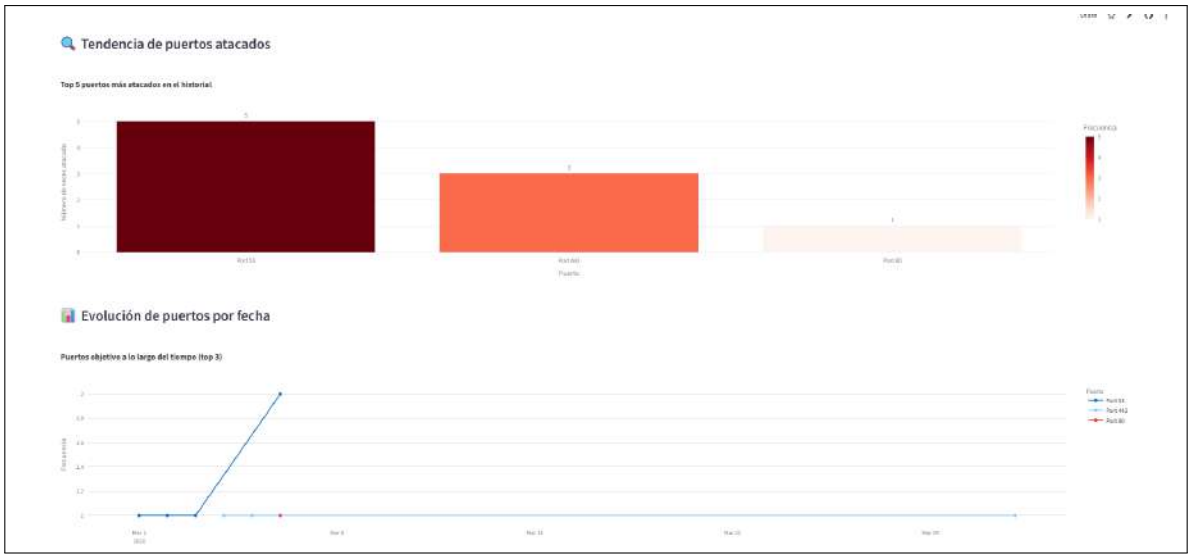

Figura 25.
Tendencia de puertos atacados detectados durante las simulaciones.

3.5.7 Evolución de la precisión del modelo

El sistema incluye gráficos que muestran la evolución del desempeño del modelo durante las simulaciones realizadas. Estos gráficos permiten observar cómo varía la precisión obtenida en diferentes ejecuciones y ayudan a identificar posibles cambios en el comportamiento del modelo.

Esta visualización resulta útil para comparar los resultados entre simulaciones y reconocer variaciones en el rendimiento del sistema.

La Figura 26 presenta la evolución de la precisión del modelo durante las simulaciones realizadas.



Figura 26.
Evolución de la precisión del modelo durante las simulaciones realizadas.

3.5.8 Generación de reportes

La tercera pestaña del sistema fue destinada a la generación de reportes personalizados. En esta sección, el usuario puede revisar la información guardada durante las simulaciones y seleccionar rangos de fechas para consultar resultados anteriores.

Además, el sistema permite exportar los datos obtenidos en formatos estructurados, lo que facilita su revisión y el análisis posterior de los resultados del modelo.

Las Figuras 27 y 28 presentan las opciones para generar reportes por intervalos de tiempo y reportes generales del sistema.

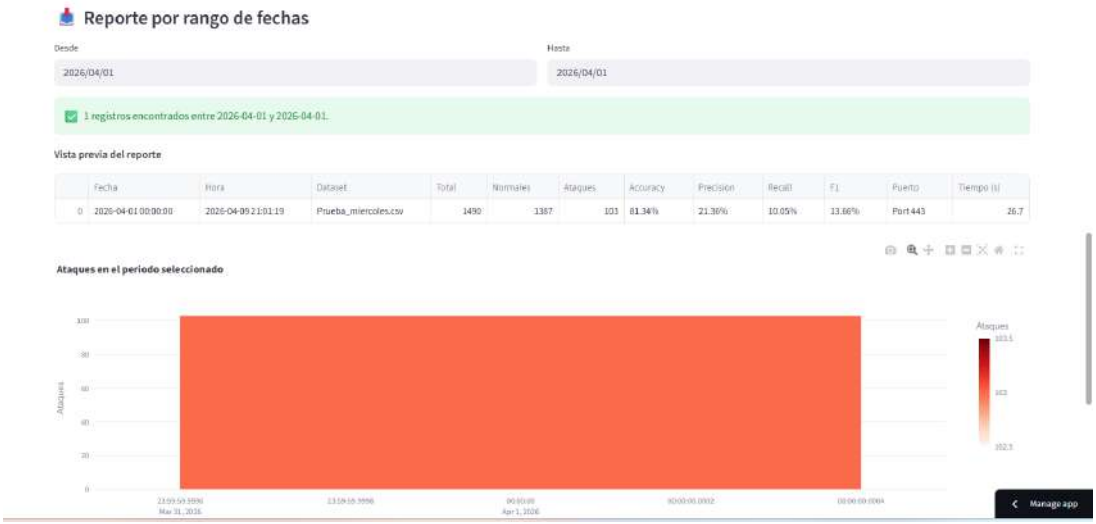

Figura 27.
Generación de reportes por intervalos de tiempo.



Figura 28.
Generación de reportes generales del sistema.

3.6 Validación del modelo

La validación del modelo permitió evaluar su comportamiento con datos que no fueron utilizados durante el entrenamiento. Esta etapa fue necesaria para comprobar si el modelo podía mantener un desempeño adecuado frente a nuevos registros de tráfico de red.

En los sistemas de detección de intrusiones, la validación tiene un papel importante porque una clasificación incorrecta puede generar dos problemas principales: falsas alarmas o ataques reales no detectados. Por ello, el modelo fue evaluado mediante métricas que permiten analizar su capacidad para distinguir entre tráfico normal y tráfico malicioso.

En este sistema se utilizaron distintas métricas de evaluación para observar el rendimiento del modelo desde varias perspectivas. Estas métricas permitieron cuantificar los resultados ob-

tenidos durante el proceso de clasificación y comparar el desempeño alcanzado en la detección de eventos maliciosos.

La Figura 29 presenta la visualización de métricas del modelo obtenidas durante la validación del sistema.

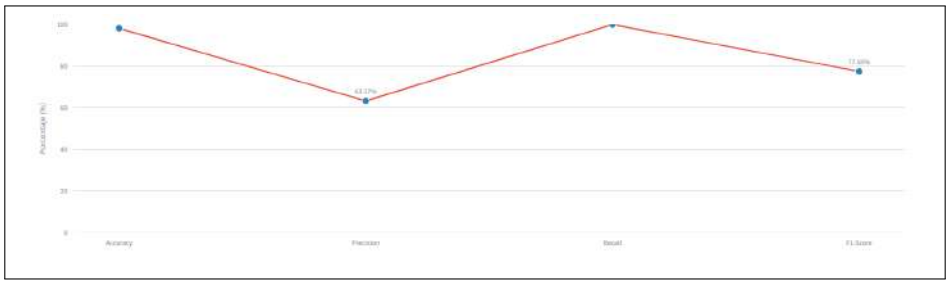

Figura 29.
Visualización de métricas del modelo obtenidas durante la validación del sistema.

Las métricas utilizadas en la investigación fueron la exactitud global, la precisión, la sensibilidad del modelo y el F1-score. En conjunto, estos indicadores permiten analizar el desempeño del modelo desde diferentes aspectos, tanto en su clasificación general como en su capacidad para reconocer correctamente los eventos maliciosos.

3.6.1 Precisión global (Accuracy)

La precisión global, conocida como *Accuracy*, mide la proporción de predicciones correctas realizadas por el modelo respecto al total de registros analizados. Esta métrica permite conocer qué porcentaje de casos fue clasificado correctamente dentro del conjunto evaluado.

Aunque la *Accuracy* ofrece una lectura general del desempeño, debe interpretarse junto con otras métricas, especialmente cuando existe desbalance entre las clases. En estos casos, un valor alto de exactitud no siempre garantiza que el modelo detecte correctamente la clase minoritaria.

La fórmula se representa así:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

3.6.2 Precisión de clasificación (Precision)

La precisión de clasificación mide qué proporción de los registros identificados como positivos por el modelo corresponde realmente a esa clase. En esta investigación, se relaciona con la capacidad del sistema para clasificar correctamente el tráfico malicioso entre todos los registros que fueron señalados como ataques.

Un valor alto de precisión indica que el modelo genera pocas falsas alarmas, es decir, que la mayoría de los eventos clasificados como maliciosos realmente pertenecen a esa categoría.

La fórmula se representa así:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3.6.3 Sensibilidad del modelo (Recall)

La sensibilidad del modelo, conocida como *Recall*, mide la proporción de ataques que fueron identificados correctamente por el sistema respecto al total de ataques presentes en el dataset. En esta investigación, esta métrica permite observar qué tan bien responde el modelo frente a registros maliciosos.

Un valor alto de *Recall* indica que el sistema deja pasar pocos ataques sin detectar. Por esta razón, su análisis es importante en sistemas de detección de intrusiones, donde no identificar una amenaza real puede representar un riesgo para la seguridad de la red.

La fórmula se representa así:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

3.6.4 Medida F1-score

La medida F1-score combina los valores de *Precision* y *Recall* en un solo indicador. Esta métrica resulta útil cuando se necesita analizar, al mismo tiempo, la cantidad de falsas alarmas y la capacidad del modelo para detectar ataques reales.

En el contexto de esta investigación, el F1-score permite valorar el equilibrio entre la precisión con la que el sistema clasifica eventos maliciosos y su sensibilidad para reconocerlos dentro del conjunto de datos. Su uso es especialmente conveniente cuando existe desbalance entre las clases, ya que ofrece una lectura más completa que la exactitud global por sí sola.

La fórmula se representa así:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

3.6.5 Matriz de confusión

La matriz de confusión se utilizó para revisar cómo se relacionan las clases reales con las predicciones generadas por el modelo. Esta herramienta permite observar, de manera directa, cuántos registros fueron clasificados correctamente y en qué casos se produjeron errores durante la evaluación.

La matriz presenta los resultados en cuatro grupos:

- **Verdaderos positivos (VP):** registros maliciosos que el modelo identificó correctamente.
- **Verdaderos negativos (VN):** registros normales clasificados correctamente como tráfico

legítimo.

- **Falsos positivos (FP):** registros normales que el modelo clasificó por error como maliciosos.
- **Falsos negativos (FN):** registros maliciosos que no fueron detectados por el sistema.

Con estos valores se puede revisar el comportamiento del clasificador. La matriz también sirve como base para calcular métricas como *Accuracy*, *Precision*, *Recall* y *F1-score*, utilizadas para evaluar el desempeño del modelo.

En el sistema desarrollado, la matriz de confusión se generó con los datos del conjunto de prueba. Su representación gráfica permitió observar la distribución de aciertos y errores durante la clasificación. Esto ayuda a identificar si el modelo confunde tráfico normal con tráfico malicioso o si deja pasar ataques sin detectarlos.

La Figura 30 presenta la estructura general de una matriz de confusión.

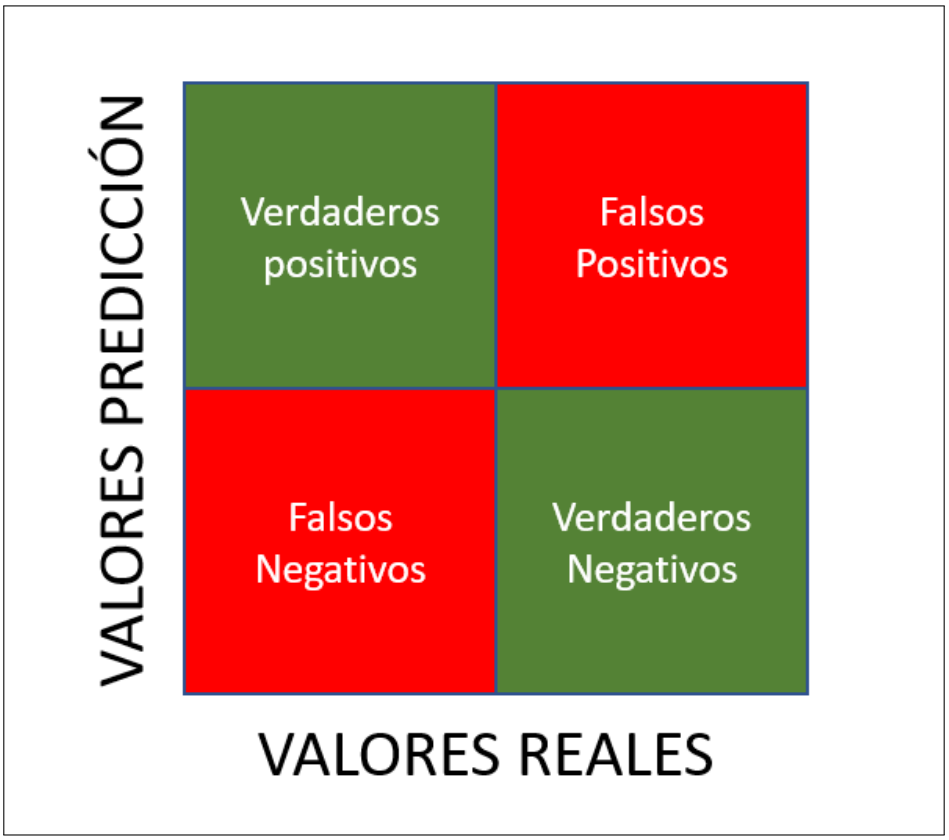

Figura 30.
Matriz de confusión [56].

La interpretación de esta matriz permite identificar posibles errores del proceso de clasificación y aporta información útil para ajustar el modelo o mejorar su desempeño en futuras pruebas.

CAPÍTULO IV

RESULTADOS Y ANÁLISIS

En este capítulo se analizan los resultados obtenidos durante la evaluación del sistema de detección de intrusiones desarrollado. Primero, se revisa el comportamiento del modelo 1D-CNN durante las pruebas realizadas. Para observar una de las características del tráfico analizado, la Figura 31 compara la duración de los flujos (*Flow Duration*) entre registros normales y maliciosos en dos escenarios de concentración de ataques.

En los gráficos se observa que el tráfico malicioso tiende a presentar valores más altos en la duración del flujo. Esta diferencia aporta información útil para que el modelo distinga entre ambos tipos de tráfico. Posteriormente, se examinan las métricas de clasificación, el comportamiento de la arquitectura implementada y la comparación con trabajos relacionados.

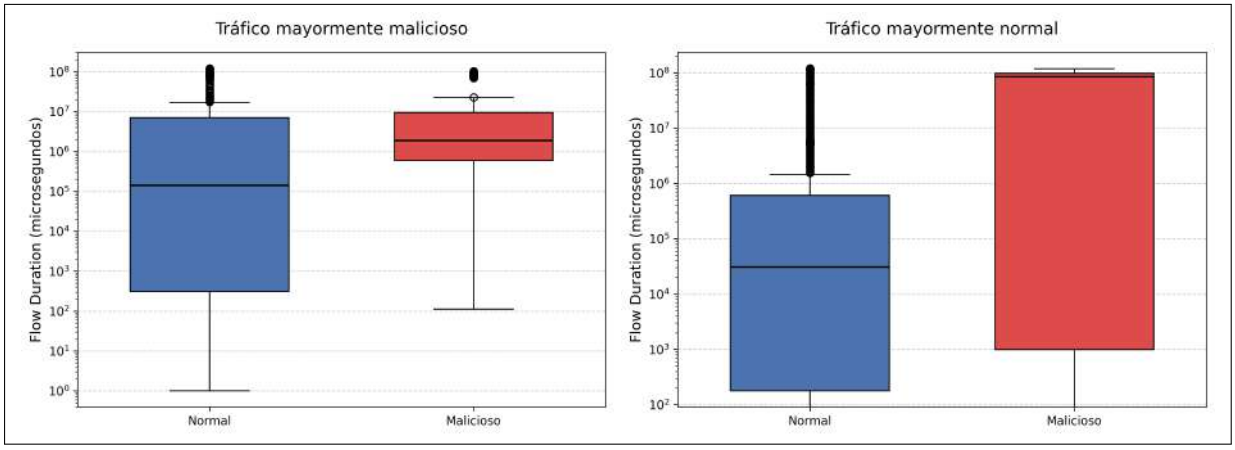

Figura 31.
Comparación visual de la duración del flujo entre tráfico normal y malicioso. Fuente: Elaboración propia.

4.1 Métricas de clasificación

Para evaluar el comportamiento del modelo en diferentes condiciones, los ataques se agruparon según su nivel de representación dentro del conjunto de datos y la dificultad observada

durante la clasificación.

Ataques mayoritarios: corresponden a DDoS, PortScan y DoS. Estos ataques tienen patrones de tráfico más fáciles de diferenciar frente al tráfico normal. Además, cuentan con más ejemplos en el conjunto de entrenamiento, lo que ayuda al modelo a reconocer mejor sus características.

Ataques minoritarios: incluyen fuerza bruta, ataques web, botnet e infiltración. Estas categorías aparecen con menor frecuencia en el conjunto de datos original y, en algunos casos, se parecen más al tráfico legítimo. Por esta razón, su clasificación resulta más difícil y puede generar falsos positivos o falsos negativos.

Las métricas se calcularon con subconjuntos de 500 a 10 000 registros por tipo de ataque, procurando mantener una proporción equilibrada entre las clases evaluadas. Esta separación permitió comparar el desempeño del modelo en ataques con patrones más claros y en otros con menor representación o mayor similitud con el tráfico normal.

La Figura 32 presenta la comparación de métricas obtenidas en ataques mayoritarios y minoritarios.



Figura 32.
Comparación de métricas en ataques mayoritarios y minoritarios. Fuente: Elaboración propia.

4.1.1 Exactitud (*Accuracy*)

Ataques mayoritarios: 98,47 %

Ataques minoritarios: 95,76 %

La exactitud mide el porcentaje global de aciertos, tanto sobre tráfico normal como sobre ataques. En los ataques mayoritarios, este valor fue alto porque el modelo clasificó correctamente la mayoría de los flujos y presentó pocos errores, con 267 falsos positivos y 2 falsos negativos.

En los ataques minoritarios, la exactitud también se mantuvo elevada. Esto muestra una buena capacidad del modelo para diferenciar entre tráfico normal y tráfico malicioso. En comparación con evaluaciones anteriores, la reducción de falsos positivos permitió obtener un rendimiento más equilibrado, sin afectar la detección de amenazas.

La Figura 33 presenta los resultados de *Accuracy* del modelo.



Figura 33.
Resultados de *Accuracy*

4.1.2 Precisión (*Precision*)

Ataques mayoritarios: 96,71 %
Ataques minoritarios: 38,95 %

La precisión indica cuántas de las alertas marcadas como ataque realmente correspondieron a eventos maliciosos. En los ataques mayoritarios, el modelo alcanzó una precisión de 96,71 %. Esto muestra que la mayoría de alertas generadas para este grupo fueron correctas, reduciendo las falsas alarmas y facilitando la revisión de los resultados por parte del analista de seguridad.

En los ataques minoritarios, la precisión fue menor, con un valor de 38,95 %. Este comportamiento indica que el modelo clasificó algunos registros benignos como maliciosos cuando se enfrentó a patrones menos frecuentes o más similares al tráfico normal. Por esta razón, la reducción de falsos positivos sigue siendo un aspecto que puede ajustarse en futuras pruebas para mejorar la diferenciación entre tráfico legítimo y tráfico malicioso.

La Figura 34 presenta los resultados de precisión del modelo.



Figura 34.
Resultados de *Precisión*.

4.1.3 Sensibilidad (*Recall*)

Ataques mayoritarios: 100 %

Ataques minoritarios: 93,73 %

El *recall* mide la proporción de ataques reales que el sistema consigue detectar. Esta métrica es importante en sistemas de defensa, porque un ataque que no se identifica a tiempo puede representar un riesgo directo para la infraestructura.

En los ataques mayoritarios, el modelo alcanzó un valor casi total. En los ataques minoritarios, aunque el porcentaje fue menor, el sistema logró reconocer la mayoría de las intrusiones. Este resultado muestra que el modelo conserva una buena capacidad de detección incluso cuando los patrones de ataque son menos frecuentes o presentan mayor complejidad.

La Figura 35 presenta los resultados de *Recall* del modelo.



Figura 35.
Resultados de *Recall*.

4.1.4 Puntuación F1 (*F1-score*)

Ataques mayoritarios: 98,31 %

Ataques minoritarios: 49,07 %

El *F1-score* combina la precisión y el *recall* mediante la media armónica. Esta métrica ayuda a revisar si el modelo mantiene un equilibrio entre detectar ataques reales y controlar las falsas alarmas.

En los ataques mayoritarios, el modelo obtuvo un *F1-score* de 98,31 %. Este valor muestra un buen equilibrio entre precisión y *recall*, ya que el sistema detectó la mayoría de los ataques y mantuvo pocas clasificaciones incorrectas en este grupo.

En los ataques minoritarios, el *F1-score* fue de 49,07 %. Este resultado refleja la dificultad de clasificar categorías con pocos ejemplos y patrones parecidos al tráfico legítimo. Aun así, muestra una mejora frente a pruebas anteriores, principalmente en la relación entre detección de ataques y control de falsos positivos.

Para mejorar este resultado, se puede ajustar el umbral de decisión, que en esta evaluación fue de 0,35. Un umbral más alto podría aumentar la precisión al reducir falsas alarmas, aunque también podría disminuir el *recall*. Por eso, la selección del umbral debe considerar el contexto de uso del sistema y el nivel de riesgo aceptable.

La Figura 36 presenta los resultados de *F1-score* del modelo.



Figura 36.
Resultados de *F1-score*.

4.2 Métricas estadísticas

Para validar la robustez del entrenamiento y la eficiencia de generalización del modelo, se analizaron indicadores que describen el funcionamiento interno del aprendizaje profundo, así como la reacción desagregada sobre diferentes categorías de ataques.

4.2.1 Función de pérdida (*Binary Cross-Entropy*)

Durante el entrenamiento se utilizó la función de pérdida *Binary Cross-Entropy*, debido a que el modelo trabaja con un problema de clasificación binaria. Esta función mide la diferencia entre la probabilidad estimada por la red neuronal y la etiqueta real de cada registro. En este caso, el valor 0 corresponde al tráfico normal y el valor 1 al tráfico malicioso. A partir de este error, el optimizador Adam ajusta progresivamente los pesos internos de la red.

Los registros obtenidos durante las 11 épocas muestran una evolución favorable del aprendizaje del modelo. La Figura 37 presenta el comportamiento de la pérdida durante el entrenamiento y la validación.

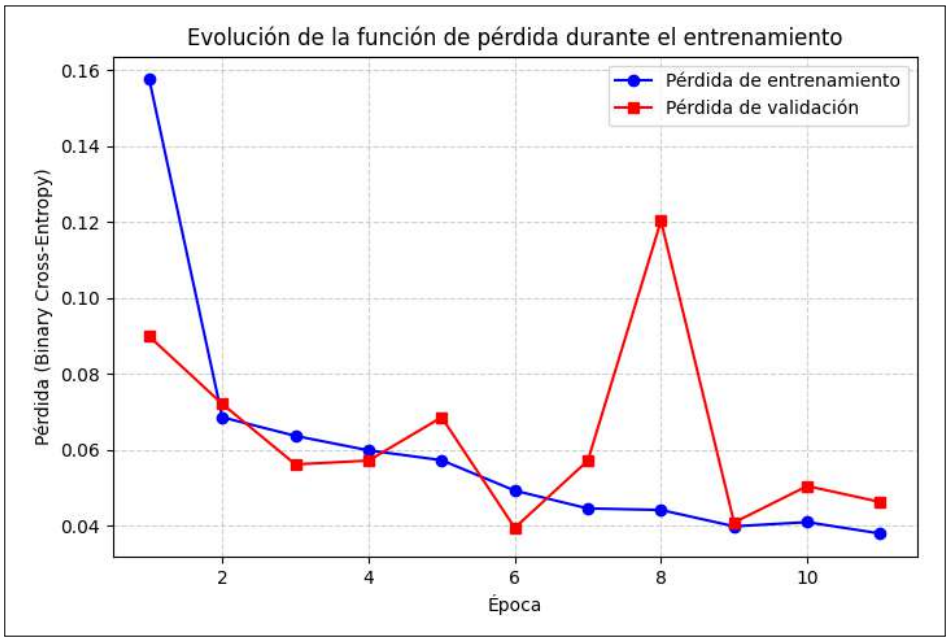

Figura 37.
Evolución de la función de pérdida durante el entrenamiento del modelo. Fuente: Elaboración propia.

La pérdida de entrenamiento y la pérdida de validación bajaron de forma progresiva y se mantuvieron en valores bajos al finalizar el proceso. Esto muestra que el modelo ajustó sus parámetros sin señales claras de sobreajuste. Además, la pérdida de validación se mantuvo estable, lo que indica que la red respondió bien ante datos que no fueron usados directamente durante el entrenamiento.

También se usó la técnica *ReduceLROnPlateau*, que baja la tasa de aprendizaje cuando el modelo deja de mejorar durante varias épocas. Esta estrategia ayudó a estabilizar el ajuste de los pesos y permitió que el entrenamiento avanzara de forma más controlada.

4.2.2 Validación Cruzada (Cross-Validation)

Aunque no se implementó una validación cruzada completa mediante k-fold, el entrenamiento incorporó una partición del 20 % de los datos destinada a validación (`validation_split = 0.2`). Este enfoque permitió evaluar el comportamiento del modelo sobre datos no vistos durante cada época de entrenamiento.

Las métricas obtenidas en el conjunto de validación fueron consistentes con las registradas durante el entrenamiento.

La Figura 38 presenta los resultados de validación obtenidos por el modelo CNN 1D.

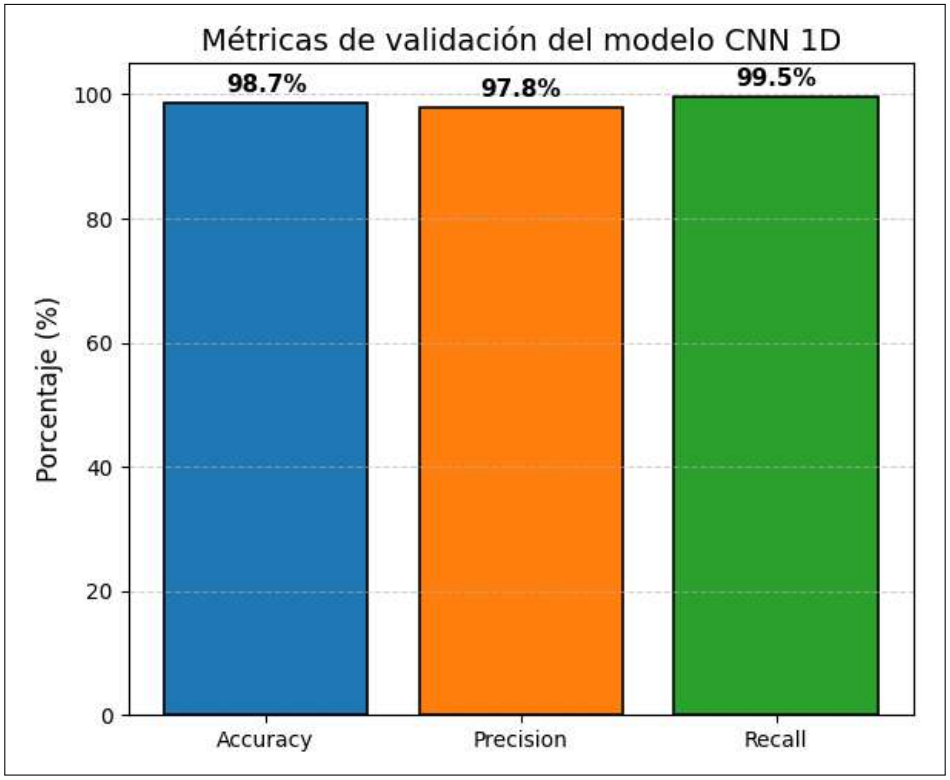

Figura 38.
Resultados de validación obtenidos por el modelo CNN 1D. Fuente: Elaboración propia.

Las diferencias entre entrenamiento y validación fueron reducidas, lo que indica que el modelo mantuvo un comportamiento estable. Este resultado sugiere que no se presentaron señales relevantes de sobreajuste y que la red conservó capacidad para responder ante datos no vistos durante el aprendizaje.

4.2.3 Análisis de las Matrices de Confusión

Con el objetivo de observar con mayor detalle la respuesta del modelo frente a diferentes tipos de ataques, se construyeron matrices de confusión para los grupos de ataques mayoritarios y minoritarios. Estas matrices permiten identificar la cantidad de registros clasificados correctamente y los errores generados durante la predicción, diferenciando entre tráfico normal y tráfico malicioso.

4.2.3.1 Ataques mayoritarios

La Figura 39 presenta la matriz de confusión para ataques mayoritarios.



Figura 39.
Matriz de confusión para ataques mayoritarios. Fuente: Elaboración propia.

En el caso de los ataques mayoritarios, la matriz de confusión muestra un comportamiento favorable del modelo. La mayor concentración de registros se ubica en las casillas correspondientes a las clasificaciones correctas, es decir, tráfico normal identificado como normal y tráfico malicioso identificado como ataque.

La casilla de verdaderos negativos muestra los registros normales que el modelo clasificó correctamente como tráfico normal. En cambio, la casilla de verdaderos positivos corresponde a los ataques que fueron reconocidos de forma correcta. Estos dos valores son importantes porque permiten revisar si el sistema diferencia adecuadamente el tráfico legítimo del tráfico malicioso.

También se observan pocos errores en comparación con la cantidad total de registros evaluados. Los falsos positivos corresponden a registros normales que fueron clasificados como ataque, mientras que los falsos negativos representan ataques reales que fueron clasificados como tráfico normal. En este grupo, la baja presencia de errores indica que el modelo respondió de manera estable frente a patrones de ataque con mayor representación en el conjunto de datos.

4.2.3.2 Ataques minoritarios

La Figura 40 presenta la matriz de confusión para ataques minoritarios.



Figura 40.
Matriz de confusión para ataques minoritarios. Fuente: Elaboración propia.

En los ataques minoritarios, la matriz de confusión evidencia un comportamiento más variable. Aunque el modelo logró identificar una parte importante del tráfico malicioso, también se observa una mayor presencia de falsos positivos en comparación con los ataques mayoritarios.

La casilla de verdaderos negativos muestra los registros normales reconocidos correctamente como tráfico normal, mientras que la casilla de verdaderos positivos representa los ataques minoritarios que fueron detectados por el sistema. Estos resultados indican que el modelo aún

detecta una parte importante de los ataques, incluso en categorías con pocos ejemplos durante el entrenamiento.

Sin embargo, los falsos positivos muestran que algunos registros legítimos fueron marcados como maliciosos. Esto puede deberse a que ciertos ataques minoritarios tienen patrones parecidos al tráfico normal. En cambio, los falsos negativos corresponden a ataques que el sistema no detectó, por lo que siguen siendo un punto a mejorar en futuras versiones del modelo.

En conjunto, la matriz permite observar que el modelo prioriza la detección de amenazas, aunque todavía presenta dificultades para reducir falsas alarmas en categorías con menor representación.

4.2.4 Curva ROC y área bajo la curva (AUC)

Para evaluar el rendimiento del modelo para diferenciar ataques, se calculó la curva ROC (Receiver Operating Characteristic) y el área bajo la curva (AUC). Esta métrica resume el comportamiento del clasificador en diferentes umbrales posibles, representando la probabilidad de que el modelo asigne una probabilidad más alta a un ejemplo malicioso que a uno normal.

La Figura 41 presenta la curva ROC del modelo CNN 1D.

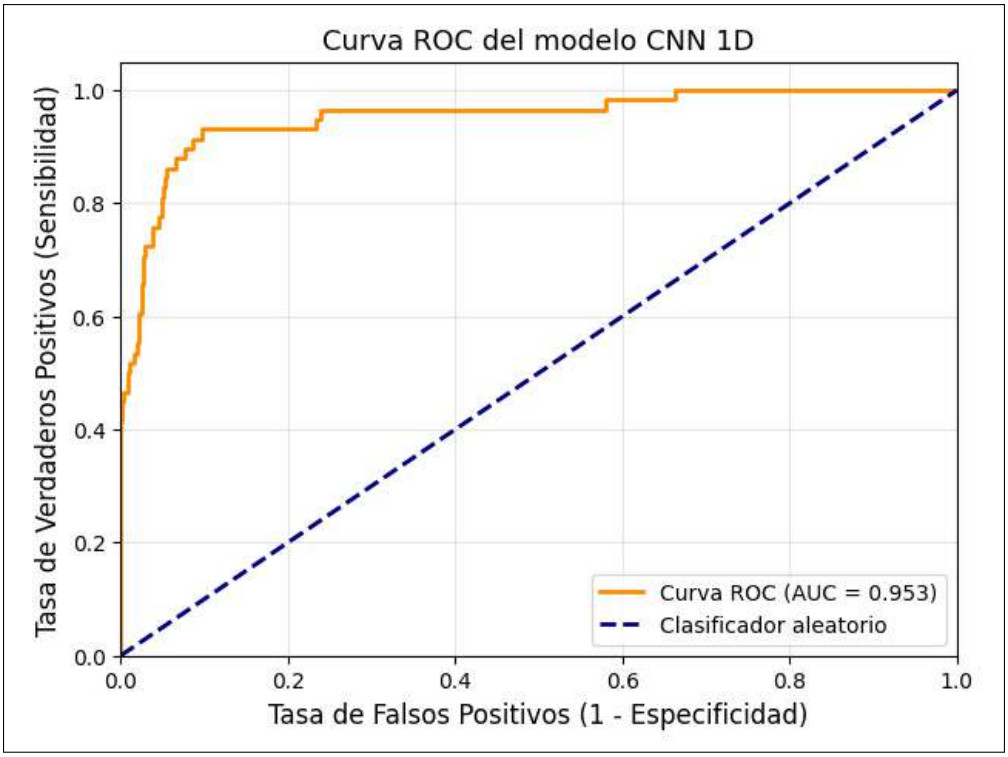

Figura 41.
Curva ROC del modelo CNN 1D. Fuente: Elaboración propia.

El valor AUC de 0.953 muestra que el modelo logró diferenciar de forma adecuada entre tráfico normal y tráfico malicioso. Este resultado indica que el clasificador separó ambas clases correctamente en la mayoría de los casos evaluados. Sin embargo, este valor debe revisarse junto con las métricas y las matrices de confusión presentadas anteriormente, ya que durante las pruebas todavía aparecieron falsos positivos y falsos negativos. Por tanto, la curva ROC muestra un buen comportamiento general del modelo, aunque también señala que el sistema puede seguir ajustándose para reducir errores de clasificación.

4.3 Comparativa del modelo

La comparación se realiza a dos niveles: primero frente a modelos base (CNN y RNN convencionales) y posteriormente frente a arquitecturas más complejas (DCNN, híbridas y optimizadas).

Se enfatiza que, a diferencia de muchos trabajos que evalúan únicamente sobre un subcon-

junto reducido de ataques mayoritarios, el presente modelo fue validado sobre todas las categorías de ataque presentes en el dataset, incluyendo aquellas con muy baja representación, como ataques web, infiltración, botnet y fuerza bruta.

4.3.1 Desempeño frente a modelos base

El modelo propuesto en esta investigación fue comparado con implementaciones reportadas en estudios previos. Según Azeez et al. [57], una arquitectura CNN convencional aplicada sobre CIC-IDS2017 alcanzó una exactitud de 95,20 %, mientras que un modelo RNN obtuvo 94,10 %.

En comparación, el modelo desarrollado en este trabajo alcanzó una exactitud global de 98,47 %, lo que representa una mejora aproximada de 3,5 puntos porcentuales frente a la CNN base.

Este resultado se relaciona con tres aspectos del diseño aplicado:

- Uso de una arquitectura 1D-CNN con capas convolucionales, Batch Normalization y Dropout.
- Aplicación de SMOTE para equilibrar las clases minoritarias.
- Normalización de datos mediante Z-score para mejorar el proceso de entrenamiento.

Varios trabajos base evalúan principalmente ataques mayoritarios. En cambio, el modelo propuesto también fue probado con ataques minoritarios. En estas condiciones, alcanzó un *recall* de 100 % en ataques mayoritarios y 93,73 % en ataques minoritarios, lo que muestra una buena capacidad de detección incluso en escenarios más difíciles.

4.3.2 Posicionamiento frente a arquitecturas complejas

La literatura reciente reporta modelos con exactitudes superiores al 99 %, entre ellos:

- Hnamte y Hussain [58]: DCNN con 99,96 %.
- Al-zubidi et al. [59]: CNN-LSTM-XGBoost con 99,2 %.
- Kilichev y Kim [60]: 1D-CNN optimizada con 99,71 %.

Sin embargo, estos modelos presentan dos limitaciones principales. La primera es la complejidad computacional, ya que algunos requieren GPU o mayores tiempos de entrenamiento. En contraste, el modelo propuesto mantiene una arquitectura más ligera, con posibilidad de ejecutarse en CPU y en entornos de prueba más accesibles.

La segunda limitación está relacionada con la cobertura de los ataques evaluados. En varios trabajos, la evaluación se reduce a un número limitado de clases, lo que puede hacer que los resultados parezcan más favorables. Por ejemplo, Qazi et al. [61] restringen su análisis a cuatro tipos de ataques y dejan fuera categorías más complejas.

En esta investigación, el modelo se evaluó con todas las clases presentes en el dataset. Aunque esto representó una mayor dificultad, el sistema alcanzó una exactitud global de 98,47 % y mantuvo niveles altos de detección en los distintos grupos de ataque.

4.3.3 Tabla comparativa

La Tabla VII presenta una comparación entre el modelo propuesto y algunos trabajos relacionados que también emplean arquitecturas basadas en redes neuronales para la detección de intrusiones.

Tabla VII.
Comparación del modelo propuesto con trabajos relacionados

Modelo / Autor	Acc.	Prec.	Rec.	F1
Modelo propuesto (CNN 1D)	98,47 %	96,71 %	100 %	98,31 %
Azeez et al. (2024) – CNN convencional [57]	95,20 %	89,97 %	92,57 %	91,25 %
Al-zubidi et al. (2024) – CNN-LSTM-XGBoost [59]	99,20 %	99,00 %	99,00 %	99,00 %
Hnamte & Hussain (2023) – DCNN [58]	99,96 %	99,96 %	99,96 %	99,96 %
Kilichev & Kim (2023) – 1D-CNN + GA/PSO [60]	99,71 %	100 %	99,00 %	99,50 %

Fuente: Elaboración propia con base en los resultados obtenidos y en los trabajos revisados [57], [58], [59], [60].

A diferencia de varios trabajos revisados, el modelo desarrollado en esta investigación fue evaluado considerando todas las clases de ataque presentes en CICIDS2017, incluidas categorías con menor representación, como ataques web, fuerza bruta, botnet, infiltración y Heartbleed. Esta condición hace que la evaluación sea más exigente, ya que no se limita únicamente a los ataques con mayor cantidad de muestras.

Aun bajo este escenario, el modelo alcanzó un *recall* del 100 % en los ataques mayoritarios y una exactitud global del 98,47 %. Estos resultados indican que la arquitectura propuesta mantiene una capacidad adecuada para reconocer distintos tipos de amenazas, aunque las clases minoritarias todavía representan un reto para mejorar la precisión y reducir falsos positivos.

4.4 Limitaciones

Aunque el modelo obtuvo resultados favorables en la detección de ataques mayoritarios, el desarrollo de la investigación presentó algunas limitaciones técnicas y operativas que condicio-

naron el proceso de entrenamiento, experimentación y validación del sistema. Estas limitaciones estuvieron relacionadas principalmente con la capacidad del hardware disponible, el tamaño del conjunto de datos, el uso del dataset seleccionado y el funcionamiento del aplicativo mediante archivos CSV.

- **Limitaciones del hardware local:** El entrenamiento de modelos CNN con grandes volúmenes de registros exigió un uso alto de la GPU y de la memoria VRAM del equipo local. Durante el desarrollo del proyecto, el equipo presentó fallas críticas en tres ocasiones, principalmente por sobrecalentamiento y uso prolongado de sus componentes. Esto afectó la continuidad del trabajo y obligó a cambiar el hardware para poder continuar con el entrenamiento y las pruebas del modelo.
- **Migración a infraestructura en la nube:** Debido a las limitaciones del equipo físico, el entrenamiento se trasladó a la plataforma Kaggle con aceleradores gráficos T4. Esto permitió trabajar en un entorno más estable, aunque también fue necesario reorganizar los archivos, respaldar el código en GitHub y guardar los modelos finales de forma más ordenada.
- **Restricciones de memoria durante el procesamiento:** Aunque se usó infraestructura en la nube, el tamaño del dataset original y la cantidad de atributos generaron una alta demanda de memoria durante la normalización y preparación de los datos. Por esta razón, se trabajó con una muestra aleatoria de 300 000 registros, para evitar errores de desbordamiento de memoria y mantener un entrenamiento viable. Esta reducción permitió ejecutar el modelo, pero también limita el análisis frente al uso completo del conjunto de datos original.
- **Limitaciones del dataset y del aplicativo:** Otra limitación del proyecto se relaciona con el uso del dataset CICIDS2017, ya que el modelo fue entrenado con los patrones presentes en ese conjunto de datos. Aunque este dataset contiene tráfico benigno y varios tipos de ataques, no necesariamente representa ataques recientes, nuevas variantes o comportamientos que no estuvieron presentes durante el entrenamiento.

Además, el aplicativo trabaja con archivos CSV cargados por el usuario, por lo que no captura tráfico directamente desde una red activa en tiempo real. También requiere que los archivos de entrada mantengan las mismas características utilizadas durante el entrenamiento del modelo. Finalmente, la clasificación principal del sistema es binaria, es decir, tráfico normal o malicioso.

Estas condiciones no invalidan los resultados obtenidos, pero deben considerarse al interpretar el alcance del modelo y del aplicativo. En futuras investigaciones sería conveniente disponer de una infraestructura computacional más robusta, utilizar datasets más recientes o tráfico capturado en un entorno real controlado, e incorporar mecanismos de captura en tiempo real y clasificación multiclase.

4.5 Impactos

4.5.1 Impacto tecnológico

Esta investigación permite automatizar parte del análisis del tráfico de red sin depender completamente de reglas manuales. Al utilizar una CNN, el sistema identifica de forma autónoma qué variables del flujo de red son maliciosas, superando a los firewalls convencionales que dependen de reglas manuales rígidas.

4.5.2 Impacto económico

El uso de herramientas de código abierto, como Python, TensorFlow y Streamlit, redujo los costos asociados a la adquisición de software. Esta decisión permite que pequeñas instituciones, laboratorios académicos o entornos de investigación puedan implementar una solución de defensa perimetral sin depender de licencias comerciales de alto costo.

4.5.3 Impacto social

Al mejorar la detección de intrusos, el proyecto aporta a la protección de la privacidad de los datos. En un entorno donde las filtraciones de información son frecuentes, contar con sistemas IDS ayuda a fortalecer la confianza de los usuarios en los servicios digitales institucionales y contribuye a proteger la información sensible de la comunidad.

CONCLUSIONES

El análisis teórico realizado permitió identificar que las redes neuronales convolucionales unidimensionales (1D-CNN) son adecuadas para el procesamiento de tráfico de red, debido a su capacidad para extraer automáticamente características relevantes de datos secuenciales sin requerir una ingeniería manual compleja. Además, la revisión de herramientas tecnológicas como TensorFlow, Keras, Streamlit y Scikit-learn, junto con el estudio del conjunto de datos CICIDS2017, proporcionó bases sólidas para el diseño e implementación del sistema de detección de intrusiones basado en aprendizaje profundo.

Se desarrolló un modelo 1D-CNN que, mediante un proceso de preprocesamiento basado en limpieza de datos, normalización Z-score y balanceo de clases con SMOTE, alcanzó un desempeño favorable en la clasificación del tráfico de red. Las pruebas realizadas sobre ataques mayoritarios, como DDoS, PortScan y DoS, evidenciaron valores altos de exactitud, precisión, sensibilidad y F1-score. La arquitectura implementada, compuesta por dos capas convolucionales y técnicas de regularización como Dropout y Early Stopping, demostró ser adecuada para diferenciar tráfico normal y tráfico malicioso en escenarios de prueba controlados.

La evaluación del modelo evidenció un mejor comportamiento en los ataques mayoritarios, donde se obtuvo un Accuracy de 98,47 %, Precision de 96,71 %, Recall de 100 % y F1-score de 98,31 %. En el caso de los ataques minoritarios, el modelo mantuvo una sensibilidad alta, con un Recall de 93,73 %, aunque presentó una Precision menor debido a la presencia de falsos positivos. Esto demuestra que el sistema prioriza la detección de amenazas, pero todavía requiere ajustes para reducir las alertas falsas en categorías con menor cantidad de muestras.

Finalmente, el aporte principal del proyecto no se centra en proponer una nueva arquitectura CNN, sino en integrar un modelo entrenado dentro de un aplicativo web funcional. Este aplicativo permite cargar archivos CSV, realizar el preprocesamiento correspondiente, clasificar el tráfico como normal o malicioso, visualizar métricas, revisar matrices de confusión, guardar historial y descargar reportes. De esta manera, el trabajo permitió validar el flujo completo de un sistema de detección de intrusiones en un entorno académico y controlado.

RECOMENDACIONES

- **Datasets recientes:** Se recomienda probar el sistema con conjuntos de datos más actuales, como CSE-CIC-IDS2018 o ToN-IoT. Esto permitiría revisar el comportamiento del modelo frente a escenarios de ataque más recientes y comprobar si mantiene un desempeño adecuado fuera del dataset utilizado en esta investigación.

- **Capturar tráfico real:** Se recomienda que futuras versiones del sistema puedan trabajar con tráfico capturado desde una red real. Para ello, se debe asegurar que los datos mantengan las mismas características utilizadas durante el entrenamiento del modelo, de manera que el proceso de clasificación sea compatible con la arquitectura implementada.

- **Ataques minoritarios:** Se recomienda incorporar una mayor cantidad de muestras de ataques con baja representación, como infiltración, ataques web y botnet. Esto ayudaría a mejorar la precisión del modelo en estas categorías y a reducir la cantidad de falsos positivos que se presentaron durante la evaluación.

- **Clasificación multiclase:** Se recomienda explorar una versión del modelo que no solo clasifique el tráfico como normal o malicioso, sino que también identifique el tipo específico de ataque. Esta mejora permitiría entregar información más detallada al usuario y facilitar el análisis de los resultados obtenidos.

- **Escenarios de prueba:** Se recomienda evaluar el sistema con diferentes archivos, tamaños de muestra y combinaciones de tráfico normal y malicioso. Esto permitiría observar el comportamiento del modelo en condiciones más variadas y fortalecer la validación del aplicativo.

- **Procesamiento del aplicativo:** Se recomienda optimizar el procesamiento de archivos con mayor cantidad de registros, especialmente cuando se trabaje con volúmenes amplios de tráfico. Esto permitiría mejorar la estabilidad del sistema y reducir posibles errores durante la carga, análisis y generación de reportes.

BIBLIOGRAFÍA

[1] A. Sharma, B. B. Gupta, A. K. Singh y V. K. Saraswat, "Advanced Persistent Threats (APT): evolution, anatomy, attribution and countermeasures," en, *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, n.º 7, págs. 9355-9381, jul. de 2023, issn: 1868-5137, 1868-5145. doi: 10.1007/s12652-023-04603-y. visitado 2 de oct. de 2025. dirección: https://link.springer.com/10.1007/s12652-023-04603-y.

[2] J. Kim, J. Kim, H. Kim, M. Shim y E. Choi, "CNN-Based Network Intrusion Detection against Denial-of-Service Attacks," en, *Electronics*, vol. 9, n.º 6, pág. 916, jun. de 2020, issn: 2079-9292. doi: 10.3390/electronics9060916. visitado 21 de ago. de 2025. dirección: https://www.mdpi.com/2079-9292/9/6/916.

[3] Y. Mo, H. Li, D. Wang y G. Liu, "An intrusion detection system based on convolution neural network," en, *PeerJ Computer Science*, vol. 10, e2152, jun. de 2024, issn: 2376-5992. doi: 10.7717/peerj-cs.2152. visitado 22 de ago. de 2025. dirección: https://peerj.com/articles/cs-2152.

[4] I. Agrafiotis, J. R. C. Nurse, M. Goldsmith, S. Creese y D. Upton, "A taxonomy of cyber-harms: Defining the impacts of cyber-attacks and understanding how they propagate," en, *Journal of Cybersecurity*, vol. 4, n.º 1, ene. de 2018, issn: 2057-2085, 2057-2093. doi: 10.1093/cybsec/tyy006. visitado 2 de oct. de 2025. dirección: https://academic.oup.com/cybersecurity/article/doi/10.1093/cybsec/tyy006/5133288.

[5] K. Lab, "State of Ransomware in 2025," en, Kaspersky Lab, Moscow, Russia, Report, mayo de 2025. dirección: https://www.kaspersky.com/reports/state-of-ransomware-2025.

[6] autor corporativo, "ESET Security Report 2024," en, ESET, spol. s r.o., Bratislava, Slovakia, Report N/A, 2024, pág. 189. dirección: https://web-assets.eset.com/fileadmin/ESET/SK/Docs/Vyrocne-spravy/vyrocna-sprava-2024-eng.pdf?utm_source=chatgpt.com.

[7] B. Guembe, A. Azeta, S. Misra, V. C. Osamor, L. Fernandez-Sanz y V. Pospelova, "The Emerging Threat of Ai-driven Cyber Attacks: A Review," en, *Applied Artificial Intelligence*, vol. 36, n.º 1, pág. 2 037 254, dic. de 2022, issn: 0883-9514, 1087-6545. doi: 10.1080/08839514.2022.2037254. visitado 2 de oct. de 2025. dirección: https://www.tandfonline.com/doi/full/10.1080/08839514.2022.2037254.

[8] Z. Li, F. Liu, W. Yang, S. Peng y J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, n.º 12, págs. 6999-7019, dic. de 2022, issn: 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2021.3084827. visitado 2 de oct. de 2025. dirección: https://ieeexplore.ieee.org/document/9451544/.

[9] L. Mohammadpour, T. C. Ling, C. S. Liew y A. Aryanfar, "A Survey of CNN-Based Network Intrusion Detection," en, *Applied Sciences*, vol. 12, n.º 16, pág. 8162, ago. de 2022, issn: 2076-3417. doi: 10.3390/app12168162. visitado 23 de ago. de 2025. dirección: https://www.mdpi.com/2076-3417/12/16/8162.

[10] P. García-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández y E. Vázquez, "Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges," *Computers & Security*, vol. 28, n.º 1–2, págs. 18-28, 2009. doi: 10.1016/j.cose.2008.08.003.

[11] M. A. Ferrag, L. Maglaras, S. Moschoyiannis y H. Janicke, "Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study," *Journal of Information Security and Applications*, vol. 50, pág. 102 419, feb. de 2020. doi: 10.1016/j.jisa.2019.102419.

[12] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj y D. J. Inman, "1D Convolutional Neural Networks and Applications: A Survey," *Mechanical Systems and Signal Processing*, vol. 151, pág. 107 398, abr. de 2021. doi: 10.1016/j.ymssp.2020.107398.

[13] I. Sharafaldin, A. H. Lashkari y A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," en *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, 2018, págs. 108-116. doi: 10.5220/0006639801080116.

[14] N. Moustafa y J. Slay, "The Evaluation of Network Anomaly Detection Systems: Statistical Analysis of the UNSW-NB15 Data Set and the Comparison with the KDD99 Data Set," *Information Security Journal: A Global Perspective*, vol. 25, n.º 1-3, págs. 18-31, 2016. doi: 10.1080/19393555.2015.1125974.

[15] A. Khraisat, I. Gondal, P. Vamplew y J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity – SpringerOpen*, vol. 2, n.º 1, pág. 23, 2019. doi: 10.1186/s42400-019-0038-7. dirección: https://link.springer.com/article/10.1186/s42400-019-0038-7.

[16] L. Diana, P. Dini y D. Paolini, "Overview on Intrusion Detection Systems for Computers," *Computers*, vol. 14, n.º 3, pág. 87, 2024. doi: 10.3390/computers14030087. dirección: https://www.mdpi.com/2073-431X/14/3/87.

[17] S. A. Bakhsh, M. A. Khan, F. Ahmed, M. S. Alshehri, H. Ali y J. Ahmad, "Enhancing IoT network security through deep learning-powered Intrusion Detection System," *Internet of Things*, vol. 24, pág. 100 936, 2023. doi: 10.1016/j.iot.2023.100936. dirección: https://www.sciencedirect.com/science/article/pii/S2542660523002597.

[18] D. Gumusbas, T. Yildirim, A. Genovese y F. Scotti, "A Comprehensive Survey of Databases and Deep Learning Methods for Cybersecurity and Intrusion Detection Systems," *IEEE Systems Journal*, vol. 15, n.º 2, págs. 1717-1731, jun. de 2021, issn: 1932-8184, 1937-9234, 2373-7816. doi: 10.1109/JSYST.2020.2992966. dirección: https://ieeexplore.ieee.org/document/9099844.

[19] P. Oscarson, "Information Security Fundamentals: Graphical Conceptualisations for Understanding," en *Security Education and Critical Infrastructures*, C. Irvine et al., eds., Springer Science+Business Media, 2003, págs. 95-107. doi: 10.1007/978-0-387-35694-5_9.

[20] National Institute of Standards and Technology (NIST), *CIA Triad*, https://www.nist.gov/image/cia-triad?utm_source=chatgpt.com, Accedido el 10 de enero de 2026, 2023.

[21] H. S. Lallie, K. Debattista y J. Bal, "A review of attack graph and attack tree visual syntax in cyber security," *Computer Science Review*, vol. 35, pág. 100 219, 2020. doi: 10.1016/j.cosrev.2020.100219. dirección: https://www.sciencedirect.com/science/article/pii/S1574013719300772.

[22] European Union Agency for Cybersecurity (ENISA), "ENISA Threat Landscape 2023," ENISA, inf. téc., 2023. dirección: https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Threat%20Landscape%202023.pdf.

[23] Z. Zhang, H. Al Hamadi, E. Damiani et al., "Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research," *Information*, vol. 13, n.º 4, pág. 180, 2022.

[24] R. Bace y P. Mell, "Intrusion Detection Systems," National Institute of Standards y Technology (NIST), inf. téc. NIST Special Publication 800-31, nov. de 2001. doi: 10.6028/NIST.SP.800-31.

[25] S. Axelsson, "The Base-Rate Fallacy and the Difficulty of Intrusion Detection," *ACM Transactions on Information and System Security*, vol. 3, n.º 3, págs. 186-205, ago. de 2000. doi: 10.1145/357830.357849.

[26] H. Debar, M. Dacier y A. Wespi, "A Revised Taxonomy for Intrusion-Detection Systems," *Annales des Télécommunications*, vol. 55, n.º 7–8, págs. 361-378, jul. de 2000. doi: 10.1007/BF02994844.

[27] K. A. Scarfone y P. M. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," National Institute of Standards y Technology, NIST Special Publication 800-94, 2007. doi: 10.6028/NIST.SP.800-94.

[28] I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters y A. Ng, "Cybersecurity Data Science: An Overview from Machine Learning Perspective," *Journal of Big Data*, vol. 7, n.º 41, 2020. doi: 10.1186/s40537-020-00318-5.

[29] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras y B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, n.º 3, págs. 343-356, 2010. doi: 10.1109/SURV.2010.032210.00054.

[30] A. Mishra, V. Gupta y B. B. Gupta, "Machine Learning and Deep Learning Methods for Cybersecurity: A Review," en *Advances in Computational Intelligence and Robotics (ACIR)*, IGI Global, 2020, págs. 1-20. doi: 10.4018/978-1-7998-7705-9.ch084.

[31] K. A. Scarfone y P. M. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," National Institute of Standards y Technology, inf. téc. 800-94, feb. de 2007, [Online]. dirección: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication80
94.pdf.

[32] M. A. Ferrag et al., "Machine Learning and Deep Learning for Network Traffic Analysis," *Applied Sciences*, vol. 15, n.º 12, pág. 6546, 2025. doi: 10.3390/app15126546.

[33] A. Thakkar y R. Lohiya, "A Review on Machine Learning and Deep Learning Perspectives of IDS for IoT: Recent Updates, Security Issues, and Challenges," *Archives of Computational Methods in Engineering*, vol. 28, págs. 3211-3243, 2021. doi: 10.1007/s11831-020-09496-0.

[34] N. Moustafa, J. Hu y J. Slay, "A Holistic Review of Network Anomaly Detection Systems: A Comprehensive Survey," *Journal of Network and Computer Applications*, vol. 204, pág. 103 419, 2022. doi: 10.1016/j.jnca.2022.103419.

[35] I. Sharafaldin, A. H. Lashkari, S. Hakak y A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," en *Proceedings of the International Carnahan Conference on Security Technology (ICCST)*, 2019, págs. 1-8. doi: 10.1109/CCST.2019.8888419.

[36] G. Karatas, O. Demir y O. K. Sahingoz, "Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset," *IEEE Access*, vol. 8, págs. 32 150-32 162, 2020. doi: 10.1109/ACCESS.2020.2973219.

[37] M. Ring, S. Wunderlich, D. Grüdl, D. Landes y A. Hotho, "A Survey of Network-based Intrusion Detection Data Sets," *Computers & Security*, vol. 86, págs. 147-167, sep. de 2019. doi: 10.1016/j.cose.2019.06.005.

[38] M. A. Al-Shareeda et al., “A Comprehensive Survey: Evaluating the Efficiency of Artificial Intelligence and Machine Learning Techniques on Cyber Security Solutions,” *IEEE Access*, vol. 12, 2024. doi: 10.1109/ACCESS.2024.3355547.

[39] N. Mohamed, “Artificial intelligence and machine learning in cybersecurity: a deep dive into state-of-the-art techniques and future paradigms,” *Knowledge and Information Systems*, vol. 67, págs. 6969-7055, 2025. doi: 10.1007/s10115-025-02429-y.

[40] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti y M. Colajanni, “Cybersecurity Meets Artificial Intelligence: A Survey of Applications, Attacks, and Challenges,” *ACM Computing Surveys*, vol. 55, n.º 12, págs. 1-36, 2023. doi: 10.1145/3571726.

[41] S. Pouyanfar et al., “A Survey on Deep Learning: Algorithms, Techniques, and Applications,” *ACM Computing Surveys*, vol. 51, n.º 5, págs. 1-36, sep. de 2018. doi: 10.1145/3234150.

[42] M. Mortezapour et al., “An Overview of Deep Learning Models for Cybersecurity Applications,” *arXiv preprint arXiv:2305.17473*, 2023.

[43] K. Zhang, H. Ying, H.-N. Dai y H. Yu, *General structures of Convolutional Neural Networks and Recurrent Neural Networks*, Diagrama disponible en ResearchGate, 2021. dirección: https://www.researchgate.net/figure/General-structures-of-CNN-and-RNN-The-main-structure-of-CNNs-consists-of-convolutional_fig3_349798432.

[44] A. del artículo (si aparece) o ResearchGate, *Proposed LSTM and GRU models architecture*, Diagrama disponible en ResearchGate, 2021. dirección: https://www.researchgate.net/figure/Proposed-LSTM-and-GRU-models-architecture_fig4_354827064.

[45] Z. Akkus, J. Cai, A. Boonrod et al., “A Schematic Illustration of a Deep Learning Training Process for a Classification Task,” *Nombre de la Revista / Fuente*, 2019, Figura de entrenamiento supervisado tomada de ResearchGate. dirección: https://www.researchgate.net/figure/A-schematic-illustration-of-a-deep-learning-training-process-for-a-classification-task_fig1_335578068.

[46] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," *SN Computer Science*, vol. 2, n.º 6, pág. 420, nov. de 2021. doi: 10.1007/s42979-021-00815-1.

[47] Y. Xin et al., "Machine Learning and Deep Learning Methods for Cybersecurity," *IEEE Access*, vol. 6, págs. 35 365-35 381, 2018. doi: 10.1109/ACCESS.2018.2836950.

[48] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez y F. Herrera, "Big Data Preprocessing: Methods and Prospects," *Big Data Analytics*, vol. 1, n.º 1, pág. 9, 2016. doi: 10.1186/s41044-016-0014-0.

[49] S. Ho, S. A. Jufout, K. Dajani y M. Mozumdar, "A Novel Intrusion Detection Model for Detecting Known and Innovative Cyberattacks Using Convolutional Neural Network," *IEEE Open Journal of the Computer Society*, vol. 2, págs. 14-25, 2021, issn: 2644-1268. doi: 10.1109/OJCS.2021.3050917. visitado 30 de ago. de 2025. dirección: https://ieeexplore.ieee.org/document/9320588/.

[50] R. A. Abed, E. K. Hamza y A. J. Humaidi, "A modified CNN-IDS model for enhancing the efficacy of intrusion detection system," en, *Measurement: Sensors*, vol. 35, pág. 101 299, oct. de 2024, issn: 26659174. doi: 10.1016/j.measen.2024.101299. visitado 28 de ago. de 2025. dirección: https://linkinghub.elsevier.com/retrieve/pii/S2665917424002757.

[51] O. A. Ayeni, S. C. Ewa y O. Owolafe, "Convolutional Neural Network Based Model for Intrusion Detection," *International Journal of RFID Security and Cryptography*, vol. 6, n.º 1, págs. 215-222, jun. de 2023, issn: 20463715. doi: 10.20533/ijrfidsc.2046.3715.2023.0025. visitado 27 de ago. de 2025. dirección: https://infonomics-society.org/wp-content/uploads/Convolutional-Neural-Network-Based-Model-for-Intrusion-Detection.pdf.

[52] I. Sharafaldin, A. H. Lashkari y A. A. Ghorbani, "Intrusion detection evaluation dataset (CICIDS2017)," *Canadian Institute for Cybersecurity*, 2017. dirección: https://www.unb.ca/cic/datasets/ids-2017.html.

[53] Z. Xu e Y. Liu, "Robust Anomaly Detection in Network Traffic: Evaluating Machine Learning Models on CICIDS2017," *arXiv preprint*, vol. arXiv:2506.19877, 2025. doi: 10.48550/arXiv.2506.19877. dirección: https://doi.org/10.48550/arXiv.2506.19877.

[54] S. S. Tripathy y B. Behera, "A Review of Various Datasets for Machine Learning Algorithm-Based Intrusion Detection System: Advances and Challenges," *arXiv preprint*, vol. arXiv:2506.02438, 2025. doi: 10.48550/arXiv.2506.02438. dirección: https://doi.org/10.48550/arXiv.2506.02438.

[55] L. Huynh, J. Hesford, D. Cheng et al., "Expectations Versus Reality: Evaluating Intrusion Detection Systems in Practice," en *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2025, págs. 56-62. doi: 10.1109/DSN-S58359.2025.00021.

[56] J. Barrios, *La matriz de confusión y sus métricas*, Accedido: 2026, 2020. dirección: https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/.

[57] N. A. Azeez, M. A. Lawal y B. Arogundade, "Performance Evaluation of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for Intrusion Detection," *Covenant Journal of Informatics and Communication Technology*, vol. 12, n.º 1, jul. de 2024. dirección: https://journals.covenantuniversity.edu.ng/index.php/cjict/article/view/4374.

[58] V. Hnamte y J. Hussain, "Network Intrusion Detection using Deep Convolution Neural Network," en *2023 4th International Conference for Emerging Technology (INCET)*, IEEE, 2023, págs. 1-6. doi: 10.1109/INCET57972.2023.10170337.

[59] A. F. Al-zubidi, A. K. Farhan y S. M. Towfek, "Predicting DoS and DDoS attacks in network security scenarios using a hybrid deep learning model," *Journal of Intelligent Systems*, vol. 33, n.º 1, mayo de 2024. doi: 10.1515/jisys-2023-0195. dirección: https://doi.org/10.1515/jisys-2023-0195.

[60] D. Kilichev y W. Kim, "Hyperparameter Optimization for 1D-CNN-Based Network Intrusion Detection Using GA and PSO," *Mathematics*, vol. 11, n.º 17, pág. 3724, ago. de

2023. doi: 10.3390/math11173724. dirección: https://doi.org/10.3390/math11173724.

[61] E. U. H. Qazi, M. Imran, A. AlGhamdi, R. Ullah y E. Ahmed, "A One-Dimensional Convolutional Neural Network (1D-CNN) Based Deep Learning System for Network Intrusion Detection," *Applied Sciences*, vol. 12, n.º 16, pág. 7986, ago. de 2022. doi: 10.3390/app12167986. dirección: https://doi.org/10.3390/app12167986.

ANEXOS

Anexo A. Repositorio del proyecto

Con el propósito de facilitar la revisión, reproducción y consulta del sistema desarrollado, el código fuente completo de la aplicación, el modelo CNN, los archivos de configuración y los recursos utilizados durante esta investigación se encuentran disponibles en el siguiente repositorio de GitHub:

https://github.com/MerlyRivera2002/IDS-CNN

Anexo B. Código fuente de la aplicación principal

A continuación, se presenta el archivo **app.py**, correspondiente a la aplicación principal desarrollada en Python mediante Streamlit. Este archivo integra la interfaz gráfica del sistema, la carga del modelo CNN, el procesamiento de los conjuntos de datos, la ejecución de las simulaciones y la visualización de los resultados obtenidos durante la investigación.

Archivo: app.py

```
import os
import io
import time
import joblib
import numpy as np
import pandas as pd
import tensorflow as tf
import plotly.express as px
import plotly.graph_objects as go
import streamlit as st
from sklearn.metrics import (
    precision_score, recall_score, f1_score,
    accuracy_score, confusion_matrix
)
import logic

```

```
# Configuración general
st.set_page_config(
    page_title="IDS · CNN Tesis 2026",
    layout="wide",
    page_icon="",
    initial_sidebar_state="expanded"
)

# Tema visual
st.markdown("""
<style>
@import url('https://fonts.googleapis.com/css2?family=Space+Mono:wght@400;700&family=DM+Sans:wght@300;400;500;600&display=swap');

html, body, [data-testid="stApp"] {
    background: #080f1a !important;
    font-family: 'DM Sans', sans-serif;
}

[data-testid="stSidebar"] {
    background: #0d1929 !important;
    border-right: 1px solid #1a3a5c;
}
[data-testid="stSidebar"] * { color: #c8d8e8 !important; }
[data-testid="stSidebar"] input {
    background: #0a1622 !important;
    border: 1px solid #1e4a7a !important;
    color: #e0eaf5 !important;
    border-radius: 6px;
}
[data-testid="stSidebar"] .stButton > button {
    background: #0a2540 !important;
    border: 1px solid #1e6fbf !important;
    color: #7ec8f7 !important;
    border-radius: 6px;
    font-family: 'Space Mono', monospace;
    font-size: 12px;
    width: 100%;
}
[data-testid="stSidebar"] .stButton > button:hover {
    background: #0f3560 !important;
    border-color: #00c8ff !important;
}

```

```
[data-testid="stTabs"] [role="tablist"] {
    background: #0d1929;
    border-radius: 10px 10px 0 0;
    padding: 6px 8px 0;
    border-bottom: 1px solid #1a3a5c;
    gap: 4px;
}
[data-testid="stTabs"] button[role="tab"] {
    color: #6a8eaa !important;
    font-family: 'Space Mono', monospace;
    font-size: 12px;
    border-radius: 7px 7px 0 0;
    padding: 8px 20px;
    background: transparent;
    border: none;
    border-bottom: 2px solid transparent;
    transition: all .2s;
}
[data-testid="stTabs"] button[role="tab"]:hover {
    color: #7ec8f7 !important;
    background: #0a1f38;
}
[data-testid="stTabs"] button[role="tab"][aria-selected="true"] {
    color: #00c8ff !important;
    background: #0a1f38;
    border-bottom: 2px solid #00c8ff;
}
[data-testid="stTabsContent"] {
    background: #0d1929;
    border-radius: 0 0 10px 10px;
    padding: 1.5rem;
    border: 1px solid #1a3a5c;
    border-top: none;
}

[data-testid="stMetric"] {
    background: #0a1f38;
    border: 1px solid #1a4a7a;
    border-radius: 10px;
    padding: 14px 18px;
}
[data-testid="stMetricLabel"] { color: #7ea8c8 !important; font-size: 12px !important; font-
    family: 'Space Mono', monospace; text-transform: uppercase; letter-spacing: .06em; }
[data-testid="stMetricValue"] { color: #00c8ff !important; font-size: 26px !important; font-
```

```
    weight: 600 !important; }
[data-testid="stMetricDelta"] svg { display: none; }
[data-testid="stMetricDelta"] { color: #ff5555 !important; font-size: 13px !important; }

h1, h2, h3 { font-family: 'Space Mono', monospace !important; color: #e0eaf5 !important; }
h1 { font-size: 22px !important; letter-spacing: .04em; }
h2 { font-size: 17px !important; color: #7ec8f7 !important; }
h3 { font-size: 14px !important; color: #6a9ec0 !important; }

p, span, label, div { color: #c8d8e8; }

[data-testid="stDataFrame"], iframe {
    border: 1px solid #1a3a5c !important;
    border-radius: 8px !important;
    background: #080f1a !important;
}
.stDataFrame td, .stDataFrame th {
    font-size: 13px !important;
    color: #c8d8e8 !important;
    background: #0a1622 !important;
}

[data-testid="stFileUploader"] {
    border: 1.5px dashed #1e4a7a !important;
    border-radius: 10px;
    background: #080f1a !important;
    padding: 12px;
}

[data-testid="stSlider"] .stSlider > div { background: #00c8ff; }

.stButton > button {
    background: linear-gradient(135deg, #0a4080, #0d5fa0) !important;
    color: #e0eaf5 !important;
    border: 1px solid #1e6fbf !important;
    border-radius: 8px !important;
    font-family: 'Space Mono', monospace;
    font-size: 12px;
    letter-spacing: .04em;
    transition: all .2s;
}
.stButton > button:hover {
    background: linear-gradient(135deg, #0d5fa0, #0a7acf) !important;
    border-color: #00c8ff !important;
```

```
    transform: translateY(-1px);
    box-shadow: 0 4px 16px rgba(0,200,255,.2);
}
[data-testid="stButton-primary"] > button {
    background: linear-gradient(135deg, #00829a, #00c8ff) !important;
    color: #060e18 !important;
    border: none !important;
    font-weight: 700 !important;
    font-size: 13px !important;
}

[data-testid="stProgress"] > div > div {
    background: linear-gradient(90deg, #0060a0, #00c8ff) !important;
}

[data-testid="stSuccess"] { background: #0a2818 !important; border-left: 3px solid #00c853 !
    important; color: #8affa0 !important; }
[data-testid="stWarning"] { background: #1e1500 !important; border-left: 3px solid #ffaa00 !
    important; color: #ffd060 !important; }
[data-testid="stError"]   { background: #1e0808 !important; border-left: 3px solid #ff3333 !
    important; color: #ff8080 !important; }
[data-testid="stInfo"]    { background: #081828 !important; border-left: 3px solid #0080c0 !
    important; color: #7ec8f7 !important; }

hr { border-color: #1a3a5c !important; }

[data-testid="stDateInput"] input {
    background: #0a1622 !important;
    border: 1px solid #1e4a7a !important;
    color: #e0eaf5 !important;
    border-radius: 6px;
}

[data-testid="stSelectbox"] > div {
    background: #0a1622 !important;
    border: 1px solid #1e4a7a !important;
    border-radius: 6px;
}

[data-testid="stDownloadButton"] > button {
    background: #0a1f38 !important;
    border: 1px solid #1e6fbf !important;
    color: #7ec8f7 !important;
}
```

```
</style>
""", unsafe_allow_html=True)

# Colores y tema de gráficos

COLOR_ATAQUE  = "#ff4f5e"
COLOR_NORMAL  = "#00c8ff"
COLOR_ACCENT  = "#7ec8f7"
COLOR_WARN    = "#ffaa00"
COLOR_SUCCESS = "#00c853"

def apply_theme(fig, height=320, title="", xaxis_title="", yaxis_title="",
                xaxis_extra=None, yaxis_extra=None, no_axes=False, **kwargs):
    fig.update_layout(
        paper_bgcolor="rgba(0,0,0,0)",
        plot_bgcolor="#080f1a",
        font=dict(family="DM Sans", color="#c8d8e8", size=13),
        margin=dict(t=40, b=30, l=10, r=10),
        title_font=dict(family="Space Mono", color="#e0eaf5", size=14),
        height=height,
        title=title,
    )
    if "legend" not in kwargs:
        fig.update_layout(legend=dict(bgcolor="rgba(0,0,0,0)", font=dict(color="#c8d8e8")))
    if kwargs:
        fig.update_layout(**kwargs)
    if not no_axes:
        ax = dict(gridcolor="#142035", linecolor="#1a3a5c",
                  tickcolor="#1a3a5c", tickfont=dict(color="#7ea8c8"))
        x = dict(ax)
        y = dict(ax)
        if xaxis_extra:  x.update(xaxis_extra)
        if yaxis_extra:  y.update(yaxis_extra)
        if xaxis_title:  x["title"] = xaxis_title
        if yaxis_title:  y["title"] = yaxis_title
        fig.update_xaxes(**x)
        fig.update_yaxes(**y)
    return fig


# Login
if "perfil" not in st.session_state:
    st.session_state.perfil = None

```

```
st.sidebar.markdown("###  Control de Acceso")
if st.session_state.perfil is None:
    u = st.sidebar.text_input("Usuario")
    p = st.sidebar.text_input("Clave", type="password")
    if st.sidebar.button("Ingresar"):
        if u == "admin" and p == "tesis2026":
            st.session_state.perfil = "Administrador"
            st.rerun()
        elif u == "viewer" and p == "visita2026":
            st.session_state.perfil = "Visualizador"
            st.rerun()
        else:
            st.sidebar.error("Credenciales incorrectas")
    st.stop()
else:
    st.sidebar.success(f" {st.session_state.perfil}")
    st.sidebar.divider()
    st.sidebar.markdown(" ** Fecha de simulación**")
    fecha_simulada = st.sidebar.date_input("", value=pd.to_datetime("2026-04-01"),
        label_visibility="collapsed")
    st.sidebar.divider()
    if st.sidebar.button("Cerrar Sesión"):
        st.session_state.clear()
        st.rerun()

# Carga del modelo
@st.cache_resource
def load_assets():
    return (
        tf.keras.models.load_model("modelo_cnn.keras"),
        joblib.load("scaler.pkl"),
        joblib.load("features.pkl"),
    )

model, scaler, features_list = load_assets()

# Encabezado
st.markdown("""
<div style="background:linear-gradient(135deg,#0a1f38,#0d2d52);border:1px solid #1a4a7a;border
    -radius:12px;padding:18px 24px;margin-bottom:1.4rem;display:flex;align-items:center;gap:16
    px;">
  <div style="font-size:36px;" ></div>
  <div>
    <div style="font-family:'Space Mono',monospace;font-size:20px;color:#e0eaf5;font-weight
```

```
        :700;letter-spacing:.05em;">SISTEMA DE DETECCIÓN DE INTRUSIONES</div>
    <div style="font-family:'DM Sans',sans-serif;color:#7ea8c8;font-size:13px;margin-top:3px;"
        >Red Neuronal Convolucional 1D · Clasificación Binaria · CICIDS2017</div>
  </div>
</div>
""", unsafe_allow_html=True)

# Pestañas
tab1, tab2, tab3 = st.tabs([
    "   SIMULACIÓN EN VIVO",
    "   ANÁLISIS Y TENDENCIAS",
    "   MOVIMIENTOS Y REPORTES",
])

# Pestaña 1: simulación
with tab1:
    if st.session_state.perfil == "Administrador":
        st.header("Monitor de Tráfico en Tiempo Real")

        col_ctrl1, col_ctrl2, col_ctrl3, col_ctrl4 = st.columns([3, 1.5, 1.5, 1])
        with col_ctrl1:
            archivo = st.file_uploader("Subir dataset CSV", type=["csv"], key="uploader_sim")
        with col_ctrl2:
            velocidad = st.slider("Seg / lote", 0.02, 0.5, 0.08, 0.02)
        with col_ctrl3:
            umbral = st.slider("Umbral (sensibilidad)", 0.01, 0.99, 0.35, 0.01,
                               help="Valor bajo → detecta más ataques (más falsos positivos).
                                   Alto → más preciso, pero puede perder ataques.")
        with col_ctrl4:
            st.markdown("<br>", unsafe_allow_html=True)
            if st.button("  Limpiar", use_container_width=True):
                st.session_state.pop("simulacion_activa", None)
                st.rerun()

        col_izq, col_der = st.columns(2)
        with col_izq:
            esp_pastel   = st.empty()
            esp_evolucion = st.empty()
        with col_der:
            esp_metricas = st.empty()
            esp_puertos  = st.empty()

        st.divider()
        st.subheader("Registro de actividad - último lote")
```

```
312         esp_tabla = st.empty()
313
314         if "simulacion_activa" not in st.session_state:
315             st.session_state.simulacion_activa = False
316             for e in [esp_pastel, esp_evolucion, esp_metricas, esp_puertos, esp_tabla]:
317                 e.info("Esperando inicio de …simulación")
318
319         if archivo:
320             if st.button("  INICIAR MONITOREO", type="primary", use_container_width=True):
321                 with st.spinner("Preparando …datos"):
322                     df_raw   = pd.read_csv(archivo)
323                     df_raw.columns = df_raw.columns.str.strip()
324                     df_clean = df_raw.replace([np.inf, -np.inf], np.nan).dropna()
325                     total    = len(df_clean)
326
327                     preds_totales = []
328                     tiempos, ataques_acum = [], []
329                     t0 = time.time()
330                     pbar = st.progress(0, text="Simulación en …curso")
331
332                     for idx, inicio in enumerate(range(0, total, 15)):
333                         chunk = df_clean.iloc[inicio:inicio+15]
334                         X_chunk = scaler.transform(
335                             chunk[features_list]
336                         ).reshape(-1, len(features_list), 1)
337                         chunk_preds = (
338                             model.predict(X_chunk, verbose=0) > umbral
339                         ).astype(int).flatten()
340                         preds_totales.extend(chunk_preds)
341
342                         ataques = int(sum(preds_totales))
343                         normales = len(preds_totales) - ataques
344                         ataques_acum.append(ataques)
345                         tiempos.append(len(preds_totales))
346
347                         prog = (inicio + len(chunk)) / total
348                         pbar.progress(min(prog, 1.0), text=f"Procesando {int(prog*100)}%")
349
350                         fig_pie = go.Figure(go.Pie(
351                             values=[normales, ataques],
352                             labels=["Normal", "Ataque"],
353                             hole=0.62,
354                             marker=dict(colors=[COLOR_NORMAL, COLOR_ATAQUE],
355                                         line=dict(color="#080f1a", width=3)),
```

```
                    textfont=dict(size=13, color="#e0eaf5"),
                ))
                apply_theme(fig_pie, height=280, no_axes=True,
                            title=f"Tráfico - {len(preds_totales)} registros",
                            showlegend=True,
                            legend=dict(orientation="h", y=-0.05,
                                        font=dict(color="#c8d8e8", size=12)),
                            annotations=[dict(
                                text=f"<b>{ataques}</b><br><span style='font-size:10px
                                    '>ataques</span>",
                                x=0.5, y=0.5, font=dict(size=16, color=COLOR_ATAQUE),
                                showarrow=False
                            )])
                esp_pastel.plotly_chart(fig_pie, use_container_width=True, key=f"pie_{
                    idx}")

                fig_evol = go.Figure()
                fig_evol.add_trace(go.Scatter(
                    x=tiempos, y=ataques_acum, mode="lines",
                    fill="tozeroy",
                    line=dict(color=COLOR_ATAQUE, width=3),
                    fillcolor="rgba(255,79,94,.15)",
                    name="Ataques acumulados"
                ))
                apply_theme(fig_evol, height=240,
                            title="Evolución de intrusiones",
                            xaxis_title="Registros procesados",
                            yaxis_title="Ataques")
                esp_evolucion.plotly_chart(fig_evol, use_container_width=True, key=f"
                    evol_{idx}")

                with esp_metricas.container():
                    m1, m2 = st.columns(2)
                    m1.metric("Conexiones", f"{len(preds_totales):,}")
                    m2.metric("Intrusiones", f"{ataques:,}", delta=f"+{int(chunk_preds
                        .sum())}")
                    elapsed = time.time() - t0
                    m3, m4 = st.columns(2)
                    m3.metric("Reg/seg", f"{len(preds_totales)/max(elapsed,0.01):.1f}"
                        )
                    col_label = next((c for c in df_clean.columns if c.lower() == "
                        label"), None)
                    if col_label:
                        y_true_p = df_clean[col_label].iloc[:len(preds_totales)].
```

```
                    astype(str).str.upper().apply(
                    lambda x: 0 if "BENIGN" in x or "NORMAL" in x else 1)
                acc_p = accuracy_score(y_true_p, preds_totales)
                m4.metric("Accuracy parcial", f"{acc_p:.2%}")

        with esp_puertos.container():
            puertos = chunk["Destination Port"].value_counts().head(5).
                reset_index()
            puertos.columns = ["Puerto", "Freq"]
            fig_p = go.Figure(go.Bar(
                x=puertos["Puerto"].astype(str),
                y=puertos["Freq"],
                marker=dict(color=puertos["Freq"],
                            colorscale=[[0,"#0a3060"],[1,COLOR_ATAQUE]],
                            line=dict(width=0)),
                text=puertos["Freq"], textposition="outside",
                textfont=dict(color="#c8d8e8", size=12),
            ))
            apply_theme(fig_p, height=210,
                        title="Top 5 puertos - lote actual",
                        xaxis_title="Puerto destino",
                        yaxis_title="Paquetes")
            st.plotly_chart(fig_p, use_container_width=True, key=f"puertos_{
                idx}")

        with esp_tabla.container():
            vista = chunk[["Destination Port"] + [c for c in chunk.columns if
                c != "Destination Port"][:4]].copy()
            vista.insert(0, "Estado", ["  ATAQUE" if p == 1 else "  NORMAL" for
                 p in chunk_preds])
            def diagnose(row):
                if "NORMAL" in row["Estado"]:
                    return "Tráfico seguro"
                p = row["Destination Port"]
                if p in [80, 443]: return "Ataque web HTTP/S"
                if p == 22: return "Fuerza bruta SSH"
                if p == 21: return "Acceso FTP no autorizado"
                return "Escaneo / puerto sospechoso"
            vista["Diagnóstico"] = vista.apply(diagnose, axis=1)
            st.dataframe(
                vista,
                use_container_width=True,
                height=400,
                hide_index=True,
```

```
                )

            time.sleep(velocidad)

        pbar.empty()
        st.success("  Simulación finalizada.")
        st.balloons()

        st.subheader("Resumen ejecutivo")
        elapsed_total = time.time() - t0
        c1, c2, c3, c4 = st.columns(4)
        c1.metric("Total registros",    f"{len(preds_totales):,}")
        c2.metric("Ataques detectados", f"{ataques:,}",
                  delta=f"{ataques/len(preds_totales):.1%}")
        c3.metric("Tiempo simulación",  f"{elapsed_total:.2f} s")
        c4.metric("Eficiencia",         f"{len(preds_totales)/elapsed_total:.1f}
            reg/s")

        col_label = next((c for c in df_clean.columns if c.lower() == "label"),
            None)
        if col_label:
            y_true = df_clean[col_label].astype(str).str.upper().apply(
                lambda x: 0 if "BENIGN" in x or "NORMAL" in x else 1
            )[:len(preds_totales)]
            acc  = accuracy_score(y_true,  preds_totales)
            prec = precision_score(y_true, preds_totales, zero_division=0)
            rec  = recall_score(y_true,    preds_totales, zero_division=0)
            f1   = f1_score(y_true,        preds_totales, zero_division=0)

            st.subheader("Evaluación del modelo CNN")
            df_met = pd.DataFrame({
                "Métrica": ["Accuracy", "Precision", "Recall", "F1-Score"],
                "Valor":   [acc*100, prec*100, rec*100, f1*100],
            })
            fig_met = go.Figure()
            colors_met = [COLOR_NORMAL, COLOR_SUCCESS, COLOR_WARN, COLOR_ATAQUE]
            for i, row in df_met.iterrows():
                fig_met.add_trace(go.Bar(
                    name=row["Métrica"],
                    x=[row["Métrica"]],
                    y=[row["Valor"]],
                    marker_color=colors_met[i],
                    text=[f"{row['Valor']:.2f}%"],
                    textposition="outside",
```

```
                textfont=dict(size=14, color="#e0eaf5"),
                width=0.5,
            ))
        apply_theme(fig_met, height=360,
                    title="Métricas de clasificación binaria",
                    showlegend=False,
                    yaxis_extra=dict(range=[0, 105], ticksuffix="%"),
                    bargap=0.35)
        st.plotly_chart(fig_met, use_container_width=True)

        st.subheader("Matriz de Confusión")
        cm = confusion_matrix(y_true, preds_totales, labels=[0, 1])
        fig_cm = go.Figure(go.Heatmap(
            z=cm,
            x=["Pred: Normal", "Pred: Ataque"],
            y=["Real: Normal", "Real: Ataque"],
            colorscale=[[0, "#0a1f38"], [0.5, "#0060a0"], [1, COLOR_ATAQUE]],
            text=cm, texttemplate="<b>%{text}</b>",
            textfont=dict(size=22, color="#ffffff"),
            showscale=True,
            colorbar=dict(tickfont=dict(color="#c8d8e8")),
        ))
        apply_theme(fig_cm, height=340,
                    title="Matriz de Confusión - Normal vs Ataque",
                    xaxis_extra=dict(title="Predicción", side="bottom",
                                     tickfont=dict(size=13, color="#7ea8c8")),
                    yaxis_extra=dict(title="Valor Real",
                                     tickfont=dict(size=13, color="#7ea8c8"),
                                     autorange="reversed"))
        st.plotly_chart(fig_cm, use_container_width=True)

        p_top = df_clean.iloc[:len(preds_totales)]["Destination Port"].mode()
            [0]
        logic.guardar_en_historial(
            "historial.csv", archivo.name, len(preds_totales), ataques,
            elapsed_total, fecha_simulada, p_top, acc,
            precision=prec, recall=rec, f1=f1,
        )
    else:
        p_top = df_clean.iloc[:len(preds_totales)]["Destination Port"].mode()
            [0]
        logic.guardar_en_historial(
            "historial.csv", archivo.name, len(preds_totales), ataques,
            elapsed_total, fecha_simulada, p_top, 0.0,
```

```
                        )
                    st.toast("Simulación guardada en bitácora ")
    else:
        st.warning("  Solo Administradores pueden ejecutar simulaciones.")
# Pestaña 2: análisis histórico
with tab2:
    st.header("Análisis Histórico y Tendencias")

    archivo_local = "historial.csv"

    if not os.path.exists(archivo_local):
        st.warning("Sin datos históricos. Ejecuta una simulación primero.")
        st.stop()

    try:
        df_h = pd.read_csv(archivo_local)
    except Exception as e:
        st.error(f"Error al leer historial: {e}")
        if st.button("  Borrar archivo corrupto"):
            os.remove(archivo_local)
            st.rerun()
        st.stop()

    if df_h.empty:
        st.warning("Historial vacío. Ejecuta una simulación.")
        st.stop()

    df_h["Fecha"] = pd.to_datetime(df_h["Fecha"], errors="coerce")
    for col in ["Accuracy", "Precision", "Recall", "F1"]:
        if col in df_h.columns:
            df_h[col] = pd.to_numeric(df_h[col], errors="coerce")
    df_h = df_h.dropna(subset=["Fecha"]).sort_values("Fecha")

    st.subheader("Resumen global")
    k1, k2, k3, k4 = st.columns(4)
    k1.metric("Total simulaciones",      len(df_h))
    k2.metric("Total ataques detectados", f"{df_h['Ataques'].sum():,}")
    k3.metric("Accuracy promedio",        f"{df_h['Accuracy'].mean():.2%}")
    k4.metric("Puerto más atacado",       df_h.loc[df_h["Ataques"].idxmax(), "Puerto"])

    st.divider()

    st.subheader("Evolución temporal de intrusiones")
    fig_line = go.Figure()
```

```
    fig_line.add_trace(go.Scatter(
        x=df_h["Fecha"], y=df_h["Ataques"],
        mode="lines+markers",
        line=dict(color=COLOR_ATAQUE, width=3),
        marker=dict(size=9, color=COLOR_ATAQUE,
                    line=dict(color="#080f1a", width=2)),
        fill="tozeroy", fillcolor="rgba(255,79,94,.12)",
        name="Ataques"
    ))
    apply_theme(fig_line, height=320,
                title="Ataques detectados a lo largo del tiempo",
                xaxis_title="Fecha", yaxis_title="Cantidad de ataques",
                xaxis_extra=dict(tickformat="%d %b %Y", tickangle=-30))
    st.plotly_chart(fig_line, use_container_width=True)

    st.divider()

    st.subheader("Distribución de puertos atacados")
    puertos_df = (
        df_h["Puerto"].value_counts().reset_index()
    )
    puertos_df.columns = ["Puerto", "Frecuencia"]
    puertos_df["Puerto"] = puertos_df["Puerto"].astype(str)

    fig_tree = go.Figure(go.Treemap(
        labels=puertos_df["Puerto"],
        parents=[""] * len(puertos_df),
        values=puertos_df["Frecuencia"],
        textinfo="label+value+percent root",
        textfont=dict(size=14, color="#ffffff"),
        marker=dict(
            colorscale=[[0, "#0a2040"], [0.4, "#0060a0"], [1, COLOR_ATAQUE]],
            colorbar=dict(tickfont=dict(color="#c8d8e8")),
            line=dict(width=2, color="#080f1a"),
        ),
        hovertemplate="<b>%{label}</b><br>Frecuencia: %{value}<extra></extra>",
    ))
    apply_theme(fig_tree, height=380, no_axes=True, title="Puertos más atacados (tamaño =
        frecuencia)")
    st.plotly_chart(fig_tree, use_container_width=True)

    st.divider()

    st.subheader("Métricas globales del modelo")
```

```
604
605     acc_g  = df_h["Accuracy"].mean()
606     prec_g = df_h["Precision"].mean()
607     rec_g  = df_h["Recall"].mean()
608     f1_g   = df_h["F1"].mean()
609
610     if "metrica_activa" not in st.session_state:
611         st.session_state.metrica_activa = None
612
613     m1, m2, m3, m4 = st.columns(4)
614     with m1:
615         st.metric("Accuracy",  f"{acc_g:.2%}")
616         if st.button("Ver evolución", key="btn_acc"): st.session_state.metrica_activa = "
                Accuracy"
617     with m2:
618         st.metric("Precision", f"{prec_g:.2%}")
619         if st.button("Ver evolución", key="btn_prec"): st.session_state.metrica_activa = "
                Precision"
620     with m3:
621         st.metric("Recall",    f"{rec_g:.2%}")
622         if st.button("Ver evolución", key="btn_rec"): st.session_state.metrica_activa = "
                Recall"
623     with m4:
624         st.metric("F1-Score",  f"{f1_g:.2%}")
625         if st.button("Ver evolución", key="btn_f1"): st.session_state.metrica_activa = "F1"
626
627     if st.session_state.metrica_activa:
628         met = st.session_state.metrica_activa
629         colors_map = {"Accuracy": COLOR_NORMAL, "Precision": COLOR_SUCCESS,
630                       "Recall": COLOR_WARN, "F1": COLOR_ATAQUE}
631         fig_ev = go.Figure()
632         fig_ev.add_trace(go.Scatter(
633             x=df_h["Fecha"], y=df_h[met],
634             mode="lines+markers",
635             line=dict(color=colors_map[met], width=3),
636             marker=dict(size=9, color=colors_map[met],
637                         line=dict(color="#080f1a", width=2)),
638             fill="tozeroy",
639             fillcolor=f"rgba({','.join(str(int(colors_map[met].lstrip('#')[i:i+2],16)) for i
                    in (0,2,4))},.12)",
640         ))
641         apply_theme(fig_ev, height=300,
642                     title=f"Evolución de {met}",
643                     yaxis_extra=dict(tickformat=".0%"),
```

```
                    xaxis_extra=dict(tickformat="%d %b %Y", tickangle=-30))
        st.plotly_chart(fig_ev, use_container_width=True)

    st.divider()

    st.subheader("Matriz de Confusión acumulada (todas las simulaciones)")

    cols_cm = ["TN", "FP", "FN", "TP"]
    if all(c in df_h.columns for c in cols_cm):
        tn = int(df_h["TN"].sum())
        fp = int(df_h["FP"].sum())
        fn = int(df_h["FN"].sum())
        tp = int(df_h["TP"].sum())
    else:
        total_s   = df_h["Total"].sum() if "Total" in df_h.columns else 1000
        ataques_s = df_h["Ataques"].sum() if "Ataques" in df_h.columns else 100
        normales_s = total_s - ataques_s
        tp = int(ataques_s * rec_g)
        fn = int(ataques_s - tp)
        fp = int(tp / max(prec_g, 0.0001) - tp)
        tn = int(normales_s - fp)

    cm_vals = [[tn, fp], [fn, tp]]
    fig_cm_hist = go.Figure(go.Heatmap(
        z=cm_vals,
        x=["Pred: Normal", "Pred: Ataque"],
        y=["Real: Normal", "Real: Ataque"],
        colorscale=[[0, "#0a1f38"], [0.5, "#0060a0"], [1, COLOR_ATAQUE]],
        text=[[f"{tn:,}", f"{fp:,}"], [f"{fn:,}", f"{tp:,}"]],
        texttemplate="<b>%{text}</b>",
        textfont=dict(size=20, color="#ffffff"),
        showscale=True,
        colorbar=dict(tickfont=dict(color="#c8d8e8")),
    ))
    apply_theme(fig_cm_hist, height=360,
                title="Matriz de Confusión acumulada - Normal vs Ataque",
                xaxis_extra=dict(title="Predicción", tickfont=dict(size=13, color="#7ea8c8")),
                yaxis_extra=dict(title="Valor Real", tickfont=dict(size=13, color="#7ea8c8"),
                    autorange="reversed"))
    st.plotly_chart(fig_cm_hist, use_container_width=True)

    if (tp + fn) > 0 and (tp + fp) > 0:
        recall_calc = tp / (tp + fn)
        prec_calc   = tp / (tp + fp)
```

```
        st.markdown(f"""
        <div style="display:flex;gap:12px;flex-wrap:wrap;margin-top:6px;">
          <div style="background:#0a1f38;border:1px solid #1a4a7a;border-radius:8px;padding:10
              px 18px;">
            <div style="font-size:11px;color:#7ea8c8;font-family:'Space Mono',monospace;text-
                transform:uppercase;">Verdaderos Negativos</div>
            <div style="font-size:22px;color:#00c8ff;font-weight:600;">{tn:,}</div>
          </div>
          <div style="background:#0a1f38;border:1px solid #1a4a7a;border-radius:8px;padding:10
              px 18px;">
            <div style="font-size:11px;color:#7ea8c8;font-family:'Space Mono',monospace;text-
                transform:uppercase;">Falsos Positivos</div>
            <div style="font-size:22px;color:#ffaa00;font-weight:600;">{fp:,}</div>
          </div>
          <div style="background:#0a1f38;border:1px solid #1a4a7a;border-radius:8px;padding:10
              px 18px;">
            <div style="font-size:11px;color:#7ea8c8;font-family:'Space Mono',monospace;text-
                transform:uppercase;">Falsos Negativos</div>
            <div style="font-size:22px;color:#ff4f5e;font-weight:600;">{fn:,}</div>
          </div>
          <div style="background:#0a1f38;border:1px solid #1a4a7a;border-radius:8px;padding:10
              px 18px;">
            <div style="font-size:11px;color:#7ea8c8;font-family:'Space Mono',monospace;text-
                transform:uppercase;">Verdaderos Positivos</div>
            <div style="font-size:22px;color:#00c853;font-weight:600;">{tp:,}</div>
          </div>
        </div>
        """, unsafe_allow_html=True)

    st.divider()

    st.subheader("Registro detallado de simulaciones")
    columnas = ["Fecha", "Hora", "Dataset", "Total", "Normales", "Ataques",
                "Accuracy", "Precision", "Recall", "F1", "Puerto", "Tiempo (s)"]
    for col in columnas:
        if col not in df_h.columns:
            df_h[col] = np.nan
    df_disp = df_h.copy()
    df_disp["Fecha"] = df_disp["Fecha"].dt.date
    for col in ["Accuracy", "Precision", "Recall", "F1"]:
        if col in df_disp.columns:
            df_disp[col] = df_disp[col].apply(lambda x: f"{x:.2%}" if pd.notnull(x) else "-")
    st.dataframe(df_disp[columnas], use_container_width=True, height=400, hide_index=True)

```

```
723
724 # Pestaña 3: reportes
725 with tab3:
726     st.header("Movimientos y Reportes")
727     df_h3 = logic.obtener_metricas_resumen("historial.csv")
728
729     if df_h3 is None or df_h3.empty:
730         st.info("  Sin datos históricos. Realiza una simulación primero.")
731         st.stop()
732
733     st.markdown("""
734     <div style="background:linear-gradient(135deg,#0a1f38,#0d2d52);border:1px solid #1a4a7a;
            border-radius:10px;padding:14px 20px;margin-bottom:1.2rem;">
735       <span style="font-family:'Space Mono',monospace;color:#7ea8c8;font-size:12px;text-
              transform:uppercase;letter-spacing:.08em;">Resumen del historial completo</span>
736     </div>
737     """, unsafe_allow_html=True)
738     s1, s2, s3, s4 = st.columns(4)
739     s1.metric("Simulaciones totales",    len(df_h3))
740     s2.metric("Registros analizados",    f"{df_h3['Total'].sum():,}" if "Total" in df_h3 else
            "-")
741     s3.metric("Total ataques detectados", f"{df_h3['Ataques'].sum():,}" if "Ataques" in df_h3
            else "-")
742     s4.metric("F1 promedio",             f"{df_h3['F1'].mean():.2%}" if "F1" in df_h3 else "-"
            )
743
744     st.divider()
745
746     st.subheader("Listado de movimientos")
747     columnas_mov = ["Fecha", "Hora", "Dataset", "Total", "Normales", "Ataques",
748                     "Accuracy", "Precision", "Recall", "F1", "Puerto", "Tiempo (s)"]
749     col_exist = [c for c in columnas_mov if c in df_h3.columns]
750     df_mov = df_h3[col_exist].copy()
751     for col in ["Accuracy", "Precision", "Recall", "F1"]:
752         if col in df_mov.columns:
753             df_mov[col] = df_mov[col].apply(lambda x: f"{x:.2%}" if pd.notnull(x) else "-")
754     st.dataframe(df_mov, use_container_width=True, height=350, hide_index=True)
755
756     st.divider()
757
758     st.subheader("Reporte por rango de fechas")
759     min_f = df_h3["Fecha"].min().date()
760     max_f = df_h3["Fecha"].max().date()
761
```

```
    fa, fb = st.columns(2)
    with fa: f_ini = st.date_input("Desde", value=min_f, min_value=min_f, max_value=max_f, key
        ="rep_ini")
    with fb: f_fin = st.date_input("Hasta", value=max_f, min_value=min_f, max_value=max_f, key
        ="rep_fin")

    mask = (df_h3["Fecha"].dt.date >= f_ini) & (df_h3["Fecha"].dt.date <= f_fin)
    df_filt = df_h3.loc[mask].copy()

    if df_filt.empty:
        st.warning("Sin registros en ese rango de fechas.")
    else:
        st.success(f" {len(df_filt)} registros entre {f_ini} y {f_fin}")
        df_prev = df_filt[col_exist].copy()
        for col in ["Accuracy", "Precision", "Recall", "F1"]:
            if col in df_prev.columns:
                df_prev[col] = df_prev[col].apply(lambda x: f"{x:.2%}" if pd.notnull(x) else "
                    -")
        st.dataframe(df_prev, use_container_width=True, hide_index=True)

        buf = io.StringIO()
        df_filt.to_csv(buf, index=False)
        st.download_button("  Descargar reporte CSV",
                           data=buf.getvalue(),
                           file_name=f"reporte_{f_ini}_{f_fin}.csv",
                           mime="text/csv",
                           use_container_width=True)

    st.divider()

    st.subheader("Exportar historial completo")
    buf_all = io.StringIO()
    df_h3.to_csv(buf_all, index=False)
    st.download_button("  Descargar todas las simulaciones",
                       data=buf_all.getvalue(),
                       file_name="historial_completo.csv",
                       mime="text/csv",
                       use_container_width=True)

    st.divider()

    with st.expander("  Zona peligrosa"):
        if st.button("Borrar todo el historial", type="secondary"):
            if os.path.exists("historial.csv"):
```

```
                os.remove("historial.csv")
                st.success("Historial eliminado.")
                st.rerun()
```