UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Aplicación de técnicas de Deep Learning en un Sistema de Detección de Intrusos con tráfico de red relacionado con la Dark Web

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Tajadura Cubillo, David

Tutor/a: Botti Navarro, Vicente Juan

Cotutor/a: Palanca Cámara, Javier

CURSO ACADÉMICO: 2021/2022

# Resumen

Los avances tecnológicos relacionados con las redes de computadores y con la tecnología de Internet de los últimos años han provocado un crecimiento exponencial de la cantidad de datos intercambiados en las comunicaciones digitales, y, por tanto, de los esfuerzos de los técnicos informáticos para proteger la seguridad, integridad y confidencialidad de los mismos. En este contexto, se han desarrollado numerosos tipos de software para prevenir y eliminar cualquier tipo de riesgo y amenazas que exponga la vulnerabilidad de los sistemas, usuarios, o cualquier tipo de dato. Con la motivación de introducirse y entender el funcionamiento de los sistemas de detección de ataques o intrusiones en redes de computadores, en este TFG se va a estudiar y utilizar técnicas de Deep Learning, tecnología base de la Inteligencia Artificial, aplicadas a un sistema de detección de intrusiones. Entre sus innumerables aplicaciones, el Deep Learning se ha utilizado con excelentes resultados en el análisis del tráfico de redes de computadoras.

Se ha escogido el conjunto de datos CIC-Darknet2020, elaborado por el Canadian Institute of Cybersecurity (CIC), que contiene conexiones de red con tráfico benigno y otras conexiones asociadas a la Dark Web. La Dark Web engloba un conjunto de redes que están ocultas para los navegadores convencionales, cuyo acceso está limitado a ciertos protocolos y softwares específicos. Su principal característica es la de guardar la privacidad del usuario que realiza la conexión, es decir, la IP de su conexión. Dentro del conjunto de datos seleccionados, estas últimas conexiones de red están clasificadas como conexiones TOR y conexiones VPN, que son consideradas como tráfico sospechoso por los ya mencionados sistemas de detección de intrusos, debido a sus características de encriptación y anonimidad. Con todo esto, el objetivo del presente trabajo consistirá en estudiar el funcionamiento e implementación de las redes neuronales utilizadas en el Deep Learning, para, posteriormente, poner en práctica los conocimientos adquiridos implementando una red neuronal. La finalidad de esta red será detectar, con la mayor efectividad posible, las conexiones maliciosas del conjunto de datos CIC-Darknet2020, haciendo uso de las características técnicas de dichas conexiones como datos de entrada.

**Palabras clave**: Inteligencia Artificial, Deep Learning, Ciberseguridad, Dark Web, redes neuronales, sistemas de detección de intrusos, TOR, VPN.

# Abstract

Technological advances related to computer networks and Internet technology in recent years have led to an exponential growth in the amount of data exchanged in digital communications, and therefore in the efforts of computer technicians to ensure the security, integrity, and confidentiality of such data. In this context, several types of software have been developed to prevent and eliminate any kind of risk and threats that expose the vulnerability of systems, users, or any type of data. With the aim of introducing and understanding the functioning of systems for detecting attacks or intrusions in computer networks, this project will study and use Deep Learning techniques, the basic technology of Artificial Intelligence, applied to an intrusion detection system. Among its countless applications, Deep Learning has been used with excellent results in the analysis of computer network traffic.

The CIC-Darknet2020 dataset, developed by the Canadian Institute of Cybersecurity (CIC), has been selected for this purpose, which contains network connections with benign traffic and other connections associated with the Dark Web. The Dark Web consists of a set of networks that are hidden from conventional browsers, access to which is limited to certain protocols and specific software. Their main characteristic is to keep the privacy of the user making the connection, i.e. the IP of his or her connection. Within the selected dataset, the latter network connections are labelled as TOR connections and VPN connections, which are considered as suspicious traffic by the above-mentioned intrusion detection systems, due to their encryption and anonymity properties. Thus, the aim of the forthcoming project will be to study the operation and implementation of the neural networks used in Deep Learning, in order to, subsequently, put the acquired knowledge into practice by deploying a neural network. The purpose of this network will be to detect, as effectively as possible, malicious connections in the CIC-Darknet2020 dataset, using the technical characteristics of these connections as input data.

**Keywords**: Artificial Intelligence, Deep Learning, Cybersecurity, Dark Web, Neural Networks, intrusion detection systems, TOR, VPN.

# Tabla de contenidos

# Tabla de Figuras

# Tabla de Tablas

# 1. INTRODUCCIÓN

## 1.1 PRESENTACIÓN

Los avances de las últimas décadas relacionados con la tecnología de Internet y el descomunal crecimiento de las comunicaciones digitales han hecho que el tamaño de la red y la cantidad de datos que se transmiten a través de ella hayan aumentado de una manera extraordinaria [1]. En misma proporción han aumentado el número y los tipos de ataques sobre las redes, poniendo en serio riesgo su seguridad, y por tanto, la seguridad de los datos que están almacenados en cualquier dispositivo conectado a ellas. Esto supone un serio riesgo para todos los usuarios de la red, tanto particulares como empresas; especialmente, para aquellos que, o bien por tener menos conocimientos tecnológicos, o por gestionar información sensible o de interés comercial, son mucho más vulnerables a los ataques de piratas informáticos o hackers.

Desde finales del siglo XX se han ido desarrollando diferentes sistemas de detección de intrusos para intentar preservar la seguridad de las redes [2]. Sin embargo, el auge desmesurado de las tecnologías y el incremento exponencial de los datos que circulan por ellas, han aumentado la vulnerabilidad del conjunto de los sistemas de forma proporcional. De hecho, hoy en día se llevan a cabo una gran cantidad de ataques informáticos de diferentes tipos cada vez más complejos, con el agravante de que, con frecuencia, son difundidos indiscriminadamente entre piratas informáticos expertos o inexpertos, lo que dificulta aún más la detección de intrusiones de tráfico malicioso en la red [3]. Es por ello que las empresas deben asegurar toda la información contenida en sus sistemas. Cualquier nodo de información que pueda ser comprometido podría causar numerosos daños en los sistemas tanto personales como empresariales, y por tanto en su organización per se. Esto podría llegar a producir ingentes pérdidas económicas para la empresa, derivadas de la violación de datos e información claves de la empresa, de los tiempos de inactividad necesarios hasta la recuperación del control, así como una caída de su reputación frente a los clientes [4].

Esta realidad ha provocado la generalización del uso de diversas herramientas de seguridad como firewalls, programas de antivirus y sistemas de detección de intrusiones. De todas estas herramientas, los sistemas de detección de intrusos basados en redes son los que proporcionan una mayor fiabilidad, al monitorizar todo el tráfico de la red en busca de anomalías y posibles riesgos para cualquier dispositivo conectado a esta [5].

En este contexto, la Inteligencia Artificial, y más concretamente el Deep Learning, han permitido a los Sistemas de Detección de Intrusos dar un salto en la eficiencia de su trabajo [6], prediciendo con antelación las conexiones sospechosas y proporcionando una seguridad añadida sin precedentes hasta ahora.

## 1.2 MOTIVACIÓN Y OBJETIVOS

La motivación principal de este trabajo es establecer un entorno de trabajo que permita estudiar y comprender el comportamiento teórico y práctico de los sistemas utilizados en el ámbito del Deep Learning aplicados a los Sistemas de detección de Intrusos basados en red. Los objetivos específicos de este proyecto son:

- Introducirse en el ámbito de la ciberseguridad y la protección de datos, enfocándose especialmente en los sistemas de detección de intrusos basados en red.
- Conocer los diferentes tipos de sistemas de detección de intrusos, así como sus características
- Analizar el funcionamiento básico de las redes neuronales aplicadas a métodos de aprendizaje supervisado en problemas reales y actuales.
- Aproximarse al conocimiento de la red Darknet, y comprender las diferencias entre Internet, Deep web y Darknet.
- Revisar y documentar el Estado del Arte de la aplicación de Deep Learning en sistemas de detección de intrusos basados en red, así como las métricas de evaluación y los resultados obtenidos en trabajos anteriores del mismo ámbito.
- Justificar los pasos a seguir para implementar una solución de Machine Learning y Deep Learning a un problema real.
- Aplicar los conocimientos adquiridos durante la carrera universitaria en la UPV para analizar, adaptar y preprocesar datos de entrada de un conjunto de datos definido.
- Implementar una solución de Inteligencia Artificial lo más precisa posible para un conjunto de datos público y conocido de un problema real, como es el tráfico de red de la Darknet.
- Analizar los resultados obtenidos en las diferentes soluciones propuestas, utilizando las métricas de evaluación documentadas.
- Implementar las mejoras oportunas en cada solución para poder realizar una retroalimentación efectiva de las mismas.

## 1.3 ESTRUCTURA DEL TRABAJO

Tras la introducción y definición de los objetivos de este trabajo, el siguiente apartado explicará el estado del arte tanto de los sistemas de detección de intrusos como del deep learning en el ámbito de la ciberseguridad. También se van a introducir unas nociones básicas de la Deep Web, así como del funcionamiento básico de las redes neuronales.

Posteriormente, pasaremos al marco experimental del trabajo, donde primero se hará una introducción al dataset escogido para la elaboración del mismo. Más adelante, se explicarán las fases que se han seguido para la obtención de una solución técnica de deep learning aplicada al dataset mencionado anteriormente, las cuales se irán explicando detalladamente en los capítulos posteriores. Estas fases del marco experimental consisten en el preprocesado de los datos de entrada, detalles del entrenamiento de los diferentes modelos de la red neuronal, y la obtención de los resultados experimentales y validación de los mismos.

Para finalizar y a modo de conclusión, se hace un resumen de los objetivos alcanzados durante el trabajo y las dificultades abordadas para el mismo. Además, se propone la continuación del trabajo futuro para seguir este trabajo. El documento se finaliza con la bibliografía utilizada durante todo el proceso y el anexo para los Objetivos de Desarrollo Sostenible.

# 2. ESTADO DEL ARTE

## 2.1 JUSTIFICACIÓN DE LOS SISTEMAS DE DETECCIÓN DE INTRUSOS

Un Sistema de Detección de Intrusiones (IDS) es una herramienta que detecta la entrada de tráfico no deseado mediante la inspección exhaustiva del tráfico de red, creando alertas dirigidas al sistema o los administradores de la red en la que se encuentran, con el fin de que se aseguren las tres principales características de las políticas de seguridad: confidencialidad, integridad y disponibilidad. Normalmente funcionan como una combinación de medidas de autentificación de los nodos de la red y medidas de control de acceso de autorización a la conexión que quiere establecer cada nodo [7].

Un IDS, como indican sus siglas, es un sistema de detección de intrusos, no un sistema de respuesta frente ataques, ya que los IDS no alteran los paquetes que circulan a través de la red. De hecho, estos sistemas normalmente forman parte de una herramienta de seguridad de un nivel superior, capaz de dar respuesta y soluciones a las situaciones detectadas mediante la monitorización ejercida por los IDS [8].

El sistema formado por un IDS y otra herramienta de respuesta o control se denomina Sistema de Prevención de Intrusiones (IPS), que, al contrario que los IDS, sí previene la entrega de paquetes maliciosos hacia los dispositivos de la red en la que se encuentra. Además, los IDS pueden estar implementados en sistemas distribuidos mediante múltiples detectores en las capas de red, núcleo y aplicación, de forma que forman un Sistema de Detección de Intrusos Colaborativo (CIDS). Cada detector del sistema CIDS emite una alarma, y de forma conjunta crean una alarma combinada, que es mucho más eficiente y precisa para este tipo de sistemas [9].

No obstante, a pesar de todos los esfuerzos para mejorar la precisión de los IDS, todavía existe el gran desafío de mejorar la tasa de falsos positivos que, generalmente, detecta este tipo de sistemas [10]. Los falsos positivos consisten en la detección por parte del IDS de la existencia de un ataque cuando en realidad no lo es.

Otro aspecto que hay que destacar para definir los IDS es la diferencia con un firewall. Generalmente, un firewall es un sistema configurado para bloquear el tipo de tráfico no deseado hacia los dispositivos de la red donde esté definido, aunque también puede estar configurado para permitir únicamente el tráfico deseado hacia dicha red. Un IDS no bloquea ni permite ningún tipo de tráfico, sino que detecta los paquetes que circulan hacia la red y son amenazas potenciales para el sistema. Por tanto, para una gestión

eficiente y eficaz de la seguridad de la red, lo más recomendable es utilizar una combinación de IDS y firewall [8]. De esta forma, el firewall sólo permitirá cierto tipo de tráfico de entrada a la red, mientras que el IDS detectará las anomalías o riesgos del tráfico permitido en la red.

Por otra parte, los IDS actuales han demostrado ser ineficientes en la detección de los denominados zero-day-attack o ataques de día cero [11], que hacen referencia a los ataques sobre vulnerabilidades de los sistemas que aún son desconocidas por los usuarios y por el desarrollador del producto, por lo que son muy dañinas para toda la red.

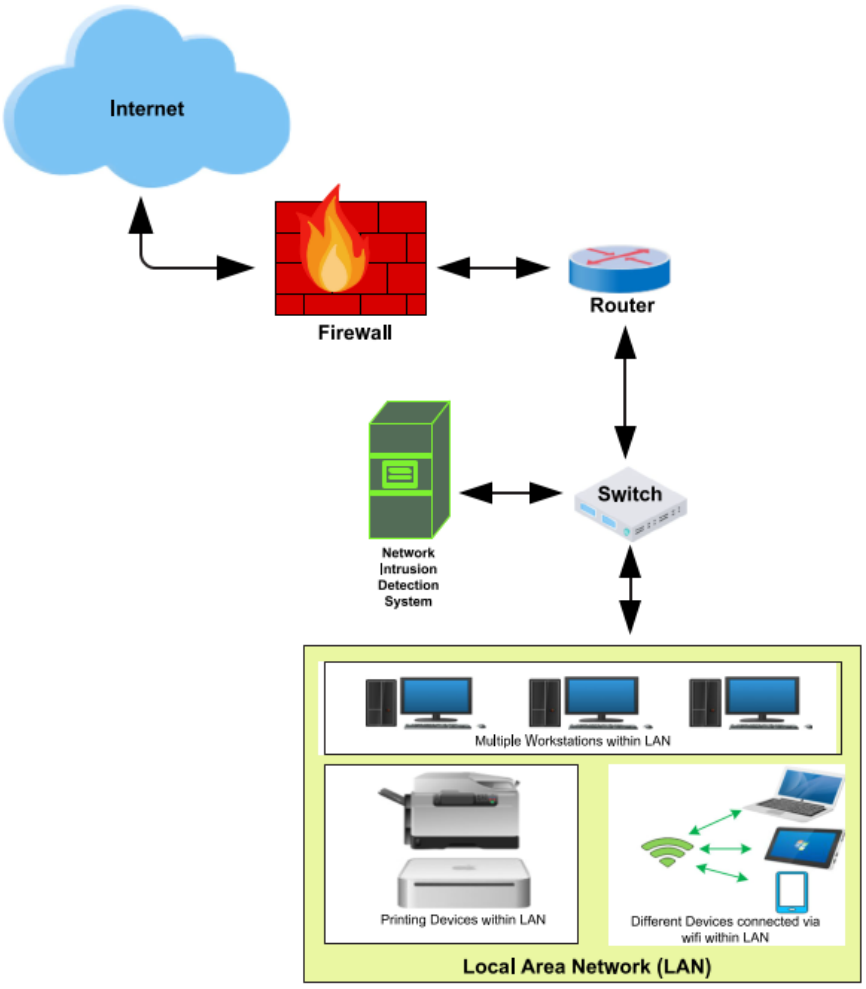


*Figura 1: Red local protegida mediante un firewall y un IDS basado en red [12]*

De lo expuesto anteriormente, se concluye que hoy en día es indispensable la utilización de técnicas de detección de intrusiones para intentar hacer frente al desmesurado ritmo de crecimiento de las potenciales vulnerabilidades y ataques en todo tipo de redes.

### 2.1.1 CLASIFICACIÓN DE LOS IDS

Los sistemas de detección de intrusos se clasifican atendiendo a diversos criterios.

Por un parte, si atendemos al método de despliegue de estos sistemas es decir, a la fuente de donde se obtengan los datos, se pueden distinguir dos tipos de IDS [2]:

A. **Host-based IDS (HIDS)**: estos sistemas basados en un host se implementan en una máquina, monitorizando toda la actividad interna del sistema. Comprueban todos los dispositivos conectados a la máquina, las aplicaciones que están siendo ejecutadas, así como los servicios que están funcionando en segundo plano en dicho sistema. Con esta información, se detecta si existe alguna condición sospechosa que pueda comprometer la seguridad del sistema. La gran desventaja de este tipo de IDS es que debe de ser implementado en todos los hosts que requieran protección, por lo que resulta una gran sobrecarga de todos los nodos, con la consecuente reducción del rendimiento total.

B. **Network-based IDS (NIDS)**: consisten en sistemas de detección de intrusos basados en red que detectan amenazas mediante el análisis del tráfico de la red. Normalmente los NIDS requieren acceso promiscuo a la red para poder analizarla en su totalidad. Trabajan de forma pasiva, sin interferir en el tráfico que monitorizan. Al implementarse sobre la red, resuelven la desventaja de los HIDS, protegiendo todos los dispositivos pertenecientes a la red y sin tener un impacto negativo en el rendimiento de los hosts del sistema. Los NIDS monitorizan constantemente la red para encontrar cualquier brecha en la seguridad que pueda comprometer a los sistemas.

Por otra parte, atendiendo a la perspectiva del modo de detección de riesgos de seguridad, los Sistemas de Detección de Intrusos se pueden clasificar en dos tipos [2]:

A. **IDS basados en firmas (Signature based IDS, SIDS)**: estos sistemas basados en el conocimiento se basan en la definición de firmas que identifican ciertos patrones comunes de ataques. Estas firmas son almacenadas en una base de datos, relacionándose con los patrones de ataques para su detección. Los sistemas que utilizan esta técnica son muy eficientes a la hora de detectar ataques conocidos que pueden relacionar con los patrones. Sin embargo, su gran desventaja es la detección de nuevos ataques de los que no tienen firmas asociadas a su patrón.

B. **IDS basados en la detección de anomalías (Anomaly Detection based IDS, AIDS)**: estos sistemas se basan en la definición de un comportamiento normal del equipo, estableciéndose unos límites que no debería sobrepasar. En el caso de que haya una desviación significativa de su comportamiento normal es considerada como una anomalía en su comportamiento. La gran ventaja de los AIDS consiste en la detección de nuevos ataques que puedan comprometer los sistemas de la red, así como su gran posibilidad de utilización en diferentes aplicaciones y dispositivos. La desventaja de estos sistemas viene dada por la dificultad de establecer unos límites entre el funcionamiento normal y el anormal del sistema.

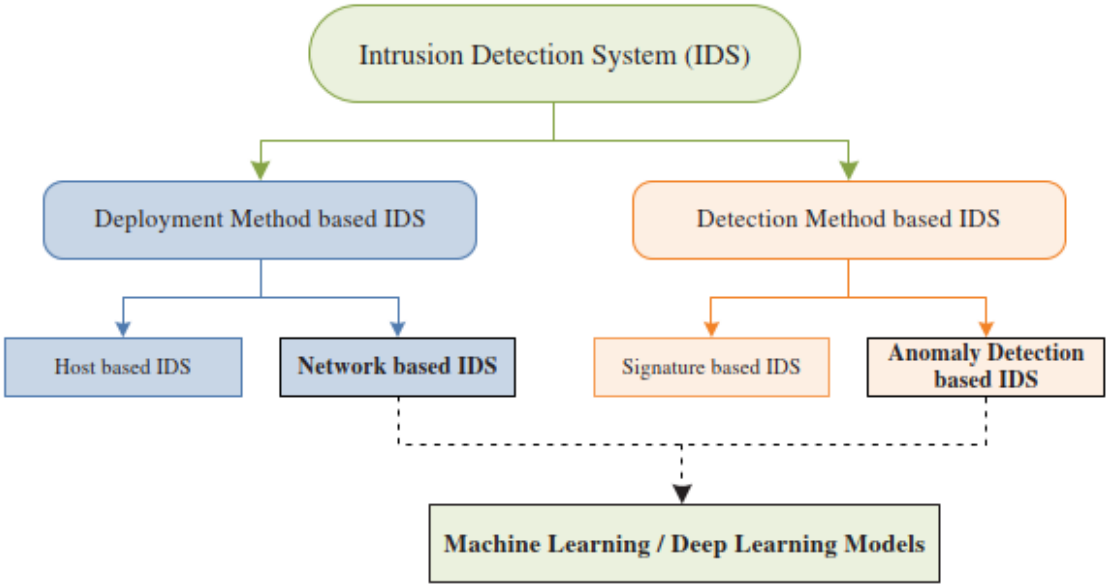


*Figura 2: Clasificación de los IDS [13]*

### 2.1.2 TIPOS DE INTRUSIONES NO DESEADAS EN LA RED

A continuación se van a describir los principales tipos de intrusiones no deseadas en un sistema o en una red, y cuya presencia debería de ser detectada por los sistemas de detección de intrusos [14].

1. **Malware**. Existen numerosas variedades de malware. Entre los más conocidos están los gusanos, troyanos y virus. Normalmente, todos los malware están diseñados para infectar los equipos y dañar o interrumpir el correcto funcionamiento del sistema que los alberga. Los tipos de malware con mayor capacidad para propagarse son los gusanos, que pueden autopropagarse, y los virus, que se ayudan de software del host para propagarse por los sistemas [8]. Por ejemplo, el ransomware es un tipo de software malicioso que impide el acceso a ciertas partes del sistema mediante técnicas muy avanzadas de encriptación, pidiendo una recompensa por la clave de encriptación del sistema. Por otra parte, los troyanos son un tipo de malware que intenta parecer legítimo, pero en realidad es malicioso, y necesitan ser propagados por los usuarios.

2. **DoS y DDoS attacks**. Un ataque DoS (Denial lof Service) consiste en la utilización de un computador para agotar la capacidad de un servidor mediante el envío continuo y masivo de paquetes hacia dicho servidor, ralentizando o incluso interrumpiendo el servicio que prestaba. Los ataques DDoS (Distributed Denial of Service), mucho más comunes que los anteriores, consisten en el mismo sistema que los DoS, pero mediante la utilización de múltiples computadores de diferentes lugares que se organizan para saturar uno o varios servidores objetivos del ataque, produciendo un bombardeo de paquetes sobre ellos, y su consecuente interrupción del servicio. Para realizar los ataques DDoS a menudo se utilizan Botnets, que están compuestas por un conjunto de dispositivos conectados a Internet, pudiendo llegar a ser miles de dispositivos, que son controlados por los propietarios para conectarse de forma conjunta a un servidor y realizar un ataque a gran escala sobre dicha víctima [15].

3. **Probe attac**k. Este tipo de ataque está especialmente elaborado para que un sistema de detección de intrusos colaborativo (CIDS) lo detecte, de forma que cuando lo haga y transmita la alarma a la infraestructura colaborativa del IDS, se transmita también parte del código, que hace que el atacante sea capaz de conocer la localización digital del sistema que le ha detectado, así como las capacidades defensivas y el resto de información del sistema integral de defensa y su red [16].

4. **Remote to Local** (R2L). Consiste en el acceso no autorizado de una máquina de forma remota al sistema víctima del ataque, con las consiguientes consecuencias que esto puede acarrear, desde pérdida de información confidencial, pérdidas económicas, o cualquier otro tipo de acción prohibida en el sistema. Entre muchos otros, destacan los ataques IMAP, que, mediante el desbordamiento de los buffers del sistema víctima, aprovechan errores de autentificación del sistema IMAP para obtener los privilegios de administrador [17]. También destacan los ataques FTP de escritura, que, gracias a una mala configuración del sistema respecto a los privilegios de escritura de usuarios en un servidor FTP, le permite al hacker subir archivos maliciosos al sistema.

5. **User to Root** (U2R). En este tipo de ataques el hacker accede al sistema con un usuario y contraseña normal, a partir del cual aprovechará las vulnerabilidades que encuentre en el sistema para tener privilegios de administrador y adueñarse de dicho sistema [17]. Puede realizarse de muchas formas, entre las que destacan los ataques de infiltración, como por ejemplo inyección SQL.

6. **Ports Scan** (Escaneo de Puertos). Los ataques de escaneo de puertos son ataques R2L que tratan de analizar el estado de los puertos de un computador conectado a una red mediante el envío de paquetes TCP o UDP, de forma que graban las respuestas del sistema, lo que les permite conocer el estado actual del sistema, así como posibles vulnerabilidades que pueden explotar para infiltrarse en el sistema [17].

7. **Brute Force Attack** (Ataques de Fuerza Bruta). Los ataques de fuerza bruta son aquellos en los que se accede a un usuario del sistema mediante la recuperación de su clave o contraseña probando las máximas combinaciones posibles hasta encontrar la adecuada que permite al atacante el acceso. Estos ataques pueden ser remotos (R2L) o no, así como desembocar en un ataque U2R [17].

8. **Fuzzing**. Esta técnica consiste en la utilización de programas automatizados que prueban datos aleatorios o inesperados de entrada en un sistema, con la intención de monitorizar las excepciones y errores que se producen, lo que posibilita el descubrimiento de problemas y vulnerabilidades de seguridad tanto en el hardware como en el software del sistema [18].

## 2.2 DEEP WEB Y DARKNET

La Deep Web es la parte de internet que sigue los protocolos del World Wide Web, pero que no están indexados por los navegadores convencionales. Por lo tanto, la Deep web engloba toda la información online que no puede ser accedida públicamente, incluyendo todo tipo de bases de datos, credenciales, datos bancarios, documentos, webs temporales y páginas privadas que se almacenan en Internet, a las que no se puede acceder si no es con sin cierta autorización o permisos. Dentro de la Deep web, una pequeña parte pertenece a la Darknet. La Darknet es un conjunto de redes ocultas para los navegadores convencionales a los cuales sólo se puede acceder con un software, configuración o protocolos especiales. Este tipo de redes tienen la característica de guardar la privacidad o anonimidad del usuario/máquina que realiza la conexión [19]. Esto da lugar a un sinfín de aplicaciones, que, evidentemente, pueden tener fines tanto legales como ilegales.

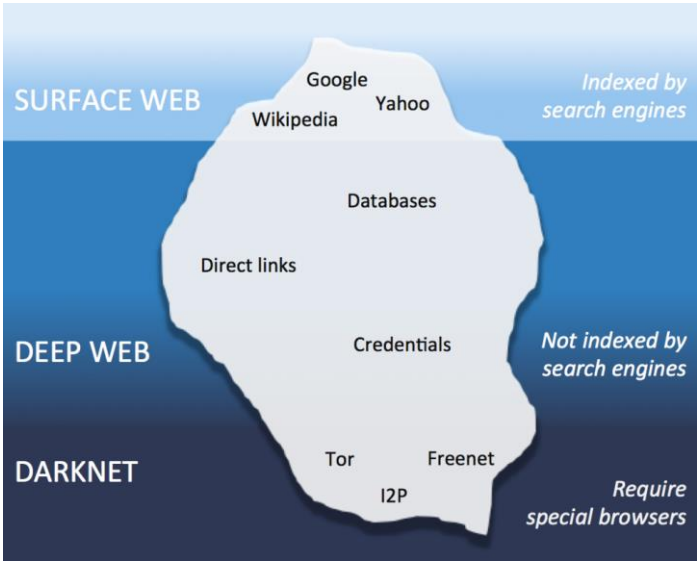


*Figura 3: Clasificación del tráfico de Internet WWW [20]*

El origen de la Darknet viene del Gobierno de Estados Unidos, quienes durante los años 90 crearon un tipo de intercambio de información de manera anónima, que utilizarían los servicios de inteligencia para proteger los documentos secretos que se intercambiaban. A este sistema lo denominaron TOR, cuya estructura está basada en una distribución en forma de cebolla que representa las diferentes capas que lo componen. De este concepto viene la extensión de sus páginas web: .onion. Más tarde, el gobierno de EEUU decidió hacerlo público con el fin de rellenar la red con el mayor

número posible de conexiones y dificultar aún más la posibilidad de identificar la información que intercambiaban a través de la misma [21].

La lógica que sigue TOR consiste en hacer pasar la información por numerosos y diferentes nodos, es decir, servidores que están conectados entre sí y encriptados (las diferentes capas), que finalmente permiten al usuario esconder las credenciales de la conexión (es decir, la IP) a través de la cual se ha conectado a la red.

Los usuarios que hacen uso de la red TOR suelen hacer uso de VPNs para cifrar sus conexiones, como si de un túnel se tratase, con el fin de protegerse de los nodos maliciosos de TOR, y también ocultar de esta forma su Proveedor de Servicios de Internet (ISP) [22]. El uso de ambas herramientas conjuntamente proporciona una privacidad y seguridad adicional al uso individual de las mismas, al proteger la identidad del usuario final de diferentes maneras.

Durante los últimos años, se han dedicado grandes esfuerzos en detectar el tráfico encriptado de la Darknet frente al tráfico convencional de una conexión estándar. Para los NIDS, cualquier comunicación de la Darknet se considera dudosa debido a su naturaleza pasiva, al aceptar tráfico entrante, pero no admitir el tráfico saliente. Por ello, uno de los últimos enfoques en esta materia ha sido la utilización de técnicas emergentes como las redes neuronales, que permiten a las aplicaciones clasificar en tiempo real el tráfico de red en benigno o maligno con cierta certeza, monitorizando así el posible malware previamente a la realización de la actividad maliciosa [23].

## 2.3 MACHINE LEARNING Y DEEP LEARNING APLICADOS A NIDS

Como ya se ha mencionado, en los últimos años las técnicas de Machine Learning (ML) y Deep Learning (DL) han ayudado a los investigadores a mejorar el rendimiento de los Sistemas de Detección de Intrusos, los cuales basándose en estas soluciones, han visto cómo su potencial ha crecido notablemente para la detección amenazas en la red. Ambas técnicas vienen de la mano de la Inteligencia Artificial, que, aprovechándose de la utilización de la información generada por la gran cantidad de datos que nos rodea, intenta imitar comportamientos inteligentes del ser humano en los ordenadores, mediante la generación de conocimiento a partir de un conjunto de experiencias que son pasadas al sistema como datos de entrada.

Tanto las herramientas de ML como las de DL tienen como objetivo generar modelos que permitan encontrar patrones en los datos con la finalidad de predecir algo, mediante la utilización de algoritmos complejos. La diferencia entre ambas radica en que, en los algoritmos de ML, los investigadores tienen que extraer las características de los datos de entrada, mientras que los algoritmos de DL aprenden automáticamente estas características, por lo que normalmente son algoritmos más complejos y se utilizan para resolver problemas a priori más complicados [24]. Los algoritmos de DL son parte del ML, y consisten en redes neuronales artificiales que aprenden ellas solas con cada nueva entrada de información que es introducida en la red. Son capaces de generar conocimiento de forma jerarquizada por niveles, dando lugar a complejos algoritmos de aprendizaje automático con muchas capas de información. Esta es la razón por la que esta técnica se denomina aprendizaje profundo o Deep Learning.

Estas técnicas han ganado mucha popularidad para la detección de intrusiones, pues han demostrado ser herramientas muy potentes en el análisis de características de las redes, así como en la predicción de su actividad normal y anormal, basándose en patrones de su comportamiento [25]. De esta forma, las investigaciones de la última década han estado centradas en encontrar soluciones de ML y DL que hagan más eficientes los NIDS, pero el creciente aumento de tráfico en la red y sus consecuentes amenazas han supuesto una gran dificultad para estos.

Se ha de destacar que, respecto a las clasificaciones de los IDS del apartado anterior según la procedencia de los datos y del modo de detección de las amenazas de seguridad, los IDS que utilizan técnicas de ML y DL están basados en red (NIDS) y en la detección de anomalías (AIDS).

Un NIDS basado en técnicas de ML y DL pasa por 3 fases generals [26]:

1. Preprocesamiento de los datos originales del dataset utilizado, cuando se transforman los datos para ser utilizados por el algoritmo.

2. Fase de entrenamiento, cuando se aplica el algoritmo de ML o DL y se genera un modelo.

3. Fase de test (prueba), cuando se demuestra el funcionamiento del modelo generado y las instancias del tráfico de red se clasifican en tráfico normal o ataques. Normalmente se utiliza el 80% del dataset para el entrenamiento y el 20% para el testing.

El tiempo que tarda el algoritmo en entrenar el modelo depende tanto del tamaño del dataset como de la complejidad de dicho modelo. Lo normal es que la fase de entrenamiento ocupe más tiempo debido a la complejidad de su estructura [26]. La siguiente figura muestra el esquema del NIDS:

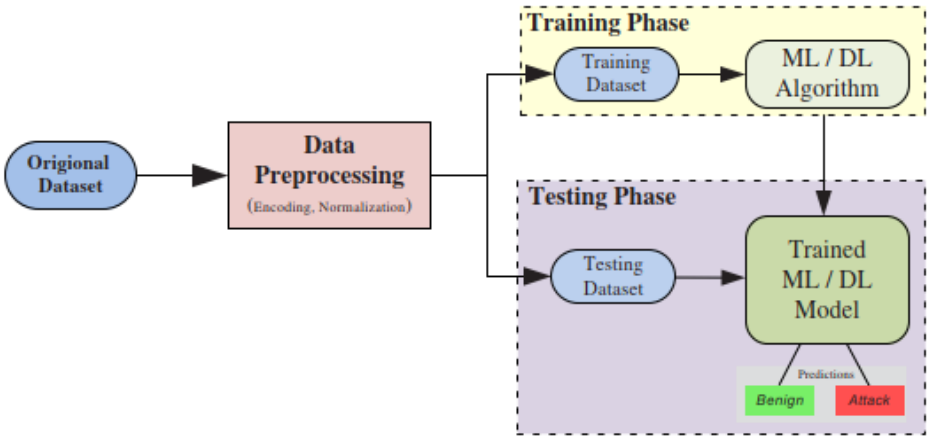


*Figura 4: Funcionamiento de un NIDS basado en técnicas de ML o DL. Fuente: [26]*

Tanto los algoritmos de ML como los de DL pueden ser supervisados o no supervisados. En los algoritmos supervisados la información se extrae de datos etiquetados, mientras que en los algoritmos no supervisados los algoritmos utilizan datos no etiquetados para extraer información [27].

A continuación se van a describir los algoritmos principales de ML y DL que son utilizados para la creación de IDS eficientes.

### 2.3.1 ALGORITMOS DE MACHINE LEARNING (ML) EN NIDS

El ML incluye los métodos y algoritmos que permiten a la máquina aprender automáticamente utilizando modelos matemáticos para extraer conocimiento de los amplios dataset con los que trabaja. Los algoritmos de ML más utilizados y eficientes para NIDS son [28]:

1. **Árbol de decisión**: es el algoritmo supervisado más básico y utilizado del ML. Utiliza la clasificación y regresión de los datos al aplicar una serie de reglas, dando como resultado una estructura de árbol con raíz, nodos, ramas y hojas. Cada nodo del árbol representa una característica diferente del conjunto de datos (dataset). Las ramas representan una decisión o regla aplicada para clasificar el árbol. Las hojas representan las posibles salidas o estados finales de los datos. El árbol de decisión selecciona automáticamente las mejores características con la información que contiene, eliminando las ramas (las decisiones) que no son relevantes. Su uso es muy común en algoritmos más complejos como el Random Forest.

2. **Random Forest**: consiste en la combinación de múltiples árboles de decisión, donde cada uno individualmente predice una clase de salida, y la que más sea votada será la predicción del algoritmo. Para que la salida del Random Forest sea fiable, los modelos de los árboles de decisión deben tener una correlación muy baja entre ellos, con la intención de protegerse ante errores individuales y valores anómalos.

3. **Naive Bayes**: es un clasificador basado en la Teoría de Bayes, cuyo fundamento es que las características de las predicciones son independientes y no afectan a la probabilidad de ninguna otra característica. La gran ventaja de Naive Bayes es que necesita pocos datos de entrada para determinar los parámetros de clasificación del modelo, que son las varianzas de cada variable.

4. **Regresión logística**: es un algoritmo de clasificación utilizado para asignar un cierto resultado a una variable dependiente a partir de un conjunto de variables características utilizando una función logística. La salida del algoritmo es la probabilidad de que ocurra la variable dependiente en función de dichas características.

5. **KNN (K-Nearest Neighbor):** este algoritmo supervisado se basa en la idea de predecir la clase de cada muestra de los datos mediante la similitud de características con otros. Clasifica las muestras calculando la distancia o diferencia de características con sus K vecinos más cercanos del conjunto de datos.

6. **Máquina de vectores de soporte (SVM)**: se trata de un algoritmo supervisado de ML que genera un modelo basado en la máxima separación marginal entre los puntos de las muestras en el espacio, separando las clases en espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector de soporte entre los puntos de las clases más cercanos. El hiperplano funciona como una frontera de decisión que utiliza el vector de soporte para clasificar las muestras.

7. **Agrupamiento K-medias:** el agrupamiento (clustering) trata de dividir las muestras de los datos en grupos (o clusters) establecidos según la similitud de sus datos. El agrupamiento K-medias es una técnica no supervisada en el que existen K centroides de clústeres del dataset. Cada muestra del conjunto de datos se representa por un punto, que es asignado a un clúster u otro de forma que la suma de las distancias entre los puntos y sus centroides sea mínima.

8. **Red neuronal artificial (Artificial Neural Network, ANN):** es un algoritmo supervisado de ML que intenta simular el funcionamiento de las neuronas del sistema nervioso humano. Está formado por un conjunto de nodos organizados en capas de entrada, internas y de salida. Una ANN se basa en el algoritmo perceptrón y en el mecanismo de Backpropagation, que permite a las neuronas aprender solas mediante los datos de entrada con los que se entrena. El algoritmo perceptrón entrena cada nodo de la red neuronal para averiguar los coeficientes de la función de peso que hace que, combinada con los datos de entrada, ofrezca un resultado concreto de salida. Durante este proceso de entrenamiento el nodo (o neurona) ajusta estos coeficientes para conseguir descubrir patrones no visibles en los datos de entrada, que permiten identificar un determinado dato. Una red neuronal artificial estará formada por un conjunto de perceptrones conectados entre ellos en estructuras complejas [29].

9. **Métodos conjuntos (Ensemble methods)**: la idea de esta técnica es aprovechar los beneficios de diferentes clasificadores para extraer el conocimiento de manera conjunta, de forma que también se minimicen las debilidades de cada uno frente a ciertos tipos de ataques, formando un

clasificador mucho más potente que se ayuda de un sistema de voto para cada decisión conjunta. Por ejemplo, podría generarse un modelo conjunto mediante la utilización de árboles de decisión, algoritmos KNN y ANN.

### 2.3.2 ALGORITMOS DE DEEP LEARNING (DL) EN NIDS

La diferencia del DL con las redes neuronales artificiales del ML radica en el procesamiento jerárquico de los datos mediante el aprendizaje por capas de la red neuronal. Son más eficientes que las técnicas de ML al tener una estructura mucho más compleja y profunda, formada por un gran número de capas de redes neuronales conectadas entre sí. Por lo tanto, el DL es una parte del ML que incluye una gran cantidad de etiquetas no visibles que caracterizan la red neuronal profunda [29]. A continuación se describen los algoritmos de DL más utilizados para NIDS [30]:

1. **Redes neuronales recurrentes:** este tipo de redes neuronales están diseñadas de forma que las salidas de cada neurona están interconectadas con la propia neurona o con otras del mismo o inferior nivel dentro de la red. Este sistema permite una especie de retroalimentación de la información que otorga una memoria temporal a la red neuronal [30]. Este tipo de redes puede ser utilizada en un IDS para la extracción de características y la clasificación supervisada del dataset del tráfico de la red.

2. **Autoencoder**: este algoritmo no supervisado es ampliamente utilizado en el entorno de la detección de intrusos. Utiliza redes neuronales con el objetivo de generar nueva información a partir de los datos de entrada, mediante la compresión de los datos con la parte “Encoder”, transformándolos en variables que caracterizan los datos. Posteriormente se reconstruyen los datos iniciales con el “Decoder” basándose en la información recogida. Lo interesante del algoritmo es que recoge las variables características más importantes de los datos, de forma que se reduce el ruido y muchas dimensiones que no son útiles para la red neuronal [31].

3. **Redes neuronales profundas**: es un algoritmo básico del DL cuyas neuronas se componen de muchas capas ocultas entre las de entrada y las de salida. Al aumentar el número de las capas ocultas se aumenta el nivel de abstracción del modelo, así como sus capacidades de procesamiento. Esto permite utilizar las redes neuronales profundas en modelos complejos de funciones no lineales. Los

estudios recientes han demostrado la robustez de este tipo de técnicas aplicadas a IDS, especialmente en aquellos modelos con un gran número de registros [30].

4. **Redes de creencia profundas**: son redes neuronales formadas por la concatenación de varias Máquinas de Boltzmann Restringidas (MBR) seguidas por una capa de clasificación. Un MBR es un modelo de dos capas de entrada y capa oculta cuyo flujo de información es bidireccional, ya que cada nodo de la capa está conectado a otros nodos de la capa anterior y siguiente, pero no está conectado con los nodos de la misma capa. Estas capas se entrenan con un algoritmo de aprendizaje no supervisado de divergencia por capas, seguido de un algoritmo supervisado de ajuste para obtener las características principales. [30] En la detección de intrusos se utilizan para la extracción de características útiles del tráfico de datos y para tareas de clasificación.

5. **Redes neuronales convolucionales**: es un algoritmo de DL normalmente utilizado para trabajar con imágenes debido a su similitud con el funcionamiento de las neuronas en la corteza visual del cerebro humano. Este algoritmo consiste en una capa de entrada, así como múltiples capas ocultas especializadas y con jerarquía, que contienen filtros convolucionales (denominados kernels) que utilizan matrices bidimensionales para la identificación de patrones característicos en las imágenes. Además, tienen una capa externa más sencilla que actúa como clasificador de las muestras según las características extraídas [32].

### 2.3.3 MÉTRICAS DE EVALUACIÓN

En este apartado se van a explicar las principales métricas de evaluación del rendimiento de los algoritmos del ML y DL. Estas métricas utilizan la matriz de confusión, que es la matriz bidimensional que compara las instancias de las clases actuales con las que se han predicho. Los atributos que aparecen en esta matriz son [33]:

- **True Positive** (TP, positivos verdaderos): son las instancias que el clasificador del modelo del IDS predice correctamente como ataques.
- **False Positive** (FP, falsos positivos): son las instancias erróneamente predichas como ataques por el clasificador del modelo del IDS, ya que el modelo ha fallado y se trata de tráfico de red normal.
- **True Negative** (TN, negativos verdaderos): son las instancias que el modelo del IDS predice correctamente como tráfico de red normal.
- **False Negative** (FN, falsos negativos): son las instancias que el clasificador del modelo del IDS predice erróneamente como tráfico normal cuando en realidad son ataques.

En la siguiente imagen se puede apreciar de forma visual los posibles resultados de las predicciones de una instancia del conjunto de datos.

| | | Instancia predicha | |
|---|---|---|---|
| | | Ataque | Tráfico normal |
| Instancia actual | Ataque | True Positive (TP) | False Negative (FN) |
| | Tráfico normal | False Positive (FP) | True Negative (TN) |

*Tabla 1: Matriz de confusión de las instancias predichas. Fuente: Elaboración propia.*

Por lo tanto, las diferentes métricas de evaluación utilizadas para los modelos de detección de intrusos son [33]:

- **Precision**: consiste en la ratio de instancias correctamente predichas como ataques (TP) frente al total de clases predichas como ataques (TP + FP). Por lo tanto, la formula será:

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

- **Accuracy**: es la ratio que mide las instancias predichas correctamente frente al número total de instancias. La fórmula se define como:

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

- **Detection rate o recall** (ratio de detección): esta ratio mide la cantidad de ataques detectados (TP) en comparación con el total de ataques del conjunto de datos (TP + FN). La fórmula vendrá dada por:

$$Detectionrate = \frac{TP}{TP + FP}$$

- **False alarm rate:** esta ratio mide la cantidad de tráfico normal que el modelo del IDS predice como ataque cuando no lo es (FP) frente al total de tráfico normal (TN + FP). Se puede obtener como la siguiente relación:

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

- **ROC**: consiste en la curva que mide la relación entre los verdaderos positivos y los falsos positivos. Es utilizado para comparar modelos óptimos frente a otros subóptimos.
- **F Measure (o F-Score)**: es una medida definida como la media armónica de la precisión y la ratio de detección, utilizada para medir el rendimiento de un modelo. Su fórmula es**:**

$$FMeassure = 2\left(\frac{Precision * Detectionrate}{Precision + Detectionrate}\right)$$

## 2.4 DESCRIPCIÓN BÁSICA DEL FUNCIONAMIENTO DE UNA RED NEURONAL

En este apartado se va a explicar de manera asequible y concisa las bases de una red neuronal. Se pretende que se pueda comprender su funcionamiento sin una formación previa especializada en esta materia. Es importante entender y mencionar que todas las explicaciones posteriores tienen una base científica y matemática con operaciones computacionales complejas, cuyo estudio no es objetivo de este capítulo. Cabe destacar que la redacción de este apartado ha sido en base a los conocimientos adquiridos durante la visualización del curso *Deep Learning Fundamentals* [34] del canal *Deep Lizard* [35] de la Plataforma Youtube.

Dicho esto, comenzamos redefiniendo el concepto de red neuronal explicado anteriormente en el apartado *2.3.1 Algoritmos de Machine Learning (ML) en NIDS*. Una red neuronal consiste en un conjunto de patrones de operaciones matemáticas mediante los cuales se obtiene un modelo computacional complejo con una determinada estructura, y que puede ser empleado bien para la clasificación de datos de entrada en diferentes tipos de datos de salida (clases), o bien para predecir un determinado valor de un tipo de dato de salida (regresión). Podemos entender la Red Neuronal como una función de clasificación con cientos de parámetros inicializados de manera aleatoria que se autoajusta en base a los resultados obtenidos para un conjunto de datos de entrada etiquetados, con el fin de dar una solución general al problema que se le plantea. Este proceso de autoajuste se llama entrenamiento, y es por esta capacidad que se encuadra dentro de las técnicas de aprendizaje automático o machine learning.

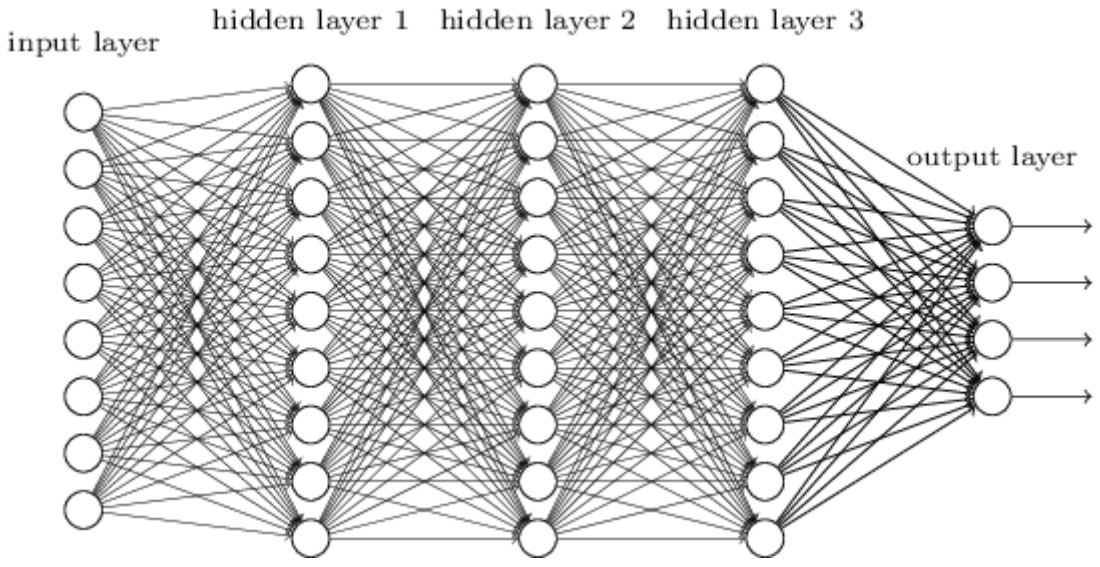


*Figura 5: Esquema del funcionamiento de una red neuronal con 5 capas [36]*

Las redes neuronales tienen una estructura característica formada por pequeñas unidades, funciones matemáticas o neuronas, que realizan operaciones sencillas, y se organizan en diferentes niveles, capas o layers, que en su conjunto forman a su vez la propia red neuronal. Cada neurona se conecta tanto a las neuronas de la siguiente capa como a las de la capa anterior, regulando la información que se va transmitiendo en cada nivel de la red. La primera capa de la red consta de los datos de entrada, mientras que la última capa está constituida por los de salida, cuyos valores indican el grado en que la red neuronal considera que los datos pertenecen a esa determinada clase de la última capa. Los valores calculados en cada función matemática de las neuronas se conocen como la activación de la neurona, calculada como la ponderación de cada variable de entrada (con tantas entradas como neuronas tenga la capa anterior de la red) multiplicadas por un vector de pesos. A este valor, a su vez, se le aplica una función de activación, que acota entre 0 y 1 el resultado obtenido, y que servirá como valor de entrada para todas las neuronas de la siguiente capa. Si esta función no existiese, la estructura de la red neuronal sería trivial, al poder ser reducida a una única función lineal. La activación de las neuronas de la última capa determina por tanto la clasificación que ha realizado la red neuronal en cuestión de unos determinados datos de entrada. A este proceso individual que realiza cada neurona se le denomina Perceptrón. Además, cabe destacar que la red neuronal se inicializa con las variables de pesos y sesgos con valores aleatorios, y se refina en cada capa para acabar ajustando los valores de manera que pueda asignar la clase correspondiente a los diferentes valores de entrada de la primera capa.

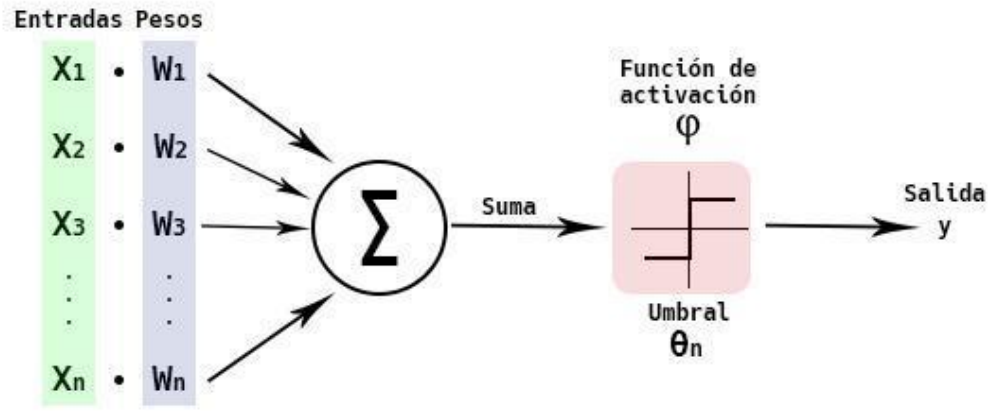


*Figura 6: Esquema simplificado del cálculo realizado por un Perceptrón [37]*

Todo este proceso, en su conjunto, se denomina entrenamiento de la red neuronal. Su funcionamiento está basado en la minimización de la función de coste de unos datos ya

etiquetados, es decir, para unos datos de entrada de los que ya se conoce la clase que les corresponde en el último nivel de la red.

Para conseguir minimizar la función de coste de cada unidad de la red, se emplean los optimizadores, que aplican el método del descenso de gradiente, obteniendo un valor de pérdida o coste que establece cómo debe variar cada variable de la red para minimizar dicha función. Se modifican de esta manera las variables y se vuelve a calcular la función de coste. En cada iteración, se van modificando estos valores, consiguiendo minimizar el coste total del problema. Este proceso reiterativo hacia atrás se conoce como Retropropagación o Backpropagation, gracias al cual las neuronas aprenden a minimizar el coste y por tanto a ajustarse más al resultado correcto, representado por la clase de la etiqueta de la última capa. La complejidad computacional de las redes neuronales es debida a este proceso de cálculo que incluye numerosas funciones aplicadas a cientos de parámetros en cada capa de la red.

A continuación se va a explicar de forma más detallada, ordenada, en lenguaje asequible y de la forma más simplificada posible, los pasos de cálculo de una red neuronal:

**A) Propagación hacia delante o Forward Pass:**

En cada neurona de la red tenemos un vector que recoge el valor de cada uno de los datos de entrada que han llegado a esta unidad. A este vector cada neurona le aplica otro vector, denominado vector de pesos, y a ese resultado una función de activación, que finalmente dará como resultado un valor que corresponderá con la activación de cada neurona. Aplicando esto consecutivamente, en la última capa de la red neuronal se obtendrán los resultados de activación para cada clase, es decir, el grado en que la red neuronal cree que los datos de entrada pertenecen a cada clase representada por cada una de las neuronas de esta capa.

**B) Descenso de gradiente empleando retropropagación o Backpropagation:**

Con el objetivo de ir ajustando progresivamente el vector de pesos de cada neurona que conforma la red neuronal, se elige una función de coste que valora los resultados obtenidos en la red neuronal en función del valor real que se debería haber obtenido (gracias a las etiquetas de la última clase). Esta función calcula un coste total medio para cada ejemplo de entrenamiento. Como el objetivo es la minimización de esta función de coste, la división entre el número de entrenamientos que se realizan es indiferente.

Además, para cada entrenamiento de la red se emplean batches, que consisten en conjuntos de datos de entrada que serán procesados por el modelo antes de aplicar los cambios correspondientes. Esto significa que aplicaremos los cambios o ajustes a las variables del modelo sólo cuando hayamos obtenido los cambios que minimizan el coste para cada ejemplo individual, y que luego agregaremos para cambiar el modelo una única vez por batch. Para ello, es imprescindible que la función de coste esté formada por la media de los costes individuales para cada ejemplo de entrenamiento. De esta manera, el empleo de batches no distorsiona los resultados finales, y a su vez se obtienen grandes ahorros computacionales, al no tener que calcular la actualización de los parámetros del modelo para cada dato de entrada. Este funcionamiento es fundamental para permitir la paralelización de los cálculos.

Por otra parte, con el fin de obtener los ajustes a realizar en los valores del vector de pesos de cada capa anterior de la red neuronal, se computan las derivadas parciales del coste total de la función respecto a cada uno de los pesos del vector que componen los parámetros de la misma, obteniendo así la variación que minimiza la función de coste de cada neurona.

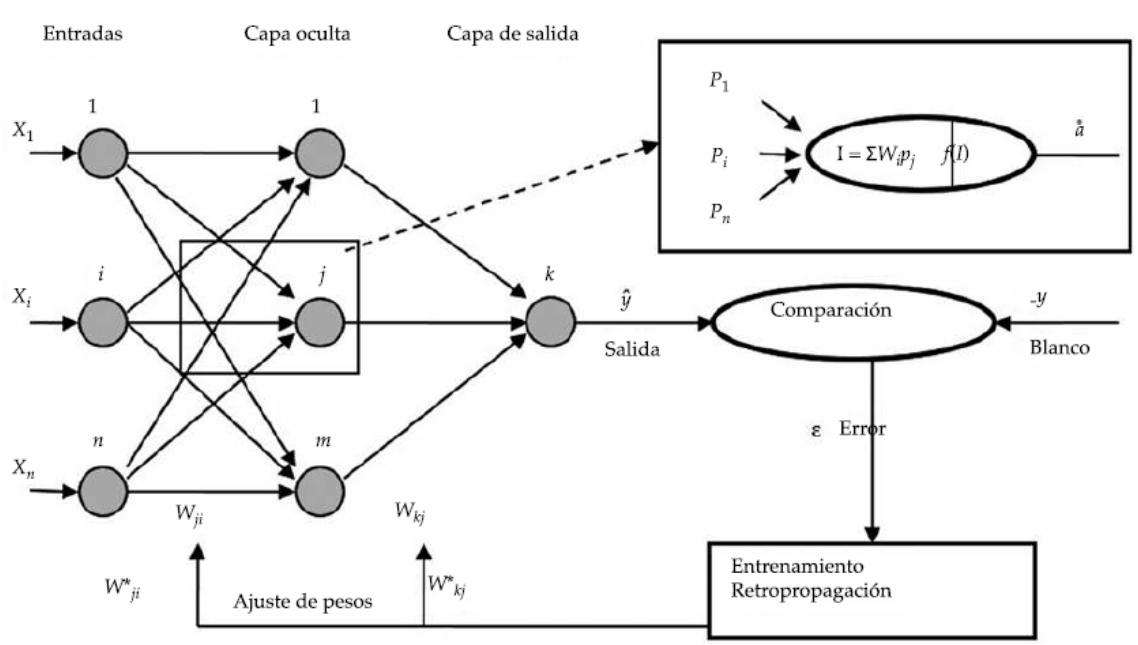


*Figura 7: Proceso backpropagation [38]*

Esta es la razón de que a este proceso se le denomine Backpropagation, ya que el cálculo se hace desde la última capa hacia la primera capa de la red neuronal. Esta

lógica permite calcular el error que tienen los parámetros de la última capa a partir del error total de la red, para posteriormente aplicar el mismo algoritmo y obtener el error de los parámetros de cada capa anterior en función del error de la capa inmediatamente posterior. Este error será eliminado modificando el vector de pesos de la neurona.

De forma resumida, el proceso de propagación hacia adelante se iterará para cada dato de entrada de la red neuronal, así como el cálculo de los gradientes. La actualización de los pesos se realizará gracias al proceso de Backpropagation una única vez por cada batch, empleando un vector de pesos de cada dato de entrada individual.

Debido a que el coste computacional de calcular todo este proceso multitud de veces para entrenar a la red neuronal es muy elevado, es usual utilizar técnicas de paralelización, que además limitan los accesos a memoria, como los denominados sistemas multiagente. Estos sistemas consisten en una composición de estructuras de inteligencia artificial individuales, que actúan cada una como un agente del sistema general, y que interactúan entre ellas mismas para resolver problemas complejos y prácticamente imposibles de resolver por un único sistema.

Tal y como se puede intuir, las aplicaciones de las redes neuronales son innumerables, pudiendo resolver problemas tanto de la vida cotidiana como de las investigaciones tecnológicas más punteras del momento.

# 3. MARCO EXPERIMENTAL

A continuación se va a describir el procedimiento llevado a cabo para la realización de este proyecto. Antes de nada, destacar que se partía de unos conocimientos muy básicos, pero tras el estudio y entrenamiento con diferentes tipos de problemas, se ha logrado comprender el funcionamiento e implementación de las redes neuronales.

## 3.1 *CIC-DARKNET2020:* JUSTIFICACIÓN Y ESTADO DEL ARTE

El primer paso de la parte práctica de este trabajo consiste en la elección del conjunto de datos (a partir de ahora lo denominaremos dataset) que se va a tratar como datos de entrada del problema. Tras una búsqueda exhaustiva de diferentes datasets que estén directamente relacionados con los sistemas de detección de intrusiones de red (NIDS), se ha encontrado el *CIC-Darknet2020* dataset [39], que consiste en dataset elaborado en 2020 que engloba a su vez otros dos conjuntos de datos: ISCXVPN2016 y ISCXTOR2017. El contenido de este dataset consta de más de 150.000 datos de entrada que contiene tanto tráfico de red benigno como conexiones TOR y VPN. Este último tipo de conexiones, tal y como se ha explicado en el capítulo *2.3 Deep Web y Dark Web* del presente trabajo, son consideradas dudosas y por tanto clasificadas como tráfico maligno. De esta forma, las conexiones de aplicaciones a través de TOR y VPN se proponen como representantes reales del tráfico de la Dark Web [39].

El dataset elegido ha sido elaborado por el Canadian Institute of Cybersecurity (CIC), que se encuentra en la Universidad de New Brunswick (UNB). Dicho dataset está disponible para descargarse de forma gratuita cumplimentando el formulario correspondiente accediendo a su página web [40]. Este instituto lidera numerosas investigaciones de ciberseguridad, innovación, privacidad y educación en Canadá, y está altamente reconocido en este mundo [41] por dichas aportaciones, así como por su contribución al Deep Learning a nivel internacional, al desarrollar numerosos datasets utilizados en numerosas universidades e investigaciones del sector, como el conocido NSL-KDD dataset de 2009.

En lo que concierne al dataset elegido para este proyecto, en la Figura 8 se puede apreciar la primera capa en que se ha dividido el tráfico de red. Cada conexión del dataset *CIC-Darknet202*0 está clasificada en conexiones benignas y Dark Web:

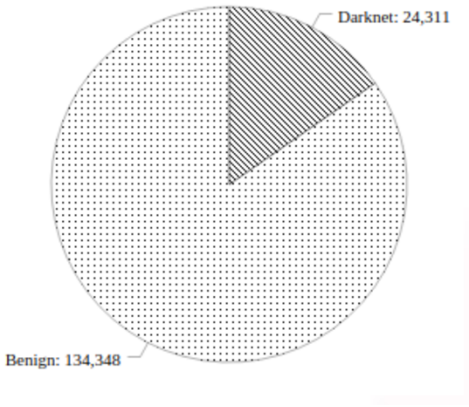


*Figura 8: Clasificación del tráfico de red presente en el dataset CIC-Darknet2020 [39]*

Las conexiones benignas tienen un “label” que indica “NonTor” o “NonVPN”, mientras que las conexiones de la Dark Web tienen el valor de “Tor” o bien “VPN”. Además, este dataset tiene una segunda capa de clasificación que divide el tráfico capturado en el tipo de conexión que se trata. Esta clasificación puede observarse en la Figura 9:

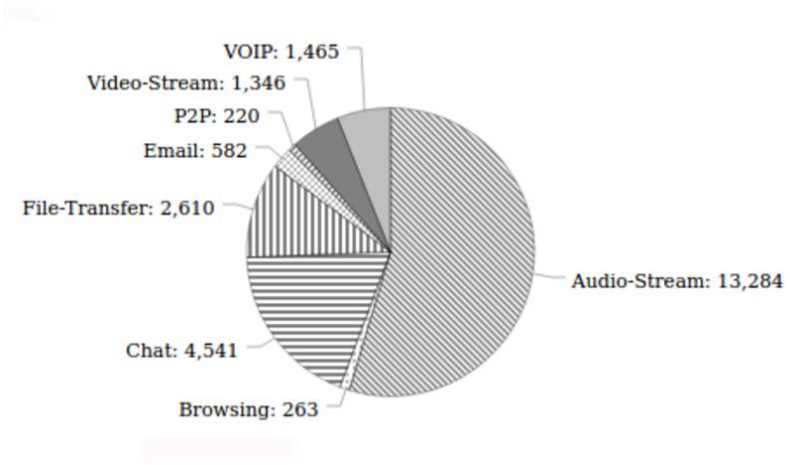


*Figura 9: Clasificación del tipo de conexiones presentes en el dataset CIC-Darknet2020 [39]*

De todas formas, en el presente trabajo, debido a su naturaleza de detectar tráfico sospechoso en un NIDS, no se va a trabajar con la segunda capa de clasificación. El objetivo será lograr la máxima efectividad en anticiparse al tráfico de Dark Web, es decir, en predecir con la máxima eficacia posible las conexiones que correspondan a conexiones TOR y conexiones VPN del dataset CIC-Darknet2020.

La aplicación de algoritmos de Deep Learning y Machine Learning para detectar tráfico de la red Darknet ha sido muy utilizada en los últimos años [42]. Además, durante los últimos años se han llevado a cabo investigaciones con resultados predictivos especialmente satisfactorios. En la siguiente tabla (Tabla 2) podemos observar un resumen de los últimos estudios realizados en este estado del arte, comparando el método de aprendizaje utilizado, la previsión para detectar y clasificar las conexiones Darknet, y el porcentaje de mejora respecto a los sistemas existentes para el problema.

| Year | Evaluation Model | Accuracy | I.F. % |
|---|---|---|---|
| 2021 | Recurring Neural Network (RNN) | 94.51% | 5.28%↑ |
| 2017 | Longitudinal Analysis of Network Traffic (LANT) | 94.00% | 5.85%↑ |
| 2020 | Hierarchical Classification Method (HCM) | 96.60% | 3.00%↑ |
| 2021 | AdaBoost Decision Trees (AB-DT) | 97.30% | 2.26%↑ |
| 2020 | Convolutional Neural Network (CNN) | 86.00% | 15.70%↑ |
| 2020 | Artificial Neural Network and Apache Spark (ANN-AS) | 94.66% | 5.11%↑ |
| 2021 | Convolutional Neural Network (CNN) and K-Means (KM) | 97.40% | 2.16%↑ |
| 2020 | Sparse Structure Learning with LASSO selection (SSL) | 97.10% | 2.47%↑ |
| 2021 | Convolutional Neural Network (CNN) | 97.65% | 1.89%↑ |
| 2019 | Random Forest Classifier (RFC) | 78.30% | 27.08%↑ |
| 2017 | Logistic Regression Classifier (LRC) | 96.60% | 3.00%↑ |
| 2022 | Bagging Decision Tree Ensembles | 99.50% | - |

*Tabla 2: Comparación de diferentes modelos de Machine Learning utilizados en la Darknet [43]*

El estudio realizado en 2022 utiliza un conjunto de árboles de decisión como método de aprendizaje, aplicado al mismo dataset de este trabajo, con una precisión del 99.50%. Tras probar una serie de 6 métodos supervisados de Machine Learning, llegaron a una solución eficiente y autónoma para el mismo problema [42]

Como se puede observar, los rendimientos obtenidos en la detección del tráfico maligno de la Darknet en los sistemas de detección de intrusos son bastante elevados, por lo que en este trabajo se pretenderá obtener la mayor tasa de precisión posible, teniendo en cuenta las limitaciones de una persona principiante en la materia, como es el caso.

## 3.2 FASES DEL DESARROLLO DEL TRABAJO

Una vez seleccionado el dataset, el siguiente paso es la preparación del equipo de trabajo. Uno de los puntos críticos de estas fases es el equipo y aplicaciones a utilizar para el desarrollo del trabajo. Los equipos locales utilizados como ordenadores personales pueden no tener la capacidad operacional suficiente para tratar este tipo de problemas tan complejos. Además, hay que controlar bien las versiones de los frameworks (entornos de trabajo) y bibliotecas a utilizar, ya que puede haber problemas de compatibilidad. Las aplicaciones que se han utilizado para el desarrollo de este trabajo son:

- **Tensorflow**: consiste en una biblioteca de software de código abierto desarrollada por Google Brain, considerada una interfaz de programación de aplicaciones (API), y aplicada al aprendizaje automático con el lenguaje de programación Python [44]. Este conjunto de funciones, procedimientos y algoritmos de computación numérica permite entrenar y ejecutar redes neuronales complejas para cualquier tipo de problema relacionado con el Machine Learning.
- **Keras**: consiste en un framework de alto nivel que permite agilizar y facilitar la creación de redes neuronales en distintos backends, como el anteriormente mencionado Tensorflow. Al igual que éste, está escrito en Python y se aplica al aprendizaje automático. La principal característica de esta biblioteca es ser user-friendly, intutitiva y extensible a diferentes aplicaciones [45].
- **Visual Studio Code**: este editor de código fuente gratuito y de código abierto fue desarrollado por Microsoft y puede utilizarse en diversos sistemas operativos, como Windows, Linux o macOS. Permite programar en diferentes lenguajes como Java, Python, JavaScript, HTML, Ruby, SQL, YAML, PHP y muchos otros [46]. Su principal característica es la personalización que ofrece al usuario y su rápido aprendizaje, al tener también una interfaz sencilla pero efectiva.

Una vez instalado todo el software necesario, y por tanto se tiene el equipo de trabajo preparado, se procede a implementar la solución al problema planteado. Las diferentes fases que se han seguido para desarrollar el trabajo constan de:

1. **Preprocesado de datos:**

   Consiste en una fase de selección, manipulación y transformación de los datos sin procesar en características que sean útiles para los algoritmos de Machine Learning.

2. **Fase de entrenamiento:**

   Durante esta fase se pretende generar un modelo de inteligencia artificial que estudie, aplique y aprenda los patrones de los datos de entrada, para poder hacer una estimación de los resultados en base a dichos datos de entrada estudiados. Por lo tanto, lo que hacemos es estudiar las correlaciones en los datos de entrada que dan lugar a los datos de salida, aplicando métodos de optimización para ajustar los parámetros (tanto el vector de pesos aplicados a cada neurona como las variables a utilizar), en un proceso de aprendizaje que dará como resultado el modelo entrenado con las combinaciones óptimas de cada variable. Para ello tenemos que medir los resultados obtenidos para ser capaces de aplicar las correspondientes mejoras que hagan el modelo lo más acertado posible. En esta fase se tienen que tomar, entre otras, las decisiones relativas al procesamiento de los datos de entrada y sus características, al tipo de modelo de clasificación a utilizar para obtener los resultados, o las funciones de pérdida a aplicar a cada capa de la red. Debido a la importancia de este punto en el desarrollo de este trabajo, en la sección *3.4 Detalles del entrenamiento* se mostrarán estos procesos con más detalle aplicados al caso de este trabajo.

3. **Fase de validación:**

   En esta fase se utiliza un conjunto de datos independientes al de entrenamiento para encontrar y optimizar el mejor modelo para resolver el problema. Gracias a esta fase intermedia entre la de entrenamiento y la de prueba, se puede elegir el modelo que mejor optimiza los resultados calculados, recurriendo a un conjunto de datos que no haya sido utilizado previamente en el entrenamiento, y por tanto evitando así problemas de overfitting o sobreajuste del modelo.

4. **Fase de test:**

   Esta fase determina el correcto funcionamiento del modelo, es decir, permite medir el grado de adecuación de nuestro modelo predictivo sobre el problema final. El conjunto de datos utilizado para esta fase debe ser independiente del utilizado tanto en la fase de entrenamiento como en la de validación. Así, podemos evaluar el rendimiento de la solución desarrollada. Las métricas de evaluación más frecuentes son la precision y la accuracy, mencionadas en el apartado *2.3.3 Métricas de evaluación* del trabajo.

## 3.3 DETALLES DEL PREPROCESADO DE LOS DATOS DE ENTRADA

Tal y como se ha ido explicando a lo largo del presente documento, el objetivo de este trabajo de investigación consiste en desarrollar un IDS con métodos de Machine Learning supervisado, con el fin de detectar conexiones de la Darknet, consideradas como potencialmente peligrosas, utilizando el dataset *CIC-Darknet2020*. Este conjunto de datos consta inicialmente de un archivo CSV con unas 141.530 entradas, que corresponden a diferentes conexiones de tráfico tanto benigno como maligno. Todas ellas están clasificadas en una de las siguientes clases: VPN, TOR, Non-VPN y Non-TOR, característica que finalmente deberemos predecir y medir durante las fases de validación y de test.

Tras investigar con detenimiento cómo conviene establecer la partición del dataset para poder establecer una comparativa de resultados consecuente, se ha escogido una partición tradicional de 80-10-10, haciendo referencia a la partición para entrenamiento, validación y prueba, respectivamente. Para ello, se ha utilizado la librería ***Scikit-learn*** (sklearn) que incluye un conjunto de funciones comúnmente utilizadas para el procesamiento de datos en Python. En concreto, se ha empleado la utilidad ***train_test_split*** para dividir el dataset de forma aleatoria y balanceada en diferentes subconjuntos de datos de entrenamiento, validación y test.

De esta forma, el número de datos de entrada para cada bloque de la partición se ha detallado en la siguiente tabla:

| | **TOTAL** | **ENTRENAMIENTO** | **VALIDACIÓN** | **PRUEBA** |
|---|---|---|---|---|
| **%** | 100 | 80 | 10 | 10 |
| **DATOS** | 141.530 | 113.224 | 14.153 | 14.153 |

*Tabla 3: Resumen de los datos utilizados para entrenamiento validación y prueba. Fuente: elaboración propia.*

Las particiones se han realizado de forma manual a partir del fichero original, utilizando la función ***shuffle()*** para barajar los datos de entrada, evitando de esta manera el orden predefinido en el que se encontraban incialmente.

Al visualizar el documento CSV, podemos observar 85 columnas, que corresponden a las 85 diferentes características de cada dato de entrada. El objetivo del preprocesado de los datos de entrada es dejar los valores de cada columna del documento con unas características que faciliten el tratamiento y procesamiento de los mismos por parte de la red neuronal. Por ello, al fijarnos en dichas columnas, se ha observado que hay datos con valores NaN (Not a Number), otros datos con valores infinitos, y alguna columna con datos en formatos no numéricos que tendremos que estudiar si aportan información útil para la red, y de ser así, ver cómo procesarlos de la mejor forma posible:

1. En primer lugar, cargamos en una variable el archivo CSV con la función ***read_csv()***, especificando la ruta de entrada donde se encuentra el documento original. Además, se han diferenciado la columna de salida (Label) en una variable "y", del resto de columnas del fichero, que guardamos en una variable "x". Esto se ha hecho para poder trabajar sobre las columnas de entrada sin tocar las columnas de salida.

2. En el fichero original la columna Label venía con los datos de entrada: Non-TOR, Non-VPN, TOR y VPN. Con el objetivo de categorizar estos datos de forma que podamos facilitar su procesamiento y análisis en la red neuronal, se han transformado los valores de la columna Label mencionados anteriormente a un vector de clases mediante la función ***utils.to_categorical()*** de la librería Keras, de forma que cada clase se procesa como:

| | **Valor categorizado** |
|---|---|
| **TOR** | [1,0,0,0] |
| **Non-VPN** | [0,1,0,0] |
| **TOR** | [0,0,1,0] |
| **VPN** | [0,0,0,1] |

*Tabla 4: Categorización de la columna Label. Fuente: elaboración propia*

En este sentido, en la Figura 10 puede observarse el número de muestras de cada clase divididas por conjuntos de datos de entrenamiento, validación y test:

```
ENTRENAMIENTO:
74659  muestras de la clase 1  65.96250353406842 %
19171  muestras de la clase 2  16.937906417868252 %
1096  muestras de la clase 3  0.9683347469607011 %
18258  muestras de la clase 4  16.13125530110263 %

VALIDACIÓN:
9283  muestras de la clase 1  65.61351427763643 %
2390  muestras de la clase 2  16.892847045518803 %
152  muestras de la clase 3  1.0743567995476393 %
2323  muestras de la clase 4  16.419281877297145 %

TEST:
9367  muestras de la clase 1  66.20255848469857 %
2300  muestras de la clase 2  16.255565764364974 %
144  muestras de la clase 3  1.0177397695950243 %
2338  muestras de la clase 4  16.524135981341438 %
```

*Figura 10: número de muestras de cada clase por conjunto de datos. Fuente: elaboración propia*

3. El fichero original contiene otra columna Timestamp con la fecha, hora, minutos y segundos, como por ejemplo el valor: 24/07/2015 04:09:78 PM. Debido a que el valor de una fecha completa no aporta información relevante, y tampoco podría procesarse para su análisis en la red neuronal al ser un valor no numérico, se ha extraído el valor de la hora ayudándonos de la función ***split()***. De esta forma, se ha creado una nueva columna en el dataset (y por tanto una nueva característica de cada dato de entrada), y eliminando la anteriormente mencionada Timestamp. El motivo de hacer hecho es que hemos encontrado interesante la hora a la cual se hace la conexión, dato que creemos que puede aportar información de valor a la red neuronal.

4. Valores NaN e infinitos. Para poder procesar correctamente los datos de entrada y evitar posibles errores en los cálculos ejecutados en la red, se han eliminado las filas de los datos de entrada con datos infinitos y NaN. Eso se consigue combinando las funciones ***replace()*** y ***dropna()*** con la librería numpy, lo cual permite dejar el dataset completo sin valores no útiles para la red neuronal.

5. IPs IPv4. Tras una intensa búsqueda de una posible solución para utilizar la información de las IPs en la red neuronal, la única solución viable que se ha

encontrado ha sido la de implementar y utilizar una función ***create_grams()*** para crear una serie de subsets que dividen las IPs en tres valores, que representan la dirección de red de cada una. Existían otras soluciones más técnicas que lograban identificar información que a priori pensaríamos que es muy útil, como el país origen de la conexión, pero esto conllevaba una suscripción a un servicio de pago que, en este caso, no se va a llevar a cabo. La idea de crear los tres grams utilizando la función mencionada anteriormente se ha recogido de una solución para preprocesar datos de entrada de la Darknet, extraída de un post dentro de la página web *KAGGLE* [47].

6. Eliminación de columnas. También se han identificado varias columnas con ningún valor útil para el procesamiento de la red neuronal, por lo que hemos procedido a eliminarlas. Tras dicha eliminación, el número de columnas finales se ha dejado en el conjunto de datos es de 82. Las columnas que se han eliminado son:

    - Flow ID: se trata simplemente de un ID de la conexión que no aporta I nformación importante a la red. Para evitar añadir información que no puede procesar o no tiene relación con el objetivo del sistema, hse ha eliminado esta columna.
    - Timestamp: su tratamiento ya ha sido explicado en este capítulo. Al haber añadido una nueva columna con la hora específica de cada conexión extraída de esta columna, se prescinde de la columna inicial.
    - Src IP: se han creado nuevas columnas con la función create_grams() explicada anteriormente. De la misma forma, se elimina esta columna del dataset.
    - Dst IP: igual que la anterior.
    - Label.1: esta columna identificaba el tipo de conexión que se había conectado. La idea inicial del dataset es que esta columna también sea objeto de estudio y sus valores se puedan predecir; pero tal y como se ha explicado anteriormente, esto no es omitido este objetivo del presente trabajo. Por tanto, también queda eliminada.

En este punto hemos terminado el preprocesamiento de los datos de entrada, y por lo tanto podemos pasar a la siguiente fase, que consistirá en estudiar en detalle y encontrar el mejor modelo que prediga las conexiones que pertenezcan a cada clase de la columna Label del dataset.

## 3.4 DETALLES DEL ENTRENAMIENTO

En este apartado se va a definir cada parte de la topología de la red neuronal construida, con la intención de explicar las razones de cada decisión que se ha tomado en su implementación y configuración para la búsqueda de la mejor solución.

### 3.4.1 TOPOLOGIA:

Respecto a la topología utilizada, en un primer lugar se han definido 5 layers o capas de 128 neuronas. Por tanto, se dispondrán de 3 capas ocultas (hidden layers), además de la capa de entrada (input layer) y la capa de salida (output layer). En la siguiente imagen (Figura 11) puede verse una figura simplificada de lo que podría ser la topología utilizada, sin tener en cuenta el número de neuronas por cada capa:

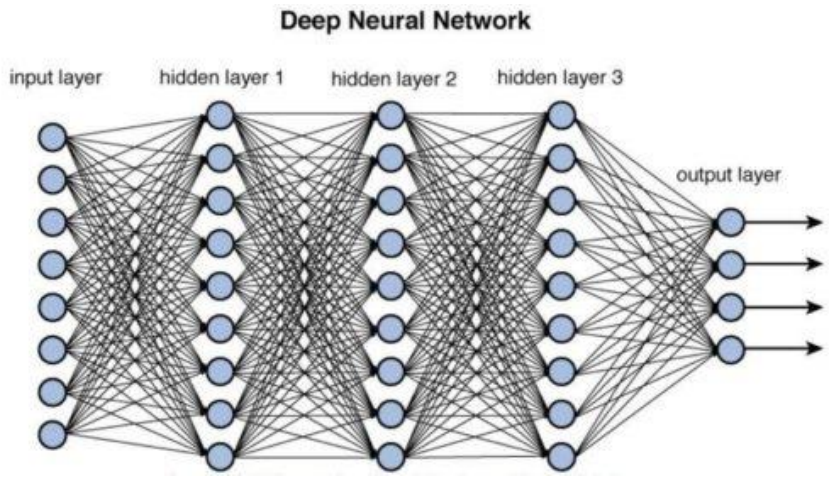


*Figura 11: Topología de la red utilizada [48]*

Además, se han definido los siguientes conceptos:

- **Epochs**: en términos coloquiales, una epoch consiste en cada vez que el conjunto de datos entero es procesado por la red neuronal. Si se establecen pocas epochs, la red neuronal tendrá un desempeño muy pequeño, pues no tiene suficiente información para balancear los pesos de las variables de cada neurona, y diríamos que el modelo está en una situación de underfitting. Por otro lado, si establecemos demasiadas epochs, el modelo tenderá a estar sobreajustado al aprender de más sobre los datos de entrada que pasamos. Se obtendrían buenos resultados para los datos de entrenamiento, pero en la fase de validación y test veríamos peores desempeños

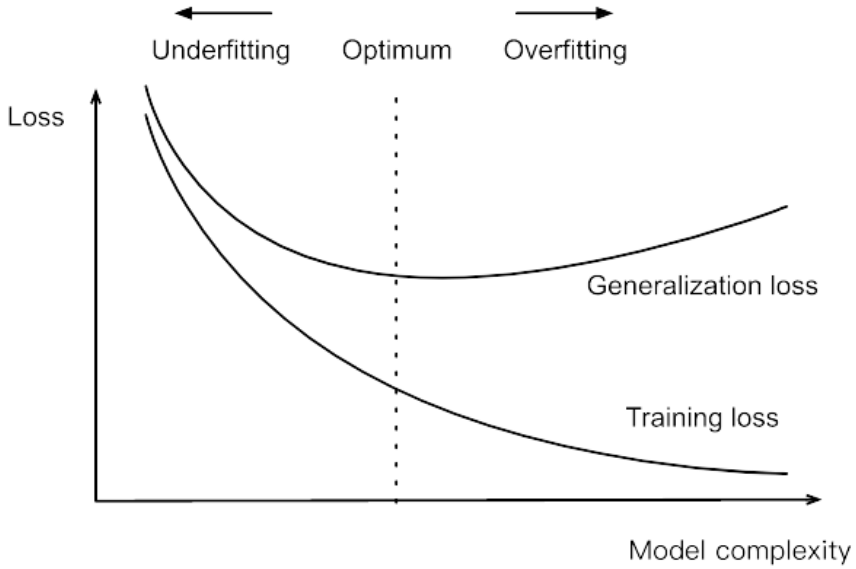


*Figura 12: Overfitting vs Underfitting [49]*

Los modelos que hemos desarrollado constan de 100 y 150 epochs, dependiendo de la curva del loss con respecto a cada epoch. Por ejemplo, en la siguiente imagen (Figura 13) Podemos ver como después de la epoch número 100 el valor de pérdida sigue descendiendo, pero a partir de la epoch número 125 se mantiene prácticamente estable:

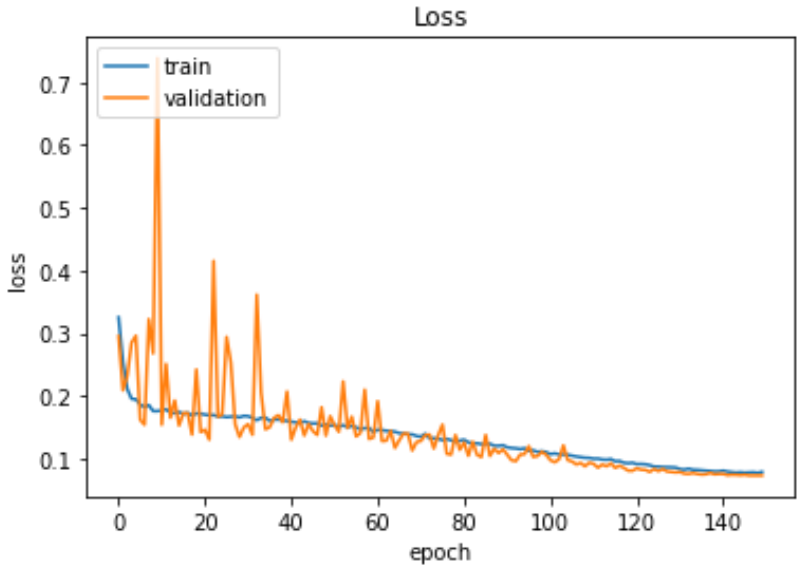


*Figura 13: Función de pérdida por cada epoch. Fuente: elaboración propia*

- **Batch size**: consiste en el número de divisiones de los datos de entrada que hacemos en cada epoch con la intención de no sobrecargar la máquina con todos los datos a la vez. La utilidad añadida de esto viene de que se actualizan un mayor número de veces los parámetros de los pesos de las neuronas, además de facilitar y agilizar los accesos a memoria de la red. Este valor también se puede modificar con la intención de buscar mejoras en los modelos.

### 3.4.2 OPTIMIZADORES Y FUNCIONES DE PÉRDIDA:

Un optimizador es un algoritmo que ajusta los parámetros o pesos de las aristas que relacionan las neuronas durante la fase de backpropagation, minimizando la función de coste o función de pérdida, con el objetivo de obtener los mejores resultados para el modelo de la manera más eficiente y eficaz posible [50]. Es el elemento que permite el descenso de gradiente explicado anteriormente en el trabajo. La función de pérdida es aquella que mide la diferencia entre el valor predicho y el valor esperado, es decir, el error de la predicción. En la siguiente imagen (Figura 14) se puede observar un esquema simplificado del funcionamiento de una neurona respecto al cálculo de la actualización de los pesos en una iteración:

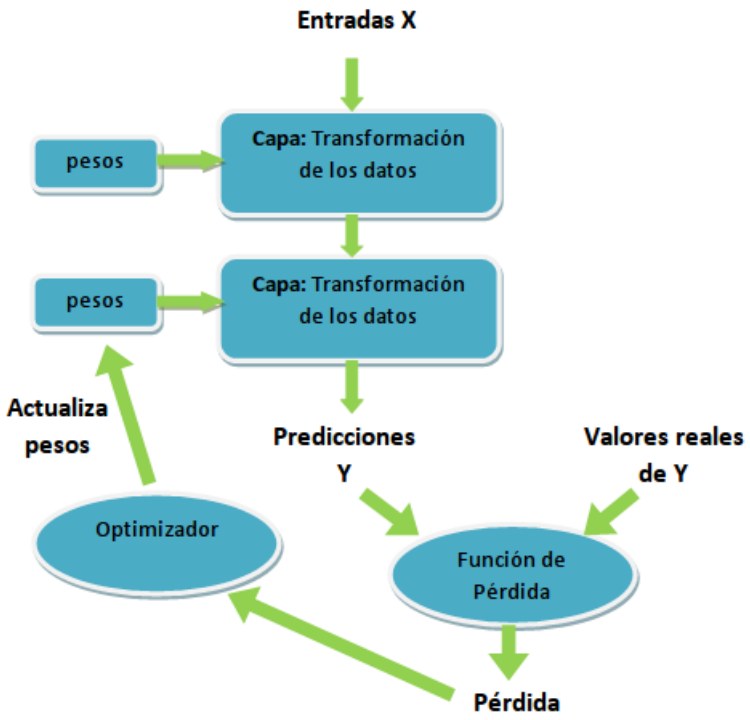


*Figura 14: Esquema del funcionamiento de la actualización de pesos [51]*

Dependiendo del tipo de problema, se pueden utilizar diferentes funciones de pérdida [52]:

- **Error Cuadrático Medio** ('Mean Square Error', **MSE**): consiste en calcular la distancia geométrica del valor predicho al valor real conocido. esta función es muy utilizada en problemas de regresión.

- **Categorical cross entropy**: consiste en medir la distancia entre distribuciones de probabilidad. Es la función de pérdida mayormente utilizada en los problemas de clasificación multiclase, como es el caso del nuestro. Por lo tanto, esta función de Perdida es la que finalmente se ha utilizado en los modelos.

Volviendo a la definición de un optimizador, en el trabajo se ha evaluado el rendimiento de los diferentes modelos con los siguientes algoritmos de optimización [50]:

1. **SGD** (Stochastic Gradient Descent): debido a la inviabilidad que implicaría realizar todos los cálculos (que incluyen derivadas de las funciones de coste respecto a cada peso de la red neuronal), este algoritmo introduce un comportamiento aleatorio al actualizar los parámetros sólo para una selección estocástica de los datos de entrada en cada iteración, utilizando la misma tasa de aprendizaje (comúnmente denominada Learning Rate). Esta operación reduce considerablemente el coste computacional de la red, obteniendo muy buenos resultados [53].

2. **ADAGRAD** (Adaptive Gradient Algorithm): este algoritmo introduce una variación muy importante que consiste en utilizar diferentes LR para cada parámetro. Además, para evitar la sobrecarga de la máquina, el Learning Rate no se calcula en cada iteración, sino que partiendo del inicial, se escala y adapta para cada parámetro respecto al gradiente acumulado del total de las iteraciones [54].

3. **ADADELTA** (ADAGRAD extendido): este optimizador sigue la misma lógica que el anterior, con la diferencia de que la Learning Rate se recalcula teniendo en cuenta el gradiente acumulado en las últimas n iteraciones, es decir, los últimos n gradients [53].

4. **ADAM** (Adaptive Moment Estimation): es el algoritmo mayormente utilizado en Deep Learning debido a sus excelentes resultados. Combina las innovaciones introducidas en los dos algoritmos anteriores con la mayor influencia de los pesos del anterior gradiente [55].

### 3.4.3 LEARNING RATE SHCEDULERS

Seguidamente, se han introducido los **Learning Rate Schedulers**, que consisten en métodos adicionales para hacer que el proceso de encontrar la tasa de aprendizaje adecuada sea mucho más fluido, rápido y preciso. La funcionalidad de estos elementos consiste en ajustar el Learning Rate durante la fase de entrenamiento reduciéndolo de acuerdo con un schedule definido previamente [56]. Los más simples son el denominado Learning Rate Scheduler lineal y el decaimiento escalonado, que, a diferencia del primero, pretende intensificar la búsqueda local de la tasa de aprendizaje. En las siguientes imágenes (Figuras 15 y 16) se puede apreciar la diferencia conceptual de la curva de la tasa de aprendizaje que implementa cada uno:

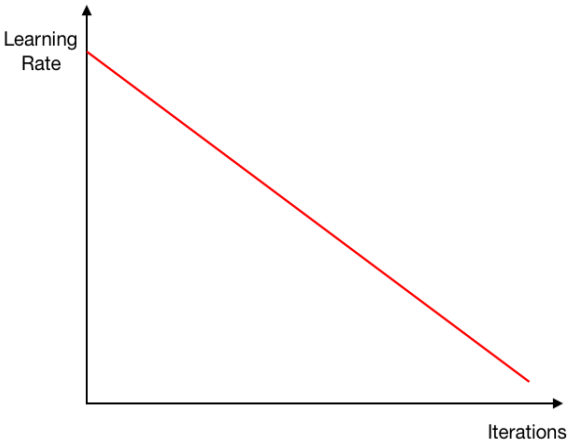


*Figura 15: Tasa de aprendizaje de un Learning Rate Scheduler lineal [56]*

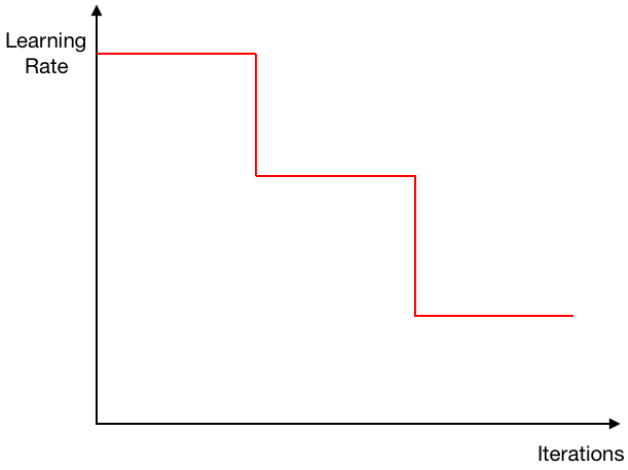


*Figura 16: Tasa de aprendizaje de un Learning rate con decaimiento escalonado*

Además, existen otros tipos de Learning Rate Schedulers que en vez de realizar un descenso del learning rate continuo (fase de enfriamiento), se reinician en cierto punto (fase de recalentamiento), de forma que se consigue optimizer la tasa de aprendizaje. En la siguiente imagen (Figura 17) se puede observar el funcionamiento del Learning Rate Scheduler with Warm Restarts:

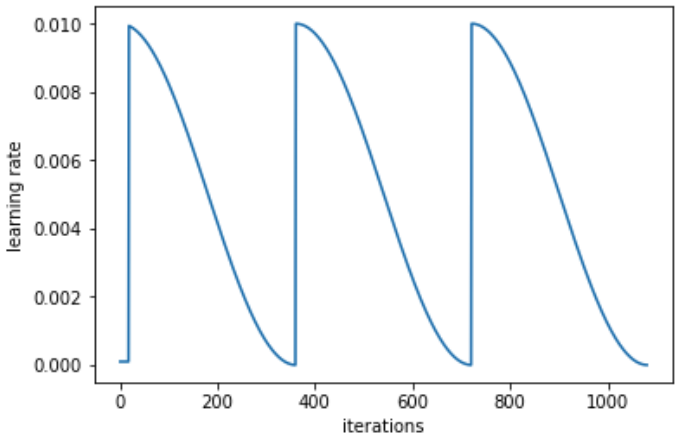


*Figura 17: Tasa de aprendizaje de un Learning Rate Scheduler with warm Restarts [56]*

En el presente trabajo hemos probado a añadir esta funcionalidad mediante un Learning Rate Scheduler lineal y un Learning Rate Scheduler with warm Restarts. Cabe destacar que en la ejecución del último tipo, no se han implementado los reinicios, debido a que nuestras ejecuciones consisten en demasiadas pocas epochs como para implementarlos [57]. Además, el código utilizado para la implementación de esta configuración se ha extraído del enlace [58].

### 3.4.3 FUNCIONES DE ACTIVACIÓN:

Otro de los elementos más importantes a la hora de parametrizar el modelo consiste en las funciones de activación. Básicamente, una función de activación es aquella que convierte en salida los datos de entrada, lo que ayuda a la red aprender relaciones complejas y patrones específicos. Dados unos datos de entrada y unos pesos en una neurona de la red, el resultado de las operaciones dependerá directamente de la función de activación que se utilice. Cabe mencionar que estas funciones no deben ser lineales, ya que, de ser así, el modelo se reduciría a una única función lineal sin capas ocultas. Esto es debido a que las difentes funciones lineales podrían combinarse para construir otra función lineal global. Las funciones de activación más importantes se describen a continuación [59]:

- **Sigmoide**: esta función consiste en una operación matemática con exponenciales que puede definirse como:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

*Figura 18: Función sigmoide [59]*

Hoy en día esta función se ha dejado de utilizar, pero ha servido como base para las funciones de activación modernas.

- **Softmax**: esta función consiste en una generalización de la función sigmoide en un entorno multiclase, en la que se coge un vector de n números reales, que se normalizan en una distribución de probabilidad de n posibilidades correspondientes a las exponenciales del número de entrada. Es muy utilizada en la última capa en problemas de clasificación en varias clases. Es por ello por lo que se ha decidido utilizar esta función en la última capa de nuestro modelo.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

*Figura 19: Función softmax [59]*

- **ReLU (**Rectified Linear Unit): es aquella función que devuelve el valor directamente si es positivo, y sino devuelve un “0”. Es la función de activación mayormente utilizada en redes neuronales debido a su facilidad de uso y cómputo, su rapidez, y sus excelentes resultados [60]. A partir de la función ReLU se han desarrollado otras diferentes mediante alguna variación en la función. Destacan entre ellas: Leaky ReLU, ELU, PreLU o Softplus.

$$f(x) = x^+ = \max(0, x),$$

*Figura 20: Función ReLU [59]*

- **Softplus**: esta función consiste en la antiderivada o integral de la función sigmoide. Puede verse como una versión similiar a ReLU. De hecho sus resultados suelen ser bastante similares, excepto para los valores cercanos a cero, donde softplus es diferenciable al dar valor a la salida, caso contrario al de ReLU. La principal desventaja de esta función consiste en el alto coste computacional que supone realizar las operaciones logarítmicas y exponenciales, a diferencia de ReLU:

$$f(x) = \ln(1 + e^x),$$

*Figura 21: Función softplus [59]*

En la siguiente imagen (Figura 21) se pueden observar las gráficas simplificadas de las funciones mencionadas anteriormente:

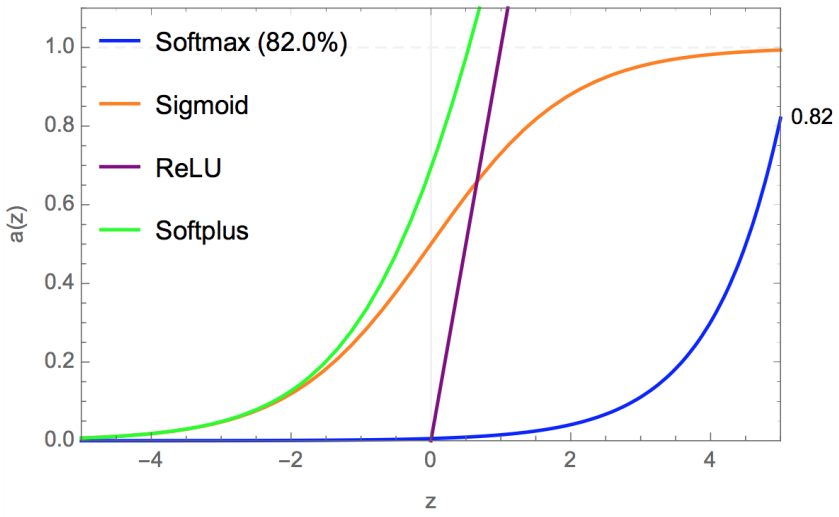


*Figura 22: Gráficas de algunas de las funciones de activación de redes neuronales [61]*

En el modelo del trabajo se ha probado a utilizar la función ReLU y Softplus en todas las capas salvo la última, que debido a sus características explicadas anteriormente, se ha utilizado siempre para dicha capa.

### 3.4.4 OTROS DETALLES DEL ENTRENAMIENTO:

1. **Regularización**: esta técnica trata de aplicar una penalización al criterio de optimización con la intención de evitar que los pesos de la red se engrandezcan hasta alcanzar valores extremos, esquivando de esta forma el overfitting del modelo. Así, las probabilidades más altas se ponderan para los valores de los pesos moderados, ayudando a reducir el sobreajuste del modelo. La regularización más común y la que se ha aplicado al modelo ha sido la L1 y L2 mediante el siguiente parámetro:

kernel_regularizer= regularizers.l1_l2(l1=1e-5, l2=1e-4)

2. **Batch normalization**: consiste en normalizar por cada batch de ejecución el output de la función de activación utilizando la media y la desviación de los resultados, con el objetivo de evitar diferencias pronunciadas entre los valores de cada operación. La red trabaja mucho mejor de esta forma al tener los datos normalizados en cada capa. Cabe destacar que se puede aplicar tanto justo antes de la ejecución de la función de activación como justo después, lo que puede arrojar resultados diferentes [62].

   Se ha probado a introducir esta funcionalidad en cada neurona del modelo mediante la expresión:

   model.add(BatchNormalization())

3. **Gaussian noise**: consiste en añadir ruido a los datos de entrada durante la fase de entrenamiento, con el objetivo de reducir el sobreajuste del modelo. Hay estudios que confirman el buen rendimiento de este parámetro, al considerarse una forma de aplicar Data Augmentation (aumento de datos) que ayuda a la generalización del modelo y a la tolerancia de los posibles fallos [63]. Al igual que la Batch Normalization, puede aplicarse tanto antes como después de la ejecución de la función de activación.

   Se ha introducido en cada neurona del modelo mediante la expresión:

   model.add(GaussianNoise(0.3))

4. **Dropout:** se basa en descartar neuronas de manera aleatoria, de forma que se fuerza que las que no han sido eliminadas representen la misma importancia que tendrían todas las neuronas en su conjunto [64]. Esta técnica es una de las soluciones posibles para lograr mejorar la generalización del modelo y reducir el overfitting o sobreajuste del mismo.

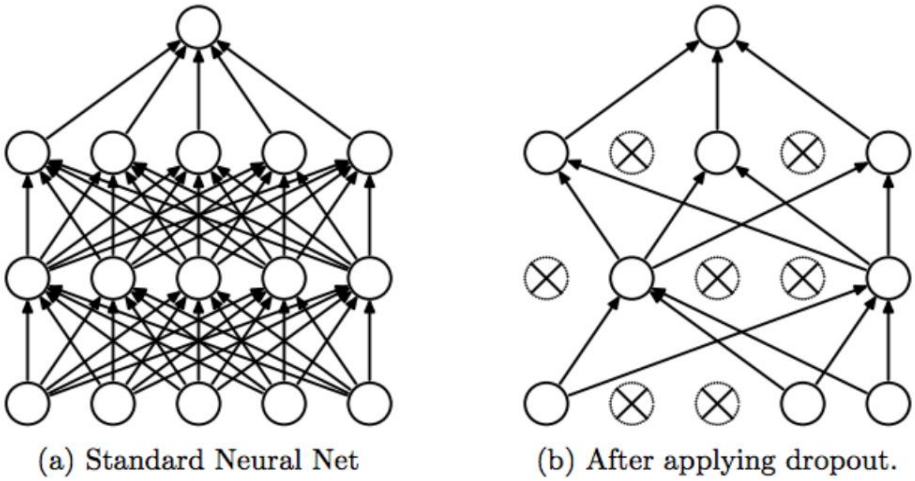


*Figura 23: Representación del funcionamiento del dropout [64]*

Se ha probado a añadir a nuestro modelo con la función:

model.add(Dropout(0.5))

Esto significaría que las neuronas desactivadas serían más o menos un 50% del total. En la experimentación de los modelos, se ha probado a parametrizar con un 30 y con un 50%.

# 4. RESULTADOS EXPERIMENTALES Y VALIDACIÓN

Tras definir la topología de la red y las posibles configuraciones, se procede a explicar todas las pruebas y mejoras llevadas a cabo durante la experimentación de la red neuronal. Cabe destacar que cada modelo ha sido ejecutado y validado 5 veces, sacando la media aritmética de los resultados obtenidos (accuracy) en las diferentes ejecuciones, para poder llevar a cabo una comparación lo más realista posible entre los diferentes modelos. Los cambios realizados en los modelos se corresponden tanto a pequeños cambios en la topología de la red como a técnicas de mejora de la generalización y optimización de estos.

En la siguiente tabla (Tabla 5) se puede observar un resumen de los diferentes modelos llevados a cabo. Posteriormente, iremos desgranando cada paso que ha permitido dar con el mejor modelo:

| ID | BATCH | EPOCHS | REG | OPT | LRS | FAct | BN | GN | DROP | Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 128 | 100 | 0 | ADAM | 0 | relu | 0 | 0 | 0 | 0.662025 |
| 2 | 128 | 100 | L1,L2 | ADAM | 0 | relu | 0 | 0 | 0 | 0.887483 |
| 3 | 128 | 100 | L1,L2 | SGD | 0 | relu | 0 | 0 | 0 | 0.693900 |
| 4 | 128 | 100 | L1,L2 | ADADELTA | 0 | relu | 0 | 0 | 0 | 0.863453 |
| 5 | 128 | 100 | L1,L2 | ADAGRAD | 0 | relu | 0 | 0 | 0 | 0.946427 |
| 6 | 128 | 150 | L1,L2 | ADAGRAD | 0 | relu | 0 | 0 | 0 | 0.952646 |
| 7 | 128 | 150 | L1,L2 | ADAGRAD | 0 | relu | 1 | 0 | 0 | 0.979857 |
| 8 | 128 | 150 | L1,L2 | ADAGRAD | 0 | relu | 0 | 1 | 0 | 0.957664 |
| 9 | 128 | 150 | L1,L2 | ADAGRAD | 0 | relu | 1 | 1 | 0 | 0.971164 |
| 10 | 128 | 150 | L1,L2 | ADAGRAD | 0 | softplus | 1 | 0 | 0 | 0.974627 |
| 11 | 128 | 150 | L1,L2 | ADAGRAD | LIN | relu | 1 | 0 | 0 | 0.985723 |
| 12 | 128 | 150 | L1,L2 | ADAGRAD | COSINE | relu | 1 | 0 | 0 | 0.987560 |
| 13 | 256 | 150 | L1,L2 | ADAGRAD | COSINE | relu | 1 | 0 | 0 | 0.985157 |
| 14 | 128 | 150 | L1,L2 | ADAGRAD | COSINE | relu | 1 | 0 | 30% | 0.979857 |
| 15 | 128 | 150 | L1,L2 | ADAGRAD | COSINE | relu | 1 | 0 | 50% | 0.977100 |
| 16 | 128 | 150 | L1,L2 | ADAM | COSINE | relu | 1 | 0 | 0 | 0.986430 |

| | |
|---|---|
| BATCH: | Batch size |
| REG: | Regularización |
| OPT: | Optimizador |
| LRS: | Learning Rate Scheduler |
| EPOCHS: | Epochs |

| | |
|---|---|
| Fact: | Función de activación |
| BN: | Batch normalization |
| GN: | Gaussian noise |
| DROP: | Dropout |

*Tabla 5: Resumen de los diferentes modelos ejecutados con el accuracy correpsondiente. Fuente: elaboración propia.*

[ID=1] En una primera instancia, se ha definido un modelo lo más sencillo posible con la topología descrita: un batch size de 128 y 100 epochs, utilizando Adam y Relu, ya que a priori parecen el mejor optimizador y la mejor función de activación, respectivamente. Sin embargo, los resultados han sido bastante desafortunados, ya que no se ha llegado al 70% de accuracy.

[ID=2] Posteriormente, se ha introducido la regularización L1,L2 en cada capa del modelo con la intención de penalizar los errores de la predicción, lo que ha resultado en un aumento considerable del accuracy, llegando a un 88.74% de las muestras de test clasificadas correctamente.

[ID=3] En este momento se ha decidido probar con diferentes optimizadores. Al probar la misma configuración con el optimizador SGD, todas las funciones de pérdida de cada epoch resultaban no tener un valor numérico, así como el accuracy final, cuyo resultado era de NAN. Tras estudiar el caso y encontrar diferentes soluciones al problema [65], se ha probado a utilizar el optimizador con el parámetro clipnorm=1, lo que impide la denominada explosion de gradientes (el gradiente numérico se desborda y crece sin control [66]). De esta forma, se ha dejado de obtener NAN en la función de pérdida, aunque el accuracy obtenido para este modelo ha sido de sólo un 69.39%.

[ID=4] Al probar un modelo con la misma configuración pero modificando el optimizador al Adadelta, observamos que el accuracy aumenta considerablemente hasta un 86.34%. Además, por otro lado, observamos que al implementar el mismo modelo con el optimizador Adagrad [ID=5], obtenemos un accuracy del 94.64%, resultado que denota una mejora muy notable respecto a los anteriores.

Una vez encontrado el mejor optimizador para la red neuronal, gracias a la imagen 24 se ha observado que en las últimas epochs se ha seguido optimizando el accuracy obtenido. Por lo tanto, se ha pensado que las 100 epochs podrían ser escasas para que el modelo converja lo máximo posible.

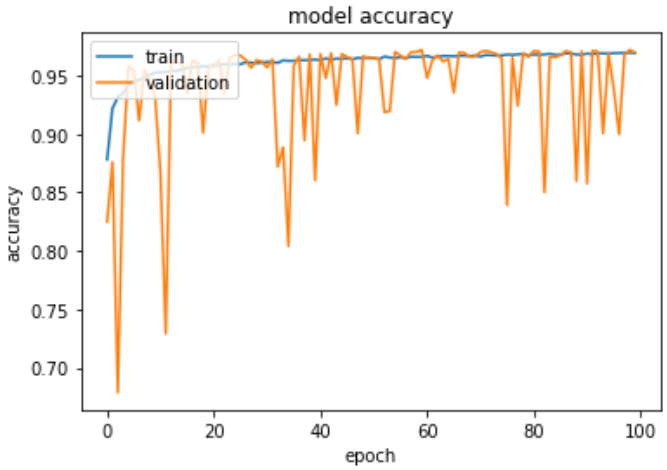


*Figura 24: Accuracy por cada epoch del modelo. Fuente: elaboración propia*

[ID=6] Por la anterior razón, se ha decidido incrementar las epochs hasta 150 con la última configuración, que se corresponde con la que mejores resultados se han obtenido hasta el momento. Tal y como Podemos ver en la tabla 5, se consigue mejorar el accuracy hasta un 95.27%.

[ID=7] Siguiendo con la misma lógica, se ha ejecutado la siguiente prueba, que consiste en aplicar Batch Normalization, técnica de optimización del modelo explicada en el apartado *3.4.4 OTROS DETALLES DEL ENTRENAMIENTO* del trabajo. Los resultados de esta prueba son bastante satisfactorios, llegando al 97.98% de accuracy.

[ID=8] También se ha intentado añadir al modelo la técnica de generalización Gaussian Noise, previamente a añadir Batch Normalization, obteniendo un accuracy de 95.26%, que mejora el modelo previo a la Batch Normalization, pero queda por debajo de los resultados obtenidos con esta técnica.

[ID=9] En la prueba posterior se han aplicado las dos últimas técnicas en conjunto (Batch Normalization y Gaussian Noise), pero el accuracy obtenido (97.11%) es ligeramente inferior al obtenido al aplicar solo la primera de ellas. Por tanto, en las siguientes pruebas se ha dejado de utilizar la técnica Gaussian Noise.

[ID=10] Otra prueba que quedaba pendiente de ser aplicada es la de probar a cambiar la función de activación, ya que en todas las pruebas anteriores se ha utilizado la función Relu. En esta prueba se ha modificado el modelo obtenido hasta ahora (ID=7), utilizando la función de activación Softplus en lugar de Relu. Sin embargo, los resultados han vuelto a ser ligeramente inferiores, por lo que en las siguientes pruebas continuaremos utilizando Relu. En este punto cabe destacar que, debido a las características descritas

en el apartado *3.4.3 FUNCIONES DE ACTIVACIÓN*, en la última capa del modelo siempre se ha utilizado Softmax,

Llegados a este punto, se ha analizado el loss obtenido en las epochs de las primeras iteraciones. Tal y como se puede observar en la Figura 24, el loss de las primeras epochs es tan elevado que no se puede apreciar la diferencia entre el loss de las siguientes ejecuciones.

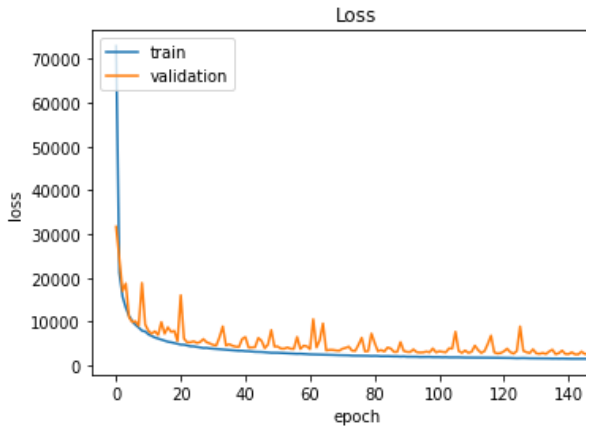


*Figura 25: Loss sin Learning Rate Scheduler. Fuente: elaboración propia.*

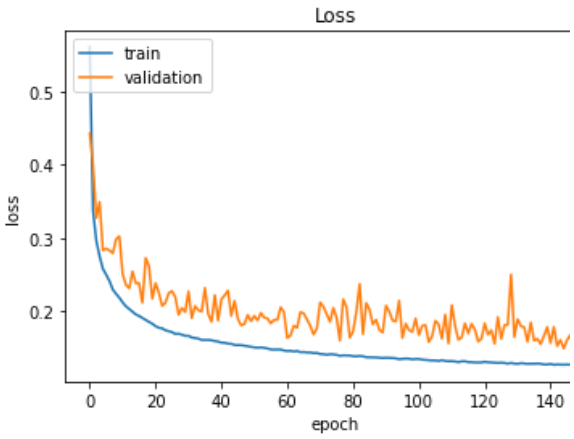


*Figura 26: Loss con Learning Rate Scheduler. Fuente: elaboración propia.*

[ID=11] Por ello, se ha decidido añadir un Learning Rate Scheduler al modelo, de forma que los parámetros de cada neurona se actualicen de forma mucho más rápida y eficiente. En las figuras 24 y 25 se puede apreciar fácilmente la diferencia del loss obtenido en las primeras iteraciones entre el modelo que utiliza un Learning Rate Scheduler y el que no.

[ID=12] Para continuar con las mejoras del modelo, se ha aplicado un Learning Rate Scheduler with Warm Restarts, explicado en el capítulo anterior del trabajo. El modelo obtenido con esta configuración ha logrado los mejores resultados hasta el momento para el accuracy: 98.76%. En la siguiente imagen (Figura 26) se puede observar la tasa de aprendizaje durante las iteraciones del modelo:

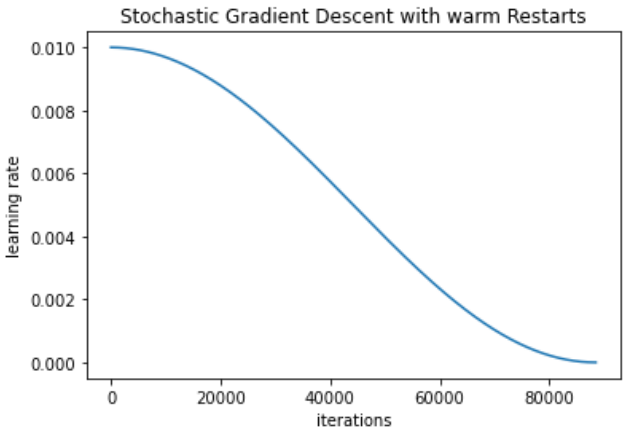


*Figura 27: Tasa de aprendizaje del modelo aplicando el Learning Rate Scheduler. Fuente: elaboración propia.*

[ID=13] Se han hecho pruebas adicionales para comprobar el impacto de aumentar el Batch size a 256 a partir del último modelo, obteniendo un accuracy del 98.52%, ligeramente inferior al modelo con ID=12.

También se ha probado a añadir al mejor modelo obtenido hasta ahora la técnica Dropout con el parámetro al 30% [ID=14] y al 50% [ID=15]. El primero de ellos ha resultado tener un accuracy de algo menos del 98%, mientras que el segundo ha obtenido un 97.71%. Por lo tanto, como no se han obtenido resultados mejores los que ya se habían alcanzado, se ha optado por no utilizar la técnica del Dropout.

[ID=16] Por último, a modo de prueba final y debido a que el optimizador Adam es el más famoso, eficiente computacionalmente, y en definitiva el preferido para la mayoría de problemas de Deep Learning [67], se ha cambiado el optimizador del mejor modelo obtenido hasta este momento (Adagrad) para ver si con toda la configuración hecha hasta ahora, obtenemos alguna mejora en el rendimiento. Sin embargo, el accuracy obtenido de esta forma es del 98.64%, resultado que, aunque se queda muy cerca, no mejora el mejor obtenido.

Por lo tanto, después de todas las pruebas realizadas, nos quedamos finalmente con el modelo con ID=12. En la siguiente gráfico (Figura 27) puede observarse la trayectoria del accuracy obtenido por cada epoch de este modelo:

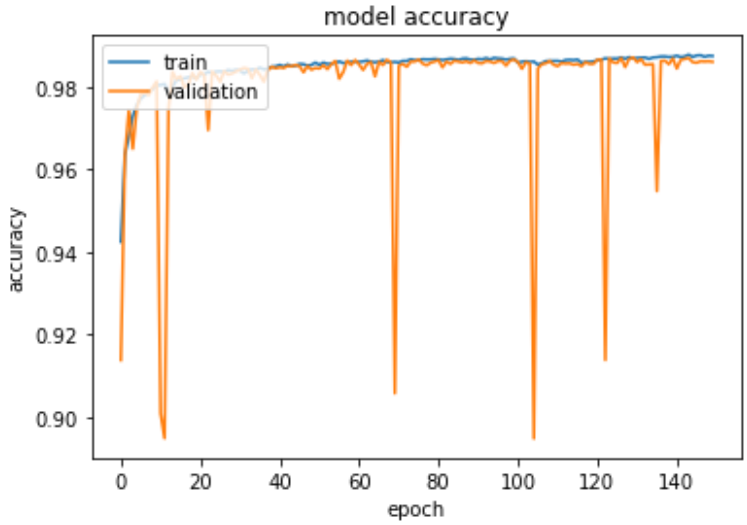


*Figura 28: Trayectoria del accuracy obtenido por el mejor modelo en cada epoch. Fuente: elaboración propia.*

Resumidamente, este modelo tiene un Batch size de 128, ejecutándose en 150 epochs, con regularización L1,L2, utilizando el optimizador Adagrad con un Learning Rate Scheduler COSINE with warm restarts, así como una función de activación Relu y batch normalization de los parámetros.

```
Predicted Class: [1 0 0 ... 1 0 0]
Confusion Matrix:
 [[9359    3    0    5]
 [   6 2224    0   70]
 [   0   26   87   31]
 [   9   16   10 2303]]
Accuracy: 0.9875609583716164
F_score: [0.999 0.974 0.722 0.97 ]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      9367
           1       0.98      0.97      0.97      2300
           2       0.90      0.60      0.72       144
           3       0.96      0.99      0.97      2338

    accuracy                           0.99     14149
   macro avg       0.96      0.89      0.92     14149
weighted avg       0.99      0.99      0.99     14149
```

*Figura 29: Resumen de los resultados del mejor modelo obtenido. Fuente: elaboración propia.*

Este modelo ha conseguido un accuracy muy elevado con un 98.76% de acierto sobre las diferentes clases. Esto significa que el 98.76% de las veces el modelo ha clasificado la conexión en la clase que le corresponde. Respecto a la precision, recall y f1 score, teniendo en cuenta la matriz de confusión que aparece en la Figura 28 se pueden observar las diferencias entre las cuatro clases:

- **Clase 0 (NON-TOR):** para esta clase ha habido 9359 conexiones (el 99.91%) que han sido clasificadas correctamente (True Positives). Sólo ha habido 8 conexiones de las 9367 totales del conjunto de datos de test clasificadas como clases diferentes a esta (False Negatives), así como 15 conexiones clasificadas como si fueran de esta clase sin realmente serlo (False Positives). Por lo tanto, tanto la medida precision, como la recall y el f1-score han resultado ser muy elevados, resultando prácticamente 1.

- **Clase 1 (NON-VPN):** para esta clase ha habido 2224 muestras de las 2300 totales del conjunto de test que han sido True Positives, lo que representa un 96.7% de acierto. De la misma forma, ha habido 76 muestras que han resultado ser False Negatives y 42 muestras de otras clases que han resultado ser False Positives. Todo esto ha resultado en una precision del 98%, una recall del 97% y un resultado f1-score del 97%.

- **Clase 2 (TOR):** esta clase ha sido la que sin duda peor se ha clasificado del modelo. La precision ha salido del 90% debido a que el número de True Positives ha sido de 87 y el número de False Positives de 10. Sin embargo, la recall ha sido del 60% debido a que 57 de las 144 muestras de esta clase del conjunto de datos de test han resultado ser False Negatives. Finalmente, el resultado f1-score para esta clase ha sido de un 72%.

- **Clase 3 (VPN):** esta clase ha obtenido muy buenos resultados, con 2303 muestras clasificadas correctamente (True Positives). Ha habido 106 muestras de otras clases clasificadas como False Positives, que, con un peso del 4.4% de las muestras, han hecho que la precision sea del 96%. Además, ha habido 35 muestras que se han clasificado como False Negatives. Todo esto ha hecho que esta clase obtenga una recall del 99% y un f1-score final del 97%.

# 5. CONCLUSIONES Y TRABAJO FUTURO

El principal objetivo de este trabajo consistía en comprender los conceptos básicos del Deep Learning para aplicarlos en un Sistema de Detección de Intrusos basados en Red, para lo cual se ha implementado una red neuronal mediante Tensorflow y Keras. De esta forma, se ha utilizado un dataset con tráfico de red relacionado con la Darknet mediante conexiones TOR y VPN. Finalmente, se ha conseguido obtener unos resultados muy satisfactorios, con un modelo final que ha sido evaluado con un 98.76% de accuracy.

Respecto a la evaluación de los resultados obtenidos con este modelo, las clases NON-TOR, NON-VPN, y VPN han sido clasificadas con una precision, recall y f1-score muy buenos. Sin embargo, la clase TOR (clase 2) ha tenido un alto porcentaje de False Negatives, lo que ha hecho que los resultados hayan sido algo inferiores a lo quesería deseable. De todas formas, el rendimiento del modelo final ha sido muy bueno.

Este trabajo ha sido un gran reto debido a los escasos conocimientos previos que se tenían tanto sobre Deep Learning como de Sistemas de Detección de Intrusos. Sin embargo, tras desempeñar un gran esfuerzo de aprendizaje y de tiempo, se ha conseguido tanto asimilar los conceptos básicos de ambos temas como unos resultados que se acercan a los del estado del arte. Particularmente, el aprendizaje de los conceptos e implementación en Keras de Deep Learning no habría sido posible sin la visualización de los video-tutoriales Deep Learning Fundamentals – Intro to Neural Networks [34] y Tensorflow – Python Deep Learning Neural Network API [68], del canal Deep Lizard [35] disponibles de forma gratuita y sin límite en la plataforma Youtube.

Por otra parte, la utilización del dataset CICDarknet2020 ha permitido introducirse en el mundo de la Ciberseguridad y la Deep Web, así como comprender las diferencias de este último concepto con Internet y con la Darknet. Por lo tanto, también se han logrado los objetivos de introducción y asimilación de estos conceptos.

También cabe destacar la utilización de una máquina remota del DSIC de la UPV para llevar a cabo los entrenamientos de cada modelo, lo que ha permitido el correcto desarrollo de este trabajo. Los entrenamientos con un dataset tan amplio en una máquina local, como un ordenador personal, consumirían una cantidad muy elevada de recursos, tanto en términos de tiempo como de computación. Sin esta facilidad proporcionada por el departamento, la parte práctica de este trabajo no podría haberse llevado a cabo.

Para elaboración de este proyecto ha sido indispensable la utilización de los conceptos adquiridos durante mi etapa en la Universidad cursando el Grado en Ingeniería Informática. Concretamente, gracias a la asignatura de Sistemas Inteligentes (SIN) aprendimos las nociones básicas del aprendizaje automático y la Inteligencia Artificial; por ejemplo, en las sesiones prácticas comprendimos y programamos un perceptrón, concepto básico de las redes neuronales explicado también a lo largo de este trabajo. Otra asignatura que ha servido de gran ayuda para este trabajo ha sido Sistemas de Información Estratégicos (SIE), en la cual aprendimos a cómo preprocesar datos de entrada y analizar qué datos son los apropiados para la minería de datos, aunque la parte práctica de esta materia la implementamos con una herramienta interactiva llamada Rapid Miner. Por último, destacar que aunque no se había aprendido a utilizar Python durante la carrera, los conocimientos adiquiridos en los distintos lenguajes de programación aprendidos en las diferentes asignaturas (Java, C, C++, Javascript, SQL, Haskell, Prolog…) han hecho que su aprendizaje no haya supuesto una dificultad añadida al trabajo.

Respecto al trabajo futuro, en cuanto a los modelos establecidos, se puede proponer alguna mejora para intentar disminuir la tasa de False Negatives de la clase TOR, como la utilización de optimizadores más avanzados derivados de Adam, como puede ser el NAdam, probar diferentes funciones de activación, como Leaky Relu, o incluso implementar diferentes modelos cambiando el orden de ejecución de las funciones de activación. Sin embargo, dado que los resultados obtenidos en el modelo final han sido bastante buenos respecto a los del estado del arte, los resultados finales tampoco podrían ser mucho mejores. Por otra parte, la clasificación que hemos hecho del dataset serviría para identificar las conexiones sospechosas tipo VPN y TOR. Sin embargo, clasificar estas conexiones como malignas puede ser algo simplista. Por lo tanto, este trabajo puede continuarse desarrollando una nuevo modelo que, partiendo de las conexiones clasificadas como sospechosas, detectase las conexiones que realmente son malignas. Con este fin y para poder entrenar el modelo, sería necesario conocer si finalmente estas conexiones han resultado ser benignas o malignas. Además, otra proposición sería la de trabajar con diferentes datasets relacionados con el tráfico de red que sean más complejos y que permitan agregar configuraciones más complicadas al modelo.

Por último, destacar que la idea principal es continuar formándose en estas disciplinas para a corto o medio plazo tener la capacidad de saber aplicar estos conceptos de forma profesional en mi trabajo.

# BIBLIOGRAFÍA


[1] S. Galeano, «El número de usuarios de internet en el mundo crece un 4% y roza los 5.000 millones (2022)», 27 de enero de 2022. https://marketing4ecommerce.net/usuarios-de-internet-mundo/ (accedido 3 de marzo de 2022).

[2] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Trans. Emerg. Telecommun. Technol.*, vol. 32, n.º 1, p. 1, ene. 2021, doi: 10.1002/ett.4150.

[3] M. Álvarez Macías, «Los ciberataques a las empresas aumentaron un 150% en 2021», *CSO España*, 20 de enero de 2022. https://cso.computerworld.es/cibercrimen/los-ciberataques-a-las-empresas-aumentaron-un-150-en-2021 (accedido 3 de marzo de 2022).

[4] J. Manzano, «NO ESPERES AL CIBERATAQUE», *BDO_ES*, 14 de febrero de 2017. https://www.bdo.es/es-es/blogs/blog-coordenadas-bdo/febrero-2017/los-riesgos-de-sufrir-un-ciberataque (accedido 5 de marzo de 2021).

[5] «UCIENCIA_2021_paper_224.pdf». Accedido: 17 de junio de 2022. [En línea]. Disponible en: https://repositorio.uci.cu/jspui/bitstream/123456789/9679/1/UCIENCIA_2021_paper_224.pdf

[6] «Catalogo_Gama_DFUSION_ES.pdf». Accedido: 17 de junio de 2022. [En línea]. Disponible en: https://www.davantis.com/sites/default/files/Catalogo_Gama_DFUSION_ES.pdf

[7] Redacción Ciberseguridad.com, «¿Cómo funciona un sistema de detección de intrusos (IDS)?», *Ciberseguridad*, 21 de marzo de 2020. https://ciberseguridad.com/servicios/sistema-deteccion-intrusos-ids/ (accedido 15 de marzo de 2021).

[8] L. González Manzano, J. M. de Fuentes, y P. Peris-Lopez, «Cyber Security Basics: A Hands-on Approach». https://www.edx.org/es/course/cyber-security-basics-a-hands-on-approach

[9] E. Vasilomanolakis, S. Habib, P. Milaszewicz, R. Malik, y M. Mühlhäuser, «Towards Trust-Aware Collaborative Intrusion Detection: Challenges and Solutions», may 2017, pp. 94-109. doi: 10.1007/978-3-319-59171-1_8.

[10] C. Alcaraz, J. Rodrıguez, R. Roman, y J. E. Rubio, «Estado y Evolución de la Detección de Intrusiones en los Sistemas Industriales», p. 20, 2017.

[11] «¿Qué es un ataque de día cero?: definición y explicación», *latam.kaspersky.com*, 13 de agosto de 2021. https://latam.kaspersky.com/resource-center/definitions/zero-day-exploit (accedido 20 de junio de 2022).

[12] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Trans. Emerg. Telecommun. Technol.*, vol. 32, n.º 1, p. 5, ene. 2021, doi: 10.1002/ett.4150.

[13] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Trans. Emerg. Telecommun. Technol.*, vol. 32, n.º 1, p. 6, ene. 2021, doi: 10.1002/ett.4150.

[14] V. Pai, N. D. Adesh, y D. Bhat, «Comparative analysis of Machine Learning algorithms for Intrusion Detection», *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1013, n.º 6, p. 6, ene. 2021, doi: 10.1088/1757-899X/1013/1/012038.

[15] «¿Qué es un ataque DDoS? Ataques distribuidos de denegación de servicio», *Akamai*, 12 de noviembre de 2020. https://www.akamai.com/es/our-thinking/ddos (accedido 21 de marzo de 2021).

[16] V. Shmatikov y M.-H. Wang, «Security against probe-response attacks in collaborative intrusion detection», en *Proceedings of the 2007 workshop on Large scale attack defense - LSAD '07*, Kyoto, Japan, 2007, p. 129. doi: 10.1145/1352664.1352673.

[17] A. Lamba, S. Singh, S. Bhardwaj, N. Dutta, S. Sai, y R. Muni, «USES OF ARTIFICIAL INTELLIGENT TECHNIQUES TO BUILD ACCURATE MODELS FOR INTRUSION DETECTION SYSTEM», ago. 2015, Accedido: 5 de marzo de 2021. [En línea]. Disponible en: https://www.researchgate.net/publication/335227510_USES_OF_ARTIFICIAL_INTELLIGENT_TECHNIQUES_TO_BUILD_ACCURATE_MODELS_FOR_INTRUSION_DETECTION_SYSTEM

[18] P. Godefroid, «Fuzzing: hack, art, and science», *Commun. ACM*, vol. 63, n.º 2, pp. 70-76, ene. 2020, doi: 10.1145/3363824.

[19] V. Ciancaglini, M. Balduzzi, R. McArdle, y M. Rösler, «Below the Surface: Exploring the Deep Web», p. 48.

[20] Y. Fernández, «Deep Web, Dark Web y Darknet: éstas son las diferencias», *Xataka*, 3 de marzo de 2020. https://www.xataka.com/servicios/deep-web-dark-web-darknet-diferencias (accedido 19 de junio de 2022).

[21] «The Tor Project | Privacy & Freedom Online». https://torproject.org (accedido 17 de marzo de 2022).

[22] M. Horowitz, «A Defensive Computing term paper on privacy: VPNs, Tor and VPN routers», *Computerworld*, 18 de septiembre de 2016. https://www.computerworld.com/article/3120997/a-defensive-computing-term-paper-on-privacy-vpns-tor-and-vpn-routers.html (accedido 17 de marzo de 2022).

[23] «Aplicaciones del deep learning en la actualidad | Tokio», *Tokio School*, 22 de enero de 2021. https://www.tokioschool.com/noticias/aplicaciones-del-deep-learning/ (accedido 20 de abril de 2022).

[24] H.-D. Wehle, «Machine Learning, Deep Learning, and AI: What's the Difference?», jul. 2017.

[25] I. Peluffo, M. Capobianco, y J. Echaiz, «Machine Learning aplicado en Sistemas de Detección de Intrusos», *Universidad Nacional del Sur*, vol. 5, pp. 812-816, 2014.

[26] Z. Ahmad, A. Shahid Khan, C. Shiang, y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches»,

*Trans. Emerg. Telecommun. Technol.*, vol. 32, n.º 1, p. 7, ene. 2021, doi: 10.1002/ett.4150.

[27] C. Serrato, «Diferencia entre aprendizaje supervisado y no supervisado: ¿Cuál es mejor para mi? - INMEDIATUM», *INMEDIATUM*, 25 de marzo de 2021. https://inmediatum.com/blog/estrategia/diferencia-entre-aprendizaje-supervisado-y-no-supervisado/, https://inmediatum.com/blog/estrategia/diferencia-entre-aprendizaje-supervisado-y-no-supervisado/ (accedido 8 de abril de 2022).

[28] V. Pai, D. Bhat, y N. Adesh, «Comparative analysis of Machine Learning algorithms for Intrusion Detection», *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1013, n.º 5, p. 5, ene. 2021, doi: 10.1088/1757-899X/1013/1/012038.

[29] P. Recuero, «¿Sabes en qué se diferencian las redes neuronales del Deep Learning?», *Blogthinkbig.com*, 26 de abril de 2019. https://blogthinkbig.com/redes-neuronales-deep-learning (accedido 8 de abril de 2022).

[30] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Trans. Emerg. Telecommun. Technol.*, vol. 32, n.º 1, p. 10, ene. 2021, doi: 10.1002/ett.4150.

[31] M. Alberti y N. Hubens, «Introducción al autoencoder – DeepLearningItalia», 8 de mayo de 2018. https://www.deeplearningitalia.com/introduzione-agli-autoencoder-2/ (accedido 15 de abril de 2022).

[32] J. Barrios, «Redes neuronales convolucionales», *Juan Barrios*, 19 de enero de 2022. https://www.juanbarrios.com/redes-neurales-convolucionales/ (accedido 17 de abril de 2022).

[33] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, y F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Trans. Emerg. Telecommun. Technol.*, vol. 32, n.º 1, p. 17, ene. 2021, doi: 10.1002/ett.4150.

[34] Deeplizard, *Deep Learning Fundamentals*, (23 de noviembre de 2017). Accedido: 5 de junio de 2021. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=gZmobeGL0Yg

[35] «Deeplizard - YouTube», *Deep Lizard*. https://www.youtube.com/c/deeplizard (accedido 5 de junio de 2022).

[36] F. Sancho Caparrini, «Entrenamiento de Redes Neuronales: mejorando el Gradiente Descendente», 20 de diciembre de 2020. http://www.cs.us.es/~fsancho/?e=165 (accedido 20 de abril de 2022).

[37] K. Pina, «Cómo entrenar a tu perceptrón», *Koldo Pina - Data Science & Inteligencia Artificial*, 10 de marzo de 2018. https://15.236.101.236/como-entrenar-a-tu-perceptron/ (accedido 20 de abril de 2022).

[38] P. A. Rousseau-Figueroa, J. Ramírez-Hernández, S. O. Infante-Prieto, R. Villa-Angulo, y M. Hallack-Alegría, «La influencia del efecto de borde en el pronóstico de precipitaciones utilizando DWT diádica, MODWT, ANN y ANFIS», *Tecnol. Cienc. Agua*, vol. VII, n.º 3, pp. 93-113, 2016.

[39] «Darknet 2020 | Datasets | Research | Canadian Institute for Cybersecurity | UNB», *Canadian Institute for Cybersecurity*. https://www.unb.ca/cic/datasets/darknet2020.html (accedido 20 de abril de 2022).

[40] «Darknet 2020 | Datasets | Research | Canadian Institute for Cybersecurity | UNB», *Canadian Institute for Cybersecurity*. http://205.174.165.80/CICDataset/CICDarknet2020/ (accedido 20 de abril de 2022).

[41] «About the CIC | Canadian Institute for Cybersecurity | UNB». https://www.unb.ca/cic/about/index.html (accedido 20 de abril de 2022).

[42] Q. Abu Al-Haija, M. Krichen, y W. Abu Elhaija, «Machine-Learning-Based Darknet Traffic Detection System for IoT Applications», *Electronics*, vol. 19, n.º 4, Art. n.º 4, ene. 2022, doi: 10.3390/electronics11040556.

[43] Q. Abu Al-Haija, M. Krichen, y W. Abu Elhaija, «Machine-Learning-Based Darknet Traffic Detection System for IoT Applications», *Electronics*, vol. 19, n.º 16, Art. n.º 4, ene. 2022, doi: 10.3390/electronics11040556.

[44] «API Documentation | TensorFlow Core v2.9.1», *TensorFlow*. https://www.tensorflow.org/api_docs (accedido 20 de junio de 2022).

[45] «Keras: the Python deep learning API», *Keras. Simple. Flexible. Powerful*. https://keras.io/ (accedido 20 de abril de 2022).

[46] «Documentation for Visual Studio Code», *Visual Studio Code*. https://code.visualstudio.com/docs (accedido 20 de junio de 2022).

[47] M. Coutinho Marim, «Darknet - preprocessing», *Kaggle*, 2021. https://kaggle.com/mateus558/darknet-preprocessing (accedido 28 de abril de 2022).

[48] «Redes neuronales profundas - Tipos y Características», *Código Fuente*, 25 de abril de 2019. https://www.codigofuente.org/redes-neuronales-profundas-tipos-caracteristicas/ (accedido 20 de mayo de 2022).

[49] A. Zhang, Z. C. Lipton, M. Li, y A. J. Smola, «Dive into Deep Learning», p. 1025, mar. 2022.

[50] C. Geek, «Implementación y comparación de optimizadores de modelos en aprendizaje profundo», *Netlify*. https://tech-es.netlify.app/articles/es525214/index.html (accedido 25 de mayo de 2022).

[51] R. López Briega, «Introducción al Deep Learning», *Matemáticas, análisis de datos y python*, 13 de marzo de 2017. https://relopezbriega.github.io/blog/2017/06/13/introduccion-al-deep-learning/ (accedido 25 de mayo de 2022).

[52] I. Gavilán, «Blue chip: Catálogo de componentes de redes neuronales (III): funciones de pérdida», *Blue chip*, 25 de mayo de 2020. http://bluechip.ignaciogavilan.com/2020/05/catalogo-de-componentes-de-redes_25.html#.YrDTAXZBzIV (accedido 25 de mayo de 2022).

[53] L. Velasco, «Optimizadores en redes neuronales profundas: un enfoque práctico», *Medium*, 26 de abril de 2020. https://velascoluis.medium.com/optimizadores-en-

redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5 (accedido 20 de junio de 2022).

[54] «Aprendiendo Inteligencia Artificial: ¿Qué es el optimizador?», *Comunidad Huawei Enterprise*, 3 de junio de 2021. https://forum.huawei.com/enterprise/es/aprendiendo-inteligencia-artificial-%C2%BFqu%C3%A9-es-el-optimizador/thread/745543-100757 (accedido 26 de mayo de 2022).

[55] Z. Global, «Brutalk - Introducción suave al algoritmo de optimización de Adam para el aprendizaje profundo», *Brutalk*, 2021. https://www.brutalk.com/en/news/brutalk-blog/view/introduccion-suave-al-algoritmo-de-optimizacion-de-adam-para-el-aprendizaje-profundo-60471b4df29fa (accedido 25 de mayo de 2022).

[56] C.-F. Wang, «A Newbie's Guide to Stochastic Gradient Descent With Restarts», *Medium*, 28 de julio de 2018. https://towardsdatascience.com/https-medium-com-reina-wang-tw-stochastic-gradient-descent-with-restarts-5f511975163 (accedido 25 de mayo de 2022).

[57] «tf.keras.optimizers.schedules.CosineDecayRestarts | TensorFlow Core v2.9.1», *TensorFlow*. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/CosineDecayRestarts (accedido 26 de mayo de 2022).

[58] J. Jordan, «Setting the learning rate of your neural network.», 1 de marzo de 2018. https://www.jeremyjordan.me/nn-learning-rate/ (accedido 28 de mayo de 2022).

[59] S. Kansal, «A Quick Guide to Activation Functions In Deep Learning», *TowardsDataScience*, 19 de agosto de 2020. https://towardsdatascience.com/a-quick-guide-to-activation-functions-in-deep-learning-4042e7addd5b (accedido 20 de junio de 2022).

[60] J. Brownlee, «A Gentle Introduction to the Rectified Linear Unit (ReLU)», *Machine Learning Mastery*, 8 de enero de 2019. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (accedido 29 de mayo de 2022).

[61] Orome, «How do I control the spacing and font size for my plot legend?», *Mathematica Stack Exchange*, 13 de diciembre de 2020. https://mathematica.stackexchange.com/q/153861 (accedido 29 de mayo de 2022).

[62] J. Huber, «Batch normalization in 3 levels of understanding», *Towards Data Science*, 6 de noviembre de 2020. https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338 (accedido 29 de mayo de 2022).

[63] J. Brownlee, «Train Neural Networks With Noise to Reduce Overfitting», *Machine Learning Mastery*, 11 de diciembre de 2018. https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/ (accedido 29 de mayo de 2022).

[64] «How to deactivate dropout layers while evaluation and prediction mode in Keras?», *Knowledge Transfer*, 6 de diciembre de 2021. https://androidkt.com/deactivate-dropout-layers-while-evaluation-prediction-mode-keras/ (accedido 29 de mayo de 2022).

[65] Othmane, «Answer to “Keras Sequential model returns loss «nan»”», *Data Science Stack Exchange*, 18 de abril de 2020. https://datascience.stackexchange.com/a/72520 (accedido 20 de junio de 2022).

[66] K. Team, «Keras documentation: Adam». https://keras.io/api/optimizers/adam/ (accedido 20 de junio de 2022).

[67] G. Mayanglambam, «Deep Learning Optimizers. SGD with momentum, Adagrad, Adadelta… | by Gunand Mayanglambam | Towards Data Science», *Towards Data Science*, 18 de noviembre de 2020. https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f (accedido 20 de junio de 2022).

[68] DeepLizard, *TensorFlow - Python Deep Learning Neural Network API*. Accedido: 27 de julio de 2021. [En línea Video]. Disponible en: https://www.youtube.com/playlist?list=PLZbbT5o_s2xrwRnXk_yCPtnqqo4_u2YGL

# ANEXO

## OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

| Objetivos de Desarrollo Sostenibles | Alto | Medio | Bajo | No Procede |
|---|---|---|---|---|
| ODS 1. **Fin de la pobreza.** | | | X | |
| ODS 2. **Hambre cero.** | | | X | |
| ODS 3. **Salud y bienestar.** | | | X | |
| ODS 4. **Educación de calidad.** | | | X | |
| ODS 5. **Igualdad de género.** | | | X | |
| ODS 6. **Agua limpia y saneamiento.** | | | X | |
| ODS 7. **Energía asequible y no contaminante.** | | | X | |
| ODS 8. **Trabajo decente y crecimiento económico.** | X | | | |
| ODS 9. **Industria, innovación e infraestructuras.** | X | | | |
| ODS 10. **Reducción de las desigualdades**. | | X | | |
| ODS 11. **Ciudades y comunidades sostenibles**. | | | X | |
| ODS 12. **Producción y consumo responsables.** | | | X | |
| ODS 13. **Acción por el clima.** | | | X | |
| ODS 14. **Vida submarina.** | | | X | |
| ODS 15. **Vida de ecosistemas terrestres.** | | | X | |
| ODS 16. **Paz, justicia e instituciones sólidas.** | | | X | |
| ODS 17. **Alianzas para lograr objetivos.** | | | X | |

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El Trabajo de Final de Grado (TFG) que se ha desarrollado está directamente relacionado con los Objetivos de Desarrollo Sostenible (ODS) en el sentido de la optimización y modernización de la tecnología, así como en la generación de un sistema de detección de intrusos más eficiente y eficaz del utilizado actualmente en este ámbito, permitiendo mejorar la protección, confidencialidad e integridad de ingentes cantidades de información, tanto de empresas como de personas particulares de todo el mundo.

Concretamente, el **ODS 8** "Trabajo decente y crecimiento económico" está relacionado con este trabajo debido al aumento de la productividad económica mediante la modernización de la tecnología y la innovación. La finalidad de este trabajo podría ayudar a muchas empresas a proteger su información de una manera mucho más eficiente y automática, lo que debe aumentar la productividad de la compañía y por tanto su consiguiente crecimiento económico.

Igualmente, el **ODS 9** "Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación" está de la misma manera relacionado con este trabajo. Hoy en día se están utilizando Sistemas de Detección de Intrusos poco (o menos) eficientes los cuales se van mejorando según van recibiendo ataques que aprovechan las vulnerabilidades de los sistemas de información. En este trabajo se ha estudiado el funcionamiento e implementación de un sistema de detección de intrusos basado en red que utiliza técnicas de aprendizaje automático, lo que supone un gran aumento de la eficiencia y detectabilidad de las amenazas para este tipo de sistemas. De esta forma, se perfecciona la operación y el desempeño de las mismas funciones que se realizaban con el anterior tipo de sistemas de detección mediante la aplicación de técnicas más sofisticadas e innovadoras.

Por otra parte, me gustaría destacar la importancia de la privacidad y la protección de datos en la sociedad en la que vivimos. En un mundo en el que la tecnología, el Internet, la publicidad digital y la cantidad de información digital con la que estamos en contacto todos los días es abismal, la privacidad de la información personal debe ser una prioridad que tratar por los gobernantes de todos los países. Por tanto, para lograr una industrialización y una digitalización sostenible, es estrictamente necesario que los datos digitales de todas las personas del mundo, especialmente las más vulnerables (no sólo vulnerables económicamente, sino también digitalmente, como pueden ser las personas mayores o los más pequeños) se encuentren seguros. Por todo esto, el trabajo realizado para la detección de intrusiones no deseadas puede ser aplicado para esta mejorar los procesos que tengan esta finalidad.

Por último, hoy en día, en los países menos desarrollados se encuentran en un proceso de transformación digital mediante la cual están comenzando a tener acceso a internet

y a dispositivos inteligentes. Al igual que en los países desarrollados, se necesita una educación digital inclusiva y total que les permita conocer los riesgos a los que están expuestos al utilizar un dispositivo conectado a Internet. En este sentido, un sistema de detección de intrusos que utiliza técnicas de aprendizaje automático, como se ha desarrollado en este trabajo, y aplicado a nivel de red, podría facilitar a gran escala este desempeño. Este concepto se relaciona con el **ODS 10** "Reducción de las desigualdades" tanto a nivel interno como externo de los países, al permitir y facilitar a las personas más desfavorecidas el acceso seguro tanto a las nuevas tecnologías como a los conocimientos que estas mismas permiten aprender a través de ellas.