Sistema preventivo contra ataques de denegación de servicio web utilizando Deep Learning

Juan Fernando Cañola García

Instituto Tecnológico Metropolitano

Facultad de Ingenierías

Medellín, Colombia

2020

Sistema preventivo contra ataques de denegación de servicio web utilizando Deep Learning

Juan Fernando Cañola García

Tesis o trabajo de investigación presentado como requisito parcial para optar al título de:

Magister en Seguridad Informática

Director:

Magister Gabriel Enrique Taborda Blandón

Línea de Investigación:

Ciencias computacionales

Instituto Tecnológico Metropolitano

Facultad de Ingenierías

Medellín, Colombia

2020

Lo que sabemos es una gota de agua, lo que ignoramos es el océano.

- *Isaac Newton*

Agradecimientos

31 de octubre del 2020, día que finalicé mi trabajo de grado y año en que la pandemia del covid-19 cambio nuestras vidas. Hoy más que agradecer, quiero felicitar a todas las personas que sin importar los cambios que esta pandemia ocasionó, salieron adelante en sus vidas, sus sueños, sus logros. ¡Ustedes son unos duros!

Como futuro magister en seguridad Informática quiero agradecer a todos los docentes de mi carrera. Comienzo con un saludo muy especial a Héctor Fernando Vargas Montoya, quien dictó dos importantes cursos de esta carrera y de quien aprendí sobre el control de riesgos cibernéticos en una organización y más importante para mí la forma de defenderme de los diferentes ataques cibernéticos que nos inundad día a día. A mis docentes de primer semestre Manuel, Juan Fernando y Francisco, gracias por darme la introducción a este maravilloso mundo de la ciberseguridad. A mis docentes de segundo semestre Leonel Marín y Javier Duran, tienen toda mi admiración por ese conocimiento técnico en forense y en sistemas operativos. las 2 materias de las cuales tenían un alta expectativa y efectivamente gracias a ellos se cumplió. A mis docentes de tercer semestre Milton y Fabio gracias por compartir su conocimiento en leyes y en detección de intrusos de máquina. A mis docentes de cuarto semestre Andrés, Rafael y Gonzalo mis más sinceros agradecimientos, grandes personas tanto personal como profesional. Finalmente, y para no olvidar a mi asesor de trabajo de grado Gabriel Taborda, quien con su conocimiento, personalidad y carisma me ha ayudado en todo este proceso del trabajo apoyándome a siempre seguir adelante.

A mis compañeros de grupo Martin, Cesar, Darling, Alex, Luis, Mario, Sebastián y Camilo espero sigamos compartiendo juntos todo el conocimiento que tenemos en diferentes áreas de tecnología y que estos años sin sus consejos, la universidad no hubiera sido lo mismo para mí. Finalmente, un agradecimiento muy especial al ITM, quien además de abrirme las puertas del conocimiento, me permitió crecer personal y laboralmente. Son 14 años que estuve en esta universidad, mi segundo hogar.

Un agradecimiento especial a mi amigo Alexis Pacheco, quien con su carisma y su espíritu de ayuda me colaboró a revisar este documento. Además de ser una gran persona, tiene un sentido de justicia el cual admiro y espero seguir aprendiendo de todas sus enseñanzas.

Ahora, debo mencionar a la persona más importante en mi vida y quien me ha apoyado en las buenas y en las malas, quien me ha visto trasnochar estudiando y jugando, y con quien nos ayudamos mutuamente para seguir cumpliendo nuestros sueños. Mi vida hermosa, bella, preciosa a quien le tengo muchos adjetivos y cada día más, *Joan Andrea Vanegas Alba* **♥.** Eres mi vida y mi mayor fortaleza para salir adelante y con quien deseo compartir el resto de mis días hasta la eternidad. Te debo esta vida y la otra, porque llegaste cuando menos lo esperaba y cuando más te necesitaba, y no tengo palabras para expresar cuanto me has cuidado, ayudado y depositado tu confianza en mí. Simplemente te amo y siempre lo haré.

Y por último a mi maravillosa familia. Mi hermano *Andrés Felipe Cañola García,* con quien disfruto jugando y viendo partidos de fútbol y *María Isabel Cañola García* quien me ayuda a cuidar de mi salud y yo le ayudo a cuidar la salud de su computador, con ellos rio, comparto, discuto y nos ayudamos para salir adelante. A mi padre *Ramón Abad Cañola Argaez* quien siempre está pendiente de nosotros su familia, y a quien recuerdo por su espíritu de ayuda y siempre al servicio de los demás. Y, por último, la persona más importante en mi vida y la causa de que yo este con vida y con buena salud, mi madre *Sonia María García Sánchez,* a quien le debo todo y que no tengo como pagarle el esfuerzo y sacrificio que ha hecho día a día.

A todos, desde el fondo de mi corazón, ¡gracias por todo!

Juan Fernando Cañola García

Resumen

El presente documento describe el proceso de investigación para la selección, entrenamiento y clasificación de un algoritmo de Deep Learning, para la creación de un sistema preventivo contra ataques de denegación de servicio hacia servidores web, denominado Dique de tipo IDS/IPS. Además, del diseño y construcción de un sistema de software, que integra una interfaz gráfica de usuario con el modelo de clasificación, y que permite verificar el funcionamiento del sistema contra ataques de denegación de servicio. Para prevenir los ataques al servidor web, el sistema Dique clasifica los paquetes que ingresan a la red en dos tipos: Maligno y NoMaligno, siendo maligno los paquetes que el sistema clasifica como posibles ataques de denegación de servicio. Esta clasificación se logró utilizando un algoritmo de Deep Learning con la red neuronal artificial Deep Feed Forward y para su entrenamiento se utilizó el Dataset CICDDoS2019, el cual contiene doce tipos de ataques de denegación de servicio, y que al final de su entrenamiento obtuvo un Accuracy de 0.994. Para demostrar el funcionamiento del sistema preventivo, se desarrolló un sistema ofensivo denominado Diluvio, el cual contiene siete tipos de ataques DoS y que se pueden lanzar al servidor web en forma selectiva. Dique posee una interfaz gráfica que permite cambiar entre modo de detección y modo de prevención y además muestra la información de los paquetes y su respectiva clasificación.

***Palabras clave*:** Ataque de denegación de servicio, Deep Learning, Sistema de detección de Intrusos, Sistema de prevención de Intrusos, Redes Neuronales

Abstract

This document describes the investigation process for a Deep Learning algorithm selection, training, and classification, to create a preventive system against web denial of service attacks called Dique type IDS / IPS. Besides, the Dique design and construction integrate a graphical user interface with the classification model, which verifies the operation of the system against denial of service attacks. In order to prevent denial of service attacks, the Dique system classifies input packets into two types: Non-Malignant and Malignant, malignant means packets that system classifies as possible denial of service attacks. This classification uses the MultiLayer Neural Network Deep Feed Forward and, for training, used Dataset called CICDDoS2019, which contains twelve types of denial of service attacks, and in the end, obtained an accuracy of 0.994. An offensive system called Diluvio was created to demonstrate how the preventive system works. Diluvio contains seven types of DoS attacks and can be targeted to the webserver. Dique has a graphical interface that allows for change between detection mode and prevention mode and displays the packages information and classification.

***KeyWords*:** Denial of Service Attack, Deep Learning, Intruder Detection System, Intruder Prevention System, Neural Networks

Contenido

Lista de Figuras

Lista de tablas

Lista de Símbolos y abreviaturas

Abreviaturas

Abreviatura	Término
Bot	Aféresis de robot
CPU	Central Processing Unit
DoS	Denial of Service
DDoS	Distributed Denial of Service
CIC	Canadian Institute for cybersecurity
CNN	Convolutional Neural Network
DARPA	Defense advanced Research Project Army
DBN	Deep Belief Networks
DE	Differential Evolution Optimizer
DFFL	Deep Feed Forward Layer
DL4J	Deep Learning for java
DNS	Domain Name System
FNN	Feed Forward Neural Network
FN	Falso Negativo
FP	Falso Positivo
GA	Algoritmo Genético
GET	Solicitud para obtener data
GRU	Gradient Recurrent Unit
HL	Hidden Layers
HU	Hidden Units
HMM	Hidden Markov Models
HTTP	Hypertext Transfer Protocol

Abreviatura	Término
IA	Inteligencia Artificial
IDE	Integrated Development Environment
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IRC	Internet Relay chat
IP	Internet Protocol
IPS	Intrusion Prevention System
Java	Lenguaje de programación de alto nivel
Kad	Implementación del protocolo Kademlia
KDD99	Knowledge Discovery in Databases
KNN	K-nearest Neighbors
LDAP	Lightweight Directory Access Protocol
LSTM	Long short term memory
MiB	Mebibyte 2^{20} bytes
MSSQL	Microsoft Structured Query Language
NB	Naive Bayes
ND4J	Librería de Java para cálculos de punto flotante
NetBIOS	Network basic input output system
NSLKDD	Dataset mejorado de KDD99
NTP	Network Time Protocol
P2P	Peer to Peer
Ping	Utilidad en diagnóstico de red
Post	Solicitud para enviar data

Abreviatura	Término
PUSH	Bandera del protocolo TCP que indica el comienzo de envío de data
Request	Solicitud
RF	Random Forest
RL	Regresión Logistica
RNN	Recurrent Neural Network
RST	Bandera del protocolo TCP que indica la llegada de un paquete mal formado
SAE	Stacked AutoEncoders
SNMP	Simple Network Management Protocol
S	Función sigmoide
SSDP	Simple Service Discovery Protocol
SVM	Support Vector Machines
SQL	Structured Query Language
SYN	Bandera del protocolo TCP que indica el comienzo de una conexión
TCP	Protocolo de control de transmisión
TN	True negative
TP	True positive
TPR	True positive rate
UDP	Protocolo de datagrama de usuario
UDP-lag	Ataque DoS utilizado en denegación de tipo slow
QOTD	Quote of the day. Protocolo antiguo para enviar la cita del día
VOIP	Voz sobre IP
WebDDoS	Ataque DoS que se realiza desde un servidor web

1. Introducción

En el campo de la seguridad informática, la detección de intrusos es el arte de captar o notar la presencia de usuarios no autorizados para acceder a un recurso informático. Generalmente, los expertos en esta área del conocimiento utilizan diferentes herramientas y técnicas para analizar el tráfico en la red y así detectar comportamientos extraños, con el objetivo de proteger la información y evitar consecuencias que esto pueda acarrear.

Con el crecimiento en tamaño y servicios del internet y a pesar de los esfuerzos que se han realizado para proteger los sistemas informáticos, los delitos informáticos siguen creciendo exponencialmente y los expertos han visto la necesidad de utilizar herramientas más sofisticadas que permitan detectar intrusos y pasar de una reacción preventiva a una reacción reactiva. En términos generales existen dos estrategias para la detección de intrusos, una basada en firmas (signature-based) que consiste en detectar intrusos a partir de una base de datos de firmas o indicadores creada a partir de ataques anteriores y la otra estrategia es conocida como detección de anomalías (anomaly-detection) que consiste en la detección de intrusos a partir del monitoreo, recolección y análisis de los paquetes de la red para un posterior análisis y clasificación de un comportamiento normal y un comportamiento anómalo. La detección de anomalías puede plantearse como un problema de clasificación con Machine Learning, que es una familia de algoritmos de la Inteligencia Artificial (IA) capaz de aprender sin estar explícitamente programada, gracias a un conjunto masivo de datos(Dataset) que permite entrenar un algoritmo diseñado para la detección de intrusos, posibilitando a una red de trabajo mantenerse alerta sobre las posibles amenazas, posteriormente y gracias a los nuevos datos que procesa, esta máquina se va entrenando más y más para mejorar su clasificación [1].

En la última década, Machine Learning se ha utilizado en tres campos de seguridad de la información [2]:

- En detección de intrusos, ayudando a identificar patrones de ataques conocidos y desconocidos.
- En análisis de malware, identificando el malware polimórfico y metamórfico los cuales no pueden ser detectados por los algoritmos tradicionales basados en reglas.
- En detección de phishing y spam, reduciendo la pérdida de tiempo y el peligro potencial causado por correos electrónicos no deseados.

Aunque Machine Learning parece ser una aproximación prometedora en diversos problemas de Ciberseguridad, presenta un inconveniente que es la gran cantidad masiva de paquetes que viaja por la red y que para Machine Learning representa un alto nivel de procesamiento, dificultando el análisis en tiempo real y puede afectar el rendimiento del sistema de cómputo. La Figura 1 muestra como el rendimiento de los métodos de Deep Learning es proporcional a la cantidad de datos procesados, comparado con los métodos de Machine Learning, el cual con el tiempo se estabiliza.

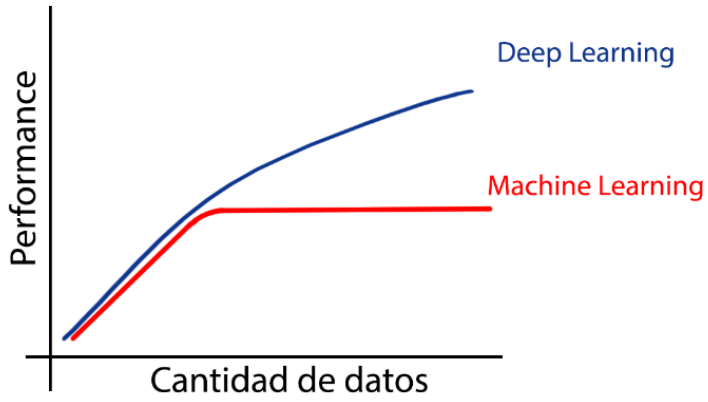


Figura 1. Comparación Machine Learning vs Deep Learning **[2].**

Deep Learning presenta muchas mejoras en la detección de intrusos con respecto a Machine Learning en [2]:

- **Tamaño**: Deep Learning funciona mucho mejor cuando se le entrega una gran cantidad de datos (millones), comparado con Machine Learning que funciona mejor con Dataset de pocos datos.
- **Tiempo**: Deep Learning requiere más tiempo para su aprendizaje, pero este es compensado en su etapa de prueba y de funcionamiento en tiempo real.
- **Dedicación**: Deep Learning escoge las características (entradas) por sí misma y es el experto en seguridad quien interpreta los resultados desde su planteamiento, comparado con Machine Learning que requiere la definición de sus características y sus etiquetas (salidas).

En los últimos años, Deep Learning se ha venido utilizando en Ciberseguridad, principalmente en la identificación de ataques de denegación de servicio, logrando una excelente clasificación de paquetes con diferentes tipos de ataques de denegación de servicio. La Tabla 1 describe los resultados de los mejores modelos entrenados mediante Deep Learning, empleados en la detección de ataques de denegación de servicio. En esta tabla las columnas F1 score, precision y Recall son medidas estadísticas utilizadas en pruebas que miden la calidad del modelo entrenado, donde 1 es el resultado más óptimo.

Tabla 1. Resultados de clasificación para ataques DoS. Tabla tomada de **[3].**

Attack Name	F1-score	Precision	Recall
DOS attempt	0.9953	0.9938	0.9969
Overflow attempt	0.9939	0.9933	0.9946
SSH Brute Force login	0.9916	0.9941	0.9892
Suspicious DNS query	0.9753	0.9953	0.9586
Cache Poisoning attempt	0.9676	0.9872	0.9506
Possible Malware infection	0.9587	0.9939	0.9337
General approach (baseline)	0.7985	0.8727	0.7360

Aunque en la literatura científica se siguen reportando datos de entrenamiento; comparando diferentes modelos con la ayuda de diversos algoritmos de Deep Learning para ataques DoS y conjuntos de datos que van surgiendo, esto solo se ha quedado en la etapa de detección, según dichos reportes, lo que conlleva a que siempre requiera de intervención humana para poder implementar controles y reglas para mitigar los ataques por denegación de servicio. Este trabajo de investigación considera ir más allá de la fase de detección de denegación de servicio utilizando Deep Learning, proponiendo un sistema preventivo; que además de analizar e identificar los ataques DoS, permita implementar controles para que autónomamente mitigue estos ataques y es otra forma de validar el modelo de Deep Learning implementado.

Objetivo General

Desarrollar un software para la prevención de ataques de denegación de servicio web utilizando Deep Learning.

Objetivos específicos

- Seleccionar los algoritmos de Deep Learning más adecuados para la identificación de ataques de denegación de servicio web.
- Construir modelos de clasificación mediante el entrenamiento de los algoritmos de Deep Learning más óptimos que serán seleccionados por medio de un cuadro comparativo que contiene las diferentes medidas de precisión utilizadas en la fase de pruebas, junto al Dataset más reciente hasta la fecha.
- Elaborar un software que integre los modelos de clasificación con una interfaz gráfica de usuario, para identificar, analizar, prevenir y mostrar gráficamente los diferentes ataques de denegación de servicio.
- Verificar el funcionamiento del sistema de prevención de ataques de denegación de servicio web, mediante un software que automatice los ataques contra el sistema informático elaborado en un ambiente controlado.

Este documento consta de los siguientes capítulos:

- El ***primer capítulo*** contiene el marco teórico donde se define los diferentes tipos de ataques de denegación de servicio, lugares en donde implementar las defensas y métodos utilizados para detectar posibles ataques DoS. También contiene el estado del arte en donde muestra un conjunto y resumen de las investigaciones realizadas por otros autores donde entrenan modelos de Deep Learning con varios Datasets, mostrando sus resultados, porcentaje de clasificación y detección.
- El ***segundo capítulo*** contiene la metodología, donde se describe el paso a paso sobre cómo se desarrolló el sistema. En esta sección se menciona el proceso de búsqueda de los algoritmos más adecuados para entrenar los modelos, las librerías de java utilizadas, los criterios de selección de los algoritmos y las herramientas con las cuales se desarrolló el sistema.
- El ***tercer capítulo*** muestra el sistema creado donde se muestra en la interfaz gráfica de usuario como el sistema analiza los paquetes de una interfaz de red y comienza a clasificarlos de acuerdo al modelo entrenado y posteriormente tomar acciones preventivas. Además, se muestra el sistema ofensivo con el cual se envían ataques de denegación de servicio para poner a prueba el sistema preventivo desarrollado.
- El ***cuarto capítulo*** describe la experiencia de la creación del sistema y las recomendaciones para crear futuros sistemas de clasificación. Además, menciona las mejoras que se le pueden hacer a este sistema y plantear los posibles trabajos futuros.

2. Marco Teórico y Estado del Arte

En esta sección se presenta los conceptos más relevantes que se emplean en este trabajo de grado y los trabajos de la literatura científica relacionados con sistemas preventivos contra ataques de denegación de servicio utilizando Deep Learning.

2.1 Marco teórico

A continuación, se describe el soporte conceptual más relevante para el desarrollo de la investigación.

2.1.1 Servidor web

Un servidor web se define como un software que responde solicitudes de páginas web utilizando el protocolo HTTP. El término servidor web puede referirse tanto a hardware como software [4]:

- Como hardware, un servidor web es un computador que almacena software de servidor web y un conjunto de archivos de sitios web como HTML, imágenes, CSS, javascript, entre otros. Un servidor web está disponible en la red y gestiona el intercambio físico de datos con otros dispositivos conectados a la red.
- Como software, un servidor web incluye toda la lógica que controla como los usuarios acceden a los archivos alojados en este servidor web. Este servidor procesa peticiones realizadas por otros dispositivos mediante direcciones web también conocidas como URLs y realiza las transferencias de la información mediante el protocolo HTTP (Hyper Text Transfer Protocol).

De forma resumida, cuando un usuario solicita un archivo mediante un navegador a un archivo alojado en un servidor web, el navegador lo realiza por medio del protocolo HTTP. Cuando la solicitud alcanza el servidor web(hardware) adecuado, el servidor web(software) acepta la solicitud, encuentra el archivo solicitado y lo envía de regreso al navegador, todo esto utilizando el protocolo HTTP. Si el servidor no encuentra el archivo solicitado retorna el error 404 not found. La figura 2 muestra el proceso de solicitud de un navegador a un servidor web.

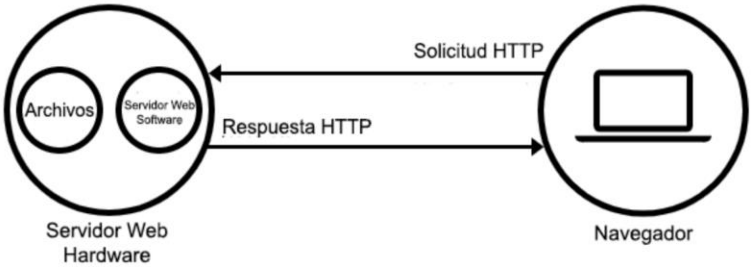


Figura 2. Proceso de solicitud a un servidor web. Adaptado de **[4]**.

2.1.2 Seguridad informática y seguridad de los datos

La seguridad Informática es la práctica de defender computadores, servidores, dispositivos móviles, sistemas electrónicos, redes y datos de ataques maliciosos. La definición aplica en una variedad de contextos y puede ser dividida en las siguientes categorías [5]:

- **Seguridad en red** consiste en asegurar una red de intrusos, ya sean atacantes o malware.
- **Seguridad de aplicación** se concentra en mantener software y dispositivos libres de amenazas. Una aplicación comprometida podría proveer acceso a los datos protegidos.
- **Seguridad de Información** consiste en proteger la integridad y privacidad de los datos, ya sea almacenados o en tránsito.
- **Seguridad operacional** incluye el proceso y decisión para manejar y proteger activos. Incluye los permisos de usuarios que acceden a una red y los procedimientos que determinan como y donde los datos pueden ser almacenados y compartidos.
- **Recuperación de desastres y continuidad de negocio** define como una organización responde a un incidente de seguridad informática o cualquier otro evento que cause perdida de operación o de datos.
  Ingeniería social está dirigida al factor más impredecible de la seguridad informática: las personas. Cualquiera puede accidentalmente introducir un virus a un sistema no seguro por no seguir las buenas practicas.

Tipos de ataques informáticos más comunes

A continuación, se mencionan algunos de los ataques informáticos utilizados en el mundo del cibercrimen [5]:

- **Malware**: Significa software malicioso. Es una de los ataques informáticos más comunes, y consiste en software creado por cibercriminales para alterar o dañar una máquina. Generalmente llega a las victimas mediante archivos adjuntos correo electrónicos o descargas que parecen legitimas.
- **SQL inyección:** Una inyección SQL (Lenguaje de consulta estructurado), es un tipo de ataque informático usado para tomar el control y robar datos de una base de datos. Los cibercriminales explotan vulnerabilidades en aplicaciones para insertar código malicioso dentro una base de datos por medio de una ejecución de instrucción. Esto le da acceso a información sensible de la base de datos.
- **Phishing:** Este ataque consiste en enviar correos electrónicos a las victimas suplantando una compañía legitima, con el objetivo de solicitar información sensible. Generalmente son utilizados para robar información de tarjetas de crédito o credenciales.
- **Hombre en el medio:** Es un tipo de ataque donde los cibercriminales interceptan la comunicación entre dos dispositivos para robar datos.

- **Denegación de servicio:** La denegación de servicio consiste en prevenir que un sistema legitimo pueda responder a las solicitudes de los usuarios, enviando una cantidad enorme de tráfico que satura la red y los servidores.

2.1.3 Seguridad Informática en servidores web

La seguridad informática en servidores web consiste en todas las medidas para proteger un servidor web frente a los diferentes tipos de amenazas informáticas. En este sentido, es un proceso continuo y parte esencial de las tareas de los administradores de servidores web [6].

La seguridad en un servidor web es importante ya que al estar desprotegido puede estar expuesto a sufrir situaciones como:

- Robo de Información.
- Explotación de datos personales (Robo de identidad, extorciones, abusos de confianza).
- Redireccionamiento a páginas web maliciosas.
- Mostrar anuncios no deseados.
- Engañar a los bots y rastreadores de los motores de búsqueda para hacer SEO (Posicionamiento en buscadores) de sombrero negro.
- Utilizar computadores de victimas para realizar otras tareas como criptomonedas.
- Recibir ataques de Denegación de Servicio para que deje de funcionar, haciéndola inaccesible para los visitantes.
- Realizar descargas de software malicioso.

A continuación, se describen algunas medidas utilizadas para la protección de servidores web [6]:

- **Certificado de seguridad:** Los certificados de seguridad permiten proteger la información que se encuentran en tránsito, es decir, que va desde el navegador de los visitantes al servidor web.
- **Firewall de aplicaciones:** Un firewall es un sistema hardware o software que permite restringir el tráfico que va de un lugar a otro. Este sistema permite filtrar y monitorear el tráfico web para identificar algunos intentos de ataques como cross site scripting, SQL inyección o denegación de servicio.
- **Escáner de seguridad:** Sirve para revisar un sitio web cada cierto tiempo con el objetivo de detectar malware o actividades sospechosas.
- **Actualizar el software**: Es indispensable actualizar el software cuando esté disponible ya que no solo permite modernizar el software, sino implementar medidas de seguridad a las vulnerabilidades encontradas en la versión anterior.
- **Contraseñas fuertes:** Entre más compleja sea una contraseña, más difícil es para los atacantes descifrarlas con ataques de fuerza bruta.

- **Limitar el acceso y los permisos:** El “principio del menor privilegio”, consiste en asignar a los usuarios solo los permisos que requieren para realizar su trabajo así no podrán realizar acciones que no les correspondan.
- **Cambiar ajustes CMS (Content Management System):** Algunos servidores web utilizan software para gestionar el contenido, pero puede ser vulnerables cuando se deja la configuración prestablecida de la cuenta.
- **Copias de seguridad:** En caso de que el servidor web haya sido comprometido, las copias de seguridad ayudaran a recuperar lo que había antes del incidente.

2.1.4 Ataques hacia servidores web

Un servidor web constituye la interfaz entre los usuarios y los aplicativos web, incluyendo las bases de datos, representando la columna vertebral al internet y las diferentes redes y aplicaciones que se conectan a través de estos. Por lo tanto, todos los servidores web al estar públicamente expuestos, son propensos a ciertos tipos de ataques. En la tabla 2, se describen los ataques más comunes hacia servidores web [7]:

Tabla 2. Ataques más comunes en un servidor web **[7].**

Ataque	Descripción
Denegación de servicio	Un servidor web es óptimo cuando puede responder a las solicitudes de los usuarios en un tiempo determinado, generalmente segundos. El ataque de denegación de servicio está diseñado para hacer lo contrario, los atacantes saturan el servidor con tantas solicitudes que las solicitudes legitimas no responden en el tiempo adecuado, y esto se debe generalmente a la cantidad excesiva de paquetes que envía el atacante.
Inyección SQL	La mayoría de sitios web contiene campos de entrada y formularios en sus aplicaciones que facilitan el proceso interactivo con los usuarios. Estos campos pueden ser utilizados para realizar consultas SQL hacia la base de datos y un atacante pueden utilizarlos para realizar sentencias maliciosas y extraer información de la base de datos.
XSS (Cross Site Scripting)	Es un ataque que consiste en ejecutar scripts maliciosos en el navegador del lado del cliente. Generalmente el navegador no sabe si el script es malicioso o no y al ser ejecutado puede secuestrar cookies de sesión, modificar un sitio web (defacement) o redireccionar el usuario a sitios maliciosos.
Autenticación y sesión rota	Los sitios web generalmente crean una cookie de sesión para cada ID de sesión y estas cookies pueden contener información sensible como usuario, password, etc. Cuando una sesión es finalizada por un deslogueo o que se cierre el navegador repentinamente, estas cookies deberían ser invalidadas. Si estas cookies no son invalidadas, la información aun existirá en el sistema y un atacante puede aprovechar esta información para reingresar al sitio donde se encontraba y extraer más información.

Referencias a objetos insegura	Esto ocurre cuando un desarrollador expone una referencia o una implementación interna como un archivo, directorio, credencial de base de datos en una URL como un parámetro. El atacante entonces puede utilizar esta información para acceder a otros objetos y crear otros ataques que accedan a información no autorizada.
Cross Site Request Forgery	Consiste en redireccionar un usuario a otro sitio suplantado, para extraer sus credenciales de login u otra información personal.
Seguridad mal configurada	La configuración de seguridad debe estar definida y desplegada para una aplicación, framework, servidor de aplicación, servidor web, servidor de aplicación, servidor de base de datos y plataforma. Si esta configuración no está bien implementada un atacante puede tener acceso no autorizado a datos sensibles o alguna funcionalidad.
Almacenamiento criptográfico inseguro	Esto ocurre cuando los datos del servidor web no están almacenados correctamente como por ejemplo nombres de usuario y password en texto plano en la base de datos sin estar encriptados.
Fallos en restricción de URLs	Las aplicaciones web verifican los permisos adecuados a las URLs antes de mostrar enlaces o botones protegidos. Si esto no está bien configurado, un atacante puede acceder a información sensible o ejecutar funciones no permitidas.
Protección insuficiente en la capa de transporte	Si el certificado de seguridad no está instalado o desactualizado, la información que transita entre cliente servidor puede ser legible y expuesta a los atacantes, quienes pueden utilizar esta información para acceder a otros recursos.
Redirecciones invalidas	Los sitios web generalmente redireccionar a otros sitios o páginas. Si no hay una validación adecuada los atacantes puede redireccionar los usuarios a sitios suplantados o que contienen malware.

2.1.5 Denegación de Servicio

La denegación de servicio(DoS) es un ataque de seguridad reconocido que intenta consumir los recursos informáticos de un host o red, haciéndolos inaccesibles para los usuarios legítimos. La denegación de servicio puede ser clasificado en dos tipos basado en la cantidad de paquetes y el número de atacantes: software exploits y flooding. El primer método, el atacante explota fallas en un software para causar a los servidores remotos, una caída en sus servicios o sustancialmente disminuir su rendimiento. El segundo método flooding, el atacante abruma los recursos del sistema CPU, memoria o la red, como resultado del envío de una gran cantidad de peticiones falsas [8]. La figura 3 muestra los métodos utilizados para realizar denegación de servicio.

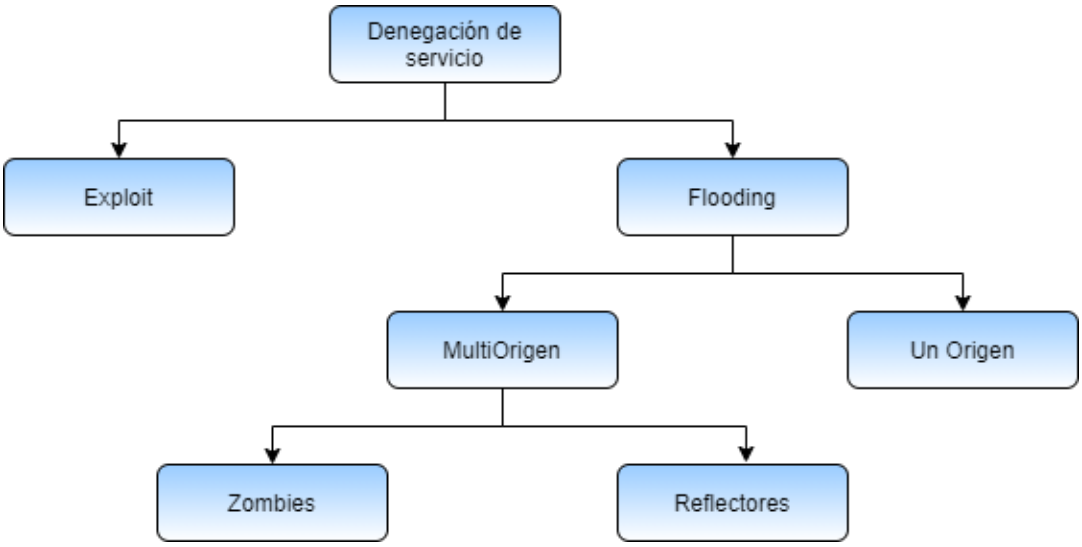


Figura 3. Clasificación de tipos de denegación de servicio. Adaptado de **[8].**

La versión más actualizada de la denegación de servicios es conocida como denegación de servicios distribuida(DDoS). Este utiliza el poder de los componentes de red para incrementar las amenazas distribuyendo los ataques desde máquinas esclavos o zombies. La figura 4 muestra los métodos utilizados para realizar una denegación de servicio distribuido.

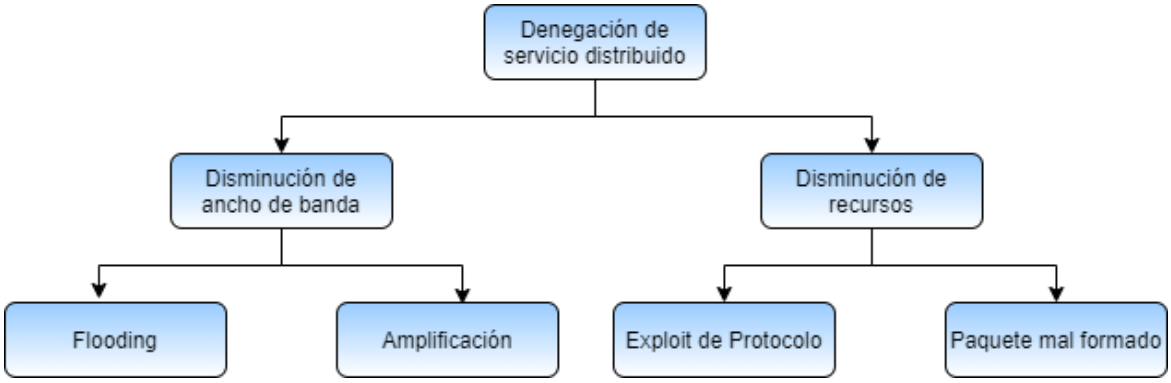


Figura 4. Clasificación de tipos de denegación de servicio distribuido. Adaptado de **[8].**

En el ataque de disminución de ancho de banda, el atacante inunda la red de la víctima con tráfico no deseado que previene al tráfico legítimo, alcanzar el sistema de la víctima. Este tipo de ataque se divide en dos: uno es el ataque de inundación, en donde las máquinas zombies envían una gran cantidad de tráfico al sistema de la víctima para congestionar su ancho de banda. El otro tipo de ataque, amplificación, las máquinas zombies envían mensajes broadcast con suplantación IP.

En el ataque de disminución de recurso, el atacante mantiene ocupado los recursos del sistema de la víctima, saturándolo para evitar solicitudes legitimas. Hay dos tipos de ataques de disminución de recurso, ataques de explotación de protocolos y ataques de

paquetes mal formados. En el primero, utilizan porciones vulnerables de un protocolo. En el segundo, el atacante utiliza zombies para enviar paquetes mal formados para colisionar el sistema de la víctima. Este último, se divide en dos: el primero consiste en enviar un paquete con la misma IP que la IP destino confundiendo el sistema operativo y logrando que colisione. El segundo consiste en asignar valores aleatorios a los campos opcionales de un paquete haciendo que el sistema de la víctima realice un procesamiento adicional, y al ser atacado por diferentes orígenes, puede ocasionar el apagado de la máquina de la víctima [8]. Además de esto, los ataques de denegación de servicio pueden variar de tipo de acuerdo al protocolo utilizado como TCP flood, ICMP flood o NTP flood. En el anexo A, se encuentra una descripción de los diferentes tipos de ataques de denegación de servicio y su clasificación.

2.1.6 Mecanismos de defensa contra DoS

Los mecanismos de defensa contra DoS se clasifican de acuerdo a dos criterios. El primer criterio de clasificación es la localización donde esta implementado el dispositivo de defensa (por ejemplo, localización de origen). El segundo criterio para la clasificación es el punto en el tiempo en el que el dispositivo de defensa DoS debería actuar en respuesta a un posible ataque DoS [10]. La figura 5 describe el perímetro de los mecanismos de defensa contra ataques de denegación de servicio:

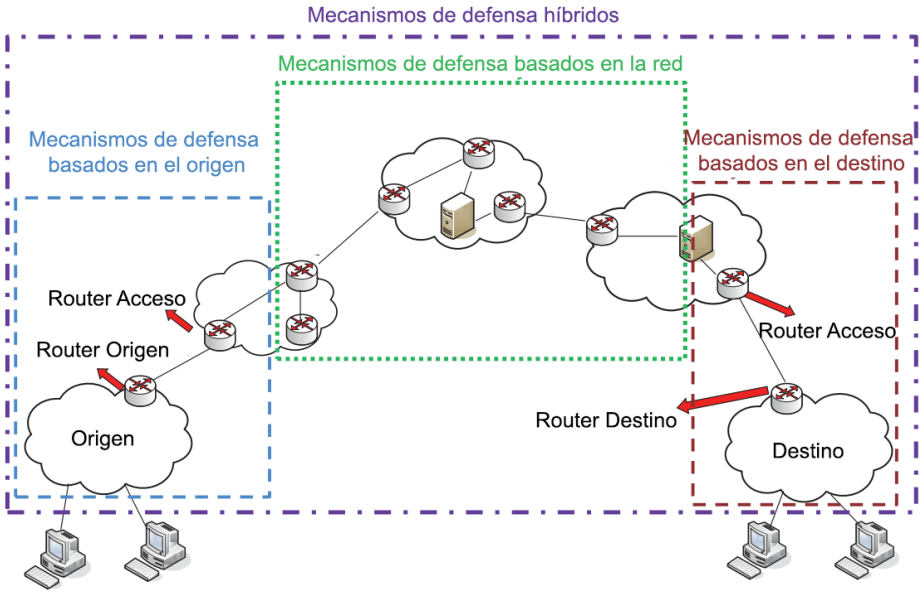


Figura 5. Mecanismos de defensa contra DoS. Adaptada de **[10].**

La tabla 3 resume las ventajas y desventajas de los lugares donde se implementa el mecanismo defensa contra DoS, los cuales se encuentran descritos en el Anexo B.

Tabla 3. Ventajas y desventajas de los mecanismos de defensa contra DDoS **[10].**

Lugar del mecanismo de defensa contra DoS	Ventajas	Desventajas
Mecanismos de defensa Hibrido (Distribuido)	1. Más robusto contra ataques DoS	1. Complejidad y gastos debido a la comunicación entre componentes distribuidos sobre todo el internet.
	2. Más recursos en varios niveles están disponibles para afrontar ataques DoS	2. Falta de inventicos por parte del proveedor de servicios para cooperar
		3. Comunicación de confianza necesaria entre varios componentes distribuidos para colaborar.
Mecanismo de defensa en la victima (Centralizado)	1. Fácil por el alto consumo de los recursos	1. El ancho de banda de la red algunas veces se satura.
	2. La mayoría son tipos de esquema de defensa prácticamente aplicables como servidores web que proporcionan servicios críticos intentan asegurar sus recursos para usuarios legítimos.	2. Esta estrategia de detección del ataque solo alcanza la víctima y detecta un ataque cuando los clientes legítimos ya han sido denegados.
Mecanismo de defensa en el origen (Centralizado)	1. Mejor defensa posible	1. Nada fácil.
	2. Previene la posibilidad de inundar no solamente en el lado de la víctima sino en toda la red intermedia.	2. Los orígenes son ampliamente distribuidos y se comportan como el tráfico normal.
		3. Es muy difícil diferencia entre el tráfico de ataque y el legítimo cerca de los orígenes, .
		4. Difícil sistema de despliegue al final de la fuente.
		5. La motivación del despliegue de los mecanismos basados en el origen es baja desde que no sea claro quien pagaría el costo asociado con estos servicios.
Mecanismo de defensa en la red intermedia (Centralizado)	1. Detección y trazabilidad de la fuente del ataque son fáciles en esta estrategia	1. Despliegue para lograr una detección completa todos los routers tendrían que utilizar este mismo esquema. Implementación de este esquema es complicada.
	2. El objetivo es detectar y responder el ataque en la red intermedia tan cerca a la fuente como sea posible.	2. Alto procesamiento y almacenamiento en los routers.

2.1.7 Machine Learning y Deep Learning

Machine Learning consiste en una forma de enseñarle a una computadora como detectar patrones y hacer conexiones mostrándole un gran volumen de datos. Así, en lugar de programar software para que cumpla una tarea específica, la máquina utiliza una gran cantidad de datos y algoritmos sofisticados para saber cómo realizar la tarea por sí misma. Machine Learning no es algo nuevo, es un concepto que lleva años perfeccionándose y que ha cogido fuerza recientemente, ya que hoy en día se cuenta con la tecnología necesaria para implementarlo. Los software tradicionales son programados para resolver problemas, mediante sus diferentes estructuras lógicas, y con reglas definidas por los ingenieros. Machine Learning es diferente, las reglas las define el sistema después de aprender con cierta cantidad de datos y se perfecciona en la medida que recibe más de estos. La Figura 6 muestra las diferencias entre un programa tradicional y Machine Learning.

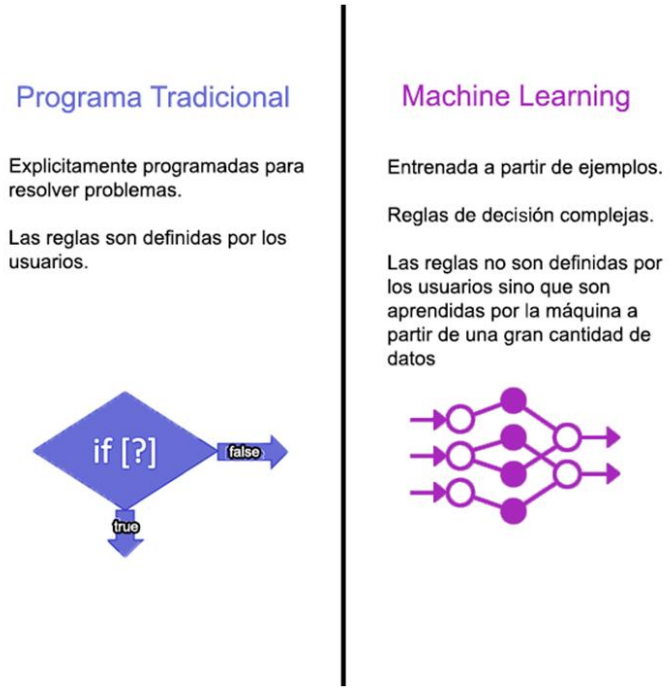


Figura 6. Programa tradicional vs Machine Learning**.**

- **Deep Learning**

El Deep Learning es una técnica mejorada de Machine Learning para la extracción de características, percepción y aprendizaje de máquinas. Los algoritmos de aprendizaje profundo realizan sus operaciones utilizando múltiples capas consecutivas. Las capas están interconectadas y cada capa recibe la salida de la capa anterior como entrada. Eso es una gran ventaja al utilizar algoritmos eficientes para extraer características jerárquicas que mejor representan datos en lugar de extraer features manuales en los métodos de Deep Learning. Hay muchas áreas de aplicación para Deep Learning como: procesamiento de imágenes, procesamiento de lenguaje natural, biomédico,

automatización de la gestión de la relación con el cliente, vehículo autónomo sistemas, entre otros.

- **Machine Learning vs Deep Learning**

Los expertos en Machine Learning y Deep Learning aún no llegan a un consenso sobre estos conceptos. Machine Learning es un concepto más antiguo que Deep Learning y el Deep Learning puede también ser una técnica que realiza el Machine Learning.

Las diferencias se enumeran a continuación [3]:

- En el Deep Learning, se necesitan demasiados datos para llevar estructura del algoritmo al ideal. En Machine Learning, el problema se puede resolver con muchos menos datos porque la persona le da las características específicas al algoritmo.
- Los algoritmos de Deep Learning intentan extraer características de datos. En el Machine Learning, las características están determinadas por el experto.
- Mientras que los algoritmos de Deep Learning funcionan en máquinas de alto rendimiento, los algoritmos de Machine Learning pueden trabajar en CPU ordinarias.
- En el Machine Learning, el problema generalmente se divide en piezas, estas partes se resuelven una por una y luego las soluciones se forman como resultado de las soluciones.
- Lleva mucho tiempo entrenar algoritmos de Deep Learning.

2.1.8 Clasificación de algoritmos de Machine Learning para la Ciberseguridad

Machine Learning se ha venido utilizando en diferentes campos como visión artificial, análisis médicos, videojuegos y marketing en las redes sociales donde ha mostrado su superioridad frente a los algoritmos tradicionales basados en reglas. Machine Learning también se está integrando en los sistemas de detección cibernética con el objetivo de apoyar o incluso reemplazar a los analistas de seguridad [3].

Los algoritmos de Machine Learning pueden ser clasificados en 2 tipos: los tradicionales **Shallow Learning**, y los más recientes **Deep Learning** como se muestra en la figura 7. Shallow Learning o aprendizaje superficial, requiere un experto en extracción de características a partir de una gran cantidad de datos relevantes. Por el contrario, Deep Learning o aprendizaje profundo consiste en un modelo multicapa que puede realizar la extracción de características automáticamente. Shallow Learning y Deep Learning pueden ser clasificados en 2 tipos: supervisados y no supervisados. La diferencia de estos dos es que el supervisado se conocen sus salidas (labels), mientras que en el no supervisado estas no son requeridas [3].

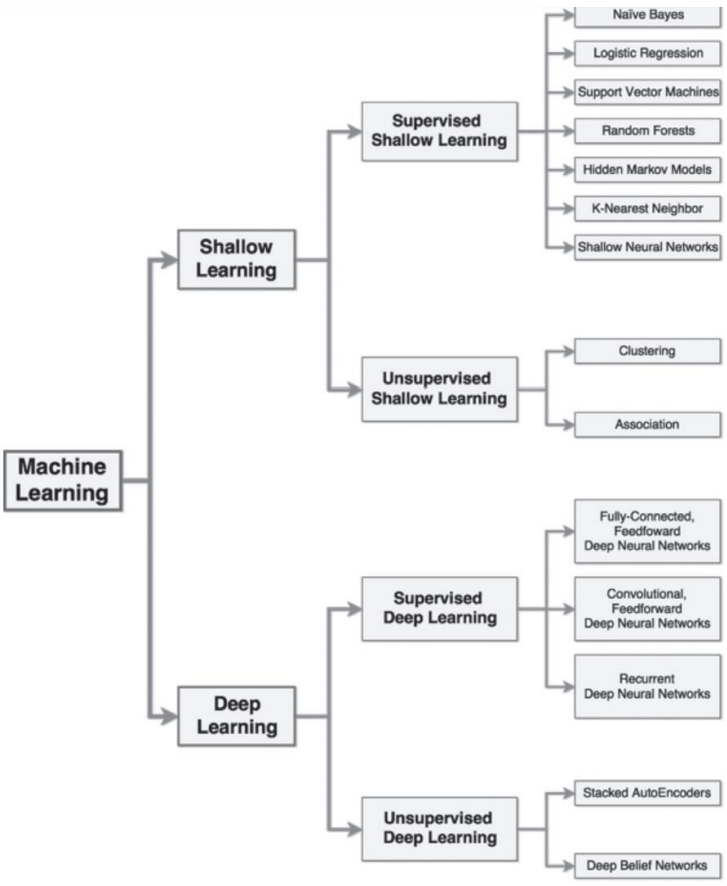


Figura 7. Clasificación de métodos de Machine Learning para la Ciberseguridad **[3].**

- **Algoritmos Shallow Learning**
  - **Algoritmos Shallow Learning Supervisados**
    - **Naïve Bayes (NB):** Son clasificadores probabilísticos que hacen el supuesto a priori de que las características del conjunto de datos de entrada son independientes de cada uno. Estos algoritmos son escalables y no requieren una gran cantidad de datos para su entrenamiento.
    - **Regresión logística (RL):** Clasificadores categóricos que adoptan un modelo discriminativo.
    - **Máquinas de vectores de soporte (SVM).** Estos son clasificadores no probabilísticos que muestra los datos en un espacio de función con el objetivo de maximizar la distancia entre cada categoría de muestras. No hacen ninguna suposición en las características de entrada, pero tienen un bajo rendimiento en las clasificaciones de varias clases. Por lo tanto, deben ser utilizados como clasificadores binarios. Su escalabilidad limitada, podría llevar a largos tiempos de procesamiento.
    - **Bosque aleatorio (RF).** Es un conjunto de árboles de decisión, y tiene en cuenta la salida de cada árbol antes de proporcionar una respuesta final unificada. Cada árbol de decisión es un clasificador condicional: el árbol se

visita desde la parte superior y, en cada nodo, una condición se verifica contra una o más características de los datos analizados. Estos métodos son eficientes para grandes conjuntos de datos y en problemas multiclase.

- **Modelos ocultos de Markov (HMM).** Estos modelan el sistema como un conjunto de estados, produciendo salidas con diferentes probabilidades; el objetivo es determinar la secuencia de estados que produjeron los resultados observados. HMM son eficaces para entender el comportamiento temporal de las observaciones, y para calcular la probabilidad de una secuencia dada de eventos. Aunque HMM pueden ser entrenados en conjuntos de datos etiquetados o no.
- **K-Vecino más cercano (KNN).** KNN se utiliza para la clasificación y pueden ser utilizados para problemas de varias clases. Sin embargo, tanto su fase de entrenamiento como la de prueba son computacionalmente exigentes en cuanto a clasificar cada muestra de prueba, ya que se compara con todas las muestras de entrenamiento.
- **Red neuronal superficial (SNN).** Estos algoritmos se basan en redes neuronales, que consisten en un conjunto de elementos de procesamiento (es decir, neuronas) organizadas en dos o más capas. SNN incluye todos esos tipos de redes neuronales con un número limitado de neuronas y capas. A pesar de la existencia de SNN sin supervisión, en seguridad informática, en su mayoría han sido utilizado para tareas de clasificación.

- **Algoritmos Shallow Learning No Supervisados**
  - **Clustering**: Algoritmo para fines de detección de anomalías, que consiste en reunir datos con características similares.
  - **Association**: Su objetivo es identificar patrones desconocidos entre los datos, para fines de predicción. Sin embargo, tienden a producir una salida excesiva de reglas no necesariamente válidas, por lo tanto, deben combinarse con inspecciones precisas por un experto humano.

- **Algoritmos Deep Learning Supervisados**
  - **Fully-connected Feedforward Deep Neural Network (FNN)** Son una variante de DNN donde cada neurona está conectada a todas las neuronas en la capa anterior. FNN no hace ninguna suposición sobre los datos de entrada y proporciona un propósito flexible y general.
  - **Convolutional Feedforward Deep Neural Networks (CNN**) Son una variante de DNN donde cada neurona recibe su entrada solo de un subconjunto de Neuronas de la capa anterior. Esta característica hace que la CNN sea efectiva en analizar datos espaciales, pero su rendimiento disminuye cuando se aplican a datos no espaciales. CNN tiene un costo de computación más bajo que FNN
  - **Recurrent Deep Neural Networks (RNN):** Una variante de DNN cuya característica es que las neuronas pueden enviar su salida también a las capas anteriores; este diseño los hace más difícil de entrenar que FNN. Sobresalen

como generadores de secuencias, especialmente su variante más reciente, long short term memory.

- **Algoritmos Deep Learning No Supervisados**
  - **Deep belief networks (DBN):** Están modelados a través de una composición de Máquinas de Boltzmann restringidas (RBM), una clase de redes neuronales sin capa de salida. DBN puede ser utilizado con éxito para tareas de pre-entrenamiento porque sobresalen en la función de extracción de características. Requieren una fase de formación, pero con conjuntos de datos sin labels.
  - **Stacked AutoEncoders (SAE):** Están compuestas por múltiples AutoEncoders, una clase de redes neuronales donde el número de entradas y las neuronas de salida son las mismas. SAE sobresale en tareas de pre-entrenamiento similar a DBN, y lograr mejores resultados en pequeños conjuntos de datos.

La tabla 4 describe el uso más óptimo de los algoritmos de Deep Learning y Shallow Learning en Ciberseguridad.

Tabla 4. Uso de técnicas de Machine Learning en ciberseguridad.Tabla tomada de **[3].**

		Intrusion Detection			Malware Analysis	Spam Detection
		Network	Botnet	DGA		
Deep Learning	Supervised	RNN	RNN		FNN CNN RNN	
	Unsupervised	DBN SAE			DBN SAE	DBN SAE
Shallow Learning	Supervised	RF NB SVM LR HMM KNN SNN	RF NB SVM LR KNN SNN	RF HMM	RF NB SVM LR HMM KNN SNN	RF NB SVM LR KNN SNN
	Unsupervised	Clustering Association	Clustering	Clustering	Clustering Association	Clustering Association

2.1.9 Datasets

La fase más desafiante para determinar el rendimiento de los sistemas de detección de intrusos es encontrar los datos apropiados. Se puede obtener la información que se utilizará para los datos observando y recolectando datos de la red. Colectar información de la red es costoso, por lo tanto, los investigadores pueden utilizar Dataset disponibles en internet. Los Dataset más utilizados son [11]:

- **KDD CUP 99:** Se creó por la razón de contar con un conjunto de datos apropiado para prueba de sistemas de detección de intrusos. Fue diseñado como datos de simulación en 1998. KDD Cup99 se utiliza especialmente en los campos de minería de datos y aprendizaje automático. Principalmente en el conjunto de datos estándar de KDD Cup99, hay alrededor de 5 millones datos. Alrededor del 80% de ellos son datos de ataque. KDD Cup 99 incluye datos de entrenamiento y prueba. Hay 41 features que pueden ser categorizado como features básicas, features de tráfico y features de contenido. Los datos en este conjunto de datos pueden ser clasificado en 5 categorías principales, 4 de ellas son de ataque, 1 es Normal.
    - Normal: datos de tipo sin ataque.
    - Ataque: DoS (Denegación de servicio), Escaneo (ataques escaneo), R2L (raíz a local) y U2R (usuario a raíz).

  KDD Cup99 contiene datos numéricos (en binario y formato de número real) e información de texto sobre categorías. Además, estos datos también contienen una característica adicional al final para mostrar el label de los datos si es un ataque o no.
- **NSL KDD:** Para que los algoritmos de aprendizaje automático funcionen mejor en KDDCup99, el tamaño de datos fue reducido al eliminar registros duplicados, y se creó el conjunto de datos NSL KDD. Contiene registros esenciales del conjunto completo de datos KDD Cup99.

  El NSL-KDD tiene las siguientes diferencias sobre el KDD Cup99 original:
    - El clasificador no da resultados sesgados porque no hay datos redundantes en el conjunto de entrenamiento.
    - La relación de reducción es menor porque no hay datos repetidos en el conjunto de prueba.
    - El número de registros de registros seleccionados de cada grupo de nivel difícil es proporcional al porcentaje de registros en el conjunto de datos KDD.

  En cada registro hay 41 atributos que se despliegan diferentes características del flujo y una etiqueta asignada a cada una como tipo de ataque o como normal. Las 4 categorías de ataque son agrupadas en DoS, Probe, R2L and U2R
- **CIC IDS 2017:** Este Dataset fue creado por el Canadian Institute for Cybersecurity (CIC). Esta Dataset contiene ataques similares a ataques del mundo real. También incluye los resultados del análisis de tráfico de red utilizando la herramienta CICFlowMeter, IPs de origen y destino, puertos, protocolos y ataque. Además, el conjunto de datos está presumiblemente disponible para cualquiera. CIC ha identificado once criterios necesarios para construir un conjunto de datos de referencia confiable. Estos criterios son: Completa configuración de red, tráfico completo, conjunto de datos etiquetados, Interacción completa, captura completa, protocolos disponibles, ataque a la diversidad, heterogeneidad, conjunto de características y metadatos. El conjunto de datos CICIDS2017 consta de flujos de red etiquetados (incluidas las cargas completas de paquetes en formato .pcap), el correspondiente perfil y los flujos etiquetados y archivos CSV para el Machine y Deep Learning.

- **CSE-CIC-IDS2018:** Dataset que también fue creado por el Canadian Institute for Cybersecurity (CIC). Y Communications Security Establishment (CSE). Este incluye información detallada de ataques con modelos de distribución abstractos. El Dataset incluye 7 diferentes tipos de ataques como Fuerza Bruta DoS, WEB Attack, Infiltración, Botnet Attack, DDoS, Hearthlech, Ataque de fuerza bruta. Utilizaron FTP Patator y SSH Herramientas de Patator para recolectar este tipo de ataques.
    - Ataque de DoS: Utilizaron Hulk, GoldenEye, Slowloris y probaron lentamente las herramientas para recopilar este tipo de ataques.
    - Ataque web: Utilizaron la aplicación web Damn Vulnerable(DVWA) y marco de selenio interno (XSS y Herramientas de fuerza bruta) para recolectar este tipo de ataques.
    - Ataque de infiltración: Utilizaron herramientas de Nmap y portscan para almacenar este tipo de ataques.
    - Ataque de Botnet: Usaron capturas de pantalla y keylogging
    - Ataque DDoS: Utilizaron Canon de iones de baja órbita (LOIC)para solicitudes UDP, TCP o HTTP.
    - Hearthlech: Es un ataque DoS.

  El equipo de CIC registro la data en crudo cada día incluyendo el tráfico de red y logs. En el proceso de extracción de features de los datos en crudo usaron CICIFlowMeter-V3 y extrajeron más de 80 features de trafico de red. Finalmente, ellos lo almacenaron en un archivo CSV por máquina.
- **MCFP Bot Traffic Merged with Benign:** Las trazas completas de tráfico de malware de Bot son proporcionadas por el Proyecto de Instalación de Captura de Malware (MCFP). Se desarrolló en ambiente laboratorio. Este conjunto de datos contiene datos sobre infección de bot real para simular un incidente, incluyendo ejemplos de malware de bot individuales que se ejecutan y se proporcionan trazas de tráfico sin tráfico benigno. Las etiquetas se pueden aplicar de manera eficiente a cada evento antes de fusionar, sin embargo, la ausencia de actividad benigna y la ausencia de advertencias falsas como resultado es un desafío al poder dar una buena representación de los datos.
- **DDoS Evaluation Dataset CICDDoS2019** [12] **:** El instituto canadiense para la ciberseguridad creó un Dataset llamado CICDDoS2019**.** Este Dataset contiene 11 tipos de ataques DoS los cuales fueron creados en un ambiente controlado. La imagen muestra los 11 tipos de ataques DoS. El Dataset contiene un total de 80 features, los cuales fueron extraídos utilizando la herramienta CICFlowMeter la cual extrae características de los archivos .pcap. Además, los expertos del instituto canadiense para la ciberseguridad, realizaron un proceso para conocer cuáles eran los features más relevantes de cada ataque DoS, dando como resultado 21 features [12]. La tabla 5 muestra los tipos de ataques y el tiempo que se realizó el ataque.

Tabla 5. Lista de ataques DDoS por hora. Tabla tomada de **[12].**

Dias	Ataques	Hora de ataque
Testing Set	PortMap	09:43 - 09:51
	NetBIOS	10:00 - 10:09
	LDAP	10:21 - 10:30
	MSSQL	10:33 - 10:42
	UDP	10:53 - 11:03
	UDP-Lag	11:14 - 11:24
	SYN	11:28 - 17:35
Training Set	NTP	10:35 - 10:45
	DNS	10:52 - 11:05
	LDAP	11:22 - 11:32
	MSSQL	11:36 - 11:45
	NetBIOS	11:50 - 12:00
	SNMP	12:12 - 12:23
	SSDP	12:27 - 12:37
	UDP	12:45 - 13:09
	UDP-Lag	13:11 - 13:15
	WebDDoS (ARME)	13:18 - 13:29
	SYN	13:29 - 13:34
	TFTP	13:35 - 17:15

Las solicitudes se envían a varios servicios que funcionan a través de UDP, porque a diferencia del Protocolo de Control de Transmisión (TCP), este protocolo de transporte no valida las direcciones de origen. Los servicios que se han utilizado para la reflexión DDoS hasta ahora incluyen el Sistema de nombres de dominio (DNS), el Protocolo de tiempo de red (NTP), el Protocolo simple de administración de redes (SNMP), el Protocolo simple de descubrimiento de servicios (SSDP) y el Protocolo generador de caracteres (CARGEN). CLDAP es solo la última incorporación a la lista. La figura 8 muestra la clasificación de ataques del Dataset.

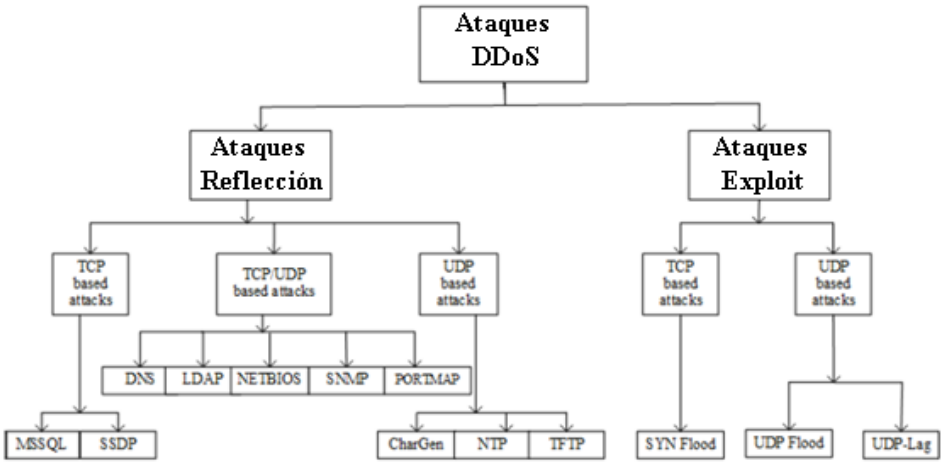


Figura 8. Tipos de ataques DDoS en el Dataset CICDoS2019. Figura tomada de **[12].**

A continuación, se describe cada uno de los ataques que contiene este Dataset:

- **NTP** Un ataque de amplificación NTP es un ataque de denegación de servicio distribuido volumétrico basado en reflexión (DDoS) en el que un atacante explota la funcionalidad de un servidor de Protocolo de tiempo de red (NTP) para abrumar una red o servidor objetivo con una cantidad amplificada de tráfico UDP, haciendo que el objetivo y su infraestructura circundante sean inaccesibles para el tráfico regular [13].
- **DNS** Los servidores del Sistema de nombres de dominio (DNS) son las "guías telefónicas" de Internet; son la ruta a través de la cual los dispositivos de Internet pueden buscar servidores web específicos para acceder al contenido de Internet. Una inundación de DNS es un tipo de ataque distribuido de denegación de servicio (DDoS) en el que un atacante inunda los servidores DNS de un dominio en particular en un intento de interrumpir la resolución de DNS para ese dominio. Si un usuario no puede encontrar la agenda telefónica, no puede buscar la dirección para llamar a un recurso en particular. Al interrumpir la resolución de DNS, un ataque de inundación de DNS comprometerá la capacidad de un sitio web, API o aplicación web para responder al tráfico legítimo. Los ataques de inundación de DNS pueden ser difíciles de distinguir del tráfico pesado normal porque el gran volumen de tráfico a menudo proviene de una multitud de ubicaciones únicas, que buscan registros reales en el dominio, imitando el tráfico legítimo [14].
- **LDAP** El protocolo LDAP es muy utilizado actualmente por empresa que apuestan por el software libre al utilizar distribuciones de Linux para ejercer las funciones propias de un directorio activo en el que se gestionarán las credenciales y permisos de los trabajadores y estaciones de trabajo en redes LAN corporativas en conexiones cliente/servidor. LDAP son las siglas de Protocolo Ligero de Acceso a Directorio, o en inglés Lightweight Directory Access Protocol. Se trata de un conjunto de protocolos de licencia abierta que son utilizados para acceder a la información que está almacenada de forma centralizada en una red. Este protocolo se utiliza a nivel de aplicación para acceder a los servicios de directorio remoto. LDAP Funciona tanto por TCP como UDP [15].
- **MSSQL** Este nuevo método de ataque se manifiesta en forma de respuestas de Microsoft SQL Server a una consulta o solicitud del cliente a través de abuso del Protocolo de resolución de Microsoft SQL Server (MC-SQLR), que escucha en UDP puerto 1434 [16].
- **NetBIOS** NetBIOS sobre TCP/IP (NetBT) es un servicio de red del nivel de sesión que se encarga de asociar los nombres a direcciones IP. En el programa SunLink Server, NetBT se implementa a través de WINS y de la resolución de nombres broadcast. Los dos aspectos más importantes de las actividades relacionadas con la asignación de nombres son el registro y la resolución:
    - El registro es el proceso por el que se especifica un nombre exclusivo para un equipo (nodo) en la red. Generalmente cada equipo se registra a sí mismo al inicio.
    - La resolución es el proceso por el que se determina la dirección que corresponde a un nombre de equipo**.**

El propósito principal de NetBIOS es permitir que las aplicaciones en computadoras separadas se comuniquen y establezcan sesiones para acceder a recursos compartidos y para encontrarse en una red de área local.
Este ataque genera 2.56 a 3.85 veces más tráfico de respuesta enviado al objetivo que las consultas iniciales enviadas por el atacante. Aunque las consultas legítimas y maliciosas del servidor de nombres NetBIOS son una ocurrencia común, una inundación de respuesta se detectó por primera vez en marzo de 2015 durante un ataque DDoS [17].

- **SNMP** Una reflexión SNMP es un tipo de ataque de denegación de servicio distribuido (DDoS) que recuerda a las generaciones anteriores de ataques de amplificación de DNS. En lugar de los Servidores de nombres de dominio (DNS), los ataques de reflexión SNMP utilizan el Protocolo simple de administración de redes (SNMP), un protocolo común de administración de redes utilizado para configurar y recopilar información de dispositivos de red como servidores, concentradores, conmutadores, enrutadores e impresoras. Los ataques de reflexión SNMP pueden generar volúmenes de ataque de cientos de gigabits por segundo, que pueden dirigirse a objetivos de ataque desde múltiples redes de banda ancha. Los ataques a veces duran horas, son muy perjudiciales para los objetivos de ataque y pueden ser muy difíciles de mitigar [18].
- **SSDP** Un ataque del Protocolo simple de descubrimiento de servicios (SSDP) es un ataque de denegación de servicio distribuido basado en reflección que explota los protocolos de red Universal Plug and Play (UPnP) para enviar una cantidad amplificada de tráfico a una víctima objetivo, abrumando la infraestructura del objetivo y desconectando su recurso web [19].
- **UDP** Una inundación UDP es un tipo de ataque de denegación de servicio en el que se envía una gran cantidad de paquetes de Protocolo de datagramas de usuario (UDP) a un servidor de destino con el objetivo de abrumar la capacidad de ese dispositivo para procesar y responder. El cortafuegos que protege el servidor de destino también puede agotarse como resultado de una inundación UDP, lo que resulta en una denegación de servicio al tráfico legítimo [20].
- **UDP-Lag** El ataque UDP-Lag es ese tipo de ataque que interrumpe la conexión entre el cliente y el servidor. Este ataque se usa principalmente en juegos en línea donde los jugadores quieren ralentizar / interrumpir el movimiento de otros jugadores para superarlos. Este ataque puede llevarse a cabo de dos maneras, es decir, usando un interruptor de hardware conocido como interruptor de retraso o por un programa de software que se ejecuta en la red y acapara el ancho de banda de otros usuarios [20].
- **WebDDoS** Ataque DoS realizado desde un servidor web por el protocolo HTTP, como la aplicación web Damn Vulnerable(DVWA).
- **SYN** Una inundación SYN (ataque medio abierto) es un tipo de ataque de denegación de servicio que tiene como objetivo hacer que un servidor no esté disponible para el tráfico legítimo al consumir todos los recursos del servidor disponibles. Al enviar repetidamente paquetes de solicitud de conexión inicial (SYN), el atacante puede abrumar todos los puertos disponibles en una máquina

de servidor de destino, lo que hace que el dispositivo de destino responda lentamente al tráfico legítimo o no [15].

- **TFTP** (Protocolo de transferencia de archivos trivial). Es un protocolo de transferencia muy simple semejante a una versión básica de FTP. Se utiliza para transferir pequeños archivos entre computadoras en una red, como una terminal Windows. Está destinado a ser utilizado para transferencias de archivos de firmware y archivos de configuración, generalmente para dispositivos en red. Sin embargo, el diseño simple del protocolo omite características tales como las capacidades de autenticación y listado de directorios. El ataque involucra a los servidores TFTP conectados a Internet. El ataque realiza una solicitud predeterminada para un archivo, y el servidor TFTP de la víctima devuelve datos al host de destino solicitante como resultado de esta solicitud, independientemente de que el nombre del archivo no coincida. TFTP solo envía datos en tamaños de bloque específicos y requiere el reconocimiento de cada bloque que se recibe [21].

2.1.10 Herramientas para entrenamiento de Machine Learning

A continuación se describen las herramientas más utilizados para el entrenamiento de modelos con Machine Learning [22]:

- **Knime** es una herramienta de aprendizaje automático de código abierto que se basa en la GUI. Lo mejor de Knime es que no requiere ningún conocimiento de programación. Todavía se pueden aprovechar las instalaciones proporcionadas por Knime. Generalmente se usa para fines relacionados a datos. Por ejemplo, manipulación de datos, minería de datos, etc. Además, procesa datos creando diferentes flujos de trabajo y luego los ejecuta. Viene con repositorios que están llenos de diferentes nodos. Estos nodos se llevan al portal Knime. Y finalmente, se crea y ejecuta un flujo de trabajo de nodos.
- **Accord.net** es un framework computacional de aprendizaje automático. Viene con una imagen y paquetes de audio. Dichos paquetes ayudan a entrenar los modelos y a crear aplicaciones interactivas. Por ejemplo, audición, visión por computadora, etc. Como .net está presente en el nombre de la herramienta, la biblioteca base de este marco es el lenguaje C #. Las bibliotecas Accord son muy útiles para probar y manipular archivos de audio.
- **Scikit-Learn** es un paquete de aprendizaje automático de código abierto. Es una plataforma unificada, ya que se utiliza para múltiples propósitos. Ayuda en la regresión, agrupamiento, clasificación, reducción de dimensionalidad y preprocesamiento. Scikit-Learn está construido sobre las tres bibliotecas principales de Python, a saber: NumPy, Matplotlib y SciPy. Junto con esto, también lo ayudará con las pruebas y el entrenamiento de sus modelos.
- **TensorFlow** es un marco de código abierto que es útil para Machine Learning a gran escala. Es una mezcla de aprendizaje automático y modelos de redes neuronales. Además, también es un buen amigo de Python. La característica más destacada de TensorFlow es que también se ejecuta en CPU y GPU. El

procesamiento del lenguaje natural, la clasificación de imágenes son los que implementan esta herramienta.

- **Weka** También es un software de código abierto. Se puede acceder a través de una interfaz gráfica de usuario. El software es muy fácil de usar. La aplicación de esta herramienta es en investigación y docencia. Junto con esto, Weka también le permite acceder a otras herramientas de aprendizaje automático. Por ejemplo, R, Scikit-learn, etc.
- **Pytorch** Pytorch es un framework de Deep Learning. Es muy rápido y flexible de usar. Esto se debe a que Pytorch tiene un buen comando sobre la GPU. Es una de las herramientas más importantes del aprendizaje automático porque se utiliza en los aspectos más vitales de Machine Learning que incluye la construcción de redes neuronales profundas y cálculos de tensor.
- **RapidMiner** es una buena noticia para los no programadores. Es una plataforma de ciencia de datos y tiene una interfaz muy sorprendente. RapidMiner es independiente de la plataforma, ya que funciona en sistemas operativos multiplataforma.
- **Google Cloud AutoML** El objetivo de Google Cloud AutoML es hacer que la inteligencia artificial sea accesible para todos. Lo que hace Google Cloud AutoML es proporcionar los modelos previamente entrenados para los usuarios a fin de crear diversos servicios. Por ejemplo, reconocimiento de texto, reconocimiento de voz, etc. Google Cloud AutoML se hizo muy popular entre las empresas. Como las empresas quieren aplicar inteligencia artificial en todos los sectores de la industria, se han enfrentado a dificultades para hacerlo porque hay una falta de personas que conozcan de Machine Learning.
- **Jupyter Notebook** El Jupyter Notebook es una de las herramientas de aprendizaje automático más utilizadas entre todas. Es un procesamiento muy rápido, así como una plataforma eficiente. Además, admite tres idiomas: Julia, R, Python. Así, el nombre de Jupyter está formado por la combinación de estos tres lenguajes de programación. Jupyter Notebook permite al usuario almacenar y compartir el código en vivo en forma de cuadernos. También se puede acceder a través de una GUI. Por ejemplo, winpython navigator, anaconda navigator, etc.
- **Apache Mahout** es parte de Apache, que es una plataforma de código abierto basada en Hadoop. Generalmente se usa para el aprendizaje automático y la minería de datos. Técnicas como la regresión, la clasificación y la agrupación se hicieron posibles con Mahout. Junto con esto, también hace uso de funciones basadas en matemáticas como vectores, etc.
- **Estudio Azure Machine Learning** El estudio Azure Machine Learning es lanzado por Microsoft. Al igual que Google Cloud AutoML, este es el producto de Microsoft que proporciona servicios de aprendizaje automático a los usuarios. Azure Machine Learning Studio es una forma muy fácil de formar conexiones de módulos y conjuntos de datos. Junto con esto, Azure también tiene como objetivo proporcionar instalaciones de inteligencia artificial al usuario. Al igual que TensorFlow, también funciona en CPU y GPU.

- **MLLIB** Al igual que Mahout, MLLIB también es un producto de Apache Spark. Se utiliza para regresión, extracción de características, clasificación, filtrado, etc. También a menudo se llama Spark MLLIB. MLLIB viene con muy buena velocidad y eficiencia.
- **Orange3** es un software de minería de datos que es la última versión del software de Orange. Orange3 ayuda en el preprocesamiento, visualización de datos y otras cosas relacionadas con los datos. Se puede acceder a Orange3 a través del navegador Anaconda. Realmente es muy útil en la programación de Python. Junto con esto, también puede ser una excelente interfaz de usuario.
- **IBM Watson** es una interfaz web proporcionada por IBM para usar Watson. Watson es un sistema de preguntas y respuestas de interacción humana que se basa en el procesamiento del lenguaje natural. Watson se aplica en varios campos, como el aprendizaje automatizado, la extracción de información, etc. IBM Watson se utiliza generalmente para fines de investigación y prueba. Su objetivo es ofrecer una experiencia humana a los usuarios.
- **Pylearn2** es una biblioteca de aprendizaje automático construida sobre Theano. Por lo tanto, hay muchas funciones que son similares entre ellas. Junto con esto, puede realizar cálculos matemáticos. Pylearn2 también es capaz de ejecutarse en la CPU y la GPU. Antes de llegar a Pylearn2, debe estar familiarizado con Theano.
- **Deeep Learning 4J:** DeepLearning4j es una biblioteca de programación de Deep Learning escrita en Java, con un amplio soporte para algoritmos de Deep Learning. DeepLearning4j incluye implementaciones de Restricted Boltzmann Machine, Deep Belief Network, Deep AutoEncoders, Stacked Denoising AutoEncoders y Recursive Neural Tensor Network. Todos estos algoritmos incluyen versiones paralelas distribuidas que se integran con Apache Hadoop y Spark. Además, es un software de código abierto lanzado bajo la licencia Apache 2.0 desarrollado principalmente por un grupo de Deep Learning con sede en San Francisco. Está respaldado comercialmente por la startup Skymind, que agrupa DL4J, TensorFlow, Keras y otras bibliotecas de Deep Learning en una distribución empresarial llamada Skymind Intelligence Layer. Se ha demostrado que los cálculos de álgebra lineal subyacente de DeepLearning4j, realizados con ND4J, se ejecutan al menos dos veces más rápido que Numpy en multiplicaciones matriciales muy grandes [22]: Además, DeepLearning4j se ha optimizado para ejecutarse en varios chips, incluidos x86 y GPU con CUDA C . Aunque la mayoridad de personas de la comunidad de Deep Learning utiliza Python, Deep Learning 4J(DL4J) tiene algunas ventajas. En el anexo C, se encuentra mas información de esta herramienta, ya que fue la seleccionada para el entrenamiento de los modelos.

2.2 Estado del arte

A continuación, se realiza una breve descripción de algunos de los trabajos publicados en la literatura científica y que están relacionados con esta investigación, es decir que hagan referencia a sistemas de DoS implementados con Machine Learning o Deep Learning. En esta síntesis se hace énfasis en los métodos, los algoritmos, los dataset empleados y los resultados obtenidos.

Un grupo de investigadores del Centro de Seguridad de Información de la universidad de Beijing [2], se enfocaron en estudiar dos métodos para el análisis de detección de intrusos en la red: Machine Learning y Deep Learning. Estos métodos fueron estudiados realizando una implementación y comparación entre las similitudes y diferencias de los algoritmos, entrenándolos con diferentes Datasets proporcionados por otros centros de seguridad como KDD99, DARPA Y NSL-KDD. La tabla 6 muestra la lista de los artículos, métodos, Dataset y Accuracy investigados. Según plantean los autores; luego de la implementación y comparación, no se establece cual es el método más efectivo, debido a que cada estrategia para implementar un sistema de detección presenta ventajas y desventajas. Además de esto, los métodos de Machine Learning y Deep Learning no funcionan correctamente sin los datos adecuados y los Datasets públicos actuales son anticuados y desiguales, y crear un Dataset nuevo representa una gran inversión de tiempo. Otro factor importante y que es una gran desventaja para estos métodos, es la actualización constante de la información en la red, lo que requiere un reentrenamiento de los modelos rápido y a largo plazo.

Tabla 6. Resumen de métodos de Deep Learning, paper , Dataset y Accuracy.Tabla tomada de **[2].**

Methods	Paper	Data used	Accuracy	Precision	FAR	F1-score
RBF-SVM	Kotpalliwar and Wajgi. [28]	10% KDD Cup 99	99.9%	-	-	-
PSO-SVM	Saxena and Richariya. [29]	KDD Cup 99	99.0%	84.2%	-	-
SVM	Pervez and Farid. [30]	NSL-KDD	82.37%	74%	-	0.77
C-SVM	Chandrasekhar and Raghuveer. [31]	10% KDD Cup 99	98.9%	99.5%	-	0.98
SVM	Yan and Liu. [32]	Part of DARPA 1998	80.1%	81.2%	0.47%	-
SVM	Kokila et al. [33]	DARPA 1998	95.11%	-	0.8%	-
IPDS-KNN	Rao and Swathi. [36]	Part of NSL-KDD	99.6%	-	-	-
Kmeans-KNN	Sharifi et al. [37]	NSL-KDD	90%	-	-	-
kFN-KNN	Shapoorifard and Shamsinejad. [38]	NSL-KDD	99%	98%	4%	-
KNN	Meng et al.[39]	DARPA 1999	85.2%	-	-	0.824
ACO-KNN	Vishwakarma et al.[40]	Part of KDD Cup 99	94.7%	-	5.82%	-
DBN	Ding et al. [56]	Netflow	96.1%	-	-	-
DBN	Nadeem et al. [57]	KDD Cup 99	99.18%	-	-	-
DBN	Gao et al. [58]	KDD Cup 99	93.49%	92.33%	0.76%	-
DBN-PNN	Zhao et al.[59]	KDD Cup 99	99.14%	93.25%	0.62%	-
LR-DBN	Alrawashdeh and Purdy [60]	10% KDD CUP 99	-	97.9%	2.47%	-
DBN	Alom et al.[61]	40% NSL-KDD	97.5%	-	-	-
DBN	Tan et al. [62]	Netflow	97.6%	-	0.9%	-
RNN	Yin et al. [63]	NSL-KDD	83.28%	-	-	-
RNN	Krishnan and Raajan [64]	KDD CUP 99	77.55%	84.6%	-	0.73
LSTM	Staudemeyer [65]	10% KDD CUP 99	93.85%	-	1.62%	-
LSTM	Kim and Jihyun [66]	10% KDD CUP 99	96.93%	98.8%	10%	-
LSTM	Kim et al. [67]	KDD Cup 99	99.8%	-	5.5%	-
LSTM	Le et al. [68]	KDD Cup 99	97.54%	98.95%	9.98%	-
GRU	Agarap [70]	Netflow	84.15%	-	-	-
CNN	Yu et al. [73]	CTU-UNB datasets	-	98.44%	-	0.98
CNN	Kolosnjaji et al. [75]	Netflow	-	93%	-	0.92
CNN	Saxe and Berlin [76]	Netflow	92%	-	0.1%	-
1D-CNN	Wang et al. [77]	ISCX dataset	-	97.3%	-	0.96
CNN	Wang et al. [78]	Netflow	99.41%	-	-	-

En otro estudio, realizado por Nvaporn y Vasaka [24], presentaron una técnica de detección de intrusos utilizando un modelo de Deep Learning que puede clasificar diferentes tipos de ataques, sin reglas creadas por los desarrolladores o firmas. Los autores utilizaron algoritmos de Deep Learning supervisado: RNN, Stacked Keras y CNN para clasificar cinco tipos de ataques usando la librería Keras en TensorFlow. Esta técnica solo requiere la información de la cabecera del paquete y ningún payload. Para verificar el rendimiento se utilizó un Dataset llamado MAWI que son archivos pcap y se compara el resultado con IDS snort. En el estudio mencionan que experimentaron con un modelo con 1000 épocas, a una tasa de aprendizaje entre 0.01 y 0.5. CNN y sus variantes mostraron un rendimiento significativo comparado con los clasificadores de Machine Learning. La tabla 7 describe la configuración utilizada para el entrenamiento de los modelos propuestos:

Tabla 7. Parámetros de configuración. Tabla tomada de [24].

Model	Parameter	Value
RNN	Epoch	100
	Batch	512
	Activation function	softmax
	Loss function	categorical_crossentropy
	Optimizer	Adam
Staked RNN	Epoch	100
	Batch	512
	Activation function	softmax
	Loss function	categorical_crossentropy
	Optimizer	Adam
CNN	Epoch	100
	Batch	512
	Activation function	Relu, softmax
	Loss function	categorical_crossentropy
	optimizer	Adam

La tabla 8 muestra los resultados de los modelos entrenados:

Tabla 8. Calificaciones de cada algoritmo. Tabla tomada de [24].

Result	Model	Attack Type				
		Port scan	Network scan ICMP	Network scan UDP	Network scan TCP	DoS
Precision	Snort	1.00	0.00	0.00	1.00	1.00
	RNN	1.00	1.00	0.99	1.00	1.00
	Stacked RNN	1.00	1.00	1.00	1.00	1.00
	CNN	1.00	1.00	0.99	0.99	1.00
Recall	Snort	0.0023	0.00	0.00	0.48	0.0004
	RNN	1.00	1.00	1.00	0.99	1.00
	Stacked RNN	1.00	1.00	1.00	1.00	1.00
	CNN	1.00	1.00	1.00	0.99	0.99
f1-score	Snort	0.0046	0.00	0.00	0.65	0.0075
	RNN	1.00	1.00	1.00	0.99	1.00
	Stacked RNN	1.00	1.00	1.00	1.00	1.00
	CNN	1.00	1.00	0.99	0.99	1.00

Los 3 métodos de Deep Learning pueden detectar diferentes tipos de ataques con una calificación de 1 sin acceder al payload, solo con información de la cabecera, siendo el algoritmo de RNN el que obtuvo mejores resultados en términos de Accuracy. Para validar el funcionamiento de detección, se validaron los modelos en un ambiente cerrado y controlado. Entonces para los investigadores el siguiente paso es implementarlo en un ambiente en tiempo real.

En el trabajo realizado por Priyadarshini y Barik [25], entrenaron un modelo utilizando el algoritmo con el método LSTM(Long Short Term Memory). Para el entrenamiento se utilizó el Dataset Hogzilla y el Dataset ISCX 2012. La red LSTM se ha configurado con 3 capas ocultas, una capa densa, 128 nodos de entrada y un dropout de 0.2 para todas las capas ocultas dando un buen indicador de rendimiento en términos de precisión creciente y tasa de error reducido. La figura 9 muestra la tasa de error en diferentes modelos. Los modelos propuestos son: LSTM sin capas ocultas, LSTM con 1 capa oculta, LSTM con 2 capas ocultas y LSTM con 3 capas ocultas y un dropout de 0.1 y 0.2 respectivamente.

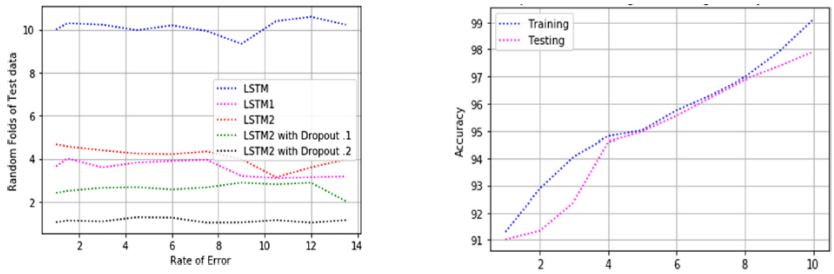


Figura 9. Gráfica lineal de Tipos de ataques en el Dataset CICDoS2019. Figura tomada de **[25]**.

Y en la tabla 9 se muestra el accuracy de los modelos planteados.

Tabla 9. Tipos de modelos con diferentes parámetros y su rendimiento. Tabla tomada de **[25].**

Model Type Activation Function	LSTM with no hidden layers Sigmoid, Tanh	LSTM with 1 hidden layers (LSTM-1) Sigmoid, Tanh	LSTM with 2 hidden layers (LSTM-2) Sigmoid, Tanh	LSTM with 3 hidden layers (LSTM-3) Sigmoid, Tanh
	Validation Accuracy Percentage			
No hidden neurons = 32	87.67	91.33	94.68	93.67
No hidden neurons = 64	87.96	96.98	95.67	97.45
No hidden neurons = 128	89.88	96.45	95.89	97.21
Dropout = 0.0	89.88	96.45	95.89	93.29
Dropout = 0.1	90.13	91.57	97.39	96.78
Dropout = 0.2	90.98	92.89	98.88	98.34

En la demostración del funcionamiento de los modelos, los investigadores modificaron el flujo de paquetes mediante una red definida por software, equipando el módulo de defensa contra ataques DDoS con el modelo entrenado, de esta manera, los paquetes infectados son denegados por la red, evitando que se propague la denegación de servicio.

Bhuvaneswari y Selvakumar [26] propusieron una estrategia para detección de ataques de DoS conocida como DeeRal (Deep Radial Intelligence) con un optimizador de weights conocido como CUl (Cumulative Incarnation), utilizando redes neuronales, la función de activación RBF (Radial basis function) y los Dataset NSL KDD y UNSW NB15 para entrenar el modelo. La tabla 10 muestra el porcentaje de detección con diferentes configuraciones en la red neuronal, donde se puede evidenciar que la red neuronal con el optimizador CUL tuvo la mejor puntuación con una tasa de 99.69 de detección. El articulo compara el resultado con otras estrategias de detección de ataques DoS, pero no demuestra su aplicación en casos concretos, ni tampoco evidencia un trabajo futuro a realizar con esta estrategia.

Tabla 10. Resultados de rendimiento. Tabla tomada de **[26].**

Dataset	Methodology	Variations	TN	FP	FN	TP	TPR	FPR	Accuracy	ER
NSL KDD	No optimization	RBF	2140	12	20	3583	99.44	0.56	99.44	0.56
		DL	2141	11	20	3583	99.44	0.51	99.46	0.54
		DeeRal	2141	11	17	3586	99.53	0.51	99.51	0.49
	GA	RBF with GA	2143	9	17	3586	99.53	0.42	99.55	0.45
		DL with GA	2143	9	13	3590	99.64	0.42	99.62	0.38
		DeeRal with GA	2145	7	12	3591	99.67	0.33	99.67	0.33
	DE	RBF with DE	2144	8	19	3584	99.47	0.37	99.53	0.47
		DL with DE	2143	9	18	3585	99.50	0.42	99.53	0.47
		DeeRal with DE	2143	9	13	3590	99.64	0.42	99.62	0.38
	CuI	RBF with CuI	2142	10	19	3584	99.47	0.46	99.50	0.50
		DL with CuI	2146	6	16	3587	99.56	0.28	99.62	0.38
		DeeRal with CuI	2146	6	12	3591	99.67	0.28	99.69	0.31
UNSW NB15	No optimization	RBF	52,362	3638	1745	10,519	85.77	6.50	92.11	7.89
		DL	52,929	3071	1574	10,690	87.17	5.48	93.20	6.80
		DeeRal	53,606	2394	1380	10,884	88.75	4.28	94.47	5.53
	GA	RBF with GA	53,125	2875	1612	10,652	86.86	5.13	93.43	6.57
		DL with GA	53,477	2523	1453	10,811	88.15	4.51	94.18	5.82
		DeeRal with GA	53,981	2019	1288	10,976	89.50	3.61	95.16	4.84
	DE	RBF with DE	53,379	2621	1572	10,692	87.18	4.68	93.86	6.14
		DL with DE	53,610	2390	1499	10,765	87.78	4.27	94.30	5.70
		DeeRal with DE	53,846	2154	1341	10,923	89.07	3.85	94.88	5.12
	CuI	RBF with CuI	53,354	2646	1270	10,994	89.64	4.73	94.26	5.74
		DL with CuI	53,979	2021	1192	11,072	90.28	3.61	95.29	4.71
		DeeRal with CuI	54,218	1782	847	11,417	93.09	3.18	96.15	3.85

En la investigación realizada por Zouhair, Noredinne, Khalid, Amina y Mohamed [27], implementaron un Network IDS utilizando Deep Learning basado en algoritmos genéticos y algoritmo de enfriamiento simulado. La tabla 11 muestra la configuración de la red neuronal y la calificación obtenida. El IDS propuesto fue instalado en un datacenter en la nube, tanto en el frontend como en el backend, en tiempo real, pero el articulo no evidencia esta implementación. Como trabajo futuro proponen adaptar la estrategia propuesta para detectar intrusos analizando paquetes encriptados.

Tabla 11. Configuración y rendimiento de las mejores configuraciones. Tabla tomada de **[27].**

Table 16 – Configuration and performances of best MLIDS_ CIDDS-001 2017.

Parameters of configuration	Value	Performance Metric	Value
Number of nodes in Input layer	10	Accuracy	99.92%
Number of nodes in Hidden layer 01	7	Precision	98.87%
Number of nodes in Hidden layer 02	4	Detection Rate (DR)	99.86%
Number of nodes in Output layer	1	False Negative Rate (FNR)	0.14%
Activation function	Sigmoid	False Positive Rate (FPR)	0.08%
Normalization of data	Min-Max	True Negative Rate (TNR)	99.92%
Learning rate	8.412874497406361E-7	F-score	0.99
Momentum term	1.264876945375064E-4	AUC (Ability to avoid misclassifications)	99.89%

En el trabajo de Sydney y Yanxia [28], diseñaron un IDS basado en un algoritmo de redes neuronales recurrentes llamado Deep Long short term memory para detectar intrusos en una red wifi. Para entrenar el modelo utilizaron el Dataset NSL KDD. La tabla 12 muestra el rendimiento del entrenamiento con diferentes configuraciones.

Tabla 12. Rendimiento de modelos DLSTM RNN**.** Tabla tomada de **[28].**

HU	HL	DFFL	Val. AC	F1-Score	Test AC
30	3	R5	99.23%	99.32%	85.34%
30	3	R10–5	98.26%	99.50%	86.34%
75	3	R25–5	99.28%	99.42%	85.84%
90	3	R30–5	99.44%	99.62%	85.45%
105	3	R35–5	99.32%	99.49%	85.51%
90	3	S30–5	99.20%	99.41%	85.34%
90	3	S29–5	99.51%	99.43%	86.99%

Los resultados presentados en la tabla 12 demuestran que el mejor rendimiento del modelo DLSTM RNN tiene 90 Hidden Units distribuidas en 3 Hidden Layers, con un Accuracy de 99,51. La estructura DFFL tiene 29 neuronas con función sigmoide en la primera capa y 5 soft Max en la última capa. El articulo compara los resultados con otros métodos de Machine Learning, mostrando un incremento signiticativo en cuanto a rendimiento. Como trabajo futuro, planean estudiar el rendimiento de cada ataque que se encuentra en el Dataset NSL-KDD.

Mientras que en el estudio realizado por Hongyu Liu, Bo Lang, Ming Liu y Hanbing Yan [29], proponen dos modelos para detección de ataques utilizando Deep Learning, que consiste en el análisis de payload llamados: Payload Convolutional Neural Networks y Payload Recurrent Neural Network. Los investigadores utilizaron los Datasets CNTC-2017, DARPA1998, KDD99 y CSIC-2010 HTTP Dataset para el entrenamiento. La tabla 13 muestra el tiempo en que los modelos detectan un ataque, lo cual permite en la práctica, detectar ataque punto a punto en milisegundos.

Tabla 13. Tiempo de detección de los modelos. Tabla tomada de [28].

Model	Stage	Time
PL-CNN	Word embedding and filtering	0.2 ms
	Resizing	4.2 ms
	Classification	4.7 ms
PL-RNN	Classification	2.8 ms

Según los invesigadores estos modelos presentan dos inconvenientes. El primero es que estos modelos contienen una gran cantidad de parámetros comparados con modelos de Machine Learning, por lo que la realización de ajustes es compleja en su entrenamiento y requiere un conocimiento técnico. Lo segundo es que se hace difícil la interpretación de los métodos de CNN y RNN porque son caja negra, es por esto que se propone interpretar estos modelos como trabajo futuro. Además, pretenden modificar los modelos para detectar partes de paquetes anómalos que puede ayudar a los analistas de seguridad a entender este tipo de paquetes.

Para Michael Siacusano, Stavros y Bogdan [30] que realizaron un entrenamiento de diferentes algoritmos de Machine Learning supervisados utilizando un conjunto de datos propios y externo para detectar ataques de denegación de servicio tipo Low. Los algoritmos utilizados para entrenar el modelo son Logistic Regresion, K-NN, SVM, Decision Trees,

Random Forest y Deep Neural Network. La tabla 14 muestra los resultados de los modelos entrenados.

Tabla 14. Lista de resultados de algoritmos de Machine Learning. Tabla tomada de **[30]**.

Dataset	LG	LG	KNN	SVC	DT	RF	DNN
1	Accuracy	95.85	99.81	97.39	99.87	99.07	97.06
	FPR	7.76	0.08	4.09	0	0.58	N/A
	FNR	1.09	0.19	0.44	0.03	4.79	N/A
	Eval time [s]	0.023	0.558	1.87	0.019	1.842	471.9
2	Accuracy	94.82	99.96	99.96	99.96	99.93	99.96
	FPR	1.8	0	0	0	0.16	N/A
	FNR	9.65	0.03	0	0	0.13	N/A
	Eval time [s]	0.017	0.221	0.092	0.013	1.261	305.7
3	Accuracy	98.75	99.86	99.77	99.92	99.41	99.47
	FPR	1.77	0.06	0.26	0	0.56	N/A
	FNR	0.62	0.06	0.11	0.01	0.57	N/A
	Eval time [s]	0.034	1.096	1.727	0.029	2.951	510.8
4	Accuracy	92.75	99.9	99.35	99.92	99.41	99.47
	FPR	21.63	0.14	0.91	0	1.31	N/A
	FNR	1.3	0.03	0.54	0	0.14	N/A
	Eval time [s]	0.51	61.21	463.38	0.505	581.84	5203

De acuerdo a los investigadores el objetivo principal del trabajo, era demostrar si los parámetros de los paquetes TCP podrían ser utilizados como features para detectar ataques LDDoS, lo cual se logró. La principal limitación de la investigación fueron los Datasets utilizados, pues algunos son datos simulados, y es por ello que como trabajo futuro se pretende utilizar datos reales y en mayor cantidad. Otra de las limitaciones fueron los recursos de hardware que impedían aumentar la cantidad de epocas para lograr un mejor rendimiento, y también la falta de mas datos que podrían permitir la creación de mas modelos aplicables a mas configuraciones del servidor y contenidos.

Narayanavadivoo y Selvakumar [31] desarrollaron una técnica llamada VCDeepFL (Vector Convolutional Deep Feature Learning) para la detección de ataques DoS. Esta técnica consiste en la combinación de dos algoritmos de Deep Learning: Vector Convolutional Neural Network (VCNN) y Fully Connected Neural Network (FCNN). VCNN extrae los features y proporciona una mejor representación del vector de entrada. FCNN es un clasificador multiclase que aumenta el rendimiento de la detección del ataque automáticamente calculando el mejor conjunto de weights del entrenamiento. El Dataset utilizado para el entrenamiento del modelo es el Dataset NSL KDD. La figura 10 muestra la configuración de la red neuronal con 11 neuronas de entrada y 6 de salida que representan los tipos de paquetes del Dataset NSL KDD.

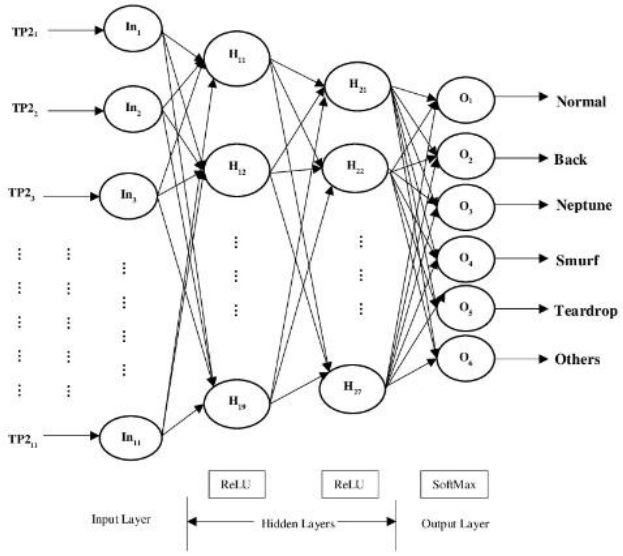


Figura 10. Configuración de red neuronal. Figura tomada de **[31].**

La figura 11 muestra la calificación del sistema propuesto frente a otros algoritmos: Multi Layer Perceptron y Support Vector Machine, donde smurf es el ataque de DoS.

Approach	Metrics	Normal	Back	Neptune	Smurf	Teardrop	Others
Proposed $VCDeepFL$	Accuracy	99.3	97.8	99.1	99.2	83.3	87.1
	Precision	99.6	95.9	97.9	91.1	19.6	97.8
	Recall	99.3	97.8	99.1	99.2	83.3	87.1
	F-score	99.4	96.8	98.5	95.0	31.7	92.1
	False Alarm	0.7	2.2	0.9	0.8	16.7	12.9
Reported Result [10]	Accuracy	97.91	36.77	98.05	99.10	100	-
	Precision	100	100	100	100	100	-
	Recall	97.91	36.77	98.05	99.10	100	-
	F-score	98.94	53.77	99.01	99.55	100	-
	False Alarm	14.70	0.00	0.00	0.15	75.51	-
MLP	Accuracy	98.9	96.4	98.8	99.5	91.7	78.8
	Precision	99.1	86.5	95.6	90.8	22.9	97.5
	Recall	98.9	96.4	98.8	99.5	91.7	78.8
	F-score	99.0	91.2	97.2	95.0	36.6	87.2
	False Alarm	1.1	3.6	1.2	0.5	8.3	20.2
SVM	Accuracy	99.0	95.5	98.5	98.7	66.7	81.5
	Precision	99.2	83.7	95.1	91.2	44.4	97.0
	Recall	99.0	95.5	98.5	98.7	66.7	81.5
	F-score	99.1	89.2	96.8	94.8	53.3	88.6
	False Alarm	1.0	4.5	1.5	1.3	33.3	18.5

Figura 11. Clasificación de los sistemas propuestos. Figura tomada de **[31].**

El estudio no demuestra la implementación del sistema de detección en un ambiente de pruebas, por lo que solo se queda en la fase de prueba del entrenamiento. Como trabajo futuro los investigadores proponen la extracción de reglas de la red VCDeepFL para cada clase del Dataset y facilitar la interpretación.

Los investigadores Alzahrani y Hong [32], propusieron un sistema donde aplican dos estrategias para detectar ataques desconocidos, una red neuronal artificial basada en anomalías y una estrategia basada en firmas. El primer paso del sistema consiste en comparar el tráfico de red con bases de datos para ataques conocidos. Si el ataque no se

encuentra en la base de datos, comienza el funcionamiento de la red neuronal artificial y posteriormente se crea una firma para el ataque desconocido y se actualiza la base de datos. La figura 12 muestra el diagrama de flujo de los ataques conocidos y desconocidos.

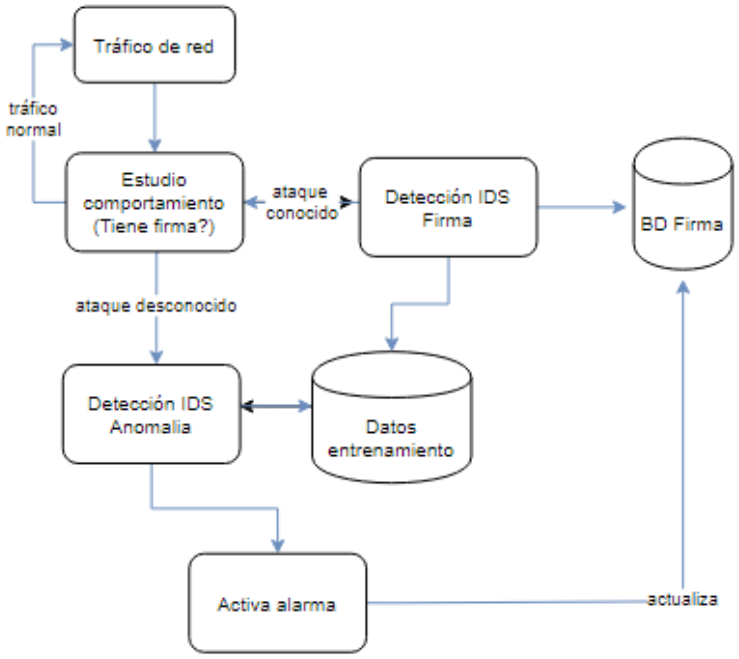


Figura 12. Flujo de detección de ataques DoS. Adaptada de **[32].**

En la prueba del experimento, se contó con un Dataset que contiene 95.128 paquetes de tráfico normal y 812 de paquetes atacantes. La tabla 15 muestra la eficiencia y precisión del sistema propuesto.

Tabla 15. Eficiencia y precisión del sistema propuesto. Adaptada de **[32].**

Métrica	Detección basada en firmas	Detección basada en anomalías	El sistema hibrido propuesto
Accuracy	97,52%	98,10%	99,98%
Tasa de detección	76,10%	80,78%	98,15%
Tasa de falsos positivos	2,30%	1,70%	0,00%

El experimiento fue llevado a cabo en una infraestructura cloud de AWS, pero no muestra evidencias de su posterior funcionamiento, ni se propone un trabajo futuro por parte de los investigadores.

En otro estudio, un grupo de investigadores del Instituto de Tecnología de la Información de Sri Lanka [33], proponen la combinación de Deep Learning con minería de datos Association Rule Mining para la detección de intrusos.

Las herramientas que utilizaron para desarrollar este sistema fueron:

- DeepLearning4j que contiene la librería para implementar algoritmos de Deep Learning en java.
- Weka para normalizar el conjunto de datos.
- Apache Spark para entrenar el algoritmo
- Dataset KDD99
- El modelo de inteligencia artificial utilizado se llama Recurrent neural Networks.

Utilizando estas herramientas los investigadores entrenaron un modelo con el algoritmo de redes neuronales recurrentes, el cual clasifica las transacciones con 2 etiquetas (1 si es ataque malicioso, 0 si es una transacción normal). En la tabla 16 se puede apreciar el resultado del modelo luego de su fase de pruebas, logrando una correcta clasificación de transacciones de 99.49%.

Tabla 16. Calificación de rendimiento. Adaptada de **[33].**

Componente	Valor
Accuracy	0,9949
Precision	0,9967
Recall	0,9894
F1 Score	0,993

Según los autores, en una de las fases de entrenamiento no utilizaron el método de Deep Learning, pues al incrementar el numero de inputs y outputs, incrementaría la cantidad de capas del modelo y podría causas problemas de rendimiento, lo que ocasionaría la caída de la aplicación. Como trabajo futuro pretenden utilizar otros modelos de Deep Learning, diferente al RNN utilizado, y evaluar la efectividad del modelo con otros Datasets.

Por otro lado, Khuphiran, Leelaprute, Uthayopas y otros investigadores [34] entrenaron un modelo con el algoritmo de Deep Learning llamado Deep Feed forward utilizando el Dataset Darpa Intrusion Detection con el cual lograron detectar ataques de denegación de servicio con un Accuracy de 99.63. El Dataset solo contenía ataques SYN Flood. La figura 13 muestra la configuración de la red neuronal utilizada.

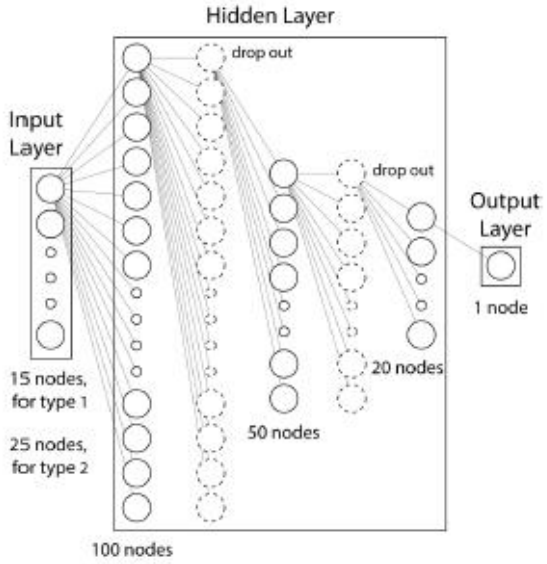


Figura 13. Configuración red neuronal utilizada. Figura tomada de **[34].**

El modelo fue comparado con un algoritmo de Machine Learning llamado SVM, el cual tuvo una diferencia significativa con este, como se muestra en la tabla 17.

Tabla 17. Mejores resultados de rendimiento para cada algoritmo. Tabla tomada de **[34].**

SVM	Model Performance				Time used (s)	
Data	A(%)	R	P	F1-score	Train	Test
Window-basis	93.01	0.922	0.933	0.927	371.118	0.003
Packet-basis(S)	92.58	0.894	0.933	0.906	379.417	0.003
Packet-basis(L)	81.23	0.927	0.756	0.826	138.260	0.500
Deep learning	Model Performance				Time used (s)	
Data	A(%)	R	P	F1-score	Train	Test
Window-basis	61.30	0.240	0.452	0.295	3.314	0.117
Packet-basis(S)	68.30	0.366	0.922	0.504	3.438	0.115
Packet-basis(L)	99.63	0.994	0.998	0.996	239.614	14.651

De acuerdo al articulo, no se demuestra el funcionamiento de los modelos en un ambiente de pruebas, por lo que los investigadores proponen como trabajo futuro utilizar los dos algoritmos en una red de datos en tiempo real.

Otro grupo de investigadores [35] describen el desarrollo de una estrategia para detección de diferentes tipos de ataques a un vehículo en movimiento utilizando Deep Learning. Los tipos de ataques utilizados fueron denegación de servicio, command injection y malware. Los ataques utilizados, fueron realizados por los mismos investigadores. La figura 14 muestra el rendimiento de la red neuronal Accuracy vs número de neuronas, donde se evidencia que el ataque DoS tiene el Accuracy más alto.

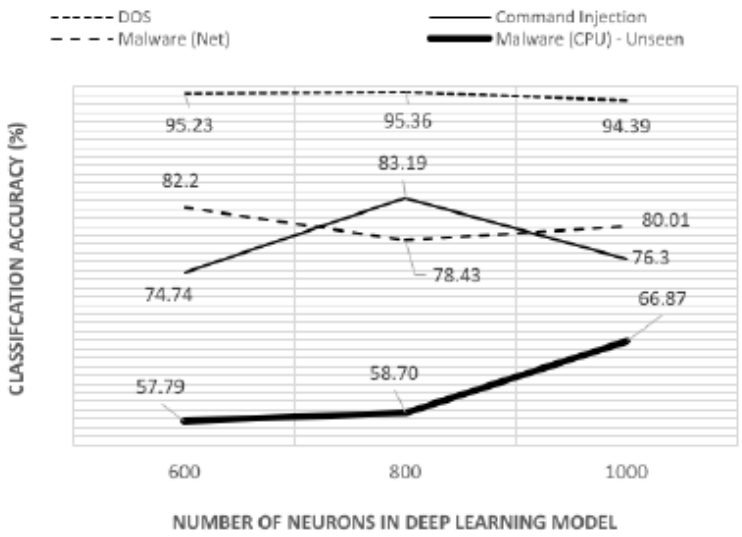


Figura 14. Accuracy vs número de neuronas. Figura tomada de **[35].**

La tabla 18 muestra el rendimiento de los algoritmos con su tipo de ataque.

Tabla 18. Accuracy de los modelos de Deep Learning vs Machine Learning. Tabla tomada de **[35].**

ML algorithm	Attack	ACC (%)	Overall ACC(%)
Logistic Regression	Denial of Service	95.5	73.7
	Command Injection	56.9	
	Malware (Net)	68.6	
Decision Tree (C5.0)	Denial of Service	73.2	74.0
	Command Injection	66.5	
	Malware (Net)	**82.4**	
Random Forest	Denial of Service	71.8	77.3
	Command Injection	78.6	
	Malware (Net)	81.6	
SVM (Radial Kernel)	Denial of Service	**97.4**	79.9
	Command Injection	67.7	
	Malware (Net)	74.5	
SVM (Linear Kernel)	Denial of Service	95.2	71.6
	Command Injection	48.0	
	Malware (Net)	71.6	
Deep learning (MLP)	Denial of Service	96.2	78.3
	Command Injection	58.0	
	Malware (Net)	80.7	
Deep learning (RNN)	Denial of Service	95.4	**86.9**
	Command Injection	**83.2**	
	Malware (Net)	82.2	

De ese articulo se demuestra que Deep Learning presenta una ventaja en el rendimiento frente a los algoritmos de Machine Learning, pero tiene la desventaja de una alta latencia en la detección debido al incremento en el procesamiento, Como trabajo futuro se consideran otros factores para la prueba como disponibilidad, costo y seguridad en la insfraestructura remota, pues en este experimiento se utilizó una nube privada, pero lo ideal es llevarlo al mundo real con vehículos sin conductor.

En otro estudio [36] desarrollaron un IDS utilizando Deep Learning con extracción de feature automática. La red neuronal contiene Gated Recurrent unit (GRU), Perceptron multicapa (MLP) y módulo softmax. Para entrenar y testear el modelo se utilizaron los Dataset NSL KDD y KDD 99 con un rendimiento de Accuracy de 99.42 % para detectar ataques DoS de 99.98 % y 99.55 % respectivamente. El estudio reveló que GRU es más apropiado como unidad de memoria que LSTM. La tabla 19 describe las diferentes medidas de los métodos propuestos con los diferentes Dataset seleccionados y que demuestra que la red MLP con el GRU bidireccional tiene un mejor rendimiento.

Tabla 19. Resultados de experimentos KDD 99 y NSL KDD. Tabla tomada de **[36].**

System	Accuracy (%)	DR (%)	FPR (%)
BGRU+MLP	**99.84**	**99.42**	**0.05**
GRU+MLP	99.28	96.73	0.07
BLSTM+MLP	98.57	93.78	0.17
LSTM+MLP	98.51	94.77	0.53
GRU	92.28	71.77	0.13
LSTM	91.91	70.77	0.10
MLP	91.88	70.92	0.31
BGRU+MLP	**99.24**	**99.31**	**0.84**
GRU+MLP	99.19	99.35	1.00
BLSTM+MLP	96.41	95.65	2.67
LSTM+MLP	95.22	93.97	3.24
GRU	94.94	94.76	4.84
LSTM	94.1	95.65	7.58
MLP	90.56	86.61	3.49

La tabla 20 muestra la comparación del puntaje obtenido con otros modelos:

Tabla 20. Comparación de rendimiento con otros estudios. Tabla tomada de **[36].**

System	Dataset	Accuracy (%)	DR (%)	FPR (%)
LSTM(2015) [20]	KDD 99	94.11	77.07	0.18
OS-ELM(2015) [38]	NSL-KDD	N/A	97.67	1.74
LSSVM+FMIFS(2016) [39]	KDD 99	99.79	99.46	0.13
LSSVM+FMIFS(2016) [39]	NSL-KDD	99.91	98.76	0.28
LSTM-RNN(2016) [21]	KDD 99	96.93	98.88	10.04
TVCPSO-MCLP(2016) [12]	NSL-KDD	N/A	97.23	2.41
Pruning VELM(2017) [15]	KDD 99	98.94	98.37	0.32
VELM(2017) [15]	NSL-KDD	97.58	97.69	2.22
GA+FLN(2018) [22]	KDD 99	99.69	N/A	N/A
PSO+FLN(2018) [22]	KDD 99	99.68	N/A	N/A
BGRU+MLP (proposed)	**KDD 99**	**99.84**	**99.42**	**0.05**
BGRU+MLP (proposed)	**NSL-KDD**	**99.24**	**99.31**	**0.84**

De acuerdo al articulo, el sistema propuesto solo quedo en la teoría y requiere de un trabajo de ingeniería para llevarlo a la practica, por lo que proponen que el siguiente paso seria optimizar el sistema para que pueda ser aplicado en ambientes del mundo real.

Kasun, Kevin y Milos [37], desarrollaron un framework para detección de anomalías utilizando Deep Learning. La red fue entrenada con el Dataset NSL KDD el cual no tiene los registros redundantes de KDD 99. El algoritmo utilizado fue Deep Feed forward. La tabla 21 contiene la configuración de las neuronas de la red neuronales en sus capas ocultas.

Tabla 21. Arquitecturas de Deep Learning testeadas. Tabla tomada de **[37].**

Model Name	Hidden Layer Architecture
3HL_Config-1	(256, 128, 64)
3HL_Config-2	(100, 100, 100)
4HL_Config-1	(256, 128, 64, 32)
4HL_Config-2	(100, 100, 100, 100)

La tabla 22 muestra el Accuracy obtenido con cada una de las configuraciones

Tabla 22. Rendimiento de arquitecturas de Deep Learning. Tabla tomada de **[37].**

Model	Basic Features		Complete Feature set	
	Train Accuracy (%)	Test Accuracy (%)	Train Accuracy (%)	Test Accuracy (%)
3HL_Config-1	95.9889	88.3191	98.6217	96.1572
3HL_Config-2	89.3754	**93.7852**	97.442	**97.2372**
4HL_Config-1	92.0553	93.4274	97.5925	97.1113
4HL_Config-2	86.8467	85.8743	97.1882	97.0318

En el articulo mencionan que el sistema de detección se encuentra en proceso de implementación y proponen como trabajo futuro, proporcionar información sobre las predicciones realizadas por el framework.

Los investigadores Yadigar y Fargana [38], utilizaron tres algoritmos de Deep Learning basados en Bolts Mann Machine para la detección de ataques de DoS. Para entrenar los algoritmos utilizaron el Dataset NSL-KDD y para entrenar la red neuronal se configuraron capas ocultas con 100 neurona, la normalización es entre 0 y 1 y los pesos iniciales fueron seleccionados aleatoriamente. La tabla 23 describe el puntaje de cada uno de los algoritmos de Deep Learning RBM. Según los autores, el modelo entrenado no es llevado a la práctica y como trabajo futuro se plantea entrenar modelos con los algoritmos LSTM decoders y bidirectional LSTM encoders.

Tabla 23. Comparación de resultados de los métodos de Deep Learning. Tabla tomada de **[38].**

Method	*Worst*	*Mean*	*Best*
Bernoulli-Bernoulli RBM	0.6422	0.7255	0.6828
Gaussian–Bernoulli RBM	0.6894	0.7591	0.7323
DBN	0.4343	0.5299	0.4809

En otro estudio [39] se utilizó el Dataset NSL-KDD para evaluar el rendimiento de un modelo entrenado para la detección DDoS basada en Deep Learning. En el estudio experimental se llevaron a cabo dos actividades diferentes. En la primera, la red neuronal propuesta detectó los ataques DoS con una precisión de clasificación de 0.988. En el segundo, el número de características de NSL-KDD se reduce a 24 logrando un Accuracy de 0.948. La figura 15 muestra el modelo propuesto para el primer experimento en donde se colocaron solo dos label de output, true or false y 23 input. Como trabajo futuro los autores proponen utilizar la estrategia propuesta con otras estrategias de Machine Learning en conjunto y cambiar los parámetros de la red neuronal para mejorar su rendimiento.

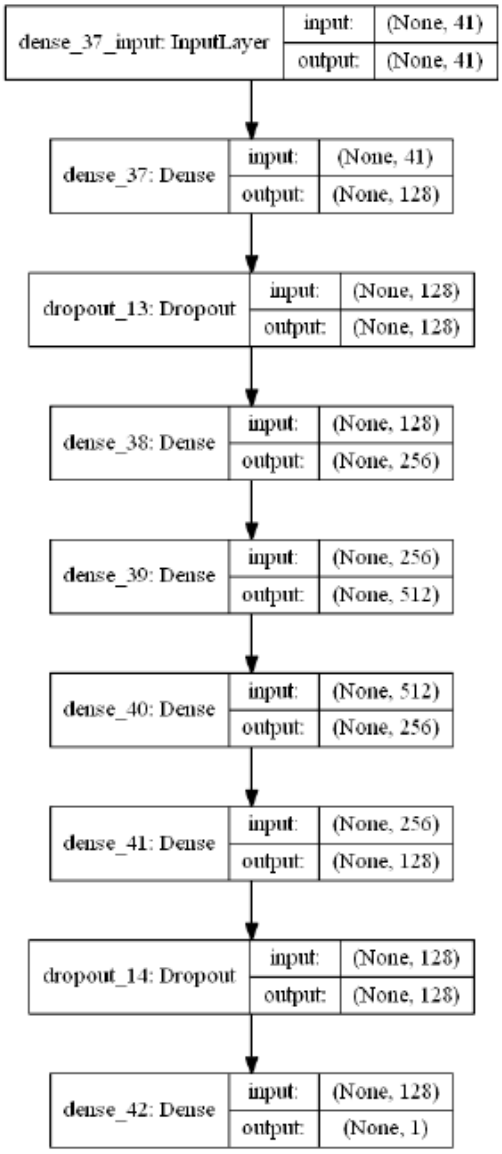


Figura 15. Modelo propuesto para el primer experimento. Figura tomada de **[39].**

En el trabajo [40], los investigadores utilizaron los métodos de Random Forest y LSTM, para identificar ataques de denegación de servicio. El resultado mostró que los algoritmos de Deep Learning presentan un mejor rendimiento frente a uno de Machine Learning. La tabla 24 muestra los resultados de la comparación donde se muestra las diferentes medidas de rendimiento. Como trabajo futuro proponen incrementar la diversidad de ataques DDoS y las configuraciones del sistema para probar el modelo en diferentes ambientes y por supuesto en un ambiente del mundo real. También proponen construir un sistema basado en el modelo creado y muy importante proponen la creación de un nuevo Dataset para dar a conocer los nuevos retos en la identificación de ataques DDoS.

Tabla 24. Comparación RF vs LSTM**.** Tabla tomada de **[40]**

ModelName	Error Rate	Accuracy	Precision	Recall	F1
LSTM	2.394%	97.606%	97.832%	97.378%	97.601%
Random Forest	6.373%	93.627%	92.532%	94.893%	93.698%

Para finalizar, Tae-Young Kim y Sung-Bae Cho [41] entrenaron un algoritmo de Deep Learning para detección de anomalías utilizando la combinación de redes neuronales convolucionales y LSTM. El resultado es comparado con otros métodos de Deep Learning en donde muestra que obtuvo un resultado significativo. El entrenamiento del algoritmo se hizo con un Batch size de 512 con 500 épocas y tomo 769 segundos de entrenamiento. La tabla 25 muestra la comparación de rendimiento de diferentes modelos.

Tabla 25. Comparación de rendimiento de diferentes modelos. Tabla tomada de **[41].**

Método	Accuracy	Precision	Recall	F1 score
LSTM	92.9	82.4	39.8	53.7
CNN	96.5	94.5	86.2	90.1
LSTM + DNN	93.1	88.2	34.5	49.6
CNN + DNN	96.7	95.1	86.5	90.6
CNN + LSTM	97.1	95.4	87.6	91.3
CNN + LSTM + DNN	**98.6**	96.2	**89.7**	92.3

En el estudio demuestran como el modelo propuesto extrae feautres que no pudieron ser extraidos en estudios de detección de anomalías anteriores usando los métodos convencionales de Machine Learning. Adicionalmente, el método presenta una demora para detectar anomalías con datos reales, debido al preprocesamiento que se requiere con los nuevos datos y lo cual representa un reto para trabajos futuro.

3. Metodología

Esta sección contiene los procedimientos, técnicas científicas, actividades y otras estrategias utilizadas durante el desarrollo la investigación. La figura 16 muestra la metodología utilizada para el proyecto.

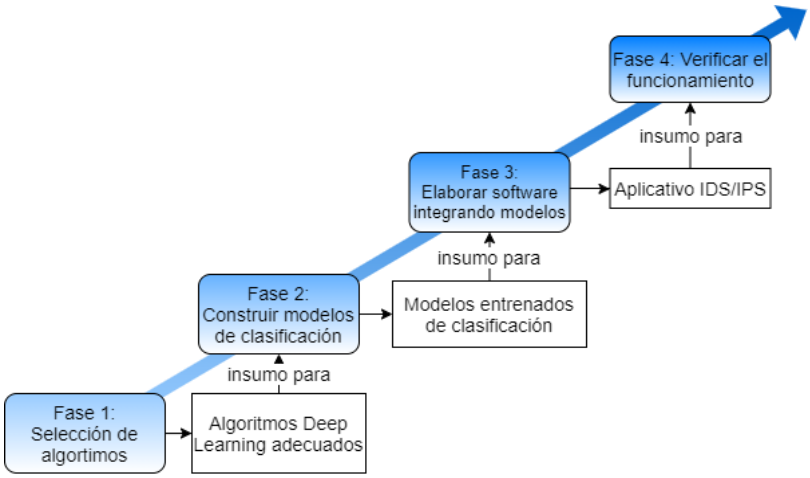


Figura 16. Fases de la metodología**.**

3.1 Fase 1: Selección de los algoritmos

En esta fase se describe el proceso para la selección de algoritmos más adecuados para la identificación de ataques de denegación de servicios web. Como fase previa a la selección del algoritmo se realizó la revisión sistemática de la literatura que se encuentra consignado en el apartado 2, bajo los ítems Marco Conceptual y el Estado del Arte. Además, se construyó a manera de síntesis la tabla 37, ubicada en el numeral 4.1, donde se encuentra el listado de algoritmos de Deep Learning, los Datasets con paquetes de ataques DoS y el Accuracy obtenido con los respectivos modelos entrenados.

3.1.1 Criterios de selección

De la tabla 37, se seleccionaron los algoritmos más adecuados de acuerdo a 4 criterios definidos por el grupo de investigadores previamente, los cuales viabilizaban el desarrollo de este trabajo y la obtención del objetivo propuesto, que se listan a continuación y se explica el porque de ellos:

- El Accuracy del modelo entrenado con el algoritmo de Deep Learning es superior al 95%, pues de acuerdo a la literatura los modelos que presentan un Accuracy mayor a este, tienden a realizar una clasificación más acertada.
- La configuración de entrenamiento del modelo que se utilizó con el algoritmo está disponible, dado que esto permite tener una guía de referencia para la construcción de los modelos que se desarrollaron en esta investigación.

- El Dataset utilizado junto al algoritmo, contiene paquetes de denegación de servicio, debido a que lo que se desea entrenar son modelos contra este tipo de ataques, y los Datasets encontrados en la literatura contienen diferentes tipos de ataques.
- El algoritmo se encuentra disponible en la librería de Java Deep Learning 4J, puesto que existen otras herramientas que permiten trabajar Deep Learning, pero en esta investigación se seleccionó DL4J de acuerdo a las ventajas que posee descritas en el Anexo C.

3.1.2 Implementación de algoritmos

Con los algoritmos seleccionados, se replicó el entrenamiento de los modelos entrenados con estos métodos, utilizando la librería de java Deep Learning 4J y el Dataset NSLKDD, para verificar el comportamiento de Deep Learning con respecto a los resultados obtenidos en la investigación [39] para el método de DFF y la investigación [2] para los métodos de RNN y LSTM reportadas en la literatura. La tabla 26 muestra la configuración del modelo DFF acompañado de la figura 17 donde se muestra el modelo implementado en java, y la tabla 27 para la configuración del modelo RNN y LSTM acompañado de la figura 18 implementados en java.

Tabla 26. Modelo propuesto por Aysegulsngr y Hacibeyoglu **[39].**

Seed		7	
Epocas		1000	
Tamaño Batch		25192	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	41
		Output	41
Layer 2	DenseLayer	Input	41
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2
		Loss Function	MCXENT

```
log.info("Build model....");
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .activation(Activation.TANH)
    .weightInit(WeightInit.XAVIER)
    .updater(new Adam(0.01))
    .list()
    .layer(new DenseLayer.Builder().nIn(numInputs).nOut(41)
        .build())
    .layer(new DenseLayer.Builder().nIn(41).nOut(128)
            .build())
    .layer(new DropoutLayer.Builder(0.5).build()
            )
    .layer(new DenseLayer.Builder().nIn(128).nOut(256)
            .build())
    .layer(new DenseLayer.Builder().nIn(256).nOut(512)
            .build())
    .layer(new DenseLayer.Builder().nIn(512).nOut(256)
            .build())
    .layer(new DenseLayer.Builder().nIn(256).nOut(128)
            .build())
    .layer(new DropoutLayer.Builder(0.5).build()
            )
    .layer( new OutputLayer.Builder(LossFunctions.LossFunction.MCXENT)
        .nIn(128).nOut(outputNum).build())
    .build();
```

Figura 17. Configuración Modelo en Java Deep Feed Forward.

Tabla 27. Modelo propuesto para los métodos LSTM y RNN.

Seed		7	
Epocas		1000	
Tamaño Batch		1000	
Funcion activación		Stochastic Gradient Descent	
Valores iniciales Weights		Xavier	
Layer 1	InputLayerLSTM	Input	41
		Output	128
		Function	softsign
Layer 2	OutputLayer RNN	Input	128
		Output	2
		Function	softmax
		Loss Function	MCXENT

```
final int numInputs = 41;
int outputNum = 2;
long seed = 7;
int nHidden = 128;
int batchSize = 1000;
int epochs = 20;
double LEARNING_RATE=0.01;

//Configuration
  NeuralNetConfiguration.Builder builder = new NeuralNetConfiguration.Builder();
    builder.seed(seed);
    builder.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT);
    builder.updater(new Sgd(LEARNING_RATE));

NeuralNetConfiguration.ListBuilder listBuilder = builder.list();

listBuilder.layer(0, new GravesLSTM.Builder().nIn(numInputs).nOut(nHidden)
    .activation(Activation.SOFTSIGN)
    .weightInit(WeightInit.XAVIER)
    .build());
listBuilder.layer(1, new RnnOutputLayer.Builder(LossFunctions.LossFunction.MCXENT)
    .activation(Activation.SOFTMAX)
    .weightInit(WeightInit.XAVIER)
    .nIn(nHidden).nOut(outputNum)
    .build());

// run the model
MultiLayerConfiguration conf = listBuilder.build();
MultiLayerNetwork model = new MultiLayerNetwork(conf);
```

Figura 18. Configuración Modelo en Java RNN y LSTM

3.2 Fase 2: Construcción modelos de clasificación

En la siguiente figura, se describe el proceso para la construcción de modelos de clasificación utilizando Deep Learning que consta de los siguientes pasos:

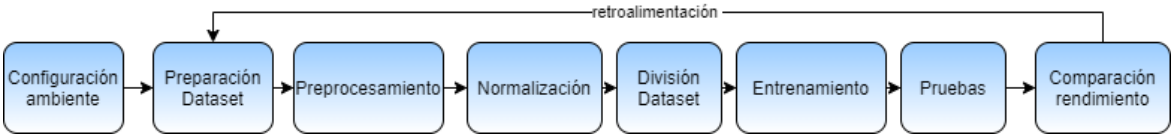


Figura 19. Pasos de la construcción de modelos**.**

3.2.1 Configuración ambiente de entrenamiento

Para el entrenamiento de los modelos se utilizó el sistema operativo Debian, debido a que es un sistema operativo que consume muy pocos recursos, los cuales son aprovechados por el algoritmo de entrenamiento. En la tabla 28 se muestra las especificaciones del servidor utilizado para el entrenamiento de los modelos.

Tabla 28. Hardware Servidor de entrenamiento**.**

Componente	Especificación
Procesador	Intel Xeon 1.90 GHz 64 bits, 8 cores
RAM	32 GB DDR4 2666
Disco Duro	Magnético 100 GB
Sistema Operativo	Debian 10
IDE	Eclipse
Librería	Java 1.8, DL4J

3.2.2 Preparación del Dataset

El Dataset seleccionado para el entrenamiento de los modelos, es el Dataset *CICDDoS2019* descrito en la sección 2.1.12 Datasets. CICDoS2019 es un Dataset creado para resolver los inconvenientes de los Datasets existentes de acuerdo a la literatura [12] y principalmente porque a comparación de otros DataSets , este contiene registros exclusivamente de ataques DoS. El Dataset esta constituido por 48.099.733 registros, divididos en 11 archivos como se evidencia en la tabla 29.

Tabla 29. Lista de archivos del Dataset CICDDoS2019.

Nombre	Tamaño	Cantidad registros
DrDoS_DNSFile	2.0 GiB	5074414
DrDoS_LDAPFile	874.8 MiB	218153
DrDoS_MSSQLFile	1.8 GiB	4524499
DrDoS_NetBIOSFile	1.6 GiB	4094987
DrDoS_NTPFile	615.1 MiB	1217008
DrDoS_SNMPFile	2.0 GiB	5161378
DrDoS_SSDPFile	1.2 GiB	2611375
DrDoS_UDPFile	1.4 GiB	3136803
SynFile	607.8 MiB	1582682
TFTPFile	8.7 GiB	20107828
UDPLagFile	150.7 MiB	370606

Cada registro representa la información de un paquete de red compuesto por 88 variables que contiene información como su ip de origen, ip destino, puerto destino, puerto origen, protocolo, etc. También contiene información del flujo que fue extraido con la herramienta CICFlow para la extracción de características. En la ultima columna de cada archivo, se encuentra la variable LABEL que representa el tipo de ataque DoS a la que pertenece cada registro, como se encuentra descrito en la sección 2.1.12, o si el paquete no hace parte de un ataque DoS, este se encuentra catalogado como BENIGN (NoMalignos). De acuerdo al estudio realizado por el Instituto Canadiense para la ciberseguridad, hay 22 variables que son más relevantes de las 88 que contiene el Dataset, con respecto a que tan importante es cada feature para cada clase. Por ejemplo, de acuerdo al estudio la variable *Packet lenght std,* es una de las más relevantes para los paquetes BENIGN. En la siguiente tabla se describen los 22 features más relevantes y los cuales serán utilizados para entrenar los modelos.

Tabla 30. Lista de features más relevantes**.**

Feature	Descripción
Destination Port	Puerto de Destino
Protocol	Número de Protocolo en la cabecera IP
Flow Duration	Duración del flujo en microsegundos
Fwd Packet Length Max	Tamaño máximo del paquete en dirección origen destino
Fwd Packet Length Min	Tamaño mínimo del paquete en dirección origen destino
Fwd Packet Length Std	Tamaño desviación estándar del paquete en dirección origen destino
Flow IAT Mean	Tiempo promedio entre dos paquetes enviados en el flujo.
Flow IAT Max	Tiempo máximo entre dos paquetes enviados en el flujo.
Fwd IAT Mean	Tiempo promedio entre dos paquetes enviados en dirección origen destino.
Fwd IAT Max	Tiempo máximo entre dos paquetes enviados en dirección origen destino.
Fwd IAT Min	Tiempo mínimo entre dos paquetes enviados en dirección origen destino.

Fwd Header Length	Total de bytes utilizados por las cabeceras en la dirección origen destino
Fwd Packets/s	Numero de paquetes enviados por segundos desde origen a destino
Min Packet Length	Longitud mínima de un paquete
Max Packet Length	Longitud máxima de un paquete
Packet Length Std	Desviación estándar de la longitud de un paquete
ACK Flag Count	Numero de paquetes con ACK
Average Packet Size	Tamaño promedio del paquete
Fwd Header Length.1	Longitud de la cabecera del paquete en dirección origen destino
Subflow Fwd Bytes	Numero promedio de bytes en un subflujo en la dirección origen destino
Init_Win_bytes_forward	Total de bytes enviados en una ventana inicial en la dirección origen destino
min_seg_size_forward	Tamaño mínimo del segmento observado en la dirección origen destino.

Para balancear y evitar el sobre entrenamiento de un solo tipo de ataque DoS, se unieron todos los archivos en un solo archivo temporal y luego se mezclaron los diferentes ataques. En la figura 20 se muestra una parte del archivo temporal creado que contiene todos los registros combinados; debido a que la cantidad de registros por ataque DoS y NoMalignos(BENIGN) deben ser proporcionales y organizados al azar. Además, la cantidad de registros NoMalignos en todo el Dataset es de 22.000, entonces para mantener la proporcionalidad se deben incluir máximo la misma cantidad de registros con diferentes ataques.

32323055,00	0,00	32323055,00	32323055	BENIGN
45056008,00	0,00	45056008,00	45056008	NTP
0,00	0,00	0,00	0	DNS
0,00	0,00	0,00	0	LDAP
1447652341254610,00	480131962459258,00	1592417574900130,00	5496094	MSSQL
0,00	0,00	0,00	0	NETBIOS
1592417576272470,00	137902981120463,00	1592417582034130,00	1,5924E+15	SNMP
1572997853628360,00	175853102919672,00	1592417581963870,00	5474628	SSDP
1592417580716410,00	166541775122856,00	1592417582543750,00	1,5924E+15	UDP
1590632358027130,00	53318084015570.3	1592417581919750,00	5480778	UDP-LAG
1573231826133590,00	174790536674286,00	1592417582030740,00	5410775	SYN
1580533867384450,00	137563926354305,00	1592417582051880,00	5447220	TFTP
1498745959734530,00	386217992200416,00	1592417582032470,00	5516838	WEBDDoS

Figura 20. Fragmento del Archivo temporal de datos**.**

3.2.3 Preprocesamiento

En este paso se eliminan las variables que no hicieron parte de los features de entrenamiento. En este paso mediante código, se eliminan las variables que no hacen parte de las 22 más relevantes como se mencionó en el paso anterior. Además, el algoritmo de entrenamiento requiere que los valores labels (valores de salida de la red neuronal) deban ser numéricos, y en el Dataset estos se encuentran en cadena de caracteres, por lo tanto, se asocia un número a cada label del ataque y al tráfico NoMaligno (BENIGN), como se ilustra en la tabla 31.

Tabla 31. Labels de la capa de salida**.**

Número	Label
0	BENIGN
1	NTP
2	DNS
3	LDAP
4	MSSQL
5	NETBIOS
6	SNMP
7	SSDP
8	UDP
9	UDP-LAG
10	SYN
11	TFTP
12	WebDDoS

3.2.4 Normalización

Las redes neuronales funcionan mejor cuando los datos que reciben están normalizados, es decir, que estén restringidos entre un rango como -1 y 1. Las razones de esto, es que las redes neuronales se entrenan usando el gradient descent y sus funciones de activación generalmente tienen un rango activo entre -1 y 1, lo cual mejora considerablemente el rendimiento de entrenamiento. Para este caso, el objetivo de este paso es restringir los 22 features a un rango para mejorar el rendimiento. La siguiente figura muestra el código en java utilizado para normalizar los datos.

```
//Proceso de normalizacion
DataNormalization normalizer = new NormalizerStandardize();
normalizer.fit(trainingData);
normalizer.transform(trainingData);
```

Figura 21. Instrucciones de java para normalización.

3.2.5 División del Dataset

El Dataset se dividió en 2; una parte fue utilizada durante el entrenamiento y la otra parte fue utilizada en las pruebas. El porcentaje asignado para entrenamiento es del 90% de los datos y el otro 10% a pruebas. Aunque la literatura recomienda utilizar un pocentaje de entrenamiento de 70% y pruebas de 30%, se tomo la decisión de utilizar mayor cantidad de datos para el entrenamiento, debido a la mejora que se obtuvo en el rendimiento. Esta división se realiza mediante código java con la siguiente instrucción.

```
//Split Data Test and Training  SplitTestAndTrain testAndTrain = allData.splitTestAndTrain(0.9);
```

3.2.6 Entrenamiento

Para el entrenamiento se utilizaron varias configuraciones de redes neuronales con el objetivo de buscar el mayor rendimiento en la métrica de Accuracy. En total, se crearon 48 modelos de clasificación, los cuales pueden consultarte en el Anexo B del trabajo. A continuación, se describe la configuración de los 2 modelos que obtuvieron el mejor Accuracy en su fase de entrenamiento; en la tabla 32 el modelo con 13 salidas de ataques DoS y la tabla 33 el modelo con 2 salidas que son Malignos y NoMalignos. La razón de cambiar la cantidad de salidas de 13 a 2, se debió a que al haber entrenado 28 modelos, superando las métricas obtenidas por el estudio del Instituto Canadiense para la Ciberseguridad, se realizó un prototipo que simulaba ataques DoS y Benignos para validar que tan bien se estaban clasificando paquetes desconocidos y el resultado no fue el esperado, debido a que el modelo siempre clasificaba los paquetes como el tipo de ataque NTP. Además, el entrenamiento con 13 salidas era un proceso demasiado costoso desde el punto de vista del consumo de recursos del sistema y dado que no se contaba con un equipo con mayor poder de computo se opto por reducirlas a 2 salidas. Por otro lado, desde el objetivo general del proyecto, lo relevante es que el sistema identifique un ataque de DoS y no tanto el tipo de ataque para poder tomar acciones y mitigar la vulnerabilidad del sistema de red.

Tabla 32. Modelo final con 13 salidas**.**

Modelo 22 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		110.000	
Tamaño Batch		10.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024

		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13
		Function	softmax

Tabla 33. Modelo final con 2 salidas**.**

Modelo 48 Propuesto			
Seed		7	
Epocas		200.000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256

		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2

3.2.7 Pruebas

Finalizada la fase de entrenamiento, el modelo entrenado es almacenado en un archivo .bin. Para la fase de pruebas, este modelo es cargado y luego se empieza a evaluar el 10% de los registros de pruebas uno a uno, que al final mostrará las medidas de rendimiento de Deep Learning, que son: Accuracy, Precision, Recall y F1 Score.

3.2.8 Comparación de rendimiento

Al terminar las pruebas con cada modelo, se tabularon los resultados y se hizo una comparación con los otros modelos, en la cual se evidencia las siguientes variables: features, labels, Accuracy, precision, Recall y F1 Score obtenido por cada modelo entrenado. La tabla 34 muestra el formato para la comparación del rendimiento.

Tabla 34. Formato para registro de rendimientos**.**

Nro. Modelo	Features	Labels	Accuracy	Precision	Recall	F1 Score
1	-	-	-	-	-	-
2	-	-	-	-	-	-

3.3 Fase 3: Elaborar un software integrando modelos

Esta sección describe el proceso para la elaboración del software para identificar y prevenir ataques de denegación de servicio web.

3.3.1 Análisis de requerimientos

Los requerimientos más relevantes para el desarrollo del sistema preventivo son:

1. El sistema debe capturar paquetes de una interfaz de red.
2. El sistema debe ser un aplicativo web para que pueda ser visualizado desde cualquier dispositivo.
3. El sistema debe tener un sistema de logueo para que solo personal autorizado pueda acceder a él.
4. El sistema debe clasificar paquetes de red en 2 tipos: Maligno y NoMaligno.
5. El sistema debe mostrar en tiempo real la clasificación de paquetes.

6. El sistema debe mostrar: el tiempo, la IP de origen, la IP destino, el puerto de origen, el puerto de destino y protocolo de los paquetes capturados.
7. El sistema debe tener un botón para comenzar y detener la captura de paquetes.
8. El sistema debe tener un botón para volver a iniciar el proceso de captura de paquetes(reiniciar).
9. El sistema debe almacenar todo el tráfico de red en un archivo. pcap para su posterior análisis.
10. El sistema debe tener una base de datos para almacenar los usuarios y además los paquetes clasificados.

3.3.2 Diseño MER y Arquitectura

- **Diseño Modelo Entidad Relación**

La siguiente figura describe las tablas creadas para almacenar la información del software

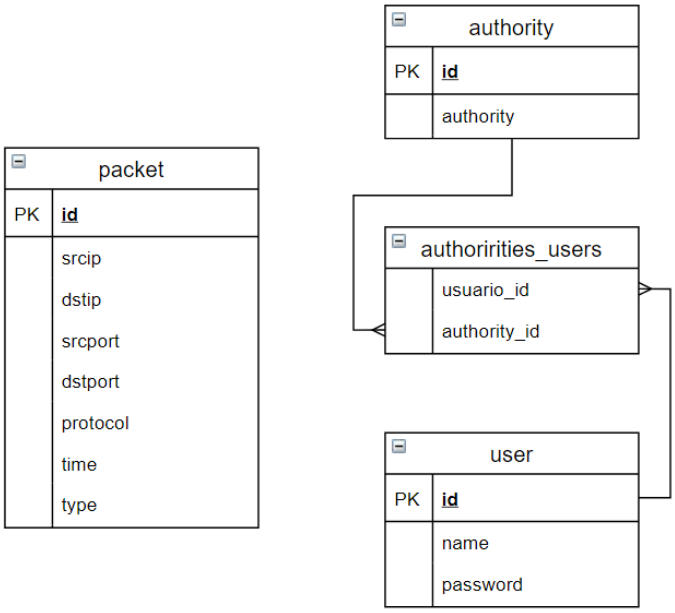


Figura 22. Modelo Entidad Relación Dique**.**

- **Packet**: Entidad que almacena algunos campos del paquete de red para mostrarlo en la gráfica del aplicativo web. Contiene los campos id, IP origen, IP destino, puerto de origen, puerto destino, número de protocolo según la IANA, hora de registro del paquete y el tipo (Maligno o NoMaligno).
- **Authority**: Entidad que almacena los roles de la aplicación. En este caso solo contiene el rol de admin.

- **Authoriries_users**: Entidad que almacena la relación entre users y authority. Aquí se define cual rol tiene cada usuario.
- **User**: Entidad que almacena el nombre y contraseña encriptada para el login de usuarios.

- **Arquitectura**

La figura 23 describe el modelo UML del sistema, en el cual se utilizó la arquitectura MVC (Modelo, Vista, Controlador). Todos los eventos ejecutados por el usuario en las diferentes paginas HTML están en la capa Vista, que son procesadas por el lenguaje javascript utilizando una librería conocida como Ajax, la cual permite realizar peticiones REST. Estas peticiones llegan a la capa Controlador, los cuales son un conjunto de servicios creados con Java, y esta capa a su vez llama a los objetos del Modelo, los cuales tienen los atributos y propiedades necesarias para ser utilizados en los métodos de la capa controlador, quien tiene toda la lógica para capturar, analizar y clasificar los paquetes de red.

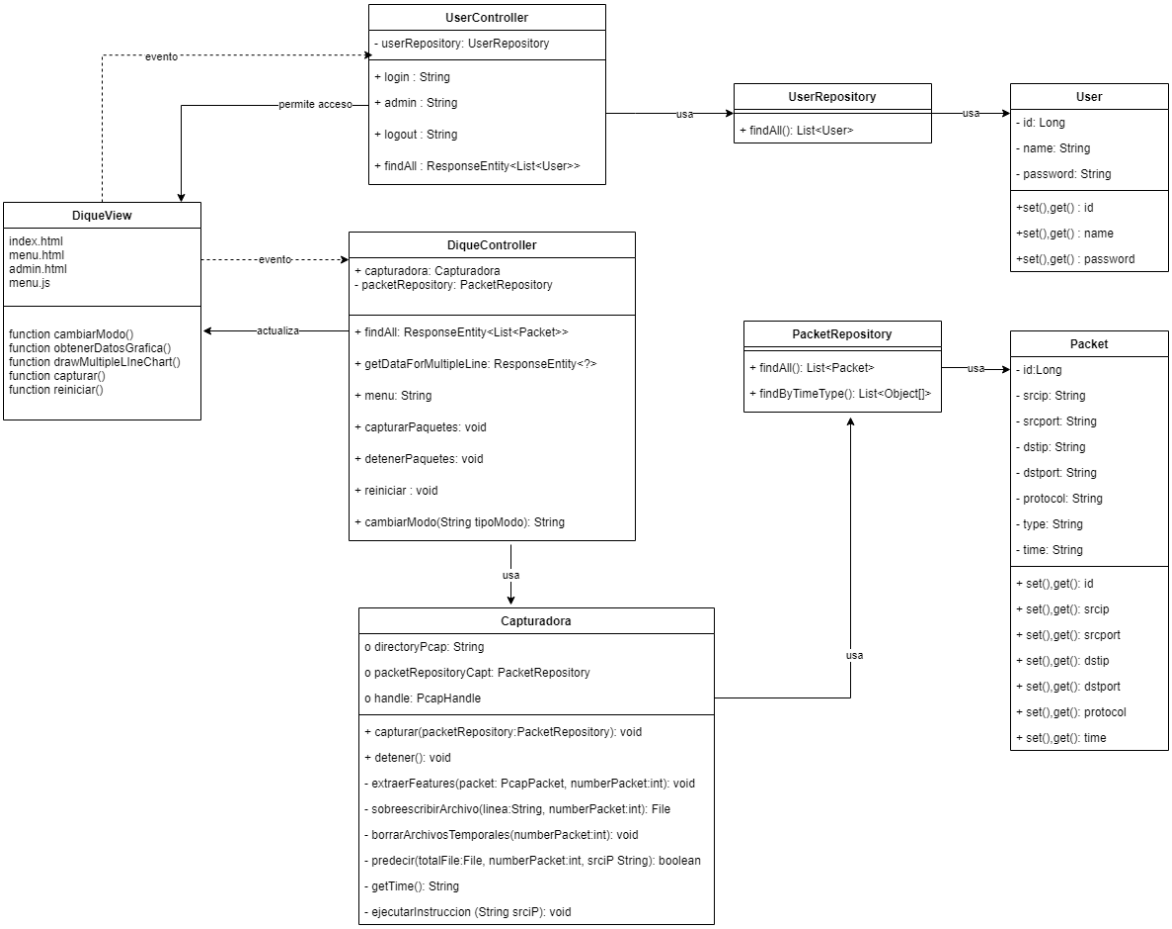


Figura 23. UML Dique**.**

3.3.3 Integración de librerías

Para el desarrollo del sistema preventivo se utilizó el IDE Eclipse con el framework spring boot. El framework spring boot, importa las librerías de java necesarias para la creación de un sitio web incluyendo la conexión a la base de datos. Estas librerías son:

- **Maven**: El cual es una herramienta para la gestión de librerías, con el objetivo de facilitar al programador la importación de librerías.
- **DL4J**: Librería de Java para entrenar modelos con algoritmos de Deep Learning. Proporciona todo un conjunto de métodos para facilitar al programador la construcción de las redes de entrenamiento.
- **PCAP4J**: Librería de Java que permite capturar, analizar, guardar y crear paquetes de una interfaz de red.
- **CICFlow**: Librería creada por el Instituto Canadiense para la ciberseguridad para extraer información de paquetes de un archivo pcap.
- **Bootstrap**: Librería de CSS para crear interfaces gráficas de usuario más amigables.
- **JQuery**: Librería creada con javascript para realizar los llamados a web Services utilizando Ajax y para ejecutar eventos realizados por el usuario.

Todo esto, con la intención de desplegar el aplicativo web en cualquier plataforma sin importar su sistema operativo. Además, el objetivo final es integrar varias librerías de Deep Learning, sniffer e interfaz gráfica de usuario en un software final, y java proporciona todas estas. La figura 24 describe la integración de todas las tecnologías para crear el sistema preventivo.

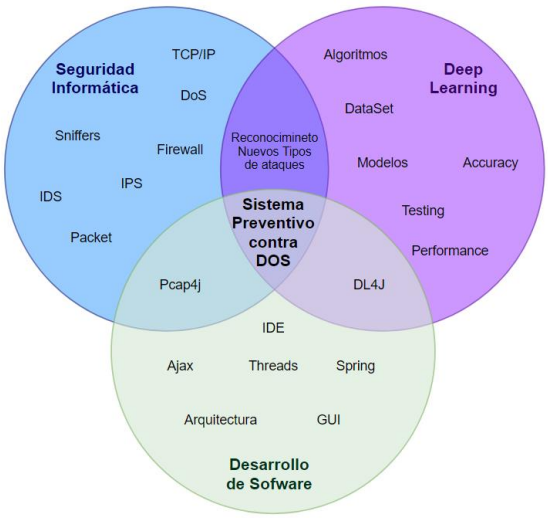


Figura 24. Tecnologías y librerías sistema preventivo**.**

3.3.4 Desarrollo de software

Los pasos para el desarrollo del software fueron los siguientes:

- **Diseño del sistema**

La siguiente figura muestra el diseño de la interfaz gráfica del sistema preventivo, que fue guía para su desarrollo.

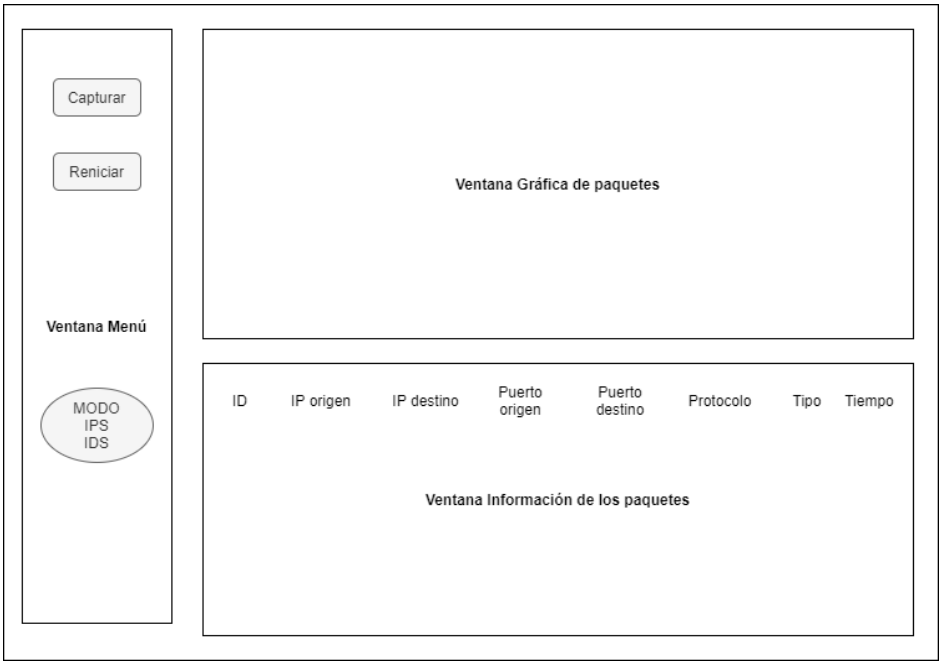


Figura 25. Diseño Interfaz gráfica del sistema preventivo**.**

La interfaz gráfica se planeó con 3 componentes principales:

- **Ventana de Menú**: Lugar donde se encuentra los botones para que el usuario pueda controlar la aplicación. En este se encuentra el botón capturar, parar, reiniciar y cambiar el modo IDS o IPS.
- **Ventana de Información de los paquetes**: Una gráfica de 2 dimensiones donde se muestra en tiempo real la cantidad de paquetes que el sistema clasifica. En el eje horizontal se encuentra el tiempo en segundos y en el eje vertical se encuentra la cantidad de paquetes en ese segundo.
- **Ventana de Información de los paquetes:** Una tabla que muestra la IP origen, IP destino, puerto origen, puerto destino, protocolo, tipo y hora en la que el paquete fue capturado.

- **Codificación**

Con los requerimientos levantados, arquitectura, diseño e integración de librerías se creó el sistema con java utilizando el IDE Eclipse. La figura 26 muestra parte del código del proyecto en el IDE eclipse.

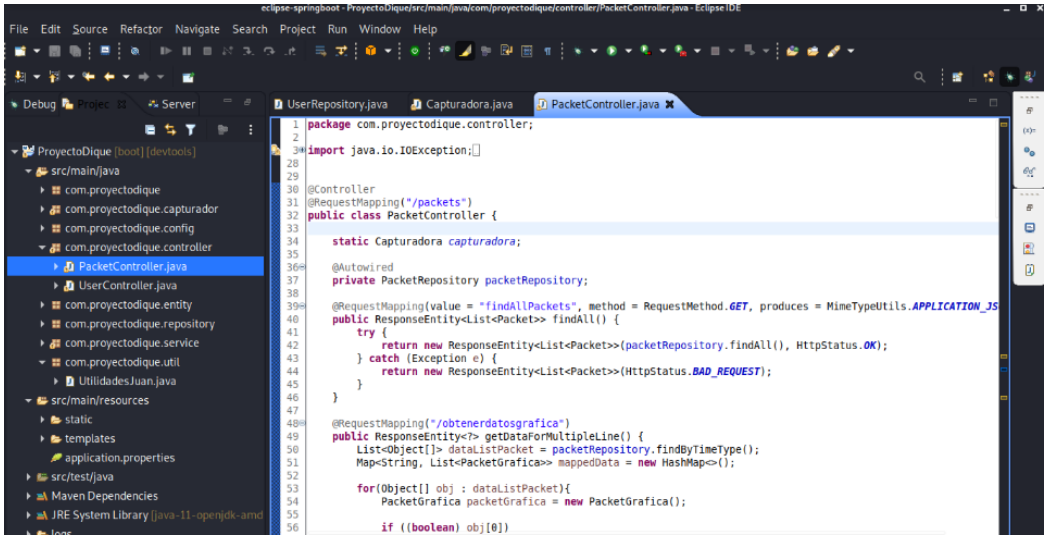


Figura 26. Codificación del sistema preventivo en IDE eclipse**.**

3.3.5 Pruebas

Una vez codificado el sistema se validó que cada uno de los requerimientos haya sido desarrollado y que funcione correctamente, incluyendo pruebas unitarias. La tabla 35 muestra el cumplimiento de los requerimientos de Dique.

Tabla 35. Requerimientos Dique**.**

Requerimientos	Cumplido
1. El sistema debe capturar paquetes de una interfaz de red.	✓
2. El sistema debe ser un aplicativo web para que pueda ser visualizado desde cualquier dispositivo.	✓
3. El sistema debe tener un sistema de logueo para que solo personal autorizado pueda acceder a él.	✓
4. El sistema debe clasificar paquetes de red en 2 tipos: Maligno y NoMaligno.	✓
5. El sistema debe mostrar en tiempo real la clasificación de paquetes.	✓
6. El sistema debe mostrar: el tiempo, la IP de origen, la IP destino, el puerto de origen, el puerto de destino y protocolo de los paquetes capturados.	✓
7. El sistema debe tener un botón para comenzar y detener la captura de paquetes.	✓
8. El sistema debe tener un botón para volver a iniciar el proceso de captura de paquetes(reiniciar).	✓
9. El sistema debe almacenar todo el tráfico de red en un archivo. pcap para su posterior análisis.	✓
10. El sistema debe tener una base de datos para almacenar los usuarios y además los paquetes clasificados.	✓

La siguiente figura muestra el cubrimiento del código realizado en las pruebas unitarias con el plugin eclEmma de eclipse y fueron satisfactorias; inicialmente, el plugin reportó algunos problemas estructurales en algunos módulos del código de Dique, los cuales se corrigieron para mejorar su funcionamiento.

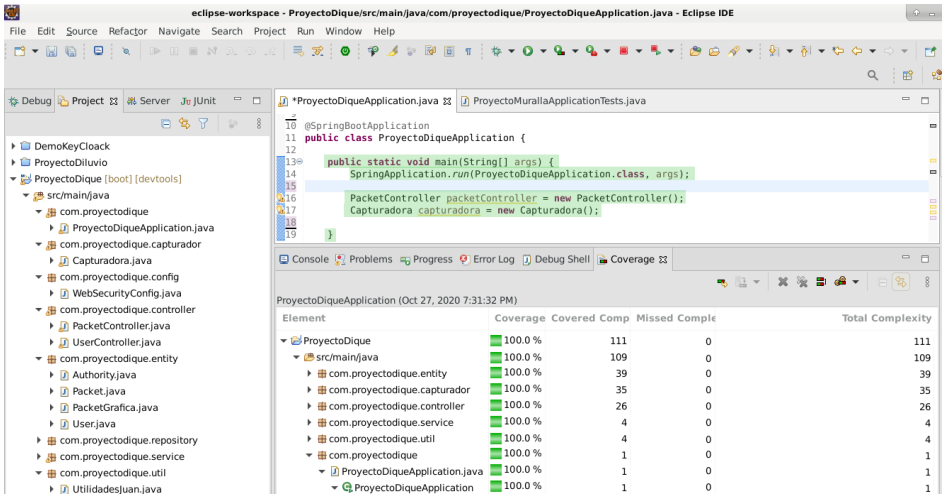


Figura 27. Resultado pruebas unitarias Dique**.**

3.4 Fase 4: Verificar el funcionamiento

Esta sección describe el proceso para verificar el funcionamiento del sistema preventivo elaborado (Dique), mediante la construcción de otro aplicativo, llamado Diluvio, que realiza ataques al sistema preventivo.

3.4.1 Análisis de requerimientos

Los requerimientos para la creación de Diluvio, está basado en diferentes ataques DoS los cuales pertenecen a las cinco categorías que se muestra en la figura 8. Por otro lado, también se consideró que Diluvio posea ataques DoS que no hacen parte del Dataset, pues lo que se pretende es que el sistema pueda clasificar y prevenir ataques DoS desconocidos. Es por esto que Diluvio está conformado por siete tipos de ataques DoS. A continuación, se describe los requerimientos más relevantes para el desarrollo del sistema Diluvio:

1. El sistema debe tener múltiples opciones de ataques DoS, donde el usuario puede seleccionar un tipo específico para realizar el ataque.
2. Debe permitir ingresar la IP hacia dónde se va a dirigir el ataque.
3. Realizar ataques DoS de la categoria Reflection con paquetes TCP/UDP.
4. Permitir ejecutar ataques DoS de la categoria Exploit con paquetes TCP.
5. Efectuar ataques DoS de la categoria Exploit con paquetes UDP.
6. Debe permitir realizar ataques DoS de la categoria Reflection con paquetes UDP.
7. Permitir ejecutar ataques DoS de la categoria Reflection con paquetes TCP.

8. Realizar dos tipos de ataques DoS con los cuales no fue entrenado el modelo.
9. Enviar paquetes NoMalignos.

3.4.2 Diseño MER y Arquitectura

- **Diseño Modelo Entidad Relación**

El sistema que realiza ataques no requirió almacenar datos por lo tanto no posee ningún diagrama para el manejo de la información persistente.

- **Arquitectura**

La figura 28 describe el diagrama UML del sistema Diluvio.

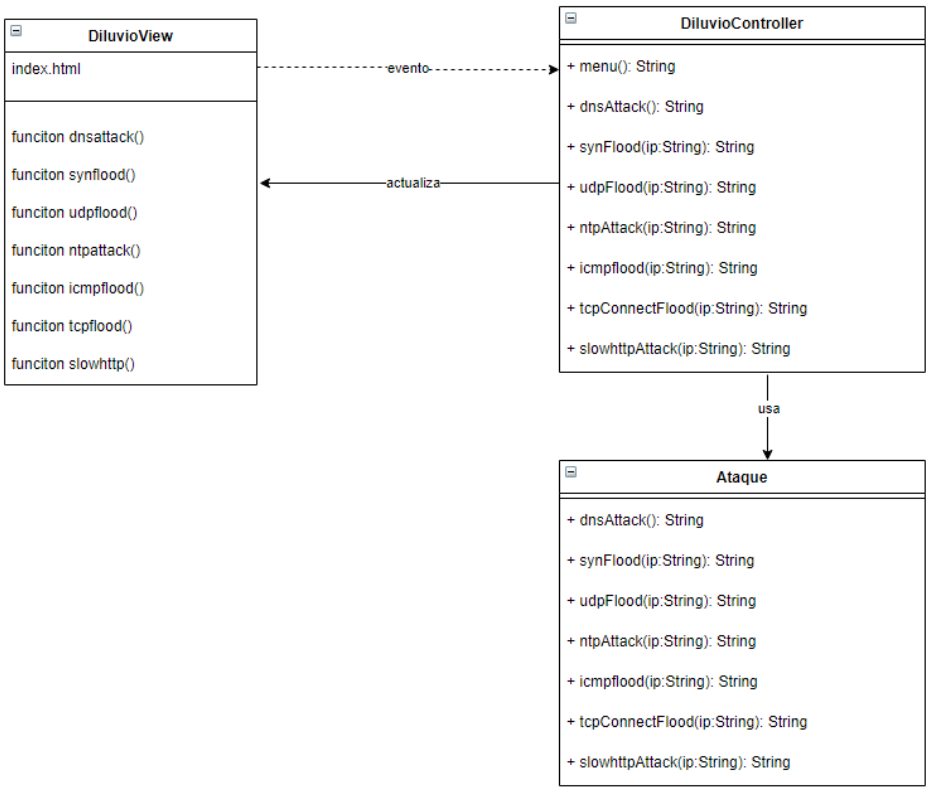


Figura 28. UML Diluvio**.**

La arquitectura del sistema Diluvio no requiere la capa Modelo del MVC. Contiene una capa de Vista que contiene los botones de los diferentes tipos de ataques DoS y la capa Controladora que recibe los llamados de los servicios para ejecutar el ataque DoS seleccionado. La clase Ataque es llamada por la capa controladora que contiene los comandos y scripts con el ataque seleccionado.

3.4.3 Librerías

Para el desarrollo del sistema ofensivo se utilizó el IDE Eclipse con el framework spring boot.

- **Maven**: El cual es una herramienta para la gestión de librerías, con el objetivo de facilitar al programador la importación de librerías.
- **Bootstrap**: Librería de CSS para crear interfaces gráficas de usuario amigables.
- **JQuery**: Librería creada con javascript para realizar los llamados a los web Services utilizando Ajax y para ejecutar eventos realizados por el usuario.

3.4.4 Desarrollo de software

- **Diseño del sistema:** La figura 29 muestra el diseño el sistema Diluvio, que fue guía para su desarrollo.

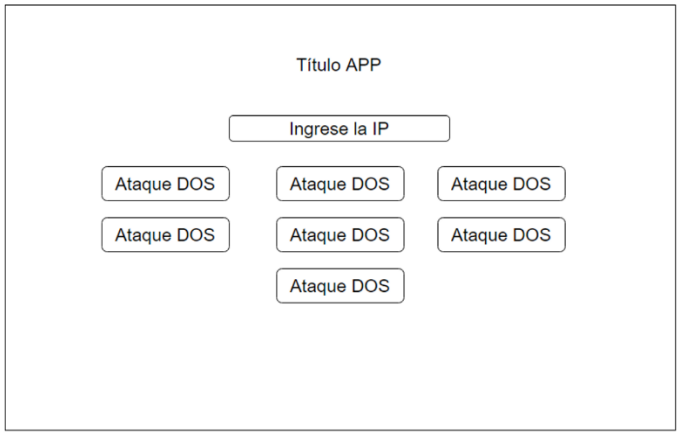


Figura 29. Diseño Sistema Ofensivo.

- **Codificación**: Con los requerimientos levantados, arquitectura, diseño e integración de librerías se procedió a crear el sistema Diluvio. La figura 30 muestra la codificación del sistema ofensivo en el IDE eclipse.

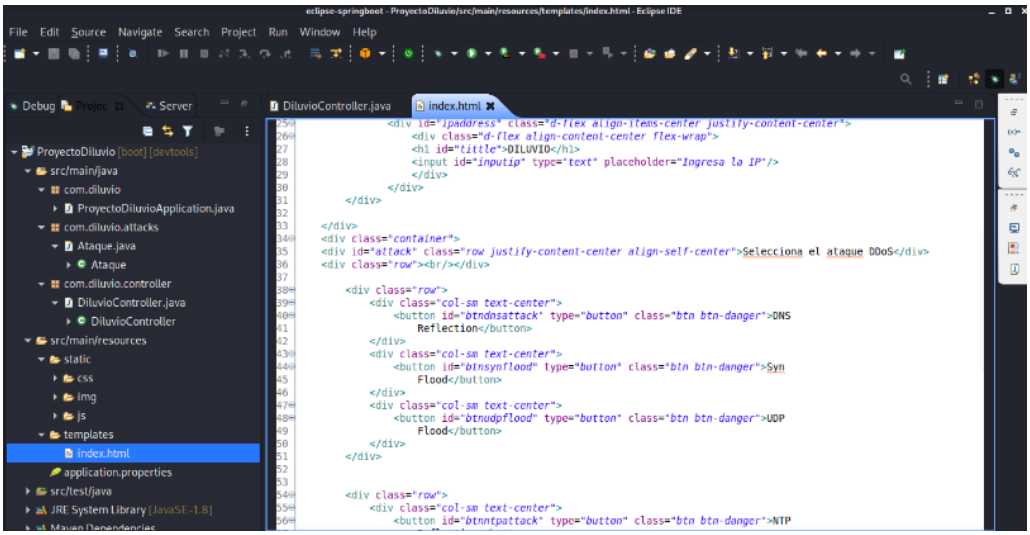


Figura 30. Codificación sistema ofensivo en IDE Eclipse**.**

A continuación, se describe la implementación de cada uno de los 7 ataques DoS que se pueden realizar con el sistema Diluvio:

- **DoS tipo Reflection con paquetes UDP:** Para realizar este tipo de ataque se utiliza la librería skapy de Python que permite manipular y crear paquetes de red. El ataque realizado consiste en realizar una consulta a un servidor DNS con ip suplantada, con el objetivo de dar respuesta a la máquina victima. La siguiente figura muestra el código en Python almacenado en un archivo dns_amplification.py, el cual es llamado luego desde java.

```
for i in range(0,len(query_type)):
    for j in range(0, len(dns_destination)):
        packet_number += 1

        # Craft the DNS query packet with scapy
        packet = IP(src=dns_source, dst=dns_destination[j], ttl=time_to_live) / UDP() / DNS(rd=1,
        qd=DNSQR(qname=query_name, qtype=query_type[i]))

        # Sending the packet
        try:
            query = sr1(packet,iface=interface,verbose=False, timeout=8)
            print("Packet #{} sent!".format(packet_number))
        except:
            print("Error sending packet #{}".format(packet_number))

        # Creating dictionary with received information
        try:
            result_dict = {
                'dns_destination':dns_destination[j],
                'query_type':query_type[i],
                'query_size':len(packet),
                'response_size':len(query),
                'amplification_factor': ( len(query) / len(packet) ),
                'packet_number':packet_number
            }
            results.append(result_dict)
        except:
            pass
```

Figura 31. Script del ataque Reflection con paquetes UDP.

- **DoS tipo Syn Flood:** Para realizar este tipo de ataque se utiliza la herramienta hping3, la cual es una herramienta de Linux que a diferencia de la herramienta ping que solo permite enviar paquetes ICMP, este envía paquetes TCP y UDP, ideal para realizar pruebas de seguridad como ataques DoS. Hping3 permite colocar en sus parámetros la cantidad de paquetes a enviar, el puerto y la bandera, que para este caso se activa el parámetro con la bandera SYN. Luego el comando es llamado desde java utilizando el método exec() de la librería Runtime. El comando implementado es *hping3 -p 80 -c 100 -S –fast.*
- **DoS tipo flood con paquetes UDP:** Igual que el ataque SYN Flood, este ataque puede ser realizado con la herramienta hping3, colocando como parámetro el número 2, que indica que los paquetes lanzados son de tipo UDP. El comando implementado con el método exec () de lal ibreria Runtime es *hping3 -c 100 -2 –faster.*
- **DoS tipo reflection con paquetes NTP:** Para realizar este tipo de ataque se utiliza la librería skapy de Python. El ataque realizado consiste en realizar una consulta a un servidor NTP con IP suplantada, con el objetivo de dar respuesta a la máquina victima. La siguiente figura muestra el código en Python almacenado en un archivo ntpattack.py, el cual es llamado luego desde java.

```
    #Magic Packet aka NTP v2 Monlist Packet
    data = "\x17\x00\x03\x2a" + "\x00" * 1

    #Hold the threads
    threads = []
    print "Starting to flood: "+ target + " using NTP list: " + ntpserverfile + " With " + str(numberthreads) + " threads"
    #print "Use CTRL+C to stop attack"

    #Thread spawner
    for n in range(numberthreads):
        thread = threading.Thread(target=deny)
        thread.daemon = True
        thread.start()

        threads.append(thread)

    #In progress!
    print "Sending..."
# Script ends here
    #Keep alive so ctrl+c still kills all them threads
    time.sleep(1)

except KeyboardInterrupt:
    print("Script Stopped [ctrl + c]... Shutting down")
```

Figura 32. Script del ataque NTP.

- **DoS tipo flood con paquetes TCP:** Para realizar este tipo de ataque se utiliza la herramienta nping, que al igual que la herramienta hping3, permite generar paquetes de red con diferentes protocolos, ideal para realizar pruebas de seguridad como ataques DoS. nping permite colocar como parámetro el tipo de paquete, la cantidad de paquetes a enviar y la tasa de envio por segundo. Luego el comando es llamado desde java utilizando el método exec() de la librería Runtime. El comando implementado es *nping --tcp-connect -rate=10 -c 100*.
- **DoS tipo flood con paquetes ICMP:** Este ataque puede ser realizado con la herramienta hping3, colocando como parámetro el número 1, que indica que los paquetes lanzados son de tipo ICMP. El comando implementado con el método exec () de la libreria Runtime es *hping3 -c 10 -1 -C 17*
- **DoS tipo Slow HTTP:** Este tipo de ataque puede ser lanzado con la herramienta slowhttppost de linux, la cual permite simular ataques de

denegación de sevicio en la capa de aplicación, de esta manera se pueden enviar paquetes a una tasa de transferencia baja. El comando implementado con el método exec () de la libreria Runtime de java es *slowhttptest -c 20 -H -i 1 -r 2 –u,* donde c representa la cantidad de paquetes, H el tipo de ataque slowloris, i el intervalo de segundos, r las conexión por segundos y u la dirección del servidor.

3.4.5 Pruebas

Una vez codificado el sistema se validó que cada uno de los requerimientos haya sido desarrollado y que funcione correctamente, incluyendo pruebas unitarias. La tabla 36 muestra la lista de requerimientos del sistema ofensivo.

Tabla 36. Lista de Requerimientos Diluvio**.**

Requerimientos	Cumplido
1. El sistema debe tener múltiples opciones de ataques DoS, donde el usuario puede seleccionar un tipo especifico para realizar el ataque.	✓
2. El sistema debe permitir ingresar la IP hacia dónde se va a dirigir el ataque.	✓
3. El sistema debe permitir realizar ataques DoS tipo reflection con paquetes UDP.	✓
4. El sistema debe permitir realizar ataques DoS tipo Syn Flood.	✓
5. El sistema debe permitir realizar ataques DoS tipo flood con paquetes UDP.	✓
6. El sistema debe permitir realizar ataques DoS tipo reflection con paquetes NTP.	✓
7. El sistema debe permitir realizar ataques DoS tipo flood con paquetes TCP.	✓
8. El sistema debe permitir realizar ataques DoS tipo flood con paquetes ICMP.	✓
9. El sistema debe permitir realizar ataques DoS tipo Slow HTTP.	✓
10. El sistema debe permitir enviar paquetes NoMalignos.	✓

La siguiente figura muestra el cubrimiento del código realizado en las pruebas unitarias con el plugin eclEmma de eclipse y fueron satisfactorias; inicialmente, el plugin reportó algunos problemas estructurales en algunos módulos del código de Diluvio, los cuales se corrigieron para mejorar su funcionamiento.

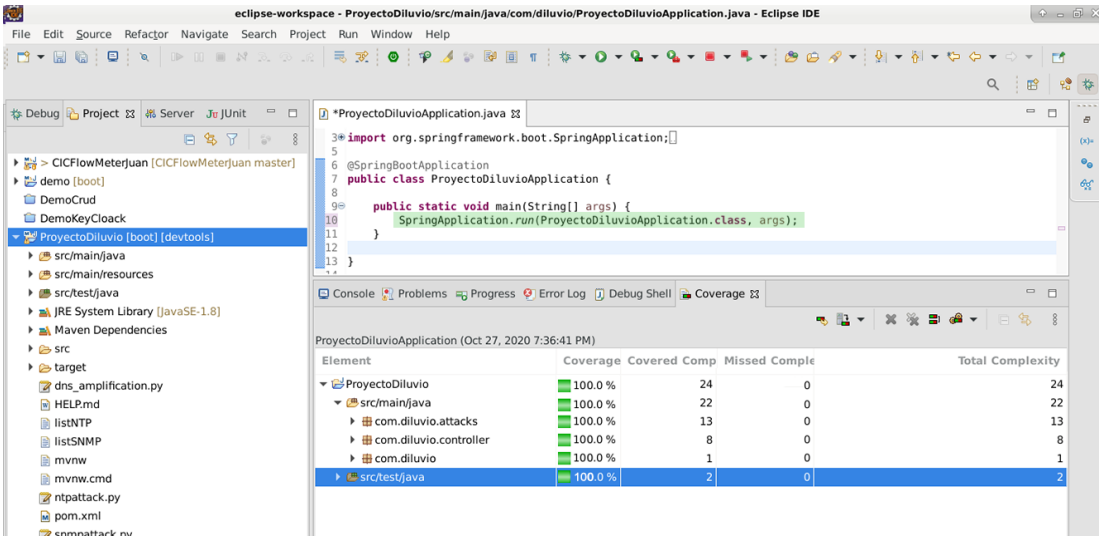


Figura 33. Resultados pruebas unitarias Diluvio**.**

4. Resultados

Esta sección describe los resultados obtenidos en cada una de las fases, cumpliendo así los objetivos propuestos.

4.1 Fase 1: Selección de los algoritmos

De acuerdo con la revisión sistemática de la literatura de los algoritmos de Deep Learning para la detección de ataques de denegación de servicio, se obtuvo la siguiente tabla donde se condensa en una matriz, las referencias encontradas en las publicaciones científicas sobre los algoritmos de Deep Learning y los Datasets utilizados para el entrenamiento de los modelos de clasificación. En esta, se muestra el Accuracy obtenido en el entrenamiento del algoritmo utilizado que corresponde a las filas y el Dataset empleado que corresponde a las columnas, además se incluye la fuente de la cual se obtuvo.

Tabla 37. Matriz de algoritmos vs Datasets.

Algoritmos \ Dataset	MawiLab 2017	ISCX 2012	NSLKDD	UNSW NB15	CICIDS2017	CNTC 2017	CSIC 2010	DARPA 1998	Netflow	KDD CUP 99	Darpa 2009 DDoS	Yahoo S5 Web scope	Dataset propio SSE
Recurrent Neural Network	100 [24]		98.8 [2]							99.49 [33]			
Convolutional Neural Network	100 [24]					94,11 [29]		99,36 [29]	99.41 [2]				
Stacked Recurrent Neural Network	100 [24]												
Long short Term Memory		98,88 [25]	99,51 [28]			95,12 [29]	96,13 [29]	99,98 [29]		99.8 [2]			
Deep Radial Intelligence			99,69 [26]	96,15 [26]									
Genetic Algorith(Simulated Annealing Algorithm (SAA)					99.92 [27]								
VCNN/FNN			99.2 [31]										
Deep Belief Network			97.5 [2]						97.6 [2]	93.49 [2]			
Gradient Recurrent Unit									84.15 [2]				
Deep Feed Forward			93.78 [37]								99.63 [34]		
(Bidireccional Gradient Recurrent Unit)BGRU + MLP			99.24 [36]							99.84 [36]			
Restricted Boltzman Machine			73.23 [38]							92,12 [42]			
CNN + LSTM												98.6 [41]	
AutoEncoders										91,86 [42]			98,99 [42]
Deep Defense(CNN, RNN, LSTM, GRU)		97,60 [42]											

Para seleccionar los algoritmos más adecuados se tuvieron en cuenta los 4 criterios definidos en la metodología, los cuales se encuentran numerados en la tabla 38. Si un algoritmo de Deep Learning cumple con los 4 criterios de selección, es considerado un algoritmo adecuado para el entrenamiento de los modelos.

Tabla 38. Criterios de selección.

Nro.	Criterio de selección
1	El algoritmo se encuentra disponible en la librería de Java Deep Learning 4J.
2	La configuración de entrenamiento del modelo que se utilizó con el algoritmo está disponible.
3	El Dataset utilizado junto al algoritmo de DL para entrenar el modelo, contiene paquetes de denegación de servicio.
4	El Accuracy del modelo entrenado con el algoritmo de Deep Learning es superior al 95%

La siguiente tabla muestra cuales algoritmos de Deep Learning, cumplieron con los criterios:

Tabla 39. Lista de algoritmos vs criterios.

Nro. Criterio / Algoritmo	1	2	3	4	Adecuado
Recurrent Neural Network	✓	✓	✓	✓	✓
Convolutional Neural Network	✓	X	✓	✓	
Stacked Recurrent Neural Network	X	X	X	X	
Long short Term Memory	✓	✓	✓	✓	✓
Deep Radial Intelligence	X	X	✓	✓	
Genetic Algorithm (IGA) y Simulated Annealing Algorithm (SAA)	X	X	✓	✓	
VCNN/FNN	X	X	✓	✓	
Deep Belief Network	✓	X	✓	✓	
Gradient Recurrent Unit	X	X	X	X	
Deep Feed Forward	✓	✓	✓	✓	✓
(Bidireccional Gradient Recurrent Unit)BGRU + MLP	X	X	✓	✓	
Restricted Boltzman Machine	X	X	✓	X	
CNN + LSTM	X	X	X	✓	
AutoEncoders	X	X	✓	✓	
Deep Defense(CNN, RNN, LSTM, GRU	X	X	✓	✓	

De la tabla anterior se concluyó que los algoritmos más adecuados para entrenar los modelos contienen los métodos de Deep Feed Forward, Recurrent Neural Network y Long Short Term Memory.

En las siguientes figuras se muestran las diferentes métricas y las matrices de confusión, resultado de haber probado el entrenamiento de los modelos Deep Feed Forward y el modelo combinado de Recurrent Neural Network y Long Short Term Memory con los algoritmos seleccionados y el Dataset NSLKDD como se reporta en las referencias bibliográficas.

- Algoritmo con red neuronal Deep Feed Forward multicapa

```
=======================Evaluation Metrics=======================
 # of classes:    2
 Accuracy:        0.9489
 Precision:       0.9353
 Recall:          0.9579
 F1 Score:        0.9464
Precision, recall & F1: reported for positive class (class 1 - "1") only


========================Confusion Matrix========================
    0    1
-----------
 3762  236 | 0 = 0
  150 3410 | 1 = 1

Confusion matrix format: Actual (rowClass) predicted as (columnClass) N times
================================================================
```

Figura 34. Resultado de entrenamiento Deep Feed forward.

- Algoritmo con red neuronal Recurrent Neural Network y LSTM multicapa

```
=======================Evaluation Metrics=======================
 # of classes:    2
 Accuracy:        0.9555
 Precision:       0.9824
 Recall:          0.9222
 F1 Score:        0.9513
Precision, recall & F1: reported for positive class (class 1 - "1") only


========================Confusion Matrix========================
    0    1
-----------
 3938   59 | 0 = 0
  277 3284 | 1 = 1

Confusion matrix format: Actual (rowClass) predicted as (columnClass) N times
================================================================
```

Figura 35. Resultado de entrenamiento Recurrent Neural Network.

La tabla 40 resume el rendimiento de los modelos entrenados.

Tabla 40. Rendimiento de algoritmos.

Algoritmo	Accuracy	Precision	Recall	F1Score
Deep Feed Forward	0.9489	0.9353	0.9579	0.9464
RNN y LSTM	0.9555	0.9824	0.9222	0.9513

4.2 Fase 2: Construcción modelos de clasificación

La siguiente tabla muestra el rendimiento obtenido por los diferentes modelos entrenados con los algoritmos seleccionados y el Dataset seleccionado, de acuerdo a la tabla de resultados definida en la metodología.

Tabla 41. Rendimiento de los modelos propuestos.

Modelo No.	Features	Labels	Accuracy	Precision	Recall	F1 Score
1	22	13	0,6557	0,7212	0,6359	0,6623
2	22	13	0,3225	0,2974	0,3101	0,3598
3	22	13	0,4256	0,4618	0,4059	0,4233
4	22	13	0,6106	0,6329	0,5894	0,558
5	22	13	0,5679	0,6369	0,5416	0,5097
6	22	13	0,5971	0,6781	0,5753	0,5906
7	22	13	0,595	0,569	0,5814	0,5279
8	22	13	0,5736	0,5494	0,5512	0,5694
9	22	13	0,5795	0,6228	0,5487	0,5192
10	22	13	0,6409	0,6509	0,6309	0,5739
11	22	13	0,3816	0,4722	0,3603	0,285
12	22	13	0,6529	0,6929	0,6538	0,6303
13	22	13	0,6456	0,7022	0,6458	0,6731
14	22	13	0,6455	0,6926	0,6622	0,6204
15	22	13	0,6524	0,7815	0,6397	0,6037
16	22	13	0,6566	0,69	0,6509	0,6865
17	22	13	0,2036	0,2832	0,1826	0,1988
18	22	13	0,2766	0,4002	0,3026	0,3154
19	22	13	0,3556	0,2527	0,3144	0,2556
20	22	13	0,3138	0,3696	0,327	0,3077
21	22	13	0,4807	0,4942	0,4455	0,4558
22	22	13	0,7293	0,8012	0,6721	0,7794
23	22	13	0,3123	0,2527	0,3144	0,2556
24	22	13	0,3949	0,4244	0,4025	0,3599
25	73	13	0,4215	0,3684	0,4182	0,3468
26	73	13	0,3151	0,3449	0,3268	0,2859
27	73	13	0,638	0,6854	0,6494	0,6022
28	73	13	0,6457	0,6891	0,6586	0,6143
29	22	2	0,9582	0,9711	0,9835	0,9772
30	22	2	0,96	1	0,9321	0,9649
31	22	2	0,9785	0,9889	0,9765	0,9836
32	22	2	0,9282	0,9997	0,9215	0,959
33	22	2	0,9266	0,9831	0,9348	0,9583

34	73	2	0,9823	0,9997	0,9807	0,9901
35	73	2	0,9847	0,9996	0,9841	0,9918
36	73	2	0,9064	0,9064	1	0,9509
37	73	2	0,9818	0,9982	0,9817	0,9899
38	73	2	0,9859	0,9972	0,9872	0,9922
39	73	2	0,9093	0,9944	0,9038	0,947
40	73	2	0,9889	0,9913	0,9965	0,9939
41	73	2	0,9814	0,9982	0,9812	0,9896
42	73	2	0,9842	0,9966	0,9859	0,9912
43	73	2	0,9722	0,9893	0,9798	0,9845
44	73	2	0,9022	0,9022	1	0,9486
45	73	2	0,9952	0,998	0,9914	0,9947
46	73	2	0,997	0,9975	0,996	0,9967
47	73	2	0,998	0,998	0,997	0,9975
48	73	2	0,9994	0,9995	0,999	0,9993

Los accuracy resaltados de la tabla anterior corresponden a los 2 modelos entrenados que obtuvieron el mejor Accuracy: el modelo No.22 con 13 labels de clasificación (12 ataques DoS y el NoMaligno) y el modelo No.48 con 2 labels de clasificación (Maligno y NoMaligno). Como se puede evidenciar en la tabla, se comenzó a entrenar modelos con los 22 features relevantes y luego se utilizó los 73 features en los entrenamientos, esto con el objetivo de intentar mejorar el desempeño de los modelos. Además, a partir del modelo No. 29 se disminuyó la cantidad de labels a 2, debido a que al evaluar los modelos de 13 labels, superando las métricas obtenidas por el estudio del Instituto Canadiense para la Ciberseguridad, se realizó un prototipo que simulaba ataques DoS y Benignos para validar que tan bien se estaban clasificando paquetes desconocidos y el resultado no fue el esperado, pues solo clasificaba los nuevos paquetes como un solo tipo (1=NTP). Por esta razón se decidió comenzar el entrenamiento de modelos con 2 salidas, dando como resultado un mejor desempeño tanto en la fase de pruebas, como en la clasificación de paquetes desconocidos. En las figuras 36 y 37, se ilustran el desempeño del modelo No.22 y No.48, que son los que reportaron mejor Accuracy.

```
=========================Evaluation Metrics=========================
 # of classes:     13
 Accuracy:         0.7293
 Precision:        0.8012        (4 classes excluded from average)
 Recall:           0.6721        (2 classes excluded from average)
 F1 Score:         0.7794        (4 classes excluded from average)
Precision, recall & F1: macro-averaged (equally weighted avg. of 13 c
```

Figura 36. Resultados de Evaluación Modelo No.22**.**

```
========================Evaluation Metrics========================
 # of classes:    2
 Accuracy:        0.9994
 Precision:       0.9995
 Recall:          0.9990
 F1 Score:        0.9993
Precision, recall & F1: reported for positive class (class 1 - "1") only
```

Figura 37. Resultados de Evaluación Modelo No.48**.**

Los algoritmos con los métodos de RNN y LSTM, no pudieron ser utilizados para entrenar los modelos. La razón de esto, es porque la librería de DL4J requeria que el Dataset ya estuviera organizado, sin ser necesario la etapa de preprocesamiento, es decir, hablando técnicamente, la clase de java SequenceRecordReaderDataSetIterator, la cual itera datos secuenciales que se necesita para utilizar los métodos de RNN y LSTM, no contenia el método para recibir data preprocesada, sino que requeria que el Dataset ya estuviera con los features correspondientes, normalizados, balanceados y con el label correspondiente. Esto se descartó entonces por limitaciones de tiempo, contrario al algoritmo con el método de Deep Feed Forward que todo el preprocesamiento se pudo realizar por código java sin ningún inconveniente.

Al finalizar el entrenamiento de cada modelo, son almacenados en un archivo .zip, el cual contiene 3 archivos: 2 archivos .bin, uno con la configuración del modelo y el otro con los pesos del modelo, el cual sirve para ser importado o cargado y utilizarlo en el sistema preventivo; y el archivo de configuración de la red neuronal en formato json. La siguiente figura muestra los archivos de cada modelo entrenado. Para el sistema preventivo solo se requiere importar el archivo .zip en el código java y este es evaluado contra nuevos datos desconocidos utilizando la librería de DL4J.

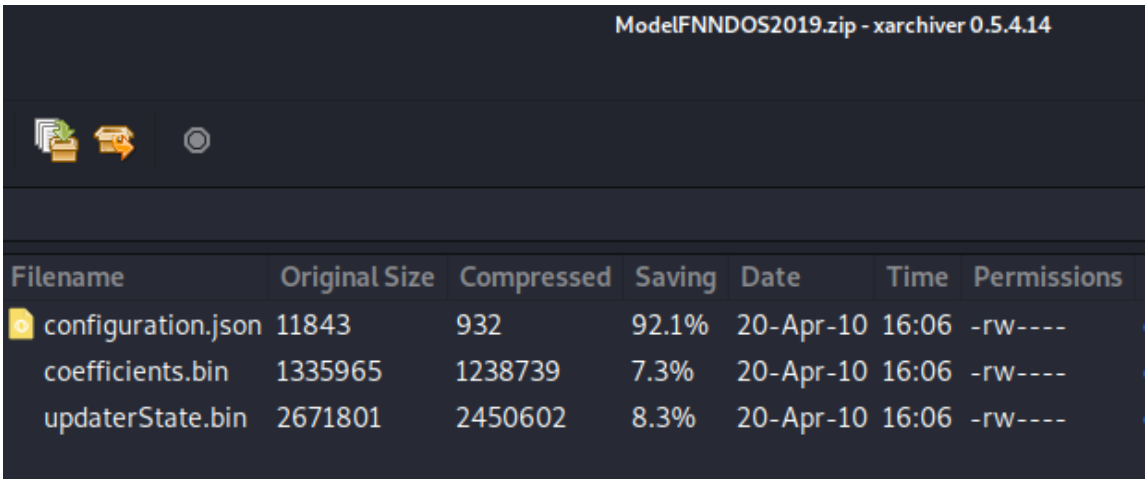


Figura 38. Archivo binario del modelo entrenado.

4.3 Fase 3: Elaborar un software integrando modelos

El sistema desarrollado consiste en un aplicativo web conocido como DIQUE. Este nombre es porque hace referencia a la función de los diques, que consiste en ser una barrera contra inundaciones, en este caso es una barrera para identificar y prevenir paquetes Malignos. Dique utiliza en su lógica el modelo entrenado de 2 labels, para clasificar los paquetes y mostrarlos en una gráfica dimensional. A continuación, se describen las diferentes secciones del sistema preventivo.

- **Página Login**

  La página login contiene el formulario mediante el cual se accede al aplicativo. Los usuarios registrados se encuentran en la base de datos con la contraseña encriptada. La siguiente figura muestra la página login de Dique.

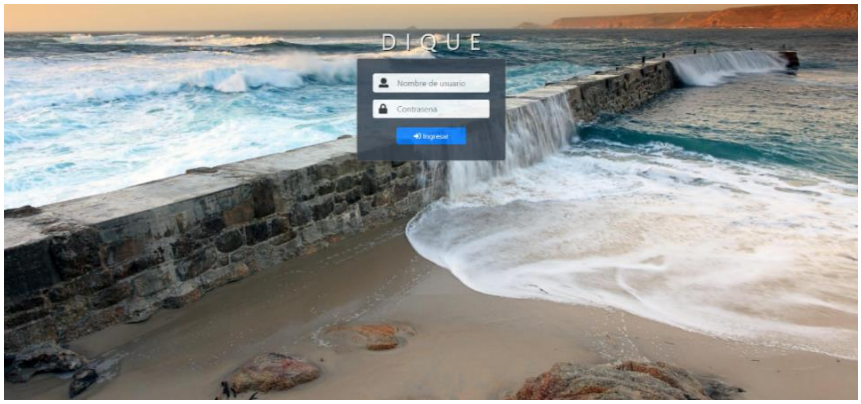


Figura 39. Página login Sistema Dique.

- **Página Principal**

  La página principal contiene toda la funcionalidad del sistema de detección y prevención contra ataques DoS. El ambiente de trabajo se encuentra dividido en 3 ventanas: Menú, gráfica e información. La siguiente figura muestra la página principal del sistema.

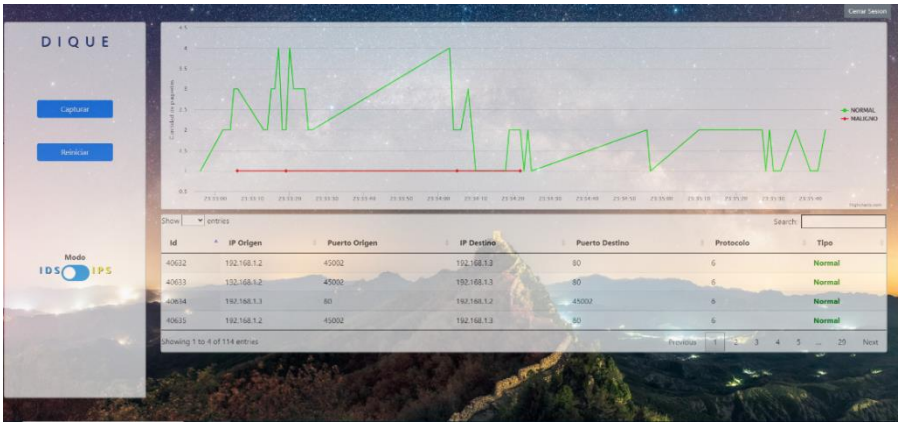


Figura 40. Página principal sistema Dique.

A continuación se describen las 3 ventanas de la página principal:

- **Ventana Menú**

  Esta sección contiene los botones desde los que se controla el aplicativo web. El primer botón que es *capturar,* permite comenzar la captura de paquetes, y este ejecuta toda la lógica interna que captura y clasifica los paquetes. Cuando se comienza a capturar, el botón se cambia por el valor *detener,* que, al dar clic sobre este, detiene la función de captura y clasificación de paquetes. El segundo botón *reiniciar* permite borrar los paquetes clasificados y prepararse para comenzar una nueva captura.

  El toogle Modo, permite cambiar el modo IDS e IPS como se evidencia en la Figura 41. En el modo de detección el sistema solo se encargará de clasificar paquetes mientras que en el modo IPS, aplicará reglas para prevenir que las fuentes de los paquetes Malignos sigan llegando.

Figura 41 Venana menú de Dique

- **Ventana Gráfica**

Sección que muestra en un plano bidimensional la cantidad de paquetes clasificados en segundos, como se evidencia en la figura 42. El eje vertical representa la cantidad de paquetes y el eje horizontal el tiempo en segundos en que se captura esa cantidad de paquetes. Los paquetes NoMalignos son mostrados en color verde y los Malignos en color rojo.

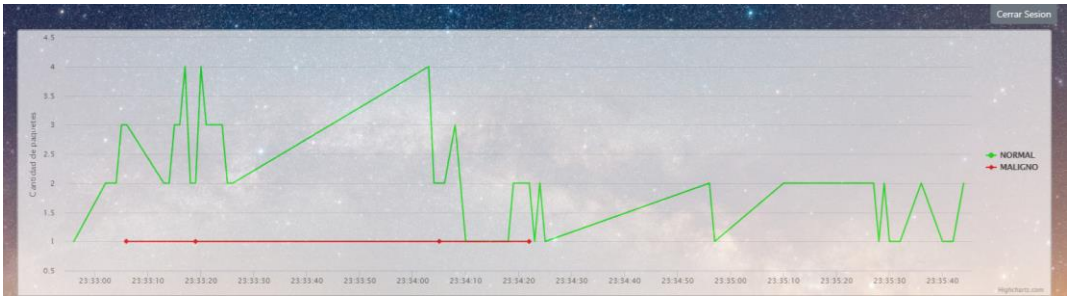


Figura 42. Sección gráfica de paquetes del sistema Dique.

- **Ventana Información**

En esta ventana se muestra una tabla con la información de los paquetes de red que son clasificados, como se muestra en la figura 43. La tabla posee las siguientes columnas:

- **ID**: Identificador único de cada paquete para ser almacenado en la base de datos.
- **IP Origen**: Dirección desde la cual se envió el paquete.
- **Puerto Origen**: Puerto desde el cual se envió el paquete.
- **IP Destino**: Dirección a la cual se envió el paquete.
- **Puerto Destino**: Puerto hacia donde se envió el paquete.
- **Protocolo**: Número de protocolo de acuerdo a la IANA.
- **Tipo**: Clasificación Maligno o Normal asignada por el modelo clasificador.

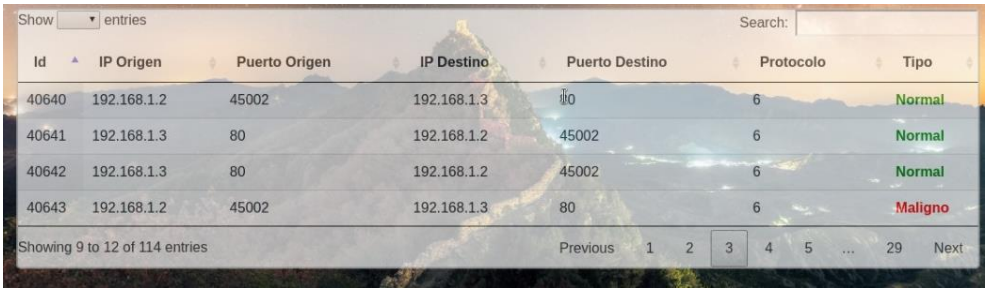


Figura 43. Sección Información de paquetes del sistema Dique.

4.4 Fase 4: Verificar el funcionamiento

El sistema que realiza ataques para verificar el funcionamiento de Dique, se denomina Diluvio, debido a la analogía con las fuerte lluvias que hacen colapsar los diques. Diluvio consiste en un aplicativo web de una única página que contiene un campo de texto en el cual se ingresa la dirección IP del dispositivo, a la que se desea enviar los paquetes. Luego se presiona uno de los botones que contiene un ataque DoS diferente. En la figura 44 se muestra la interfaz gráfica del sistema Diluvio.

Figura 44. Interfaz gráfica del sistema Diluvio.

Para verificar el funcionamiento del sistema preventivo, se ejecutó cada uno de los 7 ataques DoS y una petición HTTP para analizar como actúa frente a cada uno de ellos. Las siguientes figuras muestran los paquetes clasificados en el sistema preventivo:

- **Petición HTTP**

Al realizar una petición HTTP al servidor web, el sistema clasifica todos los paquetes como NoMaligno como se muestra en la siguiente figura 45.

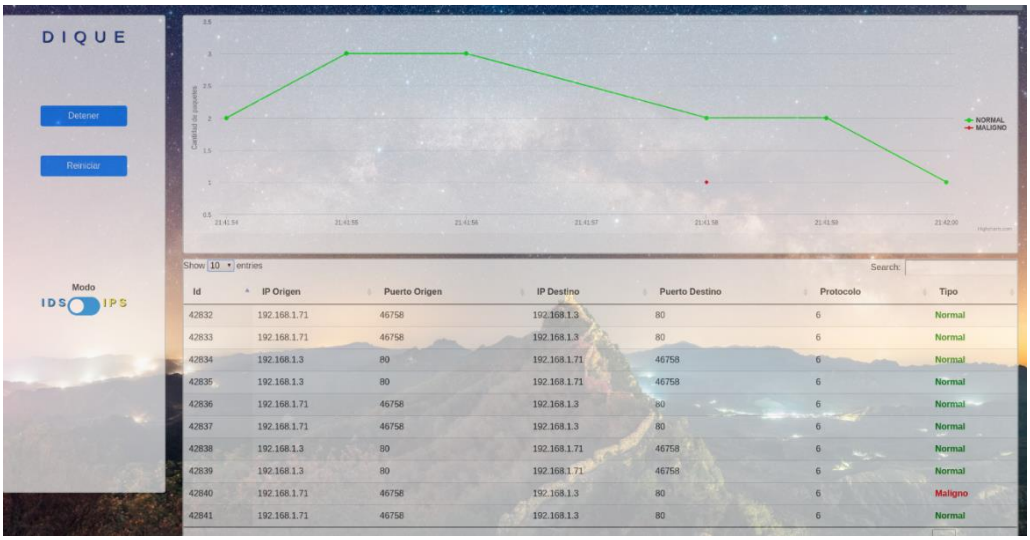


Figura 45. Petición Normal HTTP al servidor web.

Esta clasificación se basa en la salida de la red neuronal al clasificar un nuevo paquete, la cual es un vector que contiene dos elementos. Estos elementos representan los porcentajes de clasificación de las 2 clases con las que se entrenó el algoritmo, que para este caso son NoMaligno con un 99.99% de probabilidad y Maligno con un un 0.0071217% de ser Maligno.

```
**********Paquete No.52 **********************
* [[    0.9999, 7.1217e-5]]
* La probabilidad es de 0.9999288320541382
* El paquete 52 es clasificado como NO MALIGNO 0
```

Figura 46. Porcentaje de clasificación de la petición NoMaligna.

Asi, para el sistema Dique, la clasificación de los paquetes nuevos se basa en el elemento que tenga mayor probabilidad de clasificación, que para el caso de la petición HTTP, clasifica los paquetes como NoMalignos.

- **DNS REFLECTION**

En la figura 47 se muestra como el sistema Dique clasifica los paquetes del ataque DNS Reflection como Malignos y NoMalignos.

Figura 47. Ataque DNS Reflection.

La figura 48 muestra como los paquetes Malignos son clasificados con un porcentaje de clasificación de 99.66% y para los NoMalignos con un 99.99%.

```
**********Paquete No.74 **********
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 74 es clasificado como MALIGNO 1
************************************************
2021-02-17 18:58:59.313  INFO 10643 --- [     Thr
Paquete Nro 74 recibido.
Paquete Nro 75 recibido.
Paquete Nro 76 recibido.
**********Paquete No.73 **********
* [[    0.9999, 7.3992e-5]]
* La probabilidad es de 0.999925971031189
* El paquete 73 es clasificado como NO MALIGNO 0
```

Figura 48. Porcentaje de clasificación del ataque DNS Reflection.

Para este caso, hace falta entrenar la red neuronal con mas tipos de paquetes de las características de este ataque, pues el porcentaje de clasificación es muy alto para ambas clases.

- **NTP REFLECTION**

El ataque NTP Reflection es clasificado como Maligno, como se muestra en la figura 49.

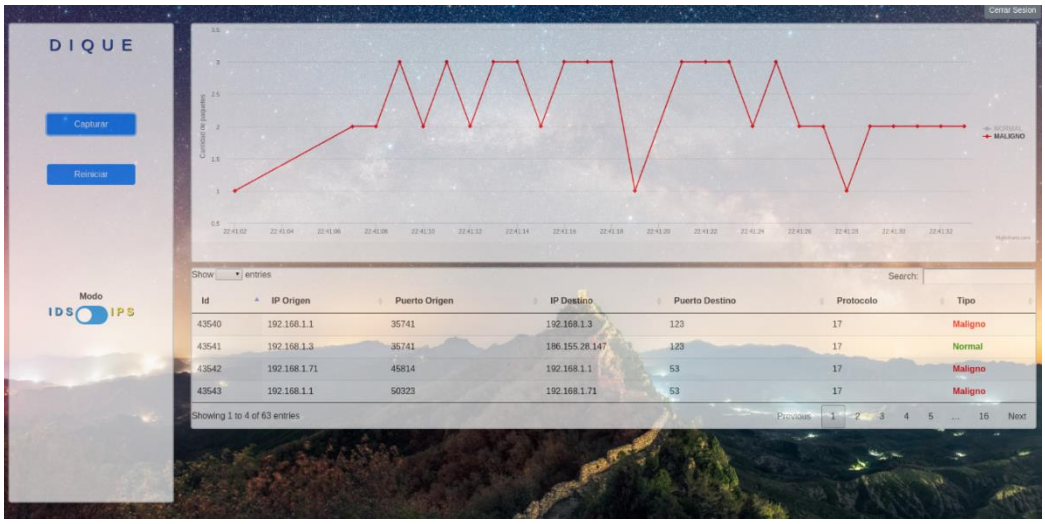


Figura 49. Ataque NTP Reflection.

La siguiente figura muestra el vector de salida de dos paquetes del ataque NTP Reflection, clasificados como Malignos con un 99.66% de probabilidad y 0,34% de probabilidad de ser NoMaligno.

```
**********Paquete No.54 **********************
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 54 es clasificado como MALIGNO 1
************************************************
Paquete Nro 54 recibido.
2021-02-17 18:58:58.244  INFO 10643 --- [     Thre
**********Paquete No.55 **********************
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 55 es clasificado como MALIGNO 1
************************************************
```

Figura 50. Porcentaje de clasificación del ataque NTP Reflection.

Esto demuestra que los features de los paquetes clasificados como NTP son relevantes para la clasificación MALIGNA, pues como lo evidencia el estudio realizado por el Instituto de CiberSeguridad de Canadá [12], los paquetes NTP y BENIGN contienen features relevantes muy diferentes.

- **SYN FLOOD**

En la siguiente figura se muestra como los paquetes del ataque Syn flood son clasificados en su mayoría como NoMaligno.

Figura 51. Ataque SYN Flood.

La siguiente figura muestra como los paquetes de este tipo de ataque son clasificados en su mayoría como No Malignos con un 96.09% de probabilidad.

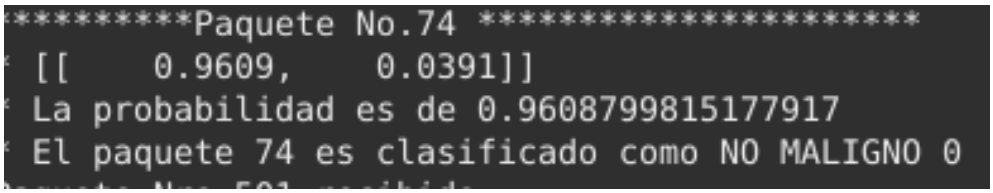


Figura 52. Porcentaje de clasificación del ataque Syn Flood.

La razón de la similitud en la clasificación con los paquetes NoMalignos, se debe a que existen 2 caracteristicas comunes relevantes que tienen estos paquetes que son ACK Flag Count y Init_Win_bytes_forward.

- **UDP FLOOD**

En la figura 53 se muestra como al realizar el ataque UDP Flood, los paquetes son clasificados en su totalidad como Maligno.

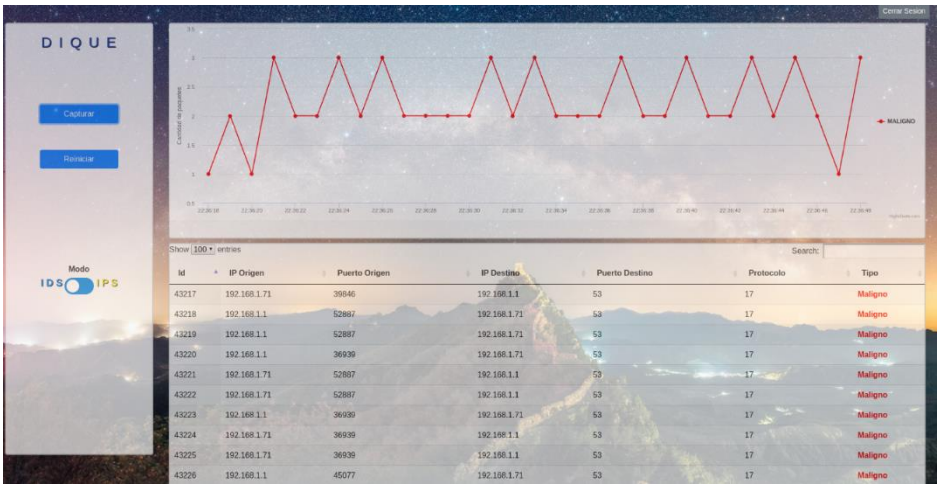


Figura 53. Ataque UDP Flood.

En la siguiente figura se evidencia que los paquetes que contienen el ataque UDP Flood, son clasificados como Malignos con una probabilidad de 99.99%, evidenciando la efectividad de la red neuronal para clasificar este tipo de ataques.

```
**********Paquete No.23 **********************
* [[ 1.1991e-5,    1.0000]]
* La probabilidad es de 0.9999879598617554
* El paquete 23 es clasificado como MALIGNO 1
************************************************
```

Figura 54. Porcentaje de clasificación del ataque UDP Flood.

- **TCP CONNECT FLOOD**

El ataque TCP Connect Flood es clasificado como NoMaligno, como se muestra en la siguiente figura.

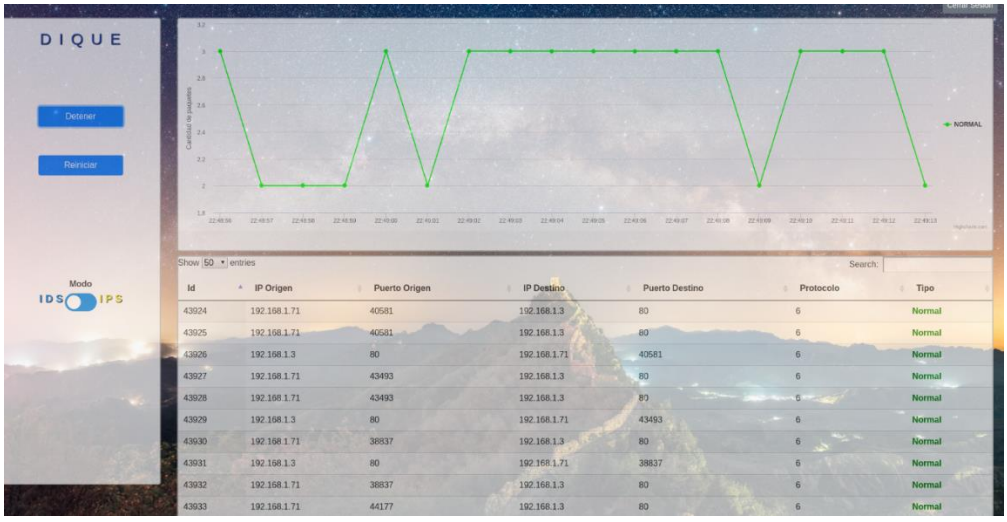


Figura 55. Ataque TCP Flood.

La siguiente figura muestra como los paquetes de este tipo de ataque son clasificados todos como Malignos con un porcentaje de probabilidad de 62.60% y 99.99%, lo que evidencia la similitud de sus features con los paquetes normales.

Figura 56. Porcentaje de clasificación del ataque TCP Flood.

- **ICMP FLOOD**

En la figura 57, se muestra como el ataque ICMP Flood es clasificado como Maligno en la mayoría de paquetes.

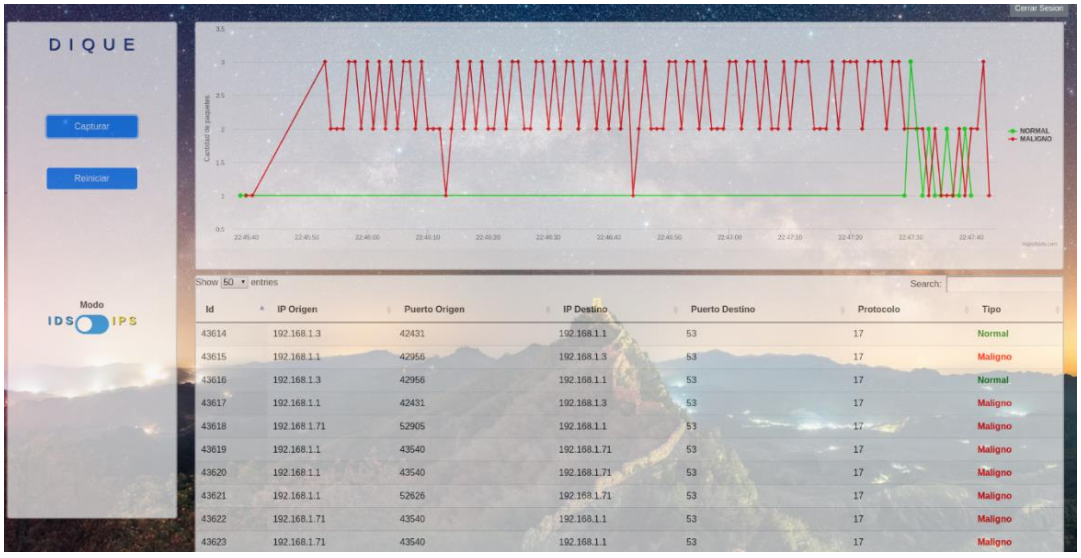


Figura 57. Ataque ICMP Flood.

La siguiente figura muestra como los paquetes con el tipo de ataque ICMP Flood son clasificados en su mayoría como Malignos con una probabilidad de 99.61%. Para este tipo de paquetes, no hubo datos de entrenamiento, por lo que se quiso evidenciar como se comportaba frente a tipos de ataques con los que la red neuronal no había sido entrenada, mostrando un porcentaje de clasificación como Malignos muy alta.

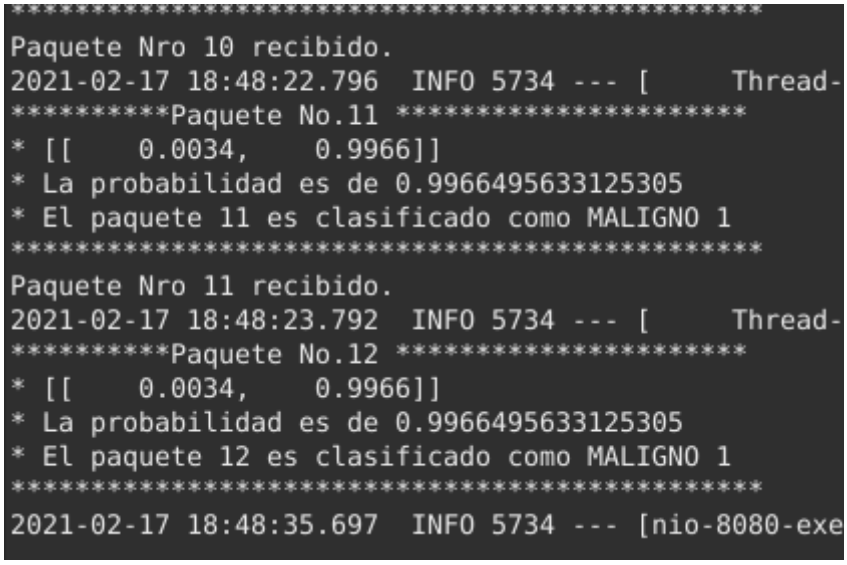


Figura 58. Porcentaje de clasificación del ataque ICMP Flood.

- **SLOW HTTP**

En la figura 59 se ilustra como el ataque SLOW HTTP clasifica paquetes como NoMaligno y Maligno.

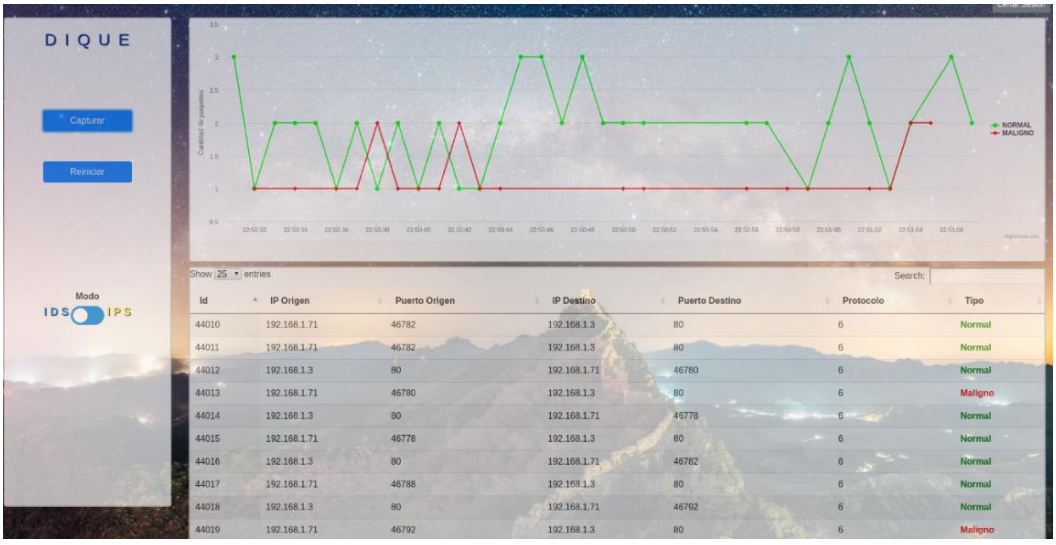


Figura 59. Ataque Slow HTTP.

La siguiente figura muestra como los paquetes de este tipo de ataque son clasificados en su mayoría como NoMalignos debido a la gran similitud que tiene con una petición HTTP normal realizada al comienzo de la fase de verificación.

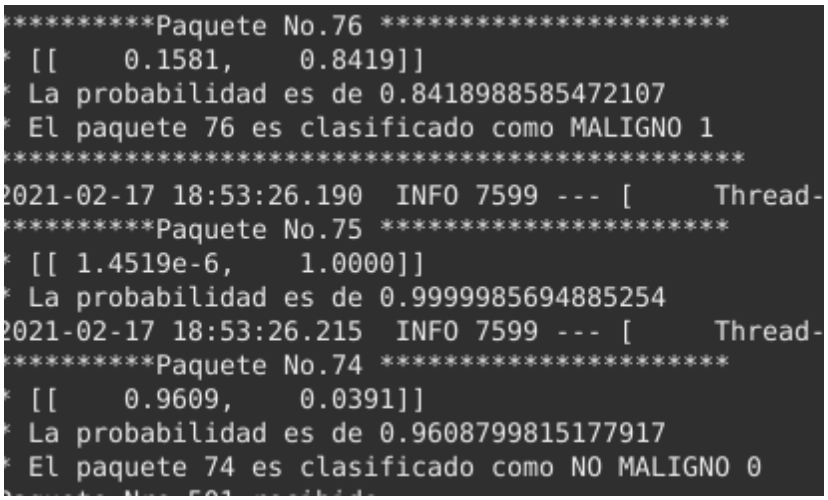


Figura 60. Porcentaje de clasificación del ataque tipo Slow HTTP.

Por último, para verificar el modo IPS de Dique, el sistema comienza a aplicar reglas automáticamente al combinar el potencial de la detección utilizando Deep Learnig, con otras herramientas como el firewall que trae por defecto del sistema. De esta manera el sistema comienza a aplicar reglas para bloquear las direcciones IP de los paquetes clasificados como Malignos, por lo que al realizar nuevos ataques DoS, el sistema no mostrará nuevos paquetes desde esa dirección IP como se evidencia en la figura 61. Asi para prevenir futuros ataques DoS, se aplicaron comandos del firewall del sistema operativo que se ejecutan desde el lenguaje java al detectar un paquete como Maligno.

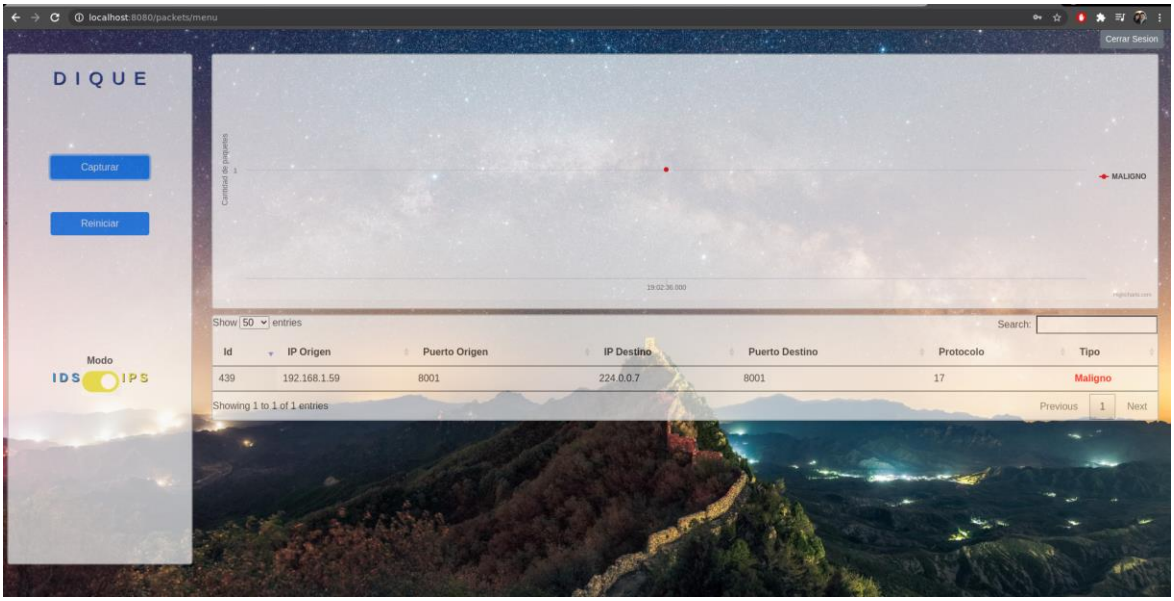


Figura 61. Modo IPS Activado**.**

5. Conclusiones y recomendaciones

A continuación, se sintetiza los puntos más relevantes obtenidos de la investigación y las recomendaciones para crear nuevos sistemas de clasificación. Además, menciona las mejoras que se le pueden hacer al sistema preventivo creado y plantear los posibles trabajos futuros.

5.1 Conclusiones

Esta sección describe los resultados más relevantes obtenidos del proyecto de grado, para su desarrollo se tomó como referente los objetivos.

Del objetivo 1; "*Seleccionar los algoritmos de Deep Learning más adecuados para la identificación de ataques de denegación de servicio web*", se concluye que los algoritmos de Deep Learning más adecuados para entrenar modelos que permitan la identificación de ataques DoS son: Deep Feed Forward, Recurrent Neural Network y Long Short Term Memory y que cumplen los criterios especificados en este proyecto. Esto se logró al haber realizado una revisión sistemática a la literatura del estado del arte de los métodos de Deep Learning para la identificación de ataques de DoS, de la cual se obtuvó 157 articulos que se encuentran en el Anexo F y que 42 cumplieron con los criterios de valoración de calidad representando las referencias bibliográficas de este trabajo. Además, se replicó el entrenamiento de los modelos con el algoritmo y el Dataset correspondiente para validar sus resultados. Por ultimo, por especificaciones técnicas de estos tres algoritmos, se seleccionó el algoritmo Deep Feed Forward el cual al ser entrenado con el Dataset *CICDDOS2019* presentó un comportamiento adecuado para la detección de ataques DoS.

Con respecto al objetivo 2, que es "*Construir modelos de clasificación mediante el entrenamiento de los algoritmos de Deep Learning más óptimos que serán seleccionados por medio de un cuadro comparativo que contiene las diferentes medidas de precisión utilizadas en la fase de pruebas, junto al Dataset más reciente hasta la fecha*", se concluye que los modelos entrenados con mejor rendimiento obtuvieron un Accuracy de 0.7293, Precision de 0.8012, Recall 0.6721 y F1 Score de 0.7794 para el modelo de clasificación de 13 clases (12 Ataques DoS y NoMaligno), en la literatura, el mejor reporte obtuvo una Precision de 0.78, Recall 0.65 y F1 score de 0.69 [12], aunque ellos utilizaron en su investigación técnicas de Machine Learning, denotando que el comportamiento de Deep Learning es superior a Machine Learning como también se reporta en la literatura. Y Accuracy de 0.9994 para el modelo de clasificación de 2 clases (Maligno y NoMaligno), y en las referencias bibliográficas hasta la elaboración de este proyecto, se reportan 0.9378 [37] y 0.9963 [34] para el algoritmo con la red neuronal Deep Feed Forward.

Del objetivo 3; "*Elaborar un software que integre los modelos de clasificación con una interfaz gráfica de usuario, para identificar, analizar, prevenir y mostrar gráficamente los diferentes ataques de denegación de servicio*", se concluye que al integrar las diferentes librerías de java como Spring Boot, DL4J, Pcap4J y el modelo entrenado, dio como resultado el aplicativo *Dique*, el cual es un sistema preventivo que permite: analizar, clasificar, detectar, prevenir y mostrar en tiempo real , ataques de denegación de servicio realizados a un servidor web. El sistema de detección se realiza a través del modelo entrenado y la prevención se ejecuta a través del firewall del sistema operativo utilizando los iptable para bloquear la ip de origen del paquete. Además, el sistema Dique puede ser desplegado en cualquier sistema operativo tipo Linux que tenga instalado java y acceder vía web desde cualquier dispositivo que tenga conexión con este.

Con respecto al objetivo 4, que es "Verificar el funcionamiento del sistema de prevención de ataques de denegación de servicio web, mediante un software que automatice los ataques contra el sistema informático elaborado en un ambiente controlado", se concluye que el aplicativo web Diluvio, permite lanzar siete tipos de ataques de denegación de servicio: cinco de estos que hacen parte del Dataset de entrenamiento y dos que no están en dicho Dataset , evidenciando que el modelo entrenado posee la característica de clasificar ataques diferentes a los cuales fue entrenado. Los ataques que no pertenecen al Dataset de entrenamiento son ICMP Flood y Slow HTTP que fueron clasificados como Malignos con una probabilidad superior al 0.84, como se evidencia en la fase 4 de *Resultados*. Además, gracias a la integración del sistema preventivo Dique con el firewall del sistema operativo, se pudo validar el modo preventivo al aplicar reglas automáticamente con el fin de prevenir futuros ataques DoS.

Como conclusión general, el trabajo realizado aporta y evidencia la aplicación de un sistema IDS/IPS utilizando modelos entrenados con Deep Learning, el cual fue probado con ataques DoS que contiene el Dataset de entrenamiento y ataques que no están en el Dataset. En la literatura no se demuestra una posterior aplicación de los modelos entrenados por los investigadores, enfocándose solamente en el rendimiento obtenido al entrenar los modelos sin llevarlos a la práctica o solo mencionando que se llevan a la práctica sin mostrar como se realiza dicho proceso, en esta investigación si se realiza una aplicación del modelo entrenado a través del aplicativo Dique. Además, a pesar de que en la literatura los modelos entrenados logran un Accuracy de alto rendimiento, esto en la práctica no es garantía de que tan óptimo realiza la clasificación de paquetes desconocidos, pues como se evidenció en la fase 4 al lanzar los ataques DoS al sistema preventivo mediante el aplicativo Diluvio, este no clasifica los paquetes en una alta probabilidad como se espera que el porcentaje lo indica, que, de acuerdo a la teoría, hace falta entrenarlo con más datos. Por último y no menos importante, los modelos fueron entrenados con un Dataset solo con información de ataques DoS, el cual se publicó cuando se estaba realizando esta investigación y que hasta la fecha solo los mismos creadores de este DataSet lo han utilizado para entrenar modelos de Machine Learning, por lo que el Accuracy obtenido mediante Deep Learning, es de gran interés para realizar comparaciones e implementaciones en las diferentes resúmenes que realiza la comunidad científica para otros estudios.

5.2 Recomendaciones, lecciones aprendidas y trabajo futuro

Habiendo cumplido el objetivo planteado en el desarrollo de este proyecto de grado, existen algunos componentes que pueden modificarse con miras a mejorar el sistema preventivo. Estos son:

- Java no contiene un API para manipular y dropear paquetes de red, por lo cual se tuvo que recurrir al firewall del sistema operativo para activar el modo preventivo. Se podría utilizar un lenguaje de programación de nivel intermedio, como C/C++, para crear funciones que permitan la manipulación de estos paquetes directamente desde el aplicativo, ejecutándolo como código nativo, sin recurrir al firewall.
- El Dataset de clasificación *CICDDoS2019* solo contiene 22.000 paquetes de red clasificados como BENIGN (NoMalignos) y 48.077.733 paquetes de ataques DoS, por lo que, si se desea aumentar el rendimiento de entrenamiento y clasificación, se deben crear nuevos paquetes de red clasificados como NoMalignos, dado que la relación entre Malignos y NoMalignos debe ser igual.
- Para el entrenamiento de modelos con Deep Learning se requiere una máquina con mejor hardware, esto porque en la fase de creación de modelos hubo una limitante de memoria RAM para aumentar la cantidad de redes neuronales e intentar mejorar el desempeño de los modelos con 13 labels.
- El sistema preventivo *Dique* requiere de una máquina con mejor hardware CPU y memoria para poder clasificar paquetes con mayor rapidez, esto incluye la fase de captura, extracción de features, clasificación y almacenamiento de paquetes.

Como recomendaciones para el desarrollo de sistemas preventivos utilizando Deep Learning se puede decir que:

- Se recomienda utilizar otros algoritmos de Deep Learning para entrenar modelos de clasificación con el Dataset *CICDDoS2019*, ya que existen muchos otros algoritmos por probar y verificar sus resultados.
- Se debe realizar una preparación del Dataset *CICDDoS2019* utilizado, ya que de esa manera se pueden utilizar otros algoritmos de Deep Learning sin requerir la fase previa de preprocesamiento y que fue uno de los inconvenientes por los que no se pudo entrenar modelos con el algoritmo *RNN* y *LSTM*.
- Utilizar otros lenguajes de programación para entrenar modelos de Deep Learning como Python o C/C++, con el objetivo de optimizar el modelo, dado que Python está orientado a trabajar Deep Learning y C/C++ es uno de los lenguajes que posee mejor aprovechamiento del hardware con respecto a la ejecución de procesos.

Como trabajo futuro, se propone profundizar en un estudio especializado de Deep Learning para:

- Elaborar un propio algoritmo de Deep Learning o mejorar uno de los existentes, con el objetivo de optimizar su desempeño.

- Mejorar el Accuracy de los modelos de clasificación para que el sistema preventivo pueda clasificar en más clases, que no sea solo Maligno y NoMaligno, sino que pueda predecir con un porcentaje a qué tipo de ataque de denegación de servicio se parece.
- Construir un Dataset propio, que puede ser a partir de los existentes, y que permita una mayor cantidad de datos para mejorar los modelos de entrenamiento.

A. Anexo: Tipos de ataques de denegación de servicio

- **Ataques a nivel de aplicación:** Este tipo de ataques DDoS tiene como objetivo una aplicación o un sitio web mal programado, con la intención de explotar alguna vulnerabilidad, dando como resultado la indisponibilidad del servidor. Wordpress y Joomla son dos ejemplos de aplicaciones que pueden ser objetivo para consumir los recursos del servidor – RAM, CPU, etc. Las bases de datos también pueden ser objetivo de inyección SQL diseñadas para explotar estas brechas. El servidor exhausto queda entonces en un estado indisponible para procesar solicitudes legitimas debido al consumo de sus recursos.
- **Día Zero DDoS:** Este es un término estandarizado usado para describir un ataque que explota nuevas vulnerabilidades. Estas vulnerabilidades no tienen parches o mecanismos efectivos de defensa.
- **Ping Flood:** Una versión avanzada de ICMP Flood. Este ataque DDoS también tiene como objetivo una aplicación específica. Cuando un servidor recibe una gran cantidad de paquetes ping falsos de una gran cantidad de IPs origen, está siendo objetivo de un ataque Ping Flood. La meta de este ataque es inundar el objetivo con paquetes ping hasta que este indisponible. Este ataque está diseñado para consumir todo el ancho de banda y los recursos en la red hasta que esté completamente ocupado y se apague. Este tipo de ataque DDoS no es fácil de detectar y puede ser muy parecido al tráfico legítimo.
- **Ataque IP Null:** Los paquetes IP versión 4 contienen información en su cabecera sobre qué tipo de protocolo de transporte está siendo utilizado. Cuando los atacantes colocan este valor en null, estos paquetes pueden llegar a saltar medidas de seguridad diseñadas para escanear TCP, IP e ICMP. Cuando el servidor objetivo intenta procesar estos paquetes, este eventualmente consumirá sus recursos y se reiniciará.
- **SNMP Flood:** El protocolo SNMP puede ser utilizado para ataques amplificados. SNMP es principalmente usado en dispositivos de red. Este ataque consiste en el envío de pequeños paquetes que contienen una IP suplantada hacia un objetivo y luego el objetivo recibe como respuesta una inundación de paquetes UDP. Este termina por agotar los recursos del objetivo para posteriormente apagarlo.
- **NTP Flood:** El protocolo NTP es un protocolo públicamente accesible. Igual que el protocolo SNMP, este ataque consiste en un envío masivo de paquetes UDP por

medio de una IP suplantada. Cuando el objetivo se da cuenta de esta cantidad de paquetes terminará por consumir sus recursos y se apagará o reiniciará.

- **Otros ataques DDoS Amplificados:** Todos los ataques amplificados utilizan la misma estrategia descrita en los protocolos SNMP Flood y NTP Flood. Otros protocolos UDP que han sido identificados para realizar este tipo de ataques son:
  - CharGEN.
  - SSDP.
  - SNMPv2.
  - NetBIOS.
  - QOTD.
  - Bit Torrent.
  - Kad.
  - Quake Network Protocol.
  - Steam Protocol.
- **HTTP Fragmented Flood:** Este ataque consiste en utilizar un conjunto de Bots con una IP válida, los cuales son usados para establecer una conexión HTTP válida con un servidor web. Entonces, los paquetes HTTP son divididos por los bots en pequeños fragmentos y enviados al objetivo tan lento como sea posible antes de que el servidor responda como un time out. Este método permite a los atacantes mantener una conexión activa por un largo periodo de tiempo sin alertar cualquier mecanismo de defensa. Un atacante puede utilizar un bot para iniciar un conjunto de sesiones extendidas, indetectables y de alto consumo de recursos.
- **HTTP Flood:** La dirección IP real del bot es utilizada para evitar cualquier sospecha. El número de bots utilizado para ejecutar este tipo de ataque es el rango de direcciones de la red en la que se encuentra el bot. Como las direcciones IP de los bots son legítimas, no hay mecanismo de defensa para evitar estas solicitudes HTTP. Solo un bot puede ser utilizado para enviar una enorme cantidad de peticiones GET, POST u otro tipo de peticiones para ejecutar un ataque entonces varios bots pueden ser combinados en un ataque DDoS HTTP para inhabilitar por completo el servidor objetivo.
- **HTTP Flood de una sola sesión:** Un atacante puede explotar una brecha de seguridad en HTTP 1.1 para enviar varias solicitudes desde una sola sesión HTTP. Esto permite a los atacantes enviar una gran cantidad de solicitudes desde un conjunto de sesiones. En otras palabras, los atacantes pueden pasar por alto las limitaciones impuestas por los mecanismos de defensa DDoS en el número de sesiones permitidas. La inundación HTTP de una sola sesión, también apunta a los recursos de un servidor para desencadenar un cierre completo del sistema o un rendimiento deficiente.
- **Single Request HTTP Flood:** Cuando los mecanismos de defensa evolucionaron para bloquear muchos paquetes entrantes, los ataques como el Single Packet HTTP Flood fueron diseñados con soluciones para esquivar estas defensas. Esta evolución de inundación de HTTP, explota otra brecha de seguridad en la

tecnología HTTP. Una sola sesión HTTP puede realizar varias solicitudes HTTP al enmascarar estas solicitudes dentro de un paquete HTTP. Esta técnica permite que un ataque permanezca invisible mientras se agotan los recursos de un servidor al mantener las tasas de paquetes dentro de los límites permitidos.

- **HTTP GET Flood Recursivo:** Para que un ataque sea altamente exitoso, debe permanecer sin ser detectado durante el mayor tiempo posible. El mejor método para pasar desapercibido es aparecer como una solicitud legítima manteniéndose dentro de todas las limitaciones mientras se ejecuta otro ataque. Este ataque GET recursivo logra esto por sí mismo al recopilar una lista de páginas o imágenes y parece que está pasando normalmente por estas páginas o imágenes. Este ataque se puede combinar con un ataque de HTTP flood para obtener el máximo impacto.
- **Random Recursive GET Flood:** Este ataque es una variación del ataque GET recursivo. Está diseñado para foros, blogs y otros sitios web que tienen páginas en secuencia. Al igual que el GET recursivo, también parece estar pasando por páginas. Dado que los nombres de las páginas están en una secuencia, para mantener la apariencia de usuario legítimo, utiliza números aleatorios de un rango de páginas válido para enviar una nueva solicitud GET. Random Recursive GET también tiene como objetivo afectar el rendimiento de su objetivo con un gran número de solicitudes GET y denegar el acceso a usuarios reales.
- **Multi-Vector Attacks:** Este tipo de ataque combina ataques GET recursivos con ataques de inundación HTTP para amplificar los efectos de un ataque. Ese es solo un ejemplo de un atacante que usa dos tipos de ataques DDoS al mismo tiempo para atacar a un servidor. Los ataques también pueden combinar varios métodos para mantener confundidos a los ingenieros que se ocupan del ataque DDoS. Estos ataques son los más difíciles de tratar y son capaces de eliminar algunos de los servidores y redes mejor protegidos.
- **SYN Flood:** Este ataque explota el diseño del proceso de comunicación TCP de tres vías entre un cliente, un host y un servidor. En este proceso, un cliente inicia una nueva sesión generando un paquete SYN, el anfitrión asigna y verifica estas sesiones hasta que son cerradas por el cliente. Para llevar a cabo un ataque SYN Flood, un atacante envía muchos paquetes SYN al servidor de destino desde direcciones IP falsificadas. Este ataque continúa hasta que agota la memoria de la tabla de conexión del servidor. El resultado es un servidor no disponible para procesar solicitudes legítimas debido al agotamiento de recursos mientras dure el ataque.
- **SYN-ACK Flood:** El segundo paso del proceso de comunicación TCP de tres vías es explotado por este ataque DDoS. En este paso, el host de escucha genera un paquete SYN-ACK para reconocer un paquete SYN entrante. Una gran cantidad de paquetes SYN-ACK falsificados se envían a un servidor de destino en un ataque de inundación SYN-ACK. El ataque intenta entonces agotar los recursos de un servidor: su RAM, CPU, etc. mientras el servidor intenta procesar este flujo de solicitudes.

- **ACK & PUSH ACK Flood:** Durante una sesión activa de TCP-SYN, los paquetes ACK o PUSH ACK transportan información hacia y desde las máquinas host y cliente hasta que dura la sesión. Durante un ataque de inundación ACK y PUSH ACK, se envía una gran cantidad de paquetes ACK falsificados al servidor de destino para afectarlo. Dado que estos paquetes no están vinculados con ninguna sesión en la lista de conexiones del servidor, el servidor invierte más recursos en procesar estas solicitudes.
- **ACK Fragmentation Flood:** Los paquetes ACK fragmentados se utilizan en esta versión de ataque que consiste en consumir el ancho de banda. Para ejecutar este ataque, se envían paquetes fragmentados de 1500 bytes al servidor de destino. Es más fácil que estos paquetes alcancen su destino sin ser detectados, ya que normalmente los enrutadores no los vuelven a ensamblar en el nivel de IP, esto permite que un atacante envíe pocos paquetes con datos irrelevantes a través de dispositivos de enrutamiento para consumir grandes cantidades de ancho de banda. Este ataque afecta a todos los servidores dentro de la red de destino al intentar consumir todo el ancho de banda disponible en la red.
- **RST / FIN Flood:** Después de una sesión TCP-SYN exitosa de tres o cuatro vías, los servidores intercambian los paquetes RST o FIN para cerrar la sesión TCP-SYN entre un host y una máquina cliente. En un ataque RST o FIN Flood, un servidor de destino recibe una gran cantidad de paquetes RST o FIN falsificados que no pertenecen a ninguna sesión en el servidor de destino. El ataque intenta agotar los recursos de un servidor: su RAM, CPU, etc. mientras el servidor intenta procesar estas solicitudes no válidas. El resultado es un servidor no disponible para procesar solicitudes legítimas.
- **Synonymous IP Attack:** Para inhabilitar un servidor, una gran cantidad de paquetes TCP-SYN que llevan la misma IP tanto de origen, como la IP de destino del servidor se envían al servidor de destino. El objetivo del ataque de Synonymous IP es agotar los recursos de un servidor: RAM, CPU, etc., mientras trata de calcular esta anomalía.
- **Spoofed Session Flood:** Algunos de los ataques DDoS anteriores no pueden engañar a los mecanismos de defensa más modernos, pero los ataques DDoS también están evolucionando para eludir estas defensas. Los ataques de sesión falsificada intentan eludir la seguridad bajo el disfraz de una sesión TCP válida llevando un SYN, múltiples ACK y uno o más paquetes RST o FIN. Este ataque puede pasar por alto los mecanismos de defensa que solo monitorean el tráfico entrante en la red. Estos ataques DDoS también pueden agotar los recursos del objetivo y resultar en un cierre completo del sistema o un rendimiento inaceptable del sistema.
- **Multiple SYN-ACK Spoofed Session Flood:** Esta versión de un ataque de sesión falsificada contiene múltiples paquetes SYN y múltiples ACK junto con uno o más paquetes RST o FIN. Una sesión falsa SYN-ACK múltiple es otro ejemplo de un ataque DDoS evolucionado. Se modifican hasta eludir los mecanismos de defensa

que se basan en reglas muy específicas para evitar tales ataques. Al igual que el ataque de sesión falsa, este ataque también puede agotar los recursos de un objetivo y dar como resultado un cierre completo del sistema o un rendimiento inaceptable del sistema.

- **Ataques de sesión:** Para evitar las defensas, en lugar de utilizar IP falsas, este ataque utiliza la dirección IP real de los Bot que se utilizan para llevar a cabo un ataque. El número de Bot utilizados para ejecutar el ataque es el mismo que el rango de IP de origen para este ataque. Este ataque se ejecuta creando una sesión TCP-SYN entre un Bot y el servidor de destino. Esta sesión se prolonga hasta que se agota el tiempo retrasando los paquetes ACK. Los ataques de sesión intentan agotar los recursos de un servidor a través de estas sesiones vacías. Eso, a su vez, da como resultado un cierre completo del sistema o un rendimiento inaceptable del sistema.
- **Misused Application Attack:** Los atacantes primero hackean máquinas cliente que alojan aplicaciones de alto tráfico como los servicios P2P. El tráfico de estas máquinas cliente se redirige al servidor de destino. El servidor de destino agota sus recursos cuando intenta aceptar y negociar el tráfico excesivo. Los mecanismos de defensa no se activan en este caso, ya que las máquinas cliente hackeadas intentan establecer una conexión válida con el servidor de destino. Después de redirigir con éxito el tráfico al objetivo, mientras el ataque continúa, el atacante abandona la red y se vuelve imposible de rastrear. El uso de Misused Application Attack apunta a los recursos de un servidor e intenta eliminarlo o destruir su rendimiento.
- **UDP Flood:** Como su nombre lo indica, en este tipo de ataque DDoS, un servidor está inundado de paquetes UDP. A diferencia de TCP, no hay un proceso de comunicación de extremo a extremo entre el cliente y el host. Esto hace que sea más difícil para los mecanismos defensivos identificar un ataque de inundación de UDP. Una gran cantidad de paquetes UDP falsificados se envían a un servidor de destino desde un conjunto masivo de IP de origen para eliminarlo. Los ataques de inundación UDP pueden apuntar a servidores aleatorios o a un servidor específico dentro de una red al incluir el puerto y la dirección IP del servidor de destino en los paquetes atacantes. El objetivo de tal ataque es consumir el ancho de banda en una red hasta que se haya agotado todo el ancho de banda disponible.
- **UDP Fragmentation Flood:** Es otro ataque DDoS inteligentemente enmascarado que no se detectan fácilmente. La actividad generada por este ataque se parece al tráfico válido y todo se mantiene dentro de los límites. Esta versión del ataque UDP Flood envía paquetes más grandes pero fragmentados para agotar más ancho de banda al enviar menos paquetes UDP fragmentados. Cuando un servidor de destino intenta juntar estos paquetes UDP fragmentados no relacionados y falsificados, no podrá hacerlo. Eventualmente, todos los recursos disponibles se agotan y el servidor puede reiniciarse.

- **DNS Flood:** Uno de los ataques DDoS más conocidos. Esta versión del ataque de inundación UDP es específica de aplicación: los servidores DNS en este caso. También es uno de los ataques DDoS más difíciles de detectar y prevenir. Para ejecutarlo, un atacante envía una gran cantidad de paquetes de solicitud DNS falsificados que no se parecen a las solicitudes reales de un conjunto muy grande de IP de origen. Esto hace que sea imposible para el servidor de destino diferenciar entre solicitudes de DNS legítimas y solicitudes de DNS que parecen ser legítimas. Al tratar de atender todas las solicitudes, el servidor agota sus recursos. El ataque consume todo el ancho de banda disponible en la red hasta que se drena por completo.
- **VoIP Flood:** Esta versión de aplicación específica de UDP se dirige a los servidores de VoIP. Un atacante envía una gran cantidad de paquetes de solicitud de VoIP falsificados desde un conjunto muy grande de IP de origen. Cuando un servidor VoIP está inundado de solicitudes falsificadas, agota todos los recursos disponibles al intentar atender las solicitudes válidas e inválidas. Esto reinicia el servidor o tiene un costo en el rendimiento del servidor y agota el ancho de banda disponible. Las inundaciones de VoIP pueden contener una fuente IP fija o aleatoria. El ataque de dirección IP de origen fijo no es fácil de detectar, ya que se enmascara y no se ve diferente al tráfico legítimo.
- **Media Data Flood:** Al igual que la inundación de VoIP, un servidor también puede ser atacado con datos de medios tales como audio y video. Un atacante envía un gran número de paquetes de datos de medios falsificados desde un conjunto muy grande de IP de origen. Cuando un servidor está inundado de solicitudes de datos de medios falsificados, agota todos los recursos disponibles y el ancho de banda de la red para procesar estas solicitudes. Este ataque es similar a las inundaciones de VoIP en todos los sentidos, excepto en el uso de paquetes de datos de medios falsificados para atacar el servidor. También puede ser difícil detectar estos ataques cuando utilizan una IP de fuente fija, ya que esto les da una apariencia legítima. El ataque está diseñado para consumir todos los recursos de servidor disponibles y el ancho de banda en la red hasta que se agote por completo
- **Direct UDP Flood:** El servidor de destino es atacado con una gran cantidad de paquetes UDP sin suplantación de identidad. Para enmascarar el ataque, el atacante no falsifica la dirección IP real de los Bot. El número de Bot utilizados para ejecutar el ataque es el mismo que el rango de IP de origen para este ataque. El ataque está diseñado para consumir todo el ancho de banda y los recursos disponibles en la red hasta que se agote por completo y se apague. Este tipo de ataque DDoS tampoco es fácil de detectar ya que se asemeja al tráfico legítimo.
- **ICMP Flood:** Al igual que UDP, ICMP tampoco tiene un proceso de extremo a extremo para el intercambio de datos. Esto hace que sea más difícil detectar un ataque de inundación de ICMP. Un atacante envía una gran cantidad de paquetes ICMP falsificados desde un conjunto muy grande de IP de origen. Cuando un servidor se inunda con cantidades masivas de paquetes ICMP falsificados, sus

recursos se agotan al tratar de procesar estas solicitudes. Los ataques de inundación de ICMP pueden dirigirse a servidores aleatorios o a un servidor específico dentro de una red al incluir el puerto y la dirección IP del servidor de destino en los paquetes. El objetivo de tal ataque es consumir ancho de banda en la red hasta que haya agotado el ancho de banda disponible.

- **ICMP Fragmentation Flood:** Esta versión del ataque de inundación de ICMP envía paquetes más grandes para agotar más ancho de banda al enviar menos paquetes de ICMP fragmentados. Cuando el servidor de destino intenta colocar estos paquetes ICMP fragmentados falsificados sin correlación juntos, no podrá hacerlo. El servidor eventualmente agota sus recursos y se reinicia.

Clasificación de Ataques DDoS tipo Flooding

La naturaleza distribuida de los ataques DDoS los hace extremadamente difícil de combatir o rastrear. Los atacantes generalmente usan direcciones IP falsas para ocultar su verdadera identidad, lo que hace que el rastreo de los ataques sea incluso más difícil. Además, existen vulnerabilidades de seguridad en muchos hosts de Internet que los intrusos pueden explotar y los incidentes de ataques que tienen como objetivo la capa de aplicación aumentan rápidamente. Los ataques DDoS Flooding pueden ser clasificados en dos categorías basados en el nivel de protocolo, como se ilustra en la tabla 42.

<table>
<tr><td rowspan="4">A nivel capa transporte</td><td colspan="3">Flooding</td></tr>
<tr><td colspan="3">Flooding basados explotación de protocolos</td></tr>
<tr><td colspan="3">Flooding basados en reflección</td></tr>
<tr><td colspan="3">Flooding basados en amplificación</td></tr>
<tr><td rowspan="7">A nivel aplicación</td><td colspan="3">Flooding</td></tr>
<tr><td rowspan="6">HTTP</td><td colspan="2">Session</td></tr>
<tr><td colspan="2">Request</td></tr>
<tr><td rowspan="2">Asymmetric</td><td>Multiple HTTP get/post</td></tr>
<tr><td>Faulty Application</td></tr>
<tr><td rowspan="3">Slow Request Response</td><td>Slow Loris</td></tr>
<tr><td>HTTP Fragmentation</td></tr>
<tr><td>Slowpost</td></tr>
</table>

- **Ataques DDoS Flooding a nivel capa / transporte**

Estos ataques se distinguen por utilizar paquetes con los protocolos TCP, UDP, ICMP y DNS. Hay 4 tipos de ataques en esta categoría:

  - **Flooding Ataques:** Los atacantes se concentran en alterar la conectividad legitima del usuario saturando el ancho de banda de la red de la víctima. Ejemplo: UDP Flood, DNS Flood.

- **Ataques Flooding explotación de protocolos:** Los atacantes explotan características específicas o bugs de algunos protocolos para consumir los recursos de la víctima. Ejemplo: TCP SYN Flood, TCP SYN-ACK Flood
- **Ataques flooding basados en reflección:** Los atacantes envían solicitudes suplantadas a servidores, en vez de enviarlas directamente a la víctima, y son reflejadas por los servidores quienes envían una respuesta a la víctima.
- **Ataques flooding basados en amplificación:** Los atacantes explotan los servicios para generar una gran cantidad de mensajes o múltiples mensajes. Los ataques de reflección y amplificación generalmente son usados en conjunto como es el smurf attack donde los atacantes envían IP suplantada (Reflection) a un gran número de reflectores explotando la característica broadcast de los paquetes (Amplificación).

- **Ataques DDoS Flooding a nivel aplicación**

  La característica de estos ataques consiste en alterar los servicios de los usuarios consumiendo los recursos del servidor como CPU, memoria, disco, ancho de banda, etc. Los ataques DDoS a nivel de aplicación generalmente consumen menos ancho de banda y son menos detectables comparados con los ataques volumétricos porque se asemejan mucho al tráfico normal.

  - **Ataques Flooding**

    Estos ataques utilizan las mismas técnicas de los ataques de nivel de red y transporte, que consiste en enviar solicitudes falsificadas a nivel de aplicación a un gran número de reflectores. Un ejemplo de este, consiste en enviar solicitudes DNS con IP suplantada, lo cual genera un gran volumen de respuesta a la IP víctima, lo cual lo satura.
  - **HTTP Flooding Attacks**

  Hay 4 tipos de ataques en esta categoria:

  - **Session Flooding Attacks**

    En este tipo de ataque, las tasas de solicitudes de sesión son más altas que las solicitudes de los usuarios legítimos, por los tanto, consume los recursos del servidor. Uno de los ataques más famosos en esta categoria es el ataque flooding HTTP get/post en donde los atacantes generan un gran número de solicitudes de HTTP válidas a un servidor web víctima.
  - **Ataques Flooding Request**

    En este tipo de ataque los atacantes envían sesiones que contienen más paquetes de lo normal. Uno de los ataques más conocidos en esta categoria es el de session flooding HTTP get/post, este ataque es una variación del ataque flooding HTTP que emplea la característica HTTP 1.1 que permite múltiples solicitudes dentro de una única sesión HTTP.
  - **Asymmetric Attacks**

    En este tipo de ataque, los atacantes envían sesiones que contienen solicitudes con alta carga de trabajo.

- **Multiple HTTP get/post**
  En este tipo de ataque, el atacante crea un paquete que contiene múltiples solicitudes. De esta manera puede realizar muchas solicitudes con poca cantidad de envío de paquetes.
- **Faulty Application**
  En este ataque, los atacantes se aprovechan del pobre diseño de la aplicación o integraciones inapropiadas con bases de datos.

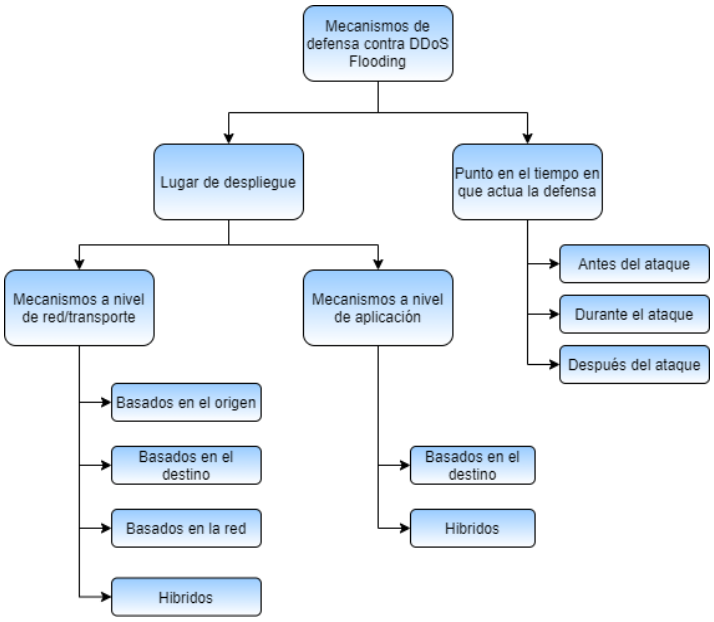
- **Slow Request Response Attacks**

En este tipo de ataque, los atacantes envían sesiones que contienen solicitudes con alta carga de trabajo.

- **Slow Loris Attack**
  Este tipo de ataque consiste en enviar solicitudes HTTP parciales que crecen continuamente, se actualizan lentamente y nunca se cierran. El ataque continuo hasta que todos los sockets disponibles son ocupados y que el servidor web sea inaccesible. Generalmente las direcciones de los atacantes no son suplantadas.
- **HTTP Fragmentation Attack**
  Similar al Slow Loris, el objetivo de este ataque es apagar un servidor realizando todas las conexiones HTTP por un largo tiempo sin ser descubierto. Los paquetes HTTP son fragmentados en pequeñas partes y envían cada fragmento lentamente hacia el servidor.
- **Slowpost Attack**
  Similar al Slow Loris, envía una solicitud post HTTP lentamente para apagar el servidor. El atacante envía una cabecera HTTP que define el campo “content-lenght” del mensaje post y envía una solicitud como tráfico normal. Entonces envían datos para completar el mensaje a una tasa de un byte cada dos minutos. Por lo tanto, el servidor espera por cada mensaje del body para ser completado.

B. Anexo: Clasificación mecanismos contra DoS

La figura describe la clasificación de los mecanismos de defensa basados en el lugar y en el punto en el tiempo en que la defensa actúa.



- **Clasificación Basada En Ubicación De Despliegue**
  - **Mecanismos de defensa contra flooding ataques nivel red/transporte**
    - **Mecanismos basados en el origen**: Los mecanismos basados en el origen se implementan cerca de las fuentes del ataque para evitar que los clientes de la red generen Ataques de inundación DDoS. Estos mecanismos pueden tener lugar ya sea en los enrutadores de borde de la red local de origen o en los enrutadores de acceso de un sistema autónomo (AS) que se conecta a los enrutadores de borde de origen. Varios dispositivos basados en el origen han sido diseñados para defenderse contra DDoS como: filtrado de entrada / salida, MULTOPS, TOPS, firewall inverso.
    - **Mecanismos basados en el destino:** En la defensa basada en los dispositivos de destino, la detección y respuesta se realiza principalmente en el destino del ataque (es decir, víctima). Existen varios mecanismos basados en el destino que pueden tener lugar ya sea en los enrutadores de borde o enrutadores de acceso de los destinos AS. Estos mecanismos pueden observar de cerca a la víctima, modelar su comportamiento y detectar cualquier anomalía. Algunos de los principales mecanismos de defensa DDoS basados en el destino son:

Mecanismos de rastreo IP, mecanismos marcado y filtrado de paquetes, Base de información de gestión (MIB), filtrado de conteo de saltos y caída de paquetes en función del nivel descongestión.

- **Mecanismos basados en la red** Estos mecanismos se implementan dentro de las redes y principalmente en los enrutadores del AS. Detectar tráfico de ataque y crear una respuesta adecuada para detenerlo en redes intermedias, es el objetivo ideal de esta categoría de mecanismos de defensa. Algunos de los principales mecanismos de defensa DDoS basados en la red son: Filtrado basado en paquetes, detección y filtrado malicioso en enrutadores.
- **Mecanismos híbridos:** En la mayoría de las categorías anteriores de mecanismos de defensa DDoS (basados en el origen, en el destino y en la red), no existe una fuerte cooperación entre los puntos. Además, la detección y la respuesta se realizan principalmente en el centro, ya sea por cada uno de los puntos de implementación (por ejemplo, mecanismos basados en el origen) o por algunos puntos responsables dentro del grupo de puntos de despliegue (por ejemplo, mecanismos basados en red).A diferencia de los mecanismos de defensa centralizados, los mecanismos híbridos de defensa se implementan en (o sus componentes son distribuidos en) múltiples ubicaciones como origen, destino o redes intermedias y generalmente hay cooperación entre los puntos de despliegue. Por ejemplo, la detección puede ser realizada en el lado de la víctima y la respuesta puede iniciarse y distribuido a otros nodos por la víctima. Algunos del mecanismo de defensa híbridos DDoS son: marcado de paquetes híbridos y mecanismos de estrangulamiento, Filtrado de tráfico, Stop It. La tabla muestra el resumen de los mecanismos de defensa basado en el lugar de despliegue.

	Accuracy (Exactitud)	**Escalabilidad**	**Rendimiento**	**Complejidad**	**Defensa Holística**
Basados en el origen	Bajo	Bajo	Moderado	Bajo	No
Basados en el destino	Alto	Bajo	Bueno	Bajo	No
Basados en la red	Bajo	Medio	Moderado	Medio	No
Híbridos	Medio	Medio-Alto	Muy bajo-Moderado	Medio-Alto	Si

- **Mecanismos de defensa contra ataques DDoS nivel de aplicación**
  - **Mecanismo basado en el destino (Lado del cliente):** Estos mecanismos de defensa son desplegados en el destino del ataque (es decir, víctima). Algunos de

estos mecanismos principales contra ataques DDoS a nivel de aplicación son: Defensa contra ataques de Reflexión / Amplificación, Escudo DDoS, detección de anomalías basadas en Hidden semi markov model.

- **Mecanismo Hibrido (Distribuido):** Estos mecanismos de defensa cooperan entre clientes y servidores para detectar y responder a los ataques. Por ejemplo, la detección se realiza en la víctima (web servidor / proxy inverso) y la respuesta se inicia y se distribuye en el lado del cliente por la víctima. Algunos de los mecanismos híbridos contra inundaciones DDoS a nivel de aplicación son: talk, diferenciar inundaciones DDoS bots de humanos, control de admisión y control de congestión, detección híbrida basada en indicadores de compromiso.

- **Clasificación Basada En El Punto En Que La Defensa Se Lleva A Cabo**
  - **Antes del ataque (Prevención de ataque):** El mejor punto para detener un ataque DDoS es en fase de lanzamiento. En otras palabras, la prevención de ataques es la mejor solución DDoS de defensa. Los mecanismos de prevención se pueden implementar en las fuentes de ataque, redes intermedias, destinos o una combinación de ellos. La mayoría de los mecanismos de prevención apuntan a corregir vulnerabilidades de seguridad (por ejemplo, protocolos inseguros, débiles esquemas de autenticación y sistemas informáticos vulnerables) que puede ser explotado para lanzar ataques DDoS. Hay algunos mecanismos de prevención general que deberían emplearse casi en todas partes (por ejemplo, servidores, hosts y redes intermedias) y en tantos lugares como sea posible, tanto por los hosts finales como por el servicio proveedores. Algunos de estos mecanismos generales de prevención son: Protección a prueba de fallas, asignación de recursos y contabilidad, mecanismos de reconfiguración, instalación de firewalls, etc.
  - **Durante el ataque (Detección de ataque):** El siguiente paso para defenderse de los ataques DDoS es la detección de ataques, que sucede durante el ataque. Estos mecanismos de detección también pueden ser desplegados en fuentes, redes intermedias, destinos o combinación de ellos.
  - **Después del ataque (Identificación de ataque y respuesta):** Después de que se detecta un ataque DDoS, el sistema de defensa debe identificar la fuente del ataque y bloquear el tráfico del ataque. Hoy, la mayoría de los mecanismos de respuesta DDoS no puede prevenir o detener completamente los ataques DDoS. Así minimizando el impacto del ataque y maximizando la disponibilidad de los servicios son el foco principal de todos los mecanismos de después del ataque.

- **Medidas para la detección de ataques**
  - **Hora de detección:** Es cuanto tiempo un sistema de detección necesita para identificar los paquetes de ataque. Una buena técnica de detección debe tener un tiempo de detección corto.
  - **Tasa de supervivencia de un paquete normal:** Es el porcentaje de paquetes normales que pueden llegar a la víctima en medio de un ataque DDoS. Un

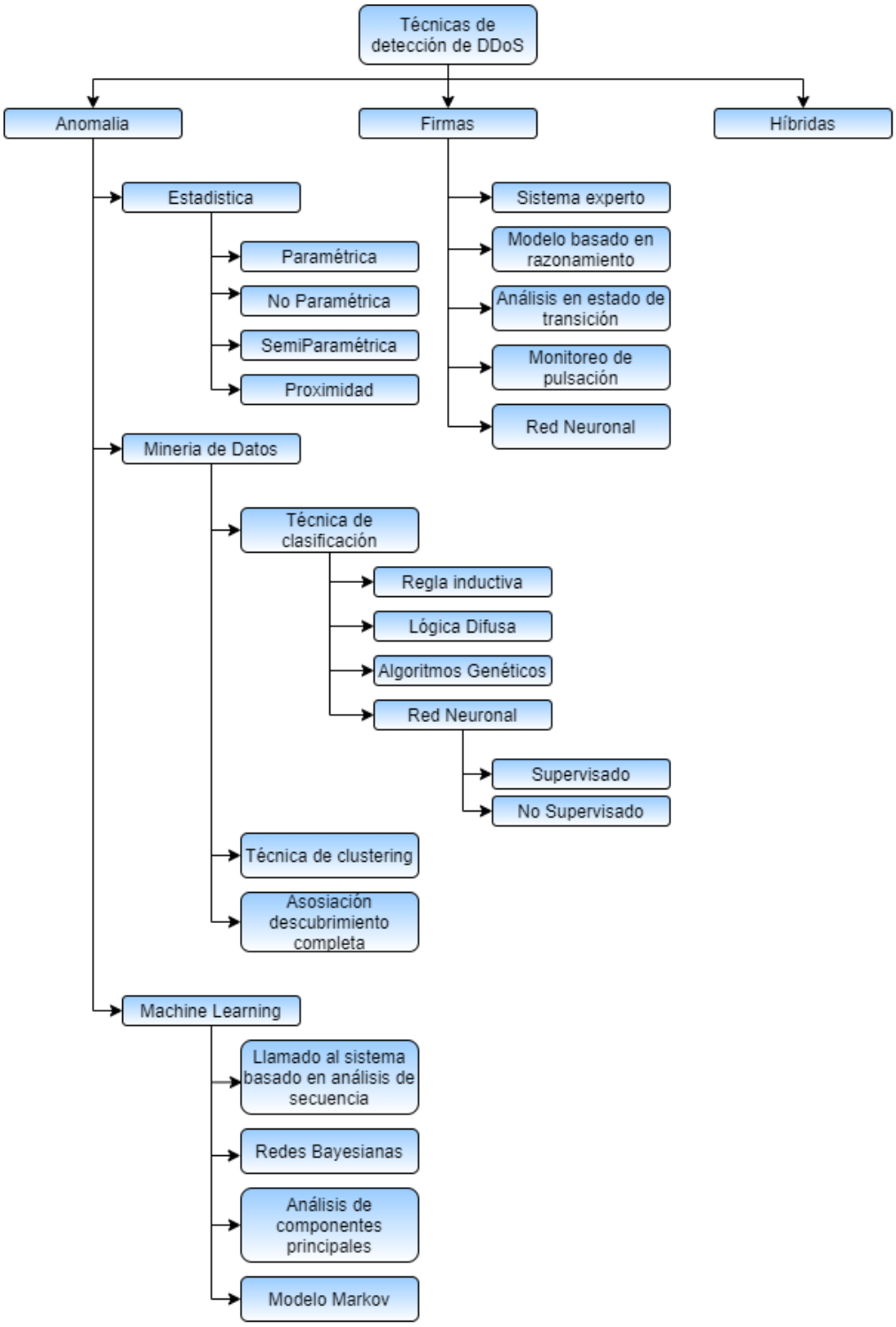
mecanismo de filtrado efectivo de paquetes debe ser capaz de lograr un alto NPSR (Normal Packet Survival Rate) durante un ataque DDoS.

- **Tasa de falsos positivos** Numero de paquetes clasificados como paquetes de ataque por el sistema de detección que son en realidad normales (negativo) sobre el Número total de paquetes confirmados como normales. Una buena técnica de detección debería tener una baja tasa de falsos positivos.
- **Tasa de falsos negativos**
  Numero de paquetes clasificados como normales (negativos) por el sistema de detección y que en realidad son ataques (positivos) sobre el número total de paquetes de ataques confirmados. Una buena técnica de detección debería tener una baja tasa de falsos negativos.

La tabla describe las características, ventajas y desventajas de los mecanismos de defensa

		Características	Ventajas	Desventajas
Centralizado	**Basado en el origen**	Detección y respuesta son desplegadas en los host origen	Las fuentes son distribuidas en diferentes dominios, por lo tanto, es difícil para cada uno de los orígenes detectar y filtrar flujos de ataques. Dificultar en diferenciar el ataque legítimo y ataque DDoS desde que el volumen del tráfico no sea tan grande.	El objetivo es detectar y responder el trafico atacante en el origen y antes de que agote todos los recursos.
	Basada en el destino	Detección y respuesta son desplegadas en los host origen	No pueden detectar y responder el ataque antes de que alcance la víctima y agote todos sus recursos.	Más fácil y menos costoso que otros mecanismos en detectar DDoS porque acceden al trafico agregado cerca del destino
	Basados en red	Detección y respuesta son desplegadas en las redes intermedias	Alto almacenamiento y procesamiento por parte de los routers	El objetivo es detectar y responder en el origen tan pronto sea posible
Distribuido	**Hibrido**	Detección y respuesta son desplegadas en varios lugares. La detección generalmente ocurre en el destino y en redes intermedias y la respuesta ocurre en el origen	Complejidad y consumo debido a la comunicación entre componentes distribuidos sobre toda la internet.	Más robusto contra DDoS. Más recursos en varios niveles son disponibles para defender contra DDoS.

La siguiente figura muestra las técnicas de detección contra ataques de denegación de servicio clasificadas en 3 tipos: anomalías, firmas e hibridas.



La tabla describe las técnicas de detección de ataques de denegación de servicio basados en anomalías y firmas.

Sistema de detección basada en anomalías	1. Habilidad para detectar ataques desconocidos	1.Complejidad intrínseca de un sistema
	2. Habilidad para detectar ataques de día Zero	2. Alto porcentaje de falsas alarmas, porque es difícil para los datos de entrenamiento proveer todo tipo de comportamiento de trafico normal. Como resultado el trafico legitimo puede ser clasificado como trafico atacante.
	3. Habilidad para detectar ataques internos.	
	4. Como los perfiles de actividad normal son personalizados para cada sistema, se hace muy difícil para el atacante saber con certeza que actividades puede realizar sin ser detectado.	3. Los malhechores quienes saben que están siendo monitoreados pueden entrenar tales sistemas sobre el tiempo donde el comportamiento intrusivo es considerado normal.
		4. Este método no ha ganado suficiente popularidad aún. Es un tema aún en investigación.
Sistema de detección basado en firmas	1. Los ataques pueden ser detectados limpiamente.	1. Firmas requeridas deben ser definidas por todos los posibles ataques. Por lo tanto requiere actualizaciones constantes para mantener la base de datos de firmas actualizada.
	2. Bajo porcentaje de falsas alarmas.	
	3. Comparativamente más simple	2. Mantener el estado de información de las firmas en que una actividad intrusiva alcance múltiples eventos discretos.
	4. Fácil para el administrador del sistema determinar exactamente que ataques están sucediendo.	
	5. Este sistema de detección comienza a proteger el computador y las redes después de instalarlo.	
	6. El sistema de detección tiene un éxito comercial.	

C. Anexo: Ventajas de DeepLearning4J

Primero, la mayoría de las grandes empresas y grandes organizaciones gubernamentales dependen en gran medida de Java o de un sistema basado en JVM. Han realizado una gran inversión, que pueden aprovechar con IA basada en JVM. Java sigue siendo el lenguaje más utilizado en las empresas. Es el lenguaje de Hadoop, ElasticSearch, Hive, Lucene y Pig, que resultan útiles para los problemas de Deep Learning. Spark y Kafka están escritos en Scala, otro lenguaje JVM. Es decir, muchos programadores que resuelven problemas del mundo real podrían beneficiarse del Deep Learning, pero están separados de él por una barrera del lenguaje con 10 millones de desarrolladores, Java es el lenguaje de programación más grande del mundo.

En segundo lugar, Java y Scala son inherentemente más rápidos que Python. Cualquier cosa escrita en Python por sí sola, sin tener en cuenta su dependencia de Cython, será más lenta. Es cierto que la mayoría de las operaciones computacionalmente costosas se escriben en C o C ++. (Cuando se habla de operaciones, también se considera de cosas como cadenas y otras tareas relacionadas con los procesos de Deep Learning de nivel superior). La mayoría de los proyectos de Deep Learning que inicialmente se escriben en Python tendrán que reescribirse si se van a poner en producción. DeepLearning4j se basa en JavaCPP para llamar a C++ nativo precompilado desde Java, acelerando sustancialmente la velocidad de entrenamiento. Muchos programadores de Python optan por hacer un Deep Learning en Scala porque prefieren la tipificación estática y la programación funcional cuando trabajan con otros en una base de código compartido.

En tercer lugar, la falta de sólidas bibliotecas informáticas científicas de Java se puede resolver escribiéndolas, lo que se ha hecho con ND4J. ND4J se ejecuta en GPU distribuidas o GPU, y puede conectarse a través de una API Java o Scala.

Finalmente, Java es un lenguaje de red seguro que funciona de manera inherente multiplataforma en servidores Linux, escritorios Windows y OSX, teléfonos Android y en los sensores de baja memoria de Internet of things a través de Java incorporado. Mientras que Torch y Pylearn2 optimizan a través de C ++, lo que presenta dificultades para aquellos que intentan optimizarlo y mantenerlo, Java es un lenguaje de "escribir una vez, ejecutar en cualquier lugar" adecuado para empresas que necesitan utilizar el aprendizaje profundo en muchas plataformas. La tabla 47 muestra la lista de características de DeepLearning4J:

Componente	Descripción
Integraciones	• Spark • Hadoop/YARN • Model Import from Keras
APIs	• Scala • Java
Librerías	• ND4J: N-dimensional arrays for the JVM • LibND4J: Native CPU/GPU operations for ND4J • DataVec: Data preparation for DL4J • DeepLearning4j
Redes	• Restricted Boltzmann Machines • Convolutional nets • Recursive AutoEncoders • Recurrent nets: Long Short-Term Memory (LSTM) (including bi-directional LSTMs) • Deep-belief networks • Denoising and Stacked Denoising AutoEncoders • Deep AutoEncoders
Herramientas	• DataVec: Machine Learning Data Pipelines for Vectorization/Tensorization • Moving-window for images • Moving-window for text • Viterbi for sequential classification • Word2Vec • Bag-of-Words encoding for word count and TF-IDF • Doc2Vec for Paragraph Vectors • Constituency parsing • DeepWalk
Algoritmos de optimización	• Stochastic gradient descent • Stochastic gradient descent with line search • Conjugate gradient line search (c.f. Hinton 2006) • L-BFGS
Actualizadores	• SGD (Learning rate only) • Nesterovs momentum • Adagrad • RMSProp • Adam • AdaDelta
Hyperparameters	• Dropout (random ommission of feature detectors to prevent overfitting • Sparsity (force activations of sparse/rare inputs) • Adagrad (feature-specific Learning-rate optimization) • L1 and L2 regularization (weight decay) • Weight transforms (useful for Deep autoencoders) • Probability distribution manipulation for initial weight generation • Gradient normalization and clipping

Loss/objective functions	• MSE: Mean Squared Error: Linear Regression • EXPLL: Exponential log likelihood: Poisson Regression • XENT: Cross Entropy: Binary Classification • MCXENT: Multiclass Cross Entropy • RMSE_XENT: RMSE Cross Entropy • SQUARED_LOSS: Squared Loss • NEGATIVELOGLIKELIHOOD: Negative Log Likelihood
Funciones de activación	• ReLU • Leaky ReLU • Tanh • Sigmoid • Hard Tanh • Softmax • Identity • ELU: Exponential Linear Units • Softsign • Softplus

D. Anexo: Código fuente en java del entrenamiento del modelo en DL4J

```
package com.proyectogrado.itm;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.LineNumberReader;
import java.io.PrintWriter;

import org.datavec.api.records.reader.RecordReader;
import org.datavec.api.records.reader.impl.csv.CSVRecordReader;
import org.datavec.api.records.reader.impl.transform.TransformProcessRecordReader;
import org.datavec.api.split.FileSplit;
import org.datavec.api.transform.TransformProcess;
import org.datavec.api.transform.schema.Schema;
import org.DeepLearning4j.Datasets.datavec.RecordReaderDataSetIterator;
import org.nd4j.evaluation.classification.Evaluation;
import org.DeepLearning4j.nn.conf.MultiLayerConfiguration;
import org.DeepLearning4j.nn.conf.NeuralNetConfiguration;
import org.DeepLearning4j.nn.conf.layers.DenseLayer;
import org.DeepLearning4j.nn.conf.layers.DropoutLayer;
import org.DeepLearning4j.nn.conf.layers.OutputLayer;
import org.DeepLearning4j.nn.multilayer.MultiLayerNetwork;
import org.DeepLearning4j.nn.weights.WeightInit;
import org.DeepLearning4j.optimize.listeners.ScoreIterationListener;
import org.DeepLearning4j.util.ModelSerializer;
```

```
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.Dataset.DataSet;
import org.nd4j.linalg.Dataset.SplitTestAndTrain;
import org.nd4j.linalg.Dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.Dataset.api.preprocessor.DataNormalization;
import org.nd4j.linalg.Dataset.api.preprocessor.NormalizerStandardize;
import org.nd4j.linalg.Learning.config.Adam;
import org.nd4j.linalg.Learning.config.Sgd;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class ClassifierFFNDoS2020All {

    private static Logger log = LoggerFactory.getLogger(ClassifierFFNDoS2020All.class);

    public static void main(String[] args) throws  Exception {

        //Leer el Dataset en un archivo separado por comas
        int numLinesToSkip = 1;
        char delimiter = ',';
        RecordReader recordReader = new CSVRecordReader(numLinesToSkip,delimiter);


        //La cantidad de archivos que se desea extraer de cada archivo del Dataset
        int cantidadRegistrosPorFile = 20000;

        //Metodo para unificar todos los archivos en uno solo
        File totalFile = cargaArchivos(cantidadRegistrosPorFile);

        recordReader.initialize(new FileSplit(totalFile));

        //Variables para definir la cantidad de features, clases, tamano del batch y epocas de
entrenamiento
        int labelIndex = 22;     //0-22 features
        int numClasses = 13;
        int batchSize = cantidadRegistrosPorFile*11;
        int epochs = 100;


        // Definir un tipo de dato para cada columna del Dataset
        Schema inputDataSchema = new Schema.Builder()
                .addColumnsInteger("Unnamed: 0")
                .addColumnsString("Flow ID")
                .addColumnsString("Source IP")
                .addColumnsInteger("Source Port")
                .addColumnsString("Destination IP")
                .addColumnsInteger("Destination Port")
                .addColumnsInteger("Protocol")
                .addColumnsString("Timestamp")
                .addColumnsInteger("Flow Duration")
                .addColumnsInteger("Total Fwd Packets")

                .addColumnDouble("Total Backward Packets")
                .addColumnDouble("Total Length of Fwd Packets")
                .addColumnDouble("Total Length of Bwd Packets")
                .addColumnDouble("Fwd Packet Length Max")
                .addColumnDouble("Fwd Packet Length Min")
                .addColumnDouble("Fwd Packet Length Mean")
                .addColumnDouble("Fwd Packet Length Std")
                .addColumnDouble("Bwd Packet Length Max")
                .addColumnDouble("Bwd Packet Length Min")
                .addColumnDouble("Bwd Packet Length Mean")

                .addColumnDouble("Bwd Packet Length Std")
                .addColumnString("Flow Bytes/s")
                .addColumnString("Flow Packets/s")
                .addColumnDouble("Flow IAT Mean")
                .addColumnDouble("Flow IAT Std")
                .addColumnDouble("Flow IAT Max")
                .addColumnDouble("Flow IAT Min")
                .addColumnDouble("Fwd IAT Total")
                .addColumnDouble("Fwd IAT Mean")
                .addColumnDouble("Fwd IAT Std")

                .addColumnDouble("Fwd IAT Max")
```

```
            .addColumnDouble("Fwd IAT Min")
            .addColumnDouble("Bwd IAT Total")
            .addColumnDouble("Bwd IAT Mean")
            .addColumnDouble("Bwd IAT Std")
            .addColumnDouble("Bwd IAT Max")
            .addColumnDouble("Bwd IAT Min")
            .addColumnDouble("Fwd PSH Flags")
            .addColumnDouble("Bwd PSH Flags")
            .addColumnDouble("Fwd URG Flags")

            .addColumnDouble("Bwd URG Flags")
            .addColumnDouble("Fwd Header Length")
            .addColumnDouble("Bwd Header Length")
            .addColumnDouble("Fwd Packets/s")
            .addColumnDouble("Bwd Packets/s")
            .addColumnDouble("Min Packet Length")
            .addColumnDouble("Max Packet Length")
            .addColumnDouble("Packet Length Mean")
            .addColumnDouble("Packet Length Std")
            .addColumnDouble("Packet Length Variance")

            .addColumnDouble("FIN Flag Count")
            .addColumnDouble("SYN Flag Count")
            .addColumnDouble("RST Flag Count")
            .addColumnDouble("PSH Flag Count")
            .addColumnDouble("ACK Flag Count")
            .addColumnDouble("URG Flag Count")
            .addColumnDouble("CWE Flag Count")
            .addColumnDouble("ECE Flag Count")
            .addColumnDouble("Down/Up Ratio")
            .addColumnDouble("Average Packet Size")

            .addColumnDouble("Avg Fwd Segment Size")
            .addColumnDouble("Avg Bwd Segment Size")
            .addColumnDouble("Fwd Header Length.1")
            .addColumnDouble("Fwd Avg Bytes/Bulk")
            .addColumnDouble("Fwd Avg Packets/Bulk")
            .addColumnDouble("Fwd Avg Bulk Rate")
            .addColumnDouble("Bwd Avg Bytes/Bulk")
            .addColumnDouble("Bwd Avg Packets/Bulk")
            .addColumnDouble("Bwd Avg Bulk Rate")
            .addColumnDouble("Subflow Fwd Packets")

            .addColumnDouble("Subflow Fwd Bytes")
            .addColumnDouble("Subflow Bwd Packets")
            .addColumnDouble("Subflow Bwd Bytes")
            .addColumnDouble("Init_Win_bytes_forward")
            .addColumnDouble("Init_Win_bytes_backward")
            .addColumnDouble("act_data_pkt_fwd")
            .addColumnDouble("min_seg_size_forward")
            .addColumnDouble("Active Mean")
            .addColumnDouble("Active Std")
            .addColumnDouble("Active Max")

            .addColumnDouble("Active Min")
            .addColumnDouble("Idle Mean")
            .addColumnDouble("Idle Std")
            .addColumnDouble("Idle Max")
            .addColumnDouble("Idle Min")
            .addColumnString("SimillarHTTP")
            .addColumnDouble("Inbound")
           .addColumnCategorical("Label", "BENIGN","DrDoS_NTP",
            "DrDoS_DNS","DrDoS_LDAP","DrDoS_MSSQL",
           "DrDoS_NetBIOS","DrDoS_SNMP","DrDoS_SSDP",
            "DrDoS_UDP","UDP-lag","Syn","TFTP","WebDDoS")
            .build();

    //Preprocesamiento para remover los features del Dataset que no se necesitan
    TransformProcess tp = new TransformProcess.Builder(inputDataSchema)
            .categoricalToInteger("Label")
            .removeColumns("Unnamed: 0")
            .removeColumns("Flow ID")
            .removeColumns("Source IP")
            .removeColumns("Source Port")
            .removeColumns("Destination IP")


            .removeColumns("Timestamp")
```

```
.removeColumns("Total Fwd Packets")

.removeColumns("Total Backward Packets")
.removeColumns("Total Length of Fwd Packets")
.removeColumns("Total Length of Bwd Packets")


.removeColumns("Fwd Packet Length Mean")

.removeColumns("Bwd Packet Length Max")
.removeColumns("Bwd Packet Length Min")
.removeColumns("Bwd Packet Length Mean")
.removeColumns("Bwd Packet Length Std")
.removeColumns("Flow Bytes/s")
.removeColumns("Flow Packets/s")

.removeColumns("Flow IAT Std")

.removeColumns("Flow IAT Min")
.removeColumns("Fwd IAT Total")

.removeColumns("Fwd IAT Std")


.removeColumns("Bwd IAT Total")
.removeColumns("Bwd IAT Mean")
.removeColumns("Bwd IAT Std")
.removeColumns("Bwd IAT Max")
.removeColumns("Bwd IAT Min")
.removeColumns("Fwd PSH Flags")
.removeColumns("Bwd PSH Flags")
.removeColumns("Fwd URG Flags")
.removeColumns("Bwd URG Flags")

.removeColumns("Bwd Header Length")

.removeColumns("Bwd Packets/s")


.removeColumns("Packet Length Mean")

.removeColumns("Packet Length Variance")
.removeColumns("FIN Flag Count")
.removeColumns("SYN Flag Count")
.removeColumns("RST Flag Count")
.removeColumns("PSH Flag Count")

.removeColumns("URG Flag Count")
.removeColumns("CWE Flag Count")
.removeColumns("ECE Flag Count")
.removeColumns("Down/Up Ratio")

.removeColumns("Avg Fwd Segment Size")
.removeColumns("Avg Bwd Segment Size")

.removeColumns("Fwd Avg Bytes/Bulk")
.removeColumns("Fwd Avg Packets/Bulk")
.removeColumns("Fwd Avg Bulk Rate")
.removeColumns("Bwd Avg Bytes/Bulk")
.removeColumns("Bwd Avg Packets/Bulk")
.removeColumns("Bwd Avg Bulk Rate")
.removeColumns("Subflow Fwd Packets")

.removeColumns("Subflow Bwd Packets")
.removeColumns("Subflow Bwd Bytes")

.removeColumns("Init_Win_bytes_backward")
.removeColumns("act_data_pkt_fwd")

.removeColumns("Active Mean")
.removeColumns("Active Std")
.removeColumns("Active Max")
.removeColumns("Active Min")
.removeColumns("Idle Mean")
.removeColumns("Idle Std")
.removeColumns("Idle Max")
.removeColumns("Idle Min")
```

```
                .removeColumns("SimillarHTTP")
                .removeColumns("Inbound")



                .build();

        //Objeto REader que luego sera utilizado para iterar
        TransformProcessRecordReader processRecordReader = new
TransformProcessRecordReader(recordReader, tp);

        //Objeto para recorrer fila por fila en el entrenamiento
        DataSetIterator iterator = new
RecordReaderDataSetIterator(processRecordReader,batchSize,labelIndex,numClasses);

        DataSet allData;
        allData = iterator.next();

        //Revolver los datos para evitar el sobreentrenamiento
        allData.shuffle();

        //Definir que porcentaje de los datos sera para entrenamiento y test
        SplitTestAndTrain testAndTrain = allData.splitTestAndTrain(0.9);  //90% de los datos para
entrenamiento

        //Se divide el Dataset en 2 partes: training y test
        DataSet trainingData = testAndTrain.getTrain();
        DataSet testData = testAndTrain.getTest();


        //Proceso de normalizacion
        DataNormalization normalizer = new NormalizerStandardize();
        normalizer.fit(trainingData);
        normalizer.transform(trainingData);
        normalizer.transform(testData);


        final int numInputs = 22;  //La cantidad de features label
        int outputNum = 13;        //La cantidad de classes

        //Iniciaizar semilla
        long seed = 7;

        //La configuracion de la red con sus hyperparametros
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(seed)
                .activation(Activation.TANH)
                .weightInit(WeightInit.XAVIER)
                .updater(new Adam(0.01))
                .list()
                .layer(new DenseLayer.Builder().nIn(numInputs).nOut(numInputs)
                    .build())
                .layer(new DenseLayer.Builder().nIn(numInputs).nOut(128)
                        .build())
                .layer(new DropoutLayer.Builder(0.5).build()
                        )
                .layer(new DenseLayer.Builder().nIn(128).nOut(256)
                        .build())
                .layer(new DenseLayer.Builder().nIn(256).nOut(512)
                        .build())
                .layer(new DenseLayer.Builder().nIn(512).nOut(256)
                        .build())
                .layer(new DenseLayer.Builder().nIn(256).nOut(128)
                        .build())
                .layer(new DropoutLayer.Builder(0.5).build()
                        )
                .layer( new OutputLayer.Builder(LossFunctions.LossFunction.MCXENT)
                    .nIn(128).nOut(outputNum).build())
                .build();


        //Asignar la configuracion al modelo
        MultiLayerNetwork model = new MultiLayerNetwork(conf);
        model.init();
        model.setListeners(new ScoreIterationListener(1));

        //Iniciar el entrenamiento
        for(int i=0; i<epochs; i++ ) {
```

```
            model.fit(trainingData);
        }

        //Guardar el modelo
        File location = new File("ModelFNNDoS2020.zip");
        boolean saveUpdater =true;
        ModelSerializer.writeModel(model, location, saveUpdater);

        //Evaluar el modelo
        Evaluation eval = new Evaluation(12);
        INDArray output = model.output(testData.getFeatures());
        eval.eval(testData.getLabels(), output);
        log.info(eval.stats());
        System.out.println(eval.stats());

    }

    //Metodo para contar lineas del archivo
    private static int countLines(File aFile) throws IOException {
        LineNumberReader reader = null;
        try {
            reader = new LineNumberReader(new FileReader(aFile));
            while ((reader.readLine()) != null);
            return reader.getLineNumber();
        } catch (Exception ex) {
            return -1;
        } finally {
            if(reader != null)
                reader.close();
        }
    }

    //Metodo para leer los Dataset y crear un unico Dataset
    private static File cargaArchivos(int cantidadLineas) throws Exception {

          File DrDoS_DNSFile     = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_DNS.csv");
          File DrDoS_LDAPFile    = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_LDAP.csv");
          File DrDoS_MSSQLFile   = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_MSSQL.csv");
          File DrDoS_NetBIOSFile = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_NetBIOS.csv");
          File DrDoS_NTPFile     = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_NTP.csv");
          File DrDoS_SNMPFile    = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_SNMP.csv");
          File DrDoS_SSDPFile    = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_SSDP.csv");
          File DrDoS_UDPFile     = new File("/home/test/Datasets/DoS2020/01-12/DrDoS_UDP.csv");
          File SynFile           = new File("/home/test/Datasets/DoS2020/01-12/Syn.csv");
          File TFTPFile          = new File("/home/test/Datasets/DoS2020/01-12/TFTP.csv");
          File UDPLagFile        = new File("/home/test/Datasets/DoS2020/01-12/UDPLag.csv");


          File fullFile = new File("/home/test/Filetemp.csv");

            fullFile.createNewFile();

              BufferedReader
brDNS,bLDAP,brMSSQL,brNetBIOS,brNTP,brSNMP,brSSDP,brUDP,brSyn,brTFTP,brUDPLag = null;
              PrintWriter pw = null;

              try {
                  brDNS     = new BufferedReader(new FileReader( DrDoS_DNSFile ));
                  bLDAP     = new BufferedReader(new FileReader( DrDoS_LDAPFile ));
                  brMSSQL   = new BufferedReader(new FileReader( DrDoS_MSSQLFile ));
                  brNetBIOS = new BufferedReader(new FileReader( DrDoS_NetBIOSFile ));
                  brNTP     = new BufferedReader(new FileReader( DrDoS_NTPFile ));
                  brSNMP    = new BufferedReader(new FileReader( DrDoS_SNMPFile ));
                  brSSDP    = new BufferedReader(new FileReader( DrDoS_SSDPFile ));
                  brUDP     = new BufferedReader(new FileReader( DrDoS_UDPFile ));
                  brSyn     = new BufferedReader(new FileReader( SynFile ));
                  brTFTP    = new BufferedReader(new FileReader( TFTPFile ));
                  brUDPLag  = new BufferedReader(new FileReader( UDPLagFile ));
                  pw =  new PrintWriter(new FileWriter( fullFile ));

                  String line;

                  brDNS.readLine();
                  bLDAP.readLine();
                  brMSSQL.readLine();
                  brNetBIOS.readLine();
                  brNTP.readLine();
                  brSNMP.readLine();
```

```
                brSSDP.readLine();
                brUDP.readLine();
                brSyn.readLine();
                brTFTP.readLine();
                brUDPLag.readLine();

                for (int i = 0; i < cantidadLineas; i++) {
                    line = brDNS.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = bLDAP.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brMSSQL.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brNetBIOS.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brNTP.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brSNMP.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brSSDP.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brUDP.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brSyn.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brTFTP.readLine();
                    pw.println(line);
                }
                for (int i = 0; i < cantidadLineas; i++) {
                    line = brUDPLag.readLine();
                    pw.println(line);
                }

                brDNS.close();
                bLDAP.close();
                brMSSQL.close();
                brNetBIOS.close();
                brNTP.close();
                brSNMP.close();
                brSSDP.close();
                brUDP.close();
                brSyn.close();
                brTFTP.close();
                brUDPLag.close();
                pw.close();
            }catch (Exception e) {
            e.printStackTrace();
            }

        System.out.println("Creacion archivo Full...DONE");
        return fullFile;

    }


}
```

E. Anexo: Lista de modelos propuestos

Modelo 1 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		110.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Registros por archivo		10.000	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	13

Modelo 2 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		110.000	
Funcion activación		softmax	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DenseLayer	Input	128
		Output	256
Layer 4	DenseLayer	Input	256
		Output	512
Layer 5	DenseLayer	Input	512
		Output	256
Layer 6	DenseLayer	Input	256
		Output	128
Layer 7	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 3 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 4 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	512
Layer 8	DenseLayer	Input	512
		Output	256
Layer 9	DenseLayer	Input	256
		Output	128
Layer 10	DropoutLayer	0.5	
Layer 11	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 5 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		550.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	512
Layer 8	DenseLayer	Input	512
		Output	256
Layer 9	DenseLayer	Input	256
		Output	128
Layer 10	DropoutLayer	0.5	
Layer 11	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 6 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		1.110.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	512
Layer 8	DenseLayer	Input	512
		Output	256
Layer 9	DenseLayer	Input	256
		Output	128
Layer 10	DropoutLayer	0.5	
Layer 11	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 7 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	512
Layer 8	DenseLayer	Input	512
		Output	256
Layer 9	DenseLayer	Input	256
		Output	128
Layer 10	DropoutLayer	0.5	
Layer 11	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 8 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	512
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	512
		Output	256
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	256
		Output	128
Layer 15	DropoutLayer	0.5	
Layer 16	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 9 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		550.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	512
Layer 8	DenseLayer	Input	512
		Output	256
Layer 9	DenseLayer	Input	256
		Output	128
Layer 10	DropoutLayer	0.5	
Layer 11	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 10 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	2048
Layer 8	DenseLayer	Input	2048
		Output	1024
Layer 9	DenseLayer	Input	1024
		Output	512
Layer 10	DenseLayer	Input	512
		Output	256
Layer 11	DenseLayer	Input	256
		Output	128
Layer 12	DropoutLayer	0.5	
Layer 13	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 11 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		550.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	2048
Layer 8	DenseLayer	Input	2048
		Output	1024
Layer 9	DenseLayer	Input	1024
		Output	512
Layer 10	DenseLayer	Input	512
		Output	256
Layer 11	DenseLayer	Input	256
		Output	128
Layer 12	DropoutLayer	0.5	
Layer 13	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 12 Propuesto			
Seed		7	
Epocas		200	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	2048
Layer 8	DenseLayer	Input	2048
		Output	1024
Layer 9	DenseLayer	Input	1024
		Output	512
Layer 10	DenseLayer	Input	512
		Output	256
Layer 11	DenseLayer	Input	256
		Output	128
Layer 12	DropoutLayer	0.5	
Layer 13	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 13 Propuesto			
Seed		7	
Epocas		200	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	2048
Layer 8	DenseLayer	Input	2048
		Output	4096
Layer 9	DenseLayer	Input	4096
		Output	2048
Layer 10	DenseLayer	Input	2048
		Output	1024
Layer 11	DenseLayer	Input	1024
		Output	512
Layer 12	DenseLayer	Input	512
		Output	256
Layer 13	DenseLayer	Input	256
		Output	128
Layer 14	DropoutLayer	0.5	
Layer 15	OutputLayer	Input	128
		Output	13
		Function	SOFTMAX

Modelo 14 Propuesto			
Seed		7	
Epocas		500	
Tamaño Batch		440.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	1024
Layer 7	DenseLayer	Input	1024
		Output	2048
Layer 8	DenseLayer	Input	2048
		Output	4096
Layer 9	DenseLayer	Input	4096
		Output	2048
Layer 10	DenseLayer	Input	2048
		Output	1024
Layer 11	DenseLayer	Input	1024
		Output	512
Layer 12	DenseLayer	Input	512
		Output	256
Layer 13	DenseLayer	Input	256
		Output	128
Layer 14	DropoutLayer	0.5	
Layer 15	OutputLayer	Input	128
		Output	13
		Activation Function	SOFTMAX

Modelo 15 Propuesto			
Seed		7	
Epocas		100	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 16 Propuesto			
Seed		7	
Epocas		200	
Tamaño Batch		220.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 SOFTMAX

Modelo 17 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		1.100.000	
Tamaño Batch		50.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 18 Propuesto			
Seed		7	
Epocas		100	
Tamaño datos		1.100.000	
Tamaño Batch		100.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 19 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		1.100.000	
Tamaño Batch		100.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 20 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		220.000	
Tamaño Batch		5.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 21 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		1.100.000	
Tamaño Batch		100.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 22 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		110.000	
Tamaño Batch		10.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 23 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		110.000	
Tamaño Batch		10.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 24 Propuesto			
Seed		7	
Epocas		100	
Tamaño datos		220.000	
Tamaño Batch		200.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 25 Propuesto			
Seed		7	
Epocas		100	
Tamaño datos		1.100.000	
Tamaño Batch		100.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 18	DenseLayer	Input	256
		Output	128
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Nota: Cada capa tiene una capa dropout de 0.5

Modelo 26 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		1.100.000	
Tamaño Batch		100.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 27 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		330.000	
Tamaño Batch		330.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

Modelo 28 Propuesto			
Seed		7	
Epocas		500	
Tamaño datos		385.000	
Tamaño Batch		385.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	13 softmax

En este punto, se comenzó a realizar una clasificación binaria y no multiclase, con el objetivo de ver los resultados con solo 2 clases.

Modelo 29 Propuesto			
Seed		7	
Epocas		50	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.08	
Features		22	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	512
Layer 12	DropoutLayer	0.5	
Layer 13	DenseLayer	Input	512
		Output	256
Layer 14	DropoutLayer	0.5	
Layer 15	DenseLayer	Input	256
		Output	128
Layer 16	DropoutLayer	0.5	
Layer 17	OutputLayer	Input	128
		Output	2 softmax

Modelo 30 Propuesto			
Seed		7	
Epocas		50	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		22	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	512
Layer 12	DropoutLayer	0.5	
Layer 13	DenseLayer	Input	512
		Output	256
Layer 14	DropoutLayer	0.5	
Layer 15	DenseLayer	Input	256
		Output	128
Layer 16	DropoutLayer	0.5	
Layer 17	OutputLayer	Input	128
		Output	2 softmax

Modelo 31 Propuesto			
Seed		7	
Epocas		100	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		22	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	512
Layer 12	DropoutLayer	0.5	
Layer 13	DenseLayer	Input	512
		Output	256
Layer 14	DropoutLayer	0.5	
Layer 15	DenseLayer	Input	256
		Output	128
Layer 16	DropoutLayer	0.5	
Layer 17	OutputLayer	Input	128
		Output	2 softmax

Modelo 32 Propuesto			
Seed		7	
Epocas		500	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		22	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	512
Layer 12	DropoutLayer	0.5	
Layer 13	DenseLayer	Input	512
		Output	256
Layer 14	DropoutLayer	0.5	
Layer 15	DenseLayer	Input	256
		Output	128
Layer 16	DropoutLayer	0.5	
Layer 17	OutputLayer	Input	128
		Output	2 softmax

Modelo 33 Propuesto			
Seed		7	
Epocas		500	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		22	
Layer 1	InputLayer	Input	22
		Output	22
Layer 2	DenseLayer	Input	22
		Output	128
Layer 3	DenseLayer	Input	128
		Output	256
Layer 4	DenseLayer	Input	256
		Output	512
Layer 5	DenseLayer	Input	512
		Output	1024
Layer 6	DenseLayer	Input	1024
		Output	2048
Layer 7	DenseLayer	Input	2048
		Output	1024
Layer 8	DenseLayer	Input	1024
		Output	512
Layer 9	DenseLayer	Input	512
		Output	256
Layer 10	DenseLayer	Input	256
		Output	128
Layer 11	OutputLayer	Input	128
		Output	2 softmax

Modelo 34 Propuesto			
Seed		7	
Epocas		500	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 19	DropoutLayer	0.5	
Layer 20	OutputLayer	Input	128
		Output	2 softmax

Modelo 35 Propuesto			
Seed		7	
Epocas		1000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DropoutLayer	0.5	
Layer 6	DenseLayer	Input	256
		Output	512
Layer 7	DropoutLayer	0.5	
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 9	DropoutLayer	0.5	
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 11	DropoutLayer	0.5	
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 13	DropoutLayer	0.5	
Layer 14	DenseLayer	Input	1024
		Output	512
Layer 15	DropoutLayer	0.5	
Layer 16	DenseLayer	Input	512
		Output	256
Layer 17	DropoutLayer	0.5	
Layer 18	DenseLayer	Input	256
		Output	128
Layer 20	OutputLayer	Input	128
		Output	2 softmax

Modelo 36 Propuesto			
Seed		7	
Epocas		2000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	1024
		Output	2048
Layer 14	DenseLayer	Input	1024
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	512
Layer 20	DenseLayer	Input	512
		Output	256
Layer 22	DenseLayer	Input	256
		Output	128
Layer 24	OutputLayer	Input	128
		Output	2 softmax
Hay una capa dropout de 0.5 entre cada capa.			

Modelo 37 Propuesto			
Seed		7	
Epocas		500	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	1024
		Output	2048
Layer 14	DenseLayer	Input	1024
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	512
Layer 20	DenseLayer	Input	512
		Output	256
Layer 22	DenseLayer	Input	256
		Output	128
Layer 24	OutputLayer	Input	128
		Output	2 softmax

Modelo 38 Propuesto			
Seed		7	
Epocas		1000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	4096
Layer 14	DenseLayer	Input	4096
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	2048
Layer 20	DenseLayer	Input	2048
		Output	1024
Layer 22	DenseLayer	Input	1024
		Output	512
Layer 24	DenseLayer	Input	512
		Output	256
Layer 26	DenseLayer	Input	256
		Output	128
Layer 28	OutputLayer	Input	128
		Output	2 softmax

Modelo 39 Propuesto			
Seed		7	
Epocas		2000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.0000001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	4096
Layer 14	DenseLayer	Input	4096
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	2048
Layer 20	DenseLayer	Input	2048
		Output	1024
Layer 22	DenseLayer	Input	1024
		Output	512
Layer 24	DenseLayer	Input	512
		Output	256
Layer 26	DenseLayer	Input	256
		Output	128
Layer 28	OutputLayer	Input	128
		Output	2 softmax

Modelo 40 Propuesto			
Seed		7	
Epocas		1000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 14	DenseLayer	Input	1024
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 20	DenseLayer	Input	1024
		Output	512
Layer 22	DenseLayer	Input	512
		Output	256
Layer 24	DenseLayer	Input	256
		Output	128
Layer 26	OutputLayer	Input	128
		Output	2 softmax

Modelo 41 Propuesto			
Seed		7	
Epocas		100	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		sigmoid	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	4096
Layer 14	DenseLayer	Input	4096
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	2048
Layer 20	DenseLayer	Input	2048
		Output	1024
Layer 22	DenseLayer	Input	1024
		Output	512
Layer 24	DenseLayer	Input	512
		Output	256
Layer 26	DenseLayer	Input	256
		Output	128
Layer 28	OutputLayer	Input	128
		Output	2 sigmoid

Modelo 42 Propuesto			
Seed		7	
Epocas		1000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		sigmoid	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	1024
Layer 14	DenseLayer	Input	1024
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 20	DenseLayer	Input	1024
		Output	512
Layer 22	DenseLayer	Input	512
		Output	256
Layer 24	DenseLayer	Input	256
		Output	128
Layer 26	OutputLayer	Input	128
		Output Function	2 sigmoid

Modelo 43 Propuesto			
Seed		7	
Epocas		1000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		sigmoid	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	4096
Layer 14	DenseLayer	Input	4096
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	512
Layer 20	DenseLayer	Input	512
		Output	256
Layer 22	DenseLayer	Input	256
		Output	128
Layer 24	OutputLayer	Input	128
		Output Function	2 sigmoid

Modelo 44 Propuesto			
Seed		7	
Epocas		200	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.01	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 4	DenseLayer	Input	128
		Output	256
Layer 6	DenseLayer	Input	256
		Output	512
Layer 8	DenseLayer	Input	512
		Output	1024
Layer 10	DenseLayer	Input	1024
		Output	2048
Layer 12	DenseLayer	Input	2048
		Output	4096
Layer 14	DenseLayer	Input	4096
		Output	2048
Layer 16	DenseLayer	Input	2048
		Output	1024
Layer 18	DenseLayer	Input	1024
		Output	512
Layer 20	DenseLayer	Input	512
		Output	256
Layer 22	DenseLayer	Input	256
		Output	128
Layer 24	OutputLayer	Input	128
		Output Function	2 Sigmoid

Modelo 45 Propuesto			
Seed		7	
Epocas		1000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.0001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output Function	2 Sigmoid

Modelo 46 Propuesto			
Seed		7	
Epocas		2000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.0001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2

Modelo 47 Propuesto			
Seed		7	
Epocas		100000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2

Modelo 48 Propuesto			
Seed		7	
Epocas		200.000	
Tamaño datos		44.000	
Tamaño Batch		44.000	
Funcion activación		tan	
Valores iniciales Weights		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2

F. Anexo: Lista de artículos de la revisión sistemática

Librería	Año	Autor	Nombre
IEEE	2019	Navaporn Chockwanich; Vasaka Visoottiviseth	Intrusion Detection by Deep Learning with TensorFlow
IEEE	2019	Olivier Brun ; Yonghua Yin	Random Neural Networks and Deep Learning for Attack Detection at the edge
IEEE	2019	Shi-Ming Xia ; Lei Zhang ; Wei Bai ; Xing-Yu Zhou ; Zhi-Song Pan	DDoS Traffic Control Using Transfer Learning DQN With Structure Information
IEEE	2019	Aymen Yahyaoui ;Takoua Abdellatif ; Rabah Attia	Hierarchical anomaly based intrusion detection and localization in IoT
IEEE	2019	Shi-Ming Xia ; Shi-Ze Guo ; Wei Bai ; Jun-Yang Qiu ; Hao Wei ; Zhi-Song Pan	A New Smart Router-Throttling Method to Mitigate DDoS Attacks
IEEE	2019	Sasanka Potluri ; Christian Diedrich	Deep Learning based Efficient Anomaly Detection for Securing Process Control Systems against Injection Attacks
IEEE	2019	Shahadate Rezvy ; Yuan Luo ; Miltos Petridis ; Aboubaker Lasebae ; Tahmina Zebin	An efficient Deep Learning model for intrusion classification and prediction in 5G and IoT networks
ACM	2019	Jinyin Chen; Yi-tao Yang; Ke-ke Hu; Hai-bin Zheng; Zhen Wang	DAD-MCNN: DDoS Attack Detection via Multi-channel CNN
ScienceDirect	2019	Rojalina Priyadarshini ; Rabindra KumarBarik	A Deep Learning based intelligent framework to mitigate DDoS attack in fog environment
Oxford Academic	2019	Muhammad Asad, Muhammad Asim, Talha Javed, Mirza O Beg, Hasan Mujtaba, Sohail Abbas	DeepDetect: Detection of Distributed Denial of Service Attacks Using Deep Learning
iopscience	2019	Ahmad Sanmorino	A study for DDoS attack classification method
ScienceDirect	2019	lBhuvaneswari AmmaN.G; aSelvakumarS	Deep Radial Intelligence with Cumulative Incarnation approach for detecting Denial of Service attacks
ScienceDirect	2019	Li Yuancheng; Zhang Pan ; MaLongqiang	Denial of service attack and defense method on load frequency control system
ScienceDirect	2019	Rojalina Priyadarshini; Rabindra KumarBarik	A Deep Learning based intelligent framework to mitigate DDoS attack in fog environment
ScienceDirect	2019	Celyn Birkinshaw ; Elpida Rouka; Vassilios G.Vassilakis	Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks
ScienceDirect	2019	Will Serrano; Erol Gelenbe	Deep Learning Clusters in the Cognitive Packet Network
ScienceDirect	2019	Sitalakshmi Venkatraman; Mamoun Alaza; R.Vinayakumarc	A hybrid Deep Learning image-based analysis for effective malware detection
ScienceDirect	2019	Soode hHosseini; Mehrdad Azizi	The hybrid technique for DDoS detection with supervised Learning algorithms

ScienceDirect	2019	Zouhair Chiba ; Noreddine Abghour; Khalid Moussaid; Amina El omri; Mohamed Rida	Intelligent approach to build a Deep Neural Network based IDS for cloud environment using combination of Machine Learning algorithms
ScienceDirect	2019	Sydney Mambwe Kasongo; Yanxia Sun	A Deep Long Short-Term Memory based classifier for Wireless Intrusion Detection System
ScienceDirect	2019	Francesco Palmieri	Network anomaly detection based on logistic regression of nonlinear chaotic invariants
ScienceDirect	2019	Wang Kangyi	Network data management model based on Naïve Bayes classifier and Deep neural networks in heterogeneous wireless networks
ScienceDirect	2019	Hongyu Liu ; Bo Lang; Ming Liu ;Hanbing Yan	CNN and RNN based payload classification methods for attack detection
ScienceDirect	2019	Junho Jeong ; Joong Yeon Lim; Yunsik Son	A data type inference method based on long short-term memory by improved feature for weakness analysis in binary code
ScienceDirect	2019	Jie Cui ; Mingjun Wang; Yonglong Luo;Hong Zhongac	DDoS detection and defense mechanism based on cognitive-inspired computing in SDN
Scopus	2019	Li Yuancheng; Zhang Pan; Ma Longqiang	Denial of service attack and defense method on load frequency control system
Scopus	2019	Aditya Kuppa; Slawomir Grzonkowski; Muhammad Rizwan Asghar; Nhien-An Le-Khac	Black box attacks on Deep anomaly detectors
Scopus	2019	M Odusami; S Misra; E Adetiba; O Abayomi-Alli; R Damasevicius and R Ahuja	An Improved Model for Alleviating Layer Seven Distributed Denial of Service Intrusion on Webserver
Scopus	2019	Bhuvaneswari Amma N.G.; Selvakumar S	Deep Radial Intelligence with Cumulative Incarnation approach for detecting Denial of Service attacks
Scopus	2019	Zhang Long; Wang Jinsong	DDoS Attack Detection Model Based on Information Entropy and DNN in SDN | [SDN中基于信息熵与DNN的DDoS攻击检测模型]
Scopus	2019	Navaporn Chockwanich ; Vasaka Visoottiviseth	Intrusion Detection by Deep Learning with TensorFlow
Scopus	2019	Michael Siracusano, Stavros Shiaeles, Bogdan Ghita	Detection of LDDoS Attacks Based on TCP Connection Parameters
Scopus	2019	Mohiuddin Ahmed; Al-Sakib Khan Pathan	Investigating Deep Learning for collective anomaly detection - An experimental study
Scopus	2019	Zakaria El Mrabet; Mehdi Ezzari; Hassan Elghazi; Badr Abou El Majd	Deep Learning-based intrusion detection system for advanced metering infrastructure
Scopus	2019	Shahadate Rezvy; Miltos Petridis; Aboubaker Lasebae; Tahmina Zebin	Intrusion detection and classification with autoencoded Deep neural network

Scopus	2019	Ili Ko; Desmond Chambers; Enda Barrett	Feature dynamic Deep Learning approach for DDoS mitigation within the ISP domain
IEEE	2019	Shaveta Gupta, Dinesh Grover, Abhinav Bhandari	Detection Techniques against DDoS Attacks: A Comprehensive Review
IEEE	2018	Narayanavadivoo Gopinathan Bhuvaneswari Amma ; Selvakumar Subramanian	VCDeepFL: Vector Convolutional Deep Feature Learning Approach for Identification of Known and Unknown Denial of Service Attacks
IEEE	2018	Panida Khuphiran ; Pattara Leelaprute ; Putchong Uthayopas ; Kohei Ichikawa ; Wassapon Watanakeesuntorn	Performance Comparison of Machine Learning Models for DDoS Attacks Detection
IEEE	2018	George Loukas ; Tuan Vuong ; Ryan Heartfield ; Georgia Sakellari ; Yongpil Yoon ; Diane Gan	Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning
IEEE	2018	Congyuan Xu ; Jizhong Shen ; Xin Du ; Fan Zhang	An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units
IEEE	2018	Yandong Liu ; Mianxiong Dong ; Kaoru Ota ; Jianhua Li ; Jun Wu ;	Deep Reinforcement Learning based Smart Mitigation of DDoS Flooding in Software-Defined Networks
IEEE	2018	Kasun Amarasinghe ; Kevin Kenney ; Milos Manic	Toward Explainable Deep Neural Network Based Anomaly Detection
IEEE	2018	Michael Siracusano ; Stavros Shiaeles ; Bogdan Ghita	Detection of LDDoS Attacks Based on TCP Connection Parameters
IEEE	2018	Baskoro Adi Pratomo ; Pete Burnap ; George Theodorakopoulos	Unsupervised Approach for Detecting Low Rate Attacks on Network Traffic with Autoencoder
IEEE	2018	Karan B. V. ; Narayan D. G. ; P. S. Hiremath	Detection of DDoS Attacks in Software Defined Networks
IEEE	2018	Gozde Karatas ; Onder Demir ; Ozgur Koray Sahingoz	Deep Learning in Intrusion Detection Systems
IEEE	2018	M. Ogawa ; H. Tanaka ; J. Muramatsu ; M. Nakano ; K. Yoshida ; T. Asakura	Application of Deep Learning to Underwater Invasion Warning System
IEEE	2018	Jing Ran ; Yexin Chen ; Shulan Li	THREE-DIMENSIONAL CONVOLUTIONAL NEURAL NETWORK BASED TRAFFIC CLASSIFICATION FOR WIRELESS COMMUNICATIONS
IEEE	2018	Sabah Alzahrani ; Liang Hong	Detection of Distributed Denial of Service (DDoS) Attacks Using Artificial Intelligence on Cloud
IEEE	2018	Bineet Kumar Joshi ; Nitin Joshi ; Mahesh Chandra Joshi	Early Detection of Distributed Denial of Service Attack in Era of Software-Defined Network
IEEE	2018	O. Boyar ; M. E. Özen ; B. Metin	Detection of Denial-of-Service Attacks with SNMP/RMON

IEEE	2018	Rajendra Patil ; Harsha Dudeja ; Snehal Gawade ; Chirag Modi	Protocol Specific Multi-Threaded Network Intrusion Detection System (PM-NIDS) for DoS/DDoS Attack Detection in Cloud
IEEE	2018	Aditya Prakash ; Rojalina Priyadarshini	An Intelligent Software defined Network Controller for preventing Distributed Denial of Service Attack
IEEE	2018	Babatunde Hafis Lawal ; A. T. Nuray	Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN)
IEEE	2018	Lanka Chris Sejaphala ; Mthulisi Velempini	Investigating high traffic rate distributed denial of service attacks detection mechanisms in software-defined networks
IEEE	2018	Gozde Karatas ; Onder Demir ; Ozgur Koray Sahingoz	Deep Learning in Intrusion Detection Systems
ACM	2018	Carla O. Camargo; Elaine R. Faria; Bruno B. Zarpelão; Rodrigo S. Miani	Qualitative evaluation of denial of service Datasets
ACM	2018	Eef van Es; Harald Vranken; Arjen Hommersom	Denial-of-Service Attacks on LoRaWAN
Scielo	2018	Zeljko Gavric , Dejan Simic	Overview of DoS attacks on wireless sensor networks and experimental results for simulation of interference attacks
ResearchGate	2018	Imamverdiyev; Abdullayeva	Deep Learning Method for Denial of Service Attack Detection Based on Restricted Boltzmann Machine
Indexive	2018	aysegulsngr;hacibeyoglu	Detection of DDoS Attacks in Network Traffic Using Deep Learning
ScienceDirect	2018	Hassan Lahza, Kenneth Radke , Ernest Foo	Applying domain-specific knowledge to construct features for detecting distributed denial-of-service attacks on the GOOSE and MMS protocols
ScienceDirect	2018	Md. ZahidHasan; K.M. Zubair Hasan, Abdus Sattar	Burst Header Packet Flood Detection in Optical Burst Switching Network Using Deep Learning Model
ScienceDirect	2018	Olivier Bruna; Yonghua Yin; Erol Gelenbe	Deep Learning with Dense Random Neural Network for Detecting Attacks against IoT-connected Home Environments
ScienceDirect	2018	Abebe Abeshu Diro; Naveen Chilamkurti	Distributed attack detection scheme using Deep Learning approach for Internet of Things
ScienceDirect	2018	MunaAL-Hawawreh; Nour Moustafa; Elena Sitnikova	Identification of malicious activities in industrial internet of things based on Deep Learning models
ScienceDirect	2018	Tae-Youn Kim; Sung-BaeCho	Web traffic anomaly detection using C-LSTM neural networks
Scopus	2018	Karan B. V. ; Narayan D. G. ; P. S. Hiremath	Detection of DDoS Attacks in Software Defined Networks
Scopus	2018	Kasun Amarasinghe; Kevin Kenney; Milos Manic	Toward explainable Deep neural network based anomaly detection
Scopus	2018	Andy Brown, Aaron Tuor, Brian Hutchinson, Nicole Nichols	Recurrent neural network attention mechanisms for interpretable system log anomaly detection
IEEE	2018	Steve Muller ;Jean Lancrenon; Carlo Harpes; Yves Le Traon; Sylvain	A training-resistant anomaly detection system

		Gombault; Jean-MarieBonninc	
IEEE	2018	Congyuan Xu ; Jizhong Shen ; Xin Du ; Fan Zhang	An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units
IEEE	2018	T. T. Teoh ; Graeme Chiew ; Edwin J Franco ; P. C. Ng ; M.P Benjamin ; Y. J. Goh	Anomaly detection in cyber security attacks on networks using MLP Deep Learning
IEEE	2018	Gozde Karatas ; Onder Demir ; Ozgur Koray Sahingoz	Deep Learning in intrusion detection systems
IEEE	2018	Abebe Abeshu ; Naveen Chilamkurti	Deep Learning The Frontier for Distributed Attack Detection in Fog-to-Things Computing
IEEE	2018	Yandong Liu ; Mianxiong Dong ; Kaoru Ota ; Jianhua Li ; Jun Wu	Deep Reinforcement Learning based Smart Mitigation of DDoS Flooding in Software-Defined Networks
IEEE	2018	Ilay Cordonsky ; Ishai Rosenberg ; Guillaume Sicard ; Eli Omid David	DeepOrigin End-to-End Deep Learning for Detection of New Malware Families
IEEE	2018	Sabah Alzahrani ; Liang Hong	Detection of distributed Denial of Service Attacks using Artificial Intelligence on Cloud
IEEE	2018	Michael Siracusano ; Stavros Shiaeles ; Bogdan Ghita	Detection of LDDoS Attacks Based on TCP Connection Parameters
IEEE	2018	Yang Xin ; Lingshuang Kong ; Zhi Liu ; Yuling Chen ; Yanmiao Li ; Hongliang Zhu ; Mingcheng Gao ; Haixia Hou ; Chunhua Wang	Machine Learning and Deep Learning Methods for Cybersecurity. IEEE
IEEE	2018	Md Zahangir Alom ; Tarek M. Taha	Network Intrusion Detection for Cyber Security using Unsupervised Deep Learning Approaches
IEEE	2018	Giovanni Apruzzese ; Michele Colajanni ; Luca Ferretti ; Alessandro Guido ; Mirco Marchetti	On the Effectiveness of Machine and Deep Learning for cybersecurity
IEEE	2018	César Byron Guevara ; Janio Jadan	Sistema de detección de Intrusos en secuencia de comandos aplicando la metología One Versus Rest. IEEE
IEEE	2018	Narayanavadivoo Gopinathan Bhuvaneswari Amma, Selvakumar Subramanian	V CDeepF L Vector Convolutional Deep Feature
IEEE	2017	Xiaoyong Yuan ; Chuanhuang Li ; Xiaolin Li	DeepDefense: Identifying DDoS Attack via Deep Learning
IEEE	2017	Supriya S. Thakare ; Parminder Kaur	Denial-of-service attack detection system
IEEE	2017	P. Hemalatha ; J. Vijithaananthi	An effective performance for Denial of Service Attack (DoS) detection

IEEE	2017	Mohamed Wasim Lorgat ; Alireza Baghai-Wadji ; Andre McDonald	Quantifying the effect of incomplete information in denial of service detection
IEEE	2017	Ahmad Riza'ain Yusof ; Nur Izura Udzir ; Ali Selamat ; Hazlina Hamdan ; Mohd Taufik Abdullah	Adaptive feature selection for denial of services (DoS) attack
ACM	2017	Haoyue Xue; Yuhong Li; Rahim Rahmani	A mechanism for mitigating DoS attack in ICN-based internet of things
ScienceDirect	2017	Erwin Adi;Zubai rBaig; Philip Hingston	Stealthy Denial of Service (DoS) attack modelling and detection for HTTP/2 services
ScienceDirect	2017	Gaurav Somani ; Manoj Singh Gaur;Dheeraj Sanghi; Mauro Conti; Rajkumar Buyya	DDoS attacks in cloud computing: Issues, taxonomy, and future directions
ScienceDirect	2017	Ahmed AlEroud;Izzat Alsmadi	Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach
IEEE	2017	Ali Ahmadian Ramaki ; Abbas Rasoolzadegan ; Abbas Javan Jafari	A systematic review on intrusion detection based on the Hidden Markov Model. Scopus
IEEE	2017	Charles Feng ; Shuning Wu ; Ningwei Liu	A UserCentric Machine Learning Framework for Cyber Security Operations Center. IEEE
IEEE	2017	Maria Rigaki; Ahmed Elragal; Luleå	Adversarial Deep Learning Against Intrusion Detection Classifiers
IEEE	2017	Steven McElwee ; Jeffrey Heaton ; James Fraley ; James Cannady	Deep Learning for Prioritizing and Responding to Intrusion Detection Alerts
IEEE	2017	Xiaoyong Yuan ; Chuanhuang Li ; Xiaolin Li	DeepDefense Identifying DDoS Attack
IEEE	2017	TarfaHamedRozitaDaraStefan C.Kremer	Network Intrusion Detection System based on recursive feature addition and bigram technique
IEEE	2016	Satyajit Yadav;Selvakumar Subramanian	Detection of Application Layer DDoS attack by feature Learning using Stacked AutoEncoder
IEEE	2016	Mikhail Zolotukhin ; Timo Hämäläinen ; Tero Kokkonen ; Jarmo Siltanen	Increasing web service availability by detecting application-layer DDoS attacks in encrypted traffic
IEEE	2016	Asantha Thilina ; Shakthi Attanayake ; Sacith Samarakoon ; Dahami Nawodya ; La	Intruder Detection Using Deep Learning and Association Rule Mining
IEEE	2016	Zhouyu Zhang ; Yunfeng Cao ; Meng Ding ; Likui Zhuang ; Weiwen Yao	An intruder detection algorithm for vision based sense and avoid system
IEEE	2016	Te-Jen Su ; Shih-Ming Wang ; Yi-Feng Chen ; Chao-Liang Liu	Attack detection of distributed denial of service based on Splunk
IEEE	2016	Raneel Kumar ; Sunil Pranit Lal ; Alok Sharma	Detecting Denial of Service Attacks in the Cloud
Scopus	2016	Mikhail Zolotukhin ; Timo Hämäläinen ; Tero Kokkonen ; Jarmo Siltanen	Increasing web service availability by detecting application-layer DDoS attacks in encrypted traffic

IEEE	2016	Asantha Thilina ; Shakthi Attanayake ; Sacith Samarakoon ; Dahami Nawodya ; Lakmal Rupasinghe	Intruder Detection Using Deep Learning and Association Rule Mining. IEEE
IEEE	2015	Zhiyuan Tan ; Aruna Jamdagni ; Xiangjian He ; Priyadarsi Nanda ; Ren Ping Liu ; Jiankun Hu	Detection of Denial-of-Service Attacks Based on Computer Vision Techniques
IEEE	2015	Supriya Pharande ; Priyanka Pawar ; P. W. Wani ; A. B. Patki	Application of Hurst parameter and fuzzy logic for denial of service attack detection
ACM	2015	Gaurav Somani; Manoj Singh Gaur; Dheeraj Sanghi	DDoS/EDoS attack in cloud: affecting everyone out there!
IEEE	2014	P. Revathi	Flow and rank correlation based detection against Distributed Reflection Denial of Service attack
IEEE	2014	Nikita Lyamin ; Alexey Vinel ; Magnus Jonsson ; Jonathan Loo	Real-Time Detection of Denial-of-Service Attacks in IEEE 802.11p Vehicular Networks
ACM	2014	Andreas Papalambrou; Kyriakos Stefanidis; John Gialelis; Dimitrios Serpanos	Detection, traceback and filtering of denial of service attacks in networked embedded systems
ACM	2014	Roland van Rijswijk-Deij; Anna Sperotto ; Aiko Pras	DNSSEC and its potential for DDoS attacks: a comprehensive measurement study
IEEE	2013	Vijay D. Katkar ; Siddhant Vijay Kulkarni	Experiments on detection of Denial of Service attacks using Naive Bayesian classifier
IEEE	2013	Vijay D. Katkar ; Siddhant Vijay Kulkarni	A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks
IEEE	2013	Vijay D. Katkar ; Siddhant Vijay Kulkarni	Experiments on detection of Denial of Service attacks using ensemble of classifiers
IEEE	2013	Vijay D. Katkar ; Deepti S. Bhatia	Experiments on detection of Denial of Service attacks using REPTree
IEEE	2013	Yonghong Chen ; Xinlei Ma ; Xinya Wu	DDoS Detection Algorithm Based on Preprocessing Network Traffic Predicted Method and Chaos Theory
ScienceDirect	2013	P.Arun Raj Kumar; S.Selvakumar	Detection of distributed denial of service attacks using an ensemble of adaptive and hybrid neuro-fuzzy systems
IEEE	2012	Zhiyuan Tan ; Aruna Jamdagni ; Xiangjian He ; Priyadarsi Nanda ; Ren Ping Liu	Triangle-Area-Based Multivariate Correlation Analysis for Effective Denial-of-Service Attack Detection
IEEE	2012	Wanlei Zhou	Keynote: Detection of and Defense Against Distributed Denial-of-Service (DDoS) Attacks
ACM	2012	Yossi Gilad; Amir Herzberg	LOT: A Defense Against IP Spoofing and Flooding Attacks
ACM	2012	Yoo Chung	Distributed denial of service is a scalability problem
ScienceDirect	2012	Hakem Beitollahi; Geert Deconinck	Analyzing well-known countermeasures against distributed denial of service attacks

IEEE	2011	Xie Chuiyi ; Zhang Yizhi ; Bai Yuan ; Luo Shuoshan ; Xu Qin	A Distributed Intrusion Detection System against flooding Denial of Services attacks
IEEE	2011	Massimo Ficco ; Massimiliano Rak	Intrusion Tolerant Approach for Denial of Service Attacks to Web Services
IEEE	2011	Rachana Yogesh Patil ; Lata Ragha	A rate limiting mechanism for defending against flooding based distributed denial of service attack
IEEE	2011	Jae-Hyun Jun ; Hyunju Oh ; Sung-Ho Kim	DDoS flooding attack detection through a step-by-step investigation
IEEE	2011	Xiang Xu ; Ding Wei ; Yuelei Zhang	Improved Detection Approach for Distributed Denial of Service Attack Based on SVM
IEEE	2010	Hsia-Hsiang Chen ; Wuu Yang	The Design and Implementation of a Practical Meta-Heuristic for the Detection and Identification of Denial-of-Service Attack Using Hybrid Approach
IEEE	2010	Asrul H. Yaacob ; Ian K.T. Tan ; Su Fong Chien ; Hon Khi Tan	ARIMA Based Network Anomaly Detection
IEEE	2010	Sahil Seth ; Anil Gankotiya	Denial of Service Attacks and Detection Methods in Wireless Mesh Networks
ACM	2010	Wonjun Lee; Anna Squicciarini; Elisa Bertino	Vulnerabilities leading to denial of services attacks in grid computing systems: a survey
ACM	2010	Zhenqian Feng; Haitao Wu; Jinshu Su	Exploring potential vulnerabilities in data center network
ACM	2010	Xin Liu; Xiaowei Yang ; Yong Xia	NetFence: preventing internet denial of service from inside out
IEEE	2009	A. B. M. Alim Al Islam ; Tishna Sabrina	Detection of various denial of service and Distributed Denial of Service attacks using RNN ensemble
IEEE	2009	G. Meera Gandhi ; S.K. Srivatsa	An Entropy Algorithm to Improve the Performance and Protection from Denial-of-Service Attacks in NIDS
IEEE	2009	Rajani Muraleedharan ; Lisa Ann Osadciw	An intrusion detection framework for Sensor Networks using Honeypot and Swarm Intelligence
IEEE	2009	Manhyun Chung ; Jaeik Cho ; Jongsub Moon	An effective denial of service detection method using kernel based data
IEEE	2009	Yong Wang ; Dawu Gu ; Xiuxia Tian ; Jing Li	Genetic Algorithm Rule Definition for Denial of Services Network Intrusion Detection
ACM	2009	José M. Garrido	Understanding distributed denial of service with object oriented simulation
IEEE	2008	Rui Gustavo Crespo	Identification of Feature Denial of Services
IEEE	2008	S. Bose ; A. Kannan	Detecting Denial of Service Attacks using Cross Layer based Intrusion Detection System in Wireless Ad Hoc Networks

Bibliografía

[1] M. Z. Alom y T. M. Taha, «Network Intrusion Detection for Cyber Security using unsupervised Deep Learning Approaches,» *Department of Electrical and Computer Engineering,* pp. 63-69, 2017.

[2] Y. Xing, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao y H. &. C. W. Hou, « Machine Learning and Deep Learning Methods for Cybersecurity,» *IEEE Access,* pp. Volume, 6, 35365 – 35381, 2018.

[3] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido y M. Marchetti, «On the Effectiveness of Machine and Deep Learning for Cyber Security,» *10th International Conference on Cyber Conflict,* 2018.

[4] M. contributors, «Mozilla Developer,» 6 Agosto 2020. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server. [Último acceso: 20 Agosto 2020].

[5] A. S. Choudhary, P. P. Choudhary y S. Salve, «A Study On Various Cyber Attacks And A Proposed Intelligent System For Monitoring Such Attacks,» *Proceedings of the International Conference on Inventive Computation Technologies,* 2018.

[6] F. P. L. Galicia, «co.godaddy,» 1 Junio 2020. [En línea]. Available: https://co.godaddy.com/blog/que-es-seguridad-en-la-web-manual-basico/. [Último acceso: 20 Agosto 2020].

[7] L. Griffin, «Study.com,» Study, [En línea]. Available: https://study.com/academy/lesson/vulnerabilities-issues-in-web-servers.html. [Último acceso: 20 Agosto 2020].

[8] A. A. S. T. Islam, «Detection of various Denial of Service and Distributed Denial of Service Attacks using RNN Ensemble,» de *Proceedings of 2009 12th International Conference on Computer and Information Technology*, Dhaka, Bangladesh, 2009.

[9] JavaPipe, «JavaPipe,» 2019. [En línea]. Available: https://javapipe.com/blog/ddos-types/.

[10] S. T. Zargar, J. Joshi y D. Tipper, «A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks,» *IEEE,* pp. 2046 - 2069, 2013.

[11] G. Karatas, O. Demir y O. K. Sahingoz, «Deep Learning in Intrusion Detection Systems,» *International Congress in Big Data, Deep Learning and fighting Cyberterrorism,* 2018.

[12] I. Sharafaldin, A. H. Lashkari, S. Hakak y A. A. Ghorbani, «Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy,» 2019.

[13] CloudFlare;, «CloudFlare NTP Attack,» [En línea]. Available: https://www.cloudflare.com/learning/ddos/ntp-amplification-ddos-attack/. [Último acceso: 4 Octubre 2020].

[14] CloudFlare, «CloudFlare DNS Attack,» [En línea]. Available: https://www.cloudflare.com/learning/ddos/dns-flood-ddos-attack/. [Último acceso: 4 Octubre 2020].

[15] Allot, «Allot DOS Glosario,» [En línea]. Available: https://www.allot.com/ddos-attack-glossary/. [Último acceso: 4 Octubre 2020].

[16] «Akamai MSSQL,» [En línea]. Available: https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/ms-sql-server-reflection-ddos-mc-sqlr-threat-advisory.pdf. [Último acceso: 28 Abril 2020].

[17] «Akamai Netbios,» [En línea]. Available: https://www.helpnetsecurity.com/2015/10/29/new-ddos-attacks-misuse-netbios-name-server-rpc-portmap-and-sentinel-licensing-servers/. [Último acceso: 28 Abril 2020].

[18] «Imperva SNMP,» [En línea]. Available: https://www.imperva.com/learn/application-security/snmp-reflection/. [Último acceso: 28 Abril 2020].

[19] «CloudFlare SSDP Attack,» [En línea]. Available: https://www.cloudflare.com/learning/ddos/ssdp-ddos-attack/. [Último acceso: 28 04 2020].

[20] «CloudFlare UDP Attack,» [En línea]. Available: https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/. [Último acceso: 28 04 2020].

[21] «Security Intelligence TFTP,» [En línea]. Available: https://securityintelligence.com/news/trivial-file-transfer-protocol-used-in-new-ddos-attack/. [Último acceso: 28 Abril 2020].

[22] G. Briceño, «Club de tecnologia,» 2 Abril 2020. [En línea]. Available: https://www.clubdetecnologia.net/blog/2020/15-herramientas-de-machine-learning-mas-utilizadas/. [Último acceso: 3 Septiembre 2020].

[23] «GitHub,» 2017. [En línea]. Available: https://jrmerwin.github.io/deeplearning4j-docs/compare-dl4j-tensorflow-pytorch. [Último acceso: 6 Septiembre 2020].

[24] N. Chockwanich y V. Visoottiviseth, «Intrusion Detection by Deep Learning with Tensorflow,» *2019 21st International Conference on Advanced Communication Technology (ICACT),* 2019.

[25] K. Barik y R. Priyadarshini, «A deep learning based intelligent framework to mitigate DDoS attack in fog environment,» *Journal of King Saud University - Computer and Information Sciences,* 2019.

[26] B. A. SelvakumarS, «Deep Radial Intelligence with Cumulative Incarnation approach for detecting Denial of Service attacks,» *Neurocomputing,* pp. 294-308, 2019.

[27] Z. Chiba, N. Abghour, K. Moussaid, A. E. omri y M. Rida, «Intelligent approach to build a Deep Neural Network based IDS for cloud environment using combination of machine learning algorithms,» *Computers & Security,* vol. 86, pp. 291-317, 2019.

[28] S. M. Kasongo y Y. Sun, A Deep Long Short-Term Memory based classifier for Wireless Intrusion Detection System, ICT Express, 2019.

[29] H. Liu, B. Lang, M. Liu y H. Yanb, «CNN and RNN based payload classification methods for attack detection,» *Knowledge-Based Systems,* pp. 322-341, 2019.

[30] M. Siracusano, S. Shiales y B. Ghita, «Detection of LDDoS Attacks based on TCP Connection Parameters,» *Global Information Infraestructure and Networking Symposium,* p. 6, 2018.

[31] N. G. B. Amma y S. Subramanian, «VCDeepFL: Vector Convolutional Deep Feature Learning Approach for Identification of Known and Unknown Denial of Service Attacks,» *TENCON 2018 - 2018 IEEE Region 10 Conference,* 2018.

[32] S. Alzahrani y L. Hong, «Detection of Distributed Denial of Service (DDoS) Attacks Using Artificial Intelligence on Cloud,» *2018 IEEE World Congress on Services (SERVICES),* 2018.

[33] A. Thilina, S. Attanayake, S. Samarakoon, D. Nawodya, L. Rupasinghe, N. Pathirage y Edirisinghe, «Intruder Detection Using Deep Learning and Association Rule

Mining,» *IEEE International Conference on Computer and Information Technology,* 2016.

[34] P. Khuphiran, P. Leelaprute, P. Uthayopas, K. Ichikawa y W. Watanakeesunt, «Performance Comparison of Machine Learning Models for DDoS Attacks Detection,» *2018 22nd International Computer Science and Engineering Conference (ICSEC),* 2018.

[35] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon y D. Gan, «Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning,» *IEEE,* pp. 3491 - 3508, 2017.

[36] C. Xu, J. Shen, X. Du y F. Zhang, «An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units,» *IEEE Access,* pp. 48697 - 48707, 2018.

[37] K. Amarasinghe, K. Kenney y M. Manic, «Toward Explainable Deep Neural Network Based Anomaly Detection,» *2018 11th International Conference on Human System Interaction (HSI),* 2018.

[38] Y. Imamverdiyev y F. Abdullayeva, «Deep Learning Method for Denial of Service Attack Detection Based on Restricted Boltzmann Machine,» *Research Gate,* 2018.

[39] M. H. A. SUNGUR UNAL, «Detection of DDOS Attacks in Network Traffic Using Deep Learning,» *International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18),* 2018.

[40] C. L. X. L. Xiaoyong Yuan, «DeepDefense Identifying DDoS Attack via Deep Learning,» *Large-scale Intelligent Systems Laboratory,,* 2017.

[41] T.-Y. Kim y S.-B. Cho, «Web traffic anomaly detection using C-LSTM neural networks,» *Expert Systems with Applications,* vol. 106, pp. 66-76, 2018.

[42] S. Yadav y S. Subramanian, «Detection of Application Layer DDoS attack by feature learning using Stacked AutoEncoder,» *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT),* 2016.