Análisis de Tráfico en Dispositivos Móviles mediante Técnicas de Aprendizaje Profundo Supervisado

Traffic Analysis on Mobile Devices Using Supervised Deep Learning Techniques

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2021–2022

Pablo Díaz Rodríguez (Grado en Ingeniería Informática)
Adina Han (Grado en Ingeniería Informática)
David Merino Mayor (Grado en Ingeniería Informática)

Directores

Luis Javier García Villalba
Daniel Povedano Álvarez

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2022

Agradecimientos

A Luis Javier García Villalba, director del Grupo de Análisis, Seguridad y Sistemas de la Facultad de Informática de la Universidad Complutense de Madrid, por confiar en nosotros desde el primer momento y animarnos a formar parte de este proyecto.

A Javier y a Daniel Povedano Álvarez, nuestros tutores por todo el apoyo recibido, y por el apoyo técnico proporcionado durante todo el proyecto.

A Luis Martínez y Sandra Pérez por su constancia durante todo el proyecto y su apoyo semanal para ir avanzado poco a poco.

A Ana L. Sandoval por su ayuda con todo el tema de documentación, además de su motivación constante.

A la Facultad de Informática de la Universidad Complutense de Madrid y a los profesores que han estado durante todos estos años de carrera y con los que hemos podido adquirir los conocimientos necesarios para poder desarrollar este trabajo.

Índice General

Índice de Figuras

Índice de Tablas

Lista de Acrónimos

AA	Aprendizaje Automático
ADN	Ácido desoxirribonucleico
AMI	Adjusted Mutual Information
ASIC	Application-Specific Integrated Circuit
CA	Certification Authority
CART	Classification and Regression Trees
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CSV	Comma Separated Values
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DGCNN	Deep Graph Convolutional Neural Networks
DL	Deep Learning
DPI	Deep Packet Inspection
E/S	Entrada/Salida
EE. UU.	Estados Unidos
EFA	Exploratory Factor Analysis
ETA	Encrypted Traffic Analytics

FN	Falsos Negativos
FP	Falsos Positivos
FTP	File Transfer Protocol
GASS	Grupo de Análisis, Seguridad y Sistemas
GB	Gigabyte
GK	Graphlet Kernel
GMM	Gaussian Mixture Models
GPU	Graphics Processing Unit
HDF5	Hierarchical Data Format version 5
HIDS	Host Intrusion Detection System
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IA	Inteligencia Artificial
IANA	Internet Assigned Numbers Authority
IBM	International Business Machines
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
ISO	Internacional Organization for Standardization

KNN	K-Nearest Neighbors
MAE	Mean Absolute Error
MB	Megabyte
ML	Machine Learning
MSE	Mean Square Error
NAT	Network Address Translation
OSI	Open Systems Interconnection
PCA	Principal Component Analysis
PCAP	Packet Capture
PK	Propagation Kernel
RAE	Real Academia Española
RF	Random Forest
RMSE	Root Mean Square Error
RW	Random Walk kernel
SAE	Stacked Autoencoder
SCP	Secure Copy Protocol
SET	Secure Electronic Transaction
SFTP	SSH File Transfer Protocol
SMTP	Simple Mail Transfer Protocolo
SSH	Secure Socket Shell

SSL	Secure Socket Layer
SVC	Support Vector Classifier
SVM	Support Vector Machine
tanh	Función tangente hiperbólica
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UCM	Universidad Complutense de Madrid
UDP	User Datagram Protocol
VE	Voting Experts
VN	Verdaderos Negativos
VoIP	Voice Over Internet Protocol
VP	Verdaderos Positivos
VPN	Virtual Private Network
WL	Weisfeiler-Lehman Subtree Kernel
XGBoost	Extreme Gradient Boosting

Abstract

In recent years, traffic between mobile applications has grown exponentially. Thus, the task of classifying traffic on the network, such as indicating which mobile application has generated the traffic, has become increasingly difficult due to the high use of security protocols to protect user and data privacy. Deep learning based traffic classification methods can cope with this impediment and for this reason the system proposed in this work is based on autoencoders, convolutional neural networks and graphical convolutional neural networks. The traffic dataset with which the experiments have been done, have been collected throughout the period of this work by a group of students and on a set of existing mobile applications on Android.

Keywords: Network traffic, Classification, Convolutional Network, Autoencoder, Graphs, Dataset, Neural Networks, Machine Learning, Artificial Intelligence, Android

Resumen

En los últimos años, el tráfico entre aplicaciones móviles ha crecido exponencialmente. Así, la tarea de clasificar el tráfico en la red, como puede ser indicar qué aplicación móvil ha generado dicho tráfico, se ha vuelto cada vez más difícil por el elevado uso de protocolos de seguridad para proteger la privacidad de los usuarios y de los datos. Los métodos de clasificación de tráfico basados en el aprendizaje profundo pueden hacer frente a este impedimento y por esta razón el sistema propuesto en este trabajo está basado en autocodificadores, redes neuronales convolucionales y redes neuronales convolucionales gráficas. El conjunto de datos de tráfico con el que se han hecho los experimentos, se ha recopilado durante todo el periodo de realización de este trabajo por un grupo de alumnos y sobre un conjunto de aplicaciones móviles existentes en *Android*.

Palabras clave: Tráfico en la red, Clasificación, Redes Convolucionales, Autocodificador, Grafos, Conjunto de datos, Redes neuronales, Aprendizaje automático, Inteligencia Artificial, Android

Capítulo 1

Introducción

1.1. Motivación

En la última década, el tráfico de red ha aumentado drásticamente, por lo que la privacidad y seguridad, tanto de los ciudadanos como de las empresas, se han visto comprometidas. Por lo tanto, ha surgido la necesidad de implantar nuevos controles de seguridad que hacen más difícil la tarea de analizar la red y, en concreto, la clasificación de aplicaciones móviles basada en el tráfico de red.

Las técnicas más conocidas para realizar dicha tarea son: inspección profunda de paquete (Deep Packet Inspection (DPI)), métodos basados en el número de puerto (*port number*), inspección de la carga útil (*payload*) y métodos basados en Aprendizaje Automático (AA), entre otros. Debido a que las características del entorno de red están en constante cambio, estas soluciones se quedan rápidamente obsoletas. El aprendizaje profundo (Deep Learning (DL)) es un nuevo enfoque que se ha utilizado en los últimos años para la clasificación del tráfico de red.

1.2. Contexto

Este Trabajo de Fin de Grado se desarrolló en el Grupo de Análisis, Seguridad y Sistemas (GASS) (`https://gass.ucm.es`) del Departamento de Ingeniería del Software e Inteligencia Artificial de la Facultad de Informática de la Universidad Complutense de Madrid (UCM).

1.3. Objeto de la Investigación

El objetivo de este proyecto es el estudio y la clasificación de aplicaciones móviles de *Android*. Para ello, se basa en otros trabajos de investigación encontrados en la literatura, utilizando algoritmos de AA como los *Autoencoders*, las Convolutional Neural Networks (CNN) y las Deep Graph Convolutional Neural Networks (DGCNN) para la clasificación.

Otro de los objetivos de la investigación es comprobar cuál de los modelos generados proporciona una mayor precisión en la información de los datos. Para ello, se modifican algunos hiperparámetros de las diferentes arquitecturas de DL mencionadas anteriormente,

como por ejemplo, el número de neuronas, el número de capas, el número de épocas y las funciones de activación, entre otros.

1.4. Plan de Trabajo

Para el desarrollo de este trabajo se han llevado a cabo las siguientes fases:

1. **Investigación**

   La fase de investigación se ha realizado durante los primeros meses de trabajo, y el tema general ha sido el análisis de tráfico en aplicaciones *Android* (análisis de red, protocolos, técnicas de análisis de tráfico) y las técnicas de AA para la realización de esta tarea (AA supervisado, no supervisado, semi-supervisado, los distintos algoritmos de AA y DL como los algoritmos de clasificación, de regresión, los basados en árboles de decisión, de agrupación y redes neuronales, entre otros).

   Con la información encontrada en los trabajos, se ha ido concretando poco a poco la temática final, centrándose finalmente en la clasificación de aplicaciones móviles en *Android*. Durante la investigación, se han identificado los diferentes trabajos que han servido como base para el desarrollo de este proyecto.

   Los artículos de revistas científicas, que se han encontrado en el buscador especializado en documentos académicos *Google Académico*, y las referencias utilizadas en estos artículos han sido la base de conocimiento de la que se ha partido. Durante las siguientes fases, se ha seguido con la investigación pero a un menor ritmo.

2. **Desarrollo**

   La parte inicial del desarrollo se ha realizado a la vez que la fase de investigación. Los trabajos académicos encontrados en la fase de investigación han sido el punto de partida en el desarrollo de la herramienta.

   Se han ido incorporando los modelos de AA en la implementación del sistema propuesto por este trabajo. Para esta parte, se han utilizado los métodos que se incluyen en el Estado del Arte ([Lotfollahi et al., 2020] o [Zhang et al., 2018], entre otros) en el Capítulo 3: Stacked Autoencoder (SAE), CNN, y DGCNN. A su vez, se han realizado capturas de tráfico de red en formato Packet Capture (PCAP) con el fin de poder contar con un conjunto de datos propio.

   Esta fase se ha realizado utilizando el lenguaje de programación *Python* y sus respectivas librerías, como por ejemplo *Numpy*, *Keras*, *Pandas*, *Scikit-Learn*, *TensorFlow*, *PyTorch*, *Sklearn* o *Matplotlib*.

3. **Experimentos y resultados**

   En esta última fase, se han realizado experimentos con distintos parámetros para los algoritmos seleccionados con el fin de obtener la combinación que proporcione los mejores resultados. Los resultados obtenidos se han analizado en base a distintas métricas y se han interpretado para poder extraer las conclusiones presentes en este trabajo.

El flujo de trabajo se puede observar en la imagen 8.1

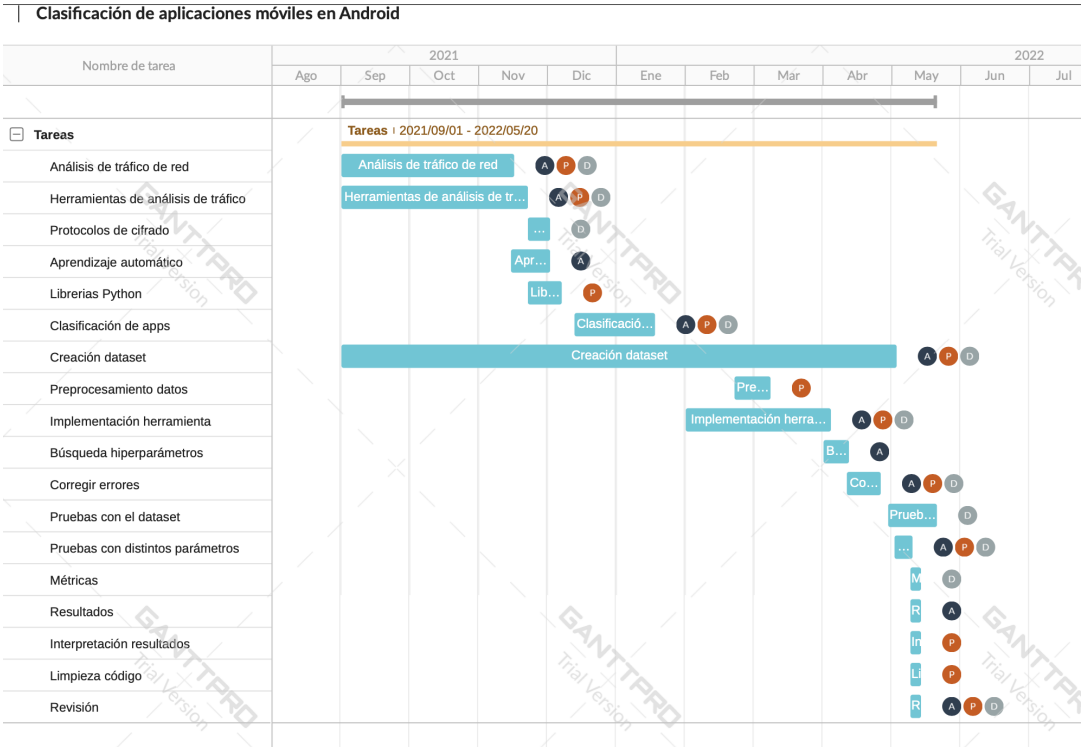

Figura 1.1: Diagrama de Gantt del flujo de trabajo

1.5. Estructura del Trabajo

La memoria de este trabajo se organiza de la siguiente manera:

El Capítulo 2 presenta el panorama de la evolución del análisis del tráfico de red, los protocolos de seguridad más conocidos y los algoritmos de AA empleados para realizar el análisis.

En el Capítulo 3 se introducen los métodos actuales utilizados para la clasificación de tráfico en aplicaciones móviles de *Android* y su funcionamiento comparando las diferentes técnicas de los trabajos relacionados con esta temática.

El sistema propuesto se detalla en el Capítulo 4 y los experimentos y los resultados obtenidos se formulan en el Capítulo 5.

Las contribuciones de cada miembro del grupo se detallan en el Capítulo 6

Por último, las conclusiones y el trabajo futuro se resumen en el Capítulo 7.

Los Capítulos 8 y 9 representan las traducciones en inglés de la Introducción, las Conclusiones y Trabajo Futuro.

Capítulo 2

Contexto de la Investigación

Este capítulo engloba una breve descripción de algunos de los protocolos de seguridad de tráfico de red, la importancia de la Inteligencia Artificial (IA) en el mundo actual y sus subcampos y los principales algoritmos de AA y DL.

Con el paso de los años, la red ha ido evolucionando en base a las necesidades de los usuarios. La preocupación por la privacidad y seguridad de los usuarios ha dado paso al estudio del tráfico de la red. Según el informe anual de 2021 de *Let's Encrypt* [Letsencrypt, 2021], se han emitido aproximadamente 2,5 millones de certificados Transport Layer Security (TLS) al día, con un total de más de 260 millones de sitios web protegidos, tal y como se puede observar en la Figura 2.1. También, se afirma que desde que lanzaron el primer proyecto en 2015, las páginas que utilizan HyperText Transfer Protocol Secure (HTTPS) han subido más del 50 % (del 40 % al 92 %) en Estados Unidos (EE. UU.) y hasta un 83 % en el resto del mundo.

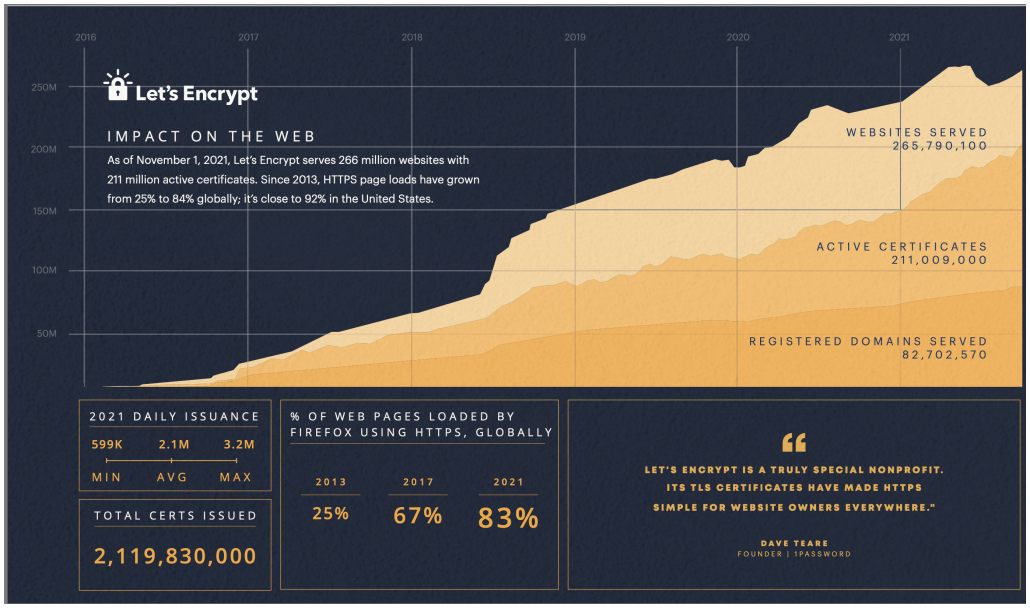

Figura 2.1: Informe anual de 2021 de Let's Encrypt

En la Figura 2.2 se puede observar cómo más de la mitad de la población mundial son usuarios de *Internet* y son usuarios activos de las redes sociales. Esto significa que se generan volúmenes masivos de datos cada día, macrodatos que hay que procesar utilizando

las herramientas de *Big Data* [Hootsuite, 2022].

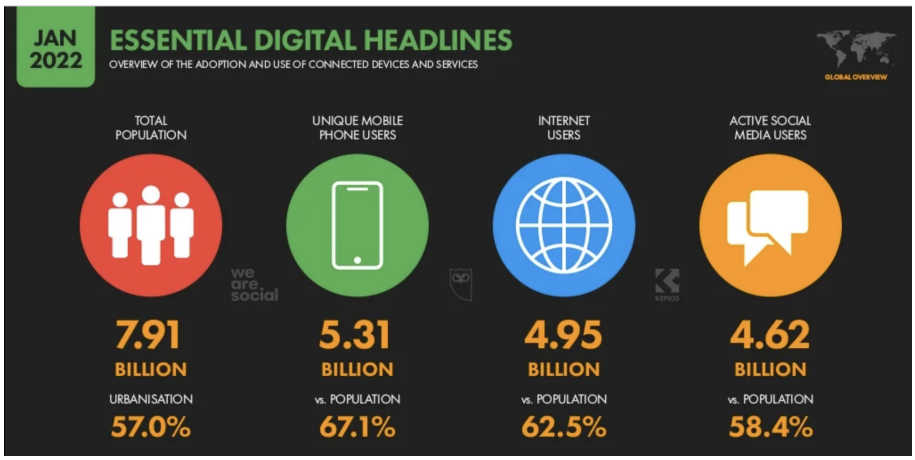

Figura 2.2: Informe de Hootsuite de 2022

Big Data es un término en desarrollo que describe, a grandes rasgos, un gran volumen de datos. Según la Real Academia Española (RAE), el término *dato* en relación al campo de la Informática se refiere a la información dispuesta de manera adecuada para su tratamiento por una computadora.

Estos datos masivos se han caracterizado hasta ahora por las conocidas como *"las tres V"*: *Volumen* (gran cantidad de datos), *Variedad* (amplia variedad de tipos de datos) y *Velocidad* (rapidez con la que deben procesarse). Además, en los últimos años se han introducido otros dos términos: *Valor* (información de valor que se puede obtener de los datos) y *Veracidad* (fiabilidad de los datos).

Con las acepciones anteriores, se podría decir que el *Big Data* está formado por conjuntos de datos de grandes dimensiones y muy complejos que vienen de fuentes de datos variadas. Tales dimensiones de datos hacen que no se puedan tratar como otro tipo de datos sencillos. Sin embargo, estos conjuntos de datos tienen la ventaja de que pueden utilizarse para abordar problemas que hasta ahora no hubiese sido posible solucionar.

2.1. Protocolos TCP/IP

Son un conjunto de protocolos usados en *Internet*. Tiene la estructura de jerarquía y está compuesto por capas o módulos que ofrecen una funcionalidad específica [TCP/IP, 2021].

- **Capa de aplicación**: Permite la comunicación extremo a extremo y el intercambio de mensajes entre dos aplicaciones, definiendo las aplicaciones y servicios de *Internet* que un usuario puede utilizar.
- **Capa de transporte**: El intercambio de la confirmación de recepción de los paquetes y la retransmisión de estos garantiza la llegada de los paquetes en secuencia y sin errores. Comprende los protocolos Transmission Control Protocol (TCP), orientado a conexión, y User Datagram Protocol (UDP), sin conexión.
- **Capa de Internet**: Responsable de enviar los paquetes por la red.

- **Capa de enlace de datos**: Responsable de la identificación del protocolo de red del paquete.
- **Capa de red física**: Se encarga de la especificación de las características hardware utilizadas por la red.

2.1.1. Modelo OSI

El modelo Open Systems Interconnection (OSI) es un estándar Internacional Organization for Standardization (ISO) que trata los aspectos de la comunicación en red. Este estándar permite la comunicación entre dos sistemas. El modelo está estructurado en capas.

En la Tabla 2.1 se puede ver una comparativa de las capas de los modelos explicados anteriormente.

Tabla 2.1: Comparación de las capas de los modelos OSI y TCP/IP

<table>
<tr><th>OSI</th><th>TCP/IP</th></tr>
<tr><td>Aplicación</td><td rowspan="3">Aplicación</td></tr>
<tr><td>Presentación</td></tr>
<tr><td>Sesión</td></tr>
<tr><td>Transporte</td><td>Transporte</td></tr>
<tr><td>Red</td><td>Internet</td></tr>
<tr><td>Enlace de datos</td><td>Enlace de datos</td></tr>
<tr><td>Física</td><td>Física</td></tr>
</table>

2.2. Protocolos

Los protocolos velan para que la información de los usuarios esté protegida y cifrada y que las páginas web a las que se accede sean lo más seguras posibles.

Algunos de los protocolos que ayudan a que esta tarea se cumpla son los explicados a continuación.

2.2.1. Protocolo SSL y TLS

Secure Socket Layer (SSL) o Capa de Puertos Seguros es el protocolo desarrollado en 1994 por la empresa *Netscape* que tenía como función proteger las comunicaciones en *Internet* proporcionando un canal seguro entre los 2 extremos de la comunicación, normalmente entre un navegador y un servidor web.

Actualmente, SSL está en desuso porque ha sido sustituido por el protocolo TLS que apareció en 1999, cuyo *handshake* o protocolo de verificación se puede ver en la Figura 2.3 [IBM, 2021].

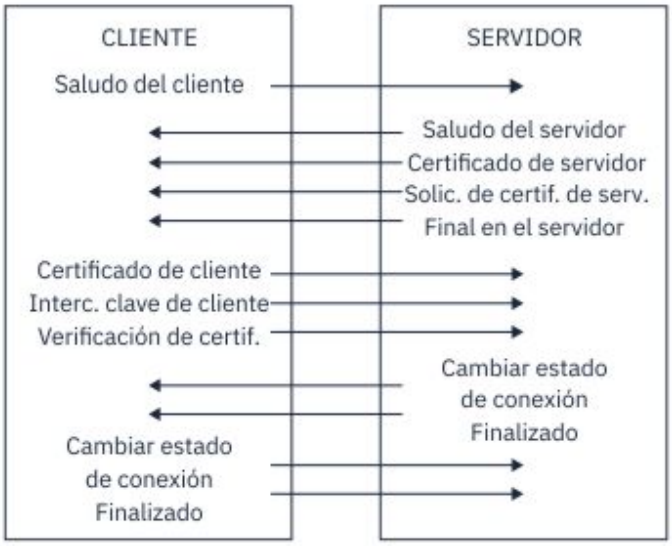

Figura 2.3: TLS

Este es también un protocolo de la capa de transporte sobre TCP y tiene la misma función: que las conexiones sean seguras y fiables proporcionando confidencialidad, integridad, autenticación, no repudio y protección contra repeticiones mediante el uso de certificados digitales. Es uno de los protocolos más utilizados y, tal y como se menciona en [Velan et al., 2015], se utiliza para asegurar sesiones HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP) y Simple Mail Transfer Protocolo (SMTP), pero también para Voice Over Internet Protocol (VoIP) y Virtual Private Network (VPN).

Desde 2018, está disponible la última versión de este protocolo: TLS 1.3.

2.2.2. Protocolo SSH

Secure Socket Shell (SSH) es el protocolo que permite a los usuarios poder manipular sus servidores remotos utilizando un mecanismo de autenticación. Este protocolo surgió como consecuencia de otros protocolos para dar una mayor seguridad al llevar cifrada toda la información que maneja.

Actualmente, este protocolo se puede utilizar también para permitir la transferencia de archivos mediante el protocolo asociado SSH File Transfer Protocol (SFTP) y Secure Copy Protocol (SCP), o mediante VPN, tal y como se recoge en [Velan et al., 2015].

Este protocolo principalmente intenta establecer una conexión modelo cliente-servidor. Para ello, la aplicación cliente, que hace referencia a las siglas del protocolo, *Secure Shell*, se conecta a un servidor SSH. Como se menciona anteriormente, la seguridad se proporciona debido al proceso de autenticación.

Esta conexión modelo cliente-servidor está representada en la Figura 2.4 [eduardocollado, 2020].

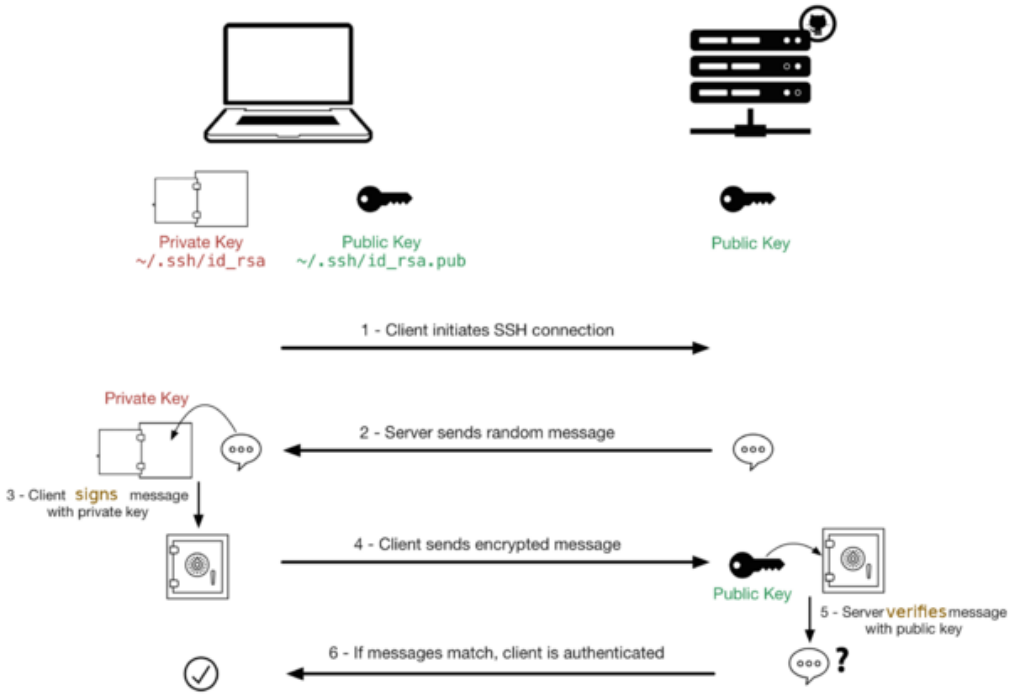

Figura 2.4: Autenticación SSH

2.2.3. Protocolo SET

El protocolo Secure Electronic Transaction (SET) es un protocolo de seguridad que utiliza métodos de **cifrado y *hash*** para garantizar que las transacciones electrónicas con tarjetas bancarias son seguras, la información es confidencial y protege la integridad de la información de los pagos, como afirman en [Paulson, 2002]. Este protocolo ha sido desarrollado por *Visa, Mastercard, American Express, IBM, Microsoft, Netscape* y otras empresas en 1996.

2.2.4. Protocolo IPsec

Internet Protocol Security (IPsec) es un conjunto de protocolos criptográficos que garantiza en la capa de red que la transmisión sobre el Internet Protocol (IP) es segura. Esto se debe a la **autenticación y cifrado** de cada paquete IP entre los 2 extremos. Al actuar en la capa de red en lugar de actuar en la capa de aplicación como lo hace SSL, cuenta con más flexibilidad y puede ser utilizado para proteger protocolos de la capa de transporte, como TCP y UDP. En [Velan et al., 2015] se indica que la principal ventaja de este protocolo es que asegura la conexión de red sin que los clientes y servidores tengan la necesidad de modificar aplicaciones.

IPsec proporciona seguridad en 2 modos: **modo transporte** (solo la carga útil del paquete IP es cifrada o autenticada) y **modo túnel** (todo el paquete IP es cifrado o autenticado). Esta es una de las formas más seguras de encriptar datos y por eso es uno de los protocolos más utilizados por las VPN.

2.2.5. Protocolo HTTPS

El protocolo de transferencia de hipertexto seguro o HTTPS es un protocolo web que utiliza un cifrado seguro para establecer una comunicación segura entre el cliente web y el servidor web, lo que impide el acceso a los datos transferidos a personas no autorizadas, como se menciona en [Prandini et al., 2010].

El icono del candado que aparece en la barra de direcciones en algunos navegadores como *Chrome, Firefox* y *Edge* indica que se está utilizando el protocolo HTTPS y por lo tanto, que la comunicación está cifrada. Esto se puede observar en la Figura 2.5.

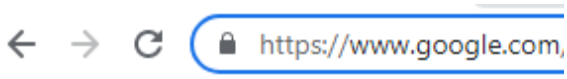

Figura 2.5: HTTPS

2.3. Encaminamiento de paquetes

El enrutamiento o encaminamiento de paquetes es un proceso realizado por el *router* para enviar los paquetes a la red de destino. El *router* se basa en la dirección *IP* del destino para que los paquetes lleguen correctamente.

Existe la denominada **tabla de encaminamiento**, que es la tabla en la que el *router* se basa para llevar a cabo el encaminamiento, es decir, gracias a ella, toma las decisiones. En esta tabla se almacena la información sobre los posibles destinos y cómo puede alcanzarlos el paquete [Encaminamiento, 2021].

El encaminamiento de paquetes puede ser de tres tipos:

1. **Encaminamiento estático**. El enrutador o enrutadores son predeterminados.
2. **Encaminamiento dinámico**. Los enrutadores están en redes locales con múltiples *hosts* y *hosts* de sistemas autónomos. Es usado en las interredes de mayor tamaño. El enrutamiento dinámico suele ser la mejor opción en la mayoría de las redes.
3. **Estático y dinámico combinados**. La combinación del enrutamiento estático y dinámico en un sistema es una práctica habitual.

2.4. Inteligencia Artificial y Aprendizaje Automático

La IA se refiere a las aplicaciones informáticas y al desarrollo de estas, que permiten que una máquina pueda imitar el comportamiento y la inteligencia humana, como se expresa en [PK, 1984].

La IA surge en la década de los 40 y unos años después, Alan Turing publica un artículo sobre números computables en el cual introduce el concepto de **algoritmo**. Debido a sus aportes, es considerado el padre de la computación o el padre de la IA.

Durante la conferencia de Darmouth en 1956, se emplea por primera vez el término de IA. Entre los años 1957 y 1974, se ha creado la primera **red neuronal artificial** (1957), el

primer ***chatbot*** *(ELIZA)* capaz de conversar debido al procesamiento de lenguaje natural integrado y se crea el **perceptrón** (1969).

En 1979 el vehículo conocido como *El cart de Stanforf* se convirtió en el primer **vehículo autónomo** y consiguió recorrer un camino con obstáculos de forma autónoma.

Otra victoria que consiguió la IA es la del programa informático *BKG 9.8*. **Backgammon** es un juego de mesa de dos jugadores que enfrentó al campeón mundial y a *BKG 9.8* y este último obtuvo la victoria.

La IA ha tenido periodos de auge y periodos de vacíos, denominados *inviernos y veranos de la IA*. Después de un periodo de vacío, en 1997 la supercomputadora ***Deep Blue*** vence al ajedrez al campeón mundial Gary Kasparov. En las últimas décadas, la IA ha seguido sumando más éxitos:

- 2005: un vehículo robot ganó una competición por recorrer más de 212 kilómetros sin la ayuda de ninguna persona.
- 2011: *Watson*, el ordenador creado por International Business Machines (IBM), vence a los campeones del concurso televisivo estadounidense Jeopardi.
- 2012: el superordenador creado por *Google* es capaz de identificar gatos y caras y cuerpos humanos.
- 2014: un *bot* llamado Eugene superó el *Test de Turing* al conseguir engañar a los jueces haciéndoles creer que era un niño ucraniano de 13 años.
- 2015: *AlphaGo* es la primera máquina en ganar a un jugador profesional de Go en un tablero de 19x19.
- 2017: *Libratus* vence al póquer a sus oponentes humanos.
- Futuro: el científico Raymond Kurzweil afirmó en 2005 que para el año 2045 las máquinas van a tener un nivel de inteligencia superior a la de un humano.

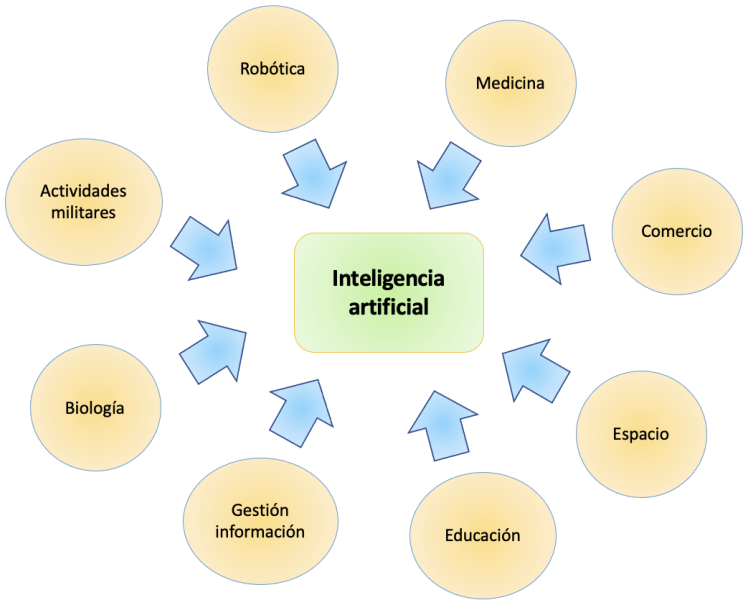

Figura 2.6: Área de aplicación de la Inteligencia Artificial

La Figura 2.6 muestra un resumen de las áreas de aplicación de la IA. Como se puede observar, tiene áreas de aplicación en la **robótica** (control de motores, visión, planificación), en la **ingeniería** (sistemas inteligentes de control, sistemas integrados de ventas), en la **medicina** (mejorar los diagnósticos médicos, diseño de prótesis), en el **comercio** (comercio electrónico), en el **espacio** (robots autónomos), en la **educación** (sistemas de tutores inteligentes), en la **gestión de la información** (minería de datos, filtrado de correo) y muchas más.

El AA es un subconjunto de la IA que incluye técnicas estadísticas que permiten que las personas entrenen a las máquinas y que estas adquieran la capacidad de aprender y de tomar ciertas decisiones en base a unos datos de entrada, como se expresa en [Vieira et al., 2020].

El DL es un subcampo del AA [Shinde and Shah, 2018] basado en redes neuronales artificiales. Estas redes neuronales artificiales están formadas por varias capas de entrada, de salida y ocultas. Así, cada capa transforma los datos de entrada en información que usa la siguiente capa para predecir su salida. Esto hace que la máquina pueda *aprender* de su propio procesamiento de datos.

La relación que existe entre la IA, el AA y el DL se puede observar en la Figura 2.7.

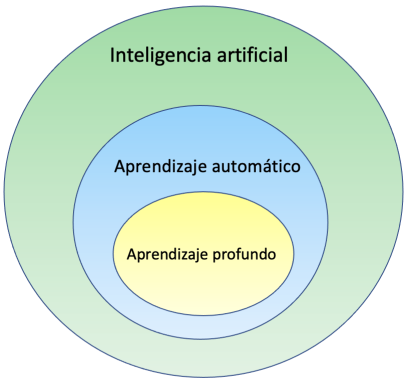

Figura 2.7: Relación entre la IA, AA y DL

En la Figura 2.8 se puede ver que el AA se puede clasificar según el tipo de aprendizaje, explicados en los apartados siguientes.

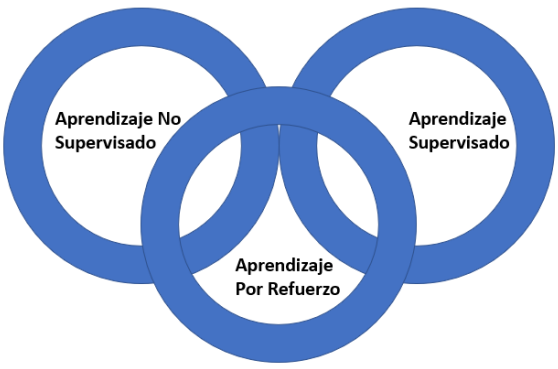

Figura 2.8: Tipos de aprendizaje

Los algoritmos de AA más comunes utilizados en resolución de problemas son los explicados en los apartados siguientes y se pueden clasificar de la manera en la que se presenta en la Figura 2.9:

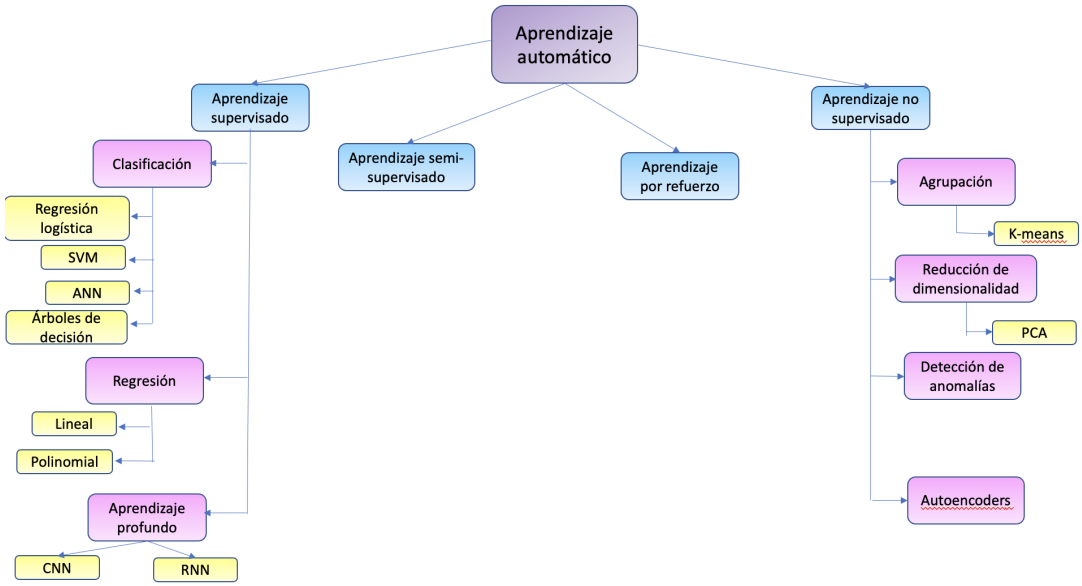

Figura 2.9: Clasificación Aprendizaje Automático

2.4.1. Aprendizaje supervisado

En el aprendizaje supervisado, al algoritmo se le proporciona un conjunto de datos etiquetados que representan las salidas deseadas, como se menciona en el trabajo [Muhammad and Yan, 2015]. Así, este tipo de aprendizaje permite que los algoritmos *aprendan* de los datos de entrenamiento y puedan realizar predicciones sobre entradas desconocidas para obtener la salida correcta. Los datos de entrenamientos tendrán que estar etiquetados correctamente para que el algoritmo pueda realizar predicciones de manera satisfactoria.

Los algoritmos más utilizados del aprendizaje supervisado son:

- *K-Nearest Neighbors (KNN)*
- *Support Vector Machine (SVM)*
- Regresión logística
- Regresión lineal
- Árboles de decisión
- *Random Forest (RF)*
- Algoritmos bayesianos
- Redes neuronales
- *DL*

Algunas de la industrias donde se puede aplicar el aprendizaje supervisado y los usos que se le puede dar son los siguientes: **bioinformática** (desarrollo de nuevas vacunas y fármacos), **reconocimiento de voz** (asistentes virtuales), **seguridad** (detección de *spam*, detección de *links* maliciosos), **medicina** (diagnósticos médicos), **comercio** (predicción de migración de clientes, detección de fraude), etc.

2.4.2. Aprendizaje no supervisado

Los algoritmos de aprendizaje no supervisado reciben los datos sin etiquetas y es el propio algoritmo el que organiza los datos de tal manera que pueda encontrar correlaciones y patrones dentro del conjunto de datos, como se indica en [Mahesh, 2020]. Como consecuencia, los datos están organizados en grupos o *clusters*.

Algoritmos más utilizados:

- *K-means*
- *Principal Component Analysis (PCA)*
- Detección de anomalías
- *Autoencoders*
- *Clusterings*

Los usos más importantes que se le da al AA no supervisado son: compresión de datos, detección de anomalías, segmentación de clientes, agrupación de patrones de Ácido desoxirribonucleico (ADN), recomendadores (recomendadores de sitios, películas, series), Internet of Things (IoT) (predicción de fallos en equipos informáticos), entre otros.

2.4.3. Aprendizaje por refuerzo

[Nian et al., 2020] indica que los algoritmos de aprendizaje por refuerzo utilizan un sistema de recompensas y junto con las acciones, parámetros y valores finales que se les proporcionan, exploran diferentes opciones a base de prueba y error para encontrar el óptimo.

Este tipo de aprendizaje se puede utilizar en **robótica** (brazos robóticos), a nivel **financiero** (para constituir una cartera de inversión), en la creación de **webs personalizadas**, en los sistemas de navegación de drones, coches ó aviones y también en los **videojuegos**.

El algoritmo más utilizado es ***Q-Learning***.

Una ejemplificación del funcionamiento de los algoritmos de este tipo se puede observar en la Figura 2.10. La interacción del agente con su entorno permite su entrenamiento y en función de las acciones que realiza y del estado del entorno en un momento dado, recibe recompensas o penalizaciones.

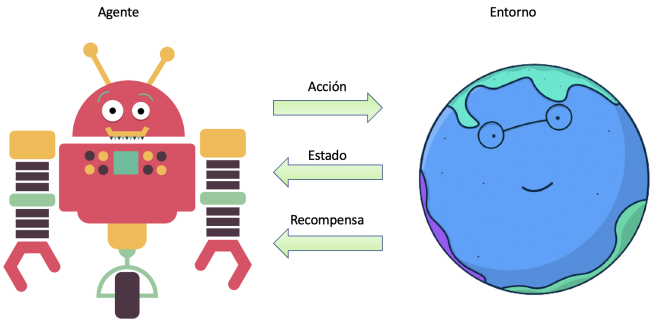

Figura 2.10: Aprendizaje por refuerzo

2.4.4. Aprendizaje semi-supervisado

En el aprendizaje semi-supervisado los algoritmos utilizan datos de entrenamiento etiquetados y no etiquetados siendo una combinación del aprendizaje supervisado y el aprendizaje no supervisado, como se menciona en [Reddy et al., 2018]. Los datos etiquetados del conjunto de entrada tienen un porcentaje bastante **mayor** al porcentaje de los datos no etiquetados, que suele ser muy bajo.

El **algoritmo de propagación de etiquetas** es el más conocido del aprendizaje semi-supervisado.

2.4.5. Algoritmos de clasificación

Los algoritmos de clasificación se emplean cuando el resultado o la variable objetivo es una etiqueta discreta, es decir, que la solución al problema planteado se encuentra en un conjunto finito de resultados posibles. Tienen como objetivo la clasificación de los datos del problema en clases preestablecidas ([Umadevi and Marseline, 2017]). La clasificación puede ser **binaria** (los datos se dividen en 2 categorías) o **multiclase** (los datos se dividen en múltiples clases).

Para poder evaluar la calidad de las predicciones de los modelos construidos, la **matriz de confusión** es una de las herramientas más importantes. Tal y como se puede comprobar en la Figura 2.11, siendo esta la representación de una clasificación binaria, las columnas representan el número de predicciones del modelo por cada clase (0 y 1), mientras que cada fila representa a las muestras reales por clase.

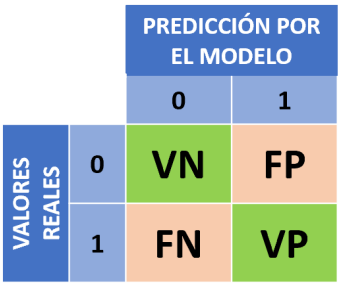

Figura 2.11: Matriz de confusión binaria

De esta forma, podemos tener un conteo del número de Verdaderos Positivos (VP), Verdaderos Negativos (VN), Falsos Positivos (FP) y Falsos Negativos (FN).

En el caso de una clasificación multiclase, se cuenta con una matriz de dimensión NxN, siendo N el número de clases a clasificar. En ambos casos, el objetivo para contar con un buen modelo clasificador es que los resultados se concentren en la **diagonal principal** de la matriz (VN y VP en la Figura 2.11).

[Umadevi and Marseline, 2017] afirma que una de las métricas para medir la calidad de un modelo de clasificación es la **precisión**. La precisión representa la proporción entre los VP y el número total de positivos predichos. Cuanto mayor es la precisión, mejor es el modelo de clasificación. Se usa la siguiente fórmula para calcular la precisión:

$$precisión = \frac{VP}{VP + FP} \tag{2.1}$$

Otra métrica es la **exhaustividad**, proveniente del término inglés **recall**, la cual informa sobre la cantidad de datos que el modelo es capaz de identificar. Por ejemplo, en este proyecto una exhaustividad de 0,4 representaría que únicamente es capaz de identificar, como tal, el 40 % de las muestras de una aplicación. Para calcularla, se usa la siguiente fórmula:

$$exhaustividad = \frac{VP}{VP + FN} \tag{2.2}$$

Además, existe una métrica que combina las dos anteriores (precisión y exhaustividad) en un único valor: el valor F1 o ***F1-score***. Este valor se calcula realizando la media armónica entre la precisión y la exhaustividad:

$$F1 = 2 \cdot \frac{precisión \cdot exhaustividad}{precisión + exhaustividad} \tag{2.3}$$

También se puede usar la *accuracy* o **exactitud**. Mide el porcentaje de casos que el modelo ha acertado de la siguiente forma:

$$exactitud = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.4}$$

A pesar de que se utiliza ampliamente, la exactitud de la clasificación es casi siempre inadecuada para la clasificación con conjuntos de datos desbalanceados (más muestras de una clase que de otra). La razón es que una alta exactitud es alcanzable por un modelo sin habilidad (mal entrenado) que sólo predice la clase mayoritaria [Ferri et al., 2009]. Por eso, es mejor tener en cuenta las otras tres métricas mencionadas anteriormente (precisión, exhaustividad y *F1-score*) para conocer la calidad del dato.

Los algoritmos más utilizados para la clasificación son:

- Regresión logística
- Árboles de decisión
- *Random forest*

- *XGBoost*
- Clasificador de Naïve Bayes
- Perceptrón Multicapa
- *KNN*
- *SVM*

2.4.6. Algoritmos de regresión

Los algoritmos de regresión se emplean cuando la variable respuesta o variable objetivo es continua y pretende encontrar relaciones y dependencias entre las variables, como se indica en [Nasteski, 2017]. Es decir, establece un método para la relación entre una variable dependiente escalar continua y una o más variables independientes. Las métricas más utilizadas para evaluar un modelo de regresión son:

- Mean Square Error (MSE) o **error cuadrático medio**: es la media de los errores al cuadrado. Esta métrica es muy sensible a los *outliers* (valores atípicos) y el cuadrado hace que los grandes errores se aumenten exponencialmente. Cuanto mayor es este valor, peor es el modelo.
- Root Mean Square Error (RMSE) o **raíz cuadrada del error cuadrático medio**: es la raíz cuadrada de los errores al cuadrado, por lo tanto la unidad de medida para RMSE es la misma que la del valor que se predice. El RMSE es más costoso que MSE.
- Mean Absolute Error (MAE) o **valor absoluto medio**: es la media del valor absoluto de los errores. Al calcular el MAE, la diferencia entre el valor predicho y el valor esperado siempre es positiva.

Existen diferentes métodos para los algoritmos de regresión:

- Regresión lineal (utiliza datos continuos)
- Árboles de decisión
- *SVM*
- *DL*

2.4.7. Algoritmos de agrupación

En los algoritmos de agrupación o ***clustering*** se agrupan los datos en *clusters* en base a patrones o correlaciones que se encuentran en el conjunto de datos, como se define en [Madhulatha, 2012].

Dado un conjunto de datos representado por puntos, el algoritmo agrupa cada punto en un *cluster*. Cada *cluster* contiene puntos con características similares, pero distintas a otros *clusters*. La Figura 2.12 muestra un ejemplo.

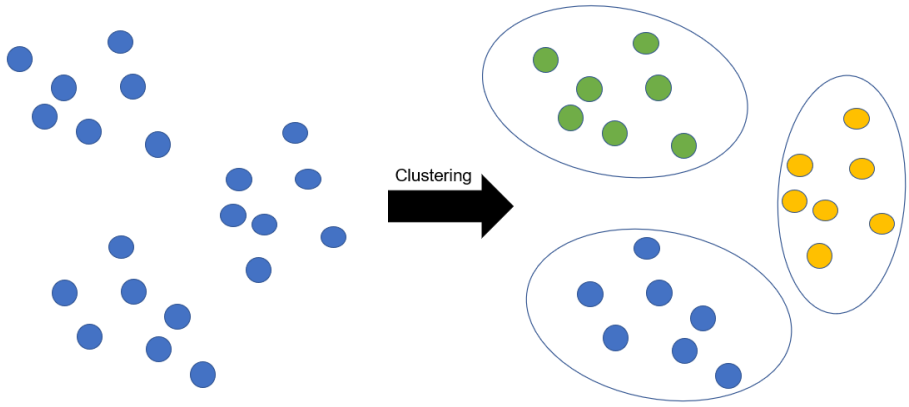

Figura 2.12: Algoritmos de agrupación

Esta técnica permite obtener información muy valiosa de los datos introducidos y por eso es muy común utilizarla en el análisis de datos estadísticos. Este tipo de algoritmos se pueden clasificar como **jerárquicos**, que pueden ser aglomerativos o divisivos y **no jerárquicos**, donde el número de grupos se sabe de antemano y un ejemplo muy conocido es el algoritmo *K-means*.

Los algoritmos de agrupación utilizan medidas de distancia. Algunas de las medidas más utilizadas son: la distancia **Manhattan** (igual a la suma de las distancias absolutas de cada variable) y la distancia **euclidiana** (la raíz cuadrada de la suma de los cuadrados de las distancias de cada variable).

Algunos de los algoritmos de esta categoría son:

- *K-means*
- *Mean-Shift*
- *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)*
- *Gaussian Mixture Models (GMM)*
- Agrupamiento jerárquico

2.4.8. Algoritmos bayesianos

Estos algoritmos están basados en el teorema de Bayes [Alzubi et al., 2018]. Partiendo de unos datos de entrada, este teorema calcula la **probabilidad condicional** de que ocurra un determinado suceso A, sabiendo con anterioridad, lo que ocurre con otro determinado suceso B.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.5}$$

En el entorno del AA, cuyo objetivo es la clasificación, es utilizado para calcular las probabilidades condicionales, como se menciona anteriormente. Es un algoritmo rápido

que establece una probabilidad independiente. Cuantos más datos se introduzcan, más preciso será el resultado.

Los clasificadores bayesianos son métodos simples de alta precisión, pero se debe de tener en cuenta que son apropiados cuando el supuesto de independencia condicional de clase es **válido**.

Por lo tanto, los algoritmos bayesianos se utilizan para predecir ocurrencias y unos ejemplos concretos de esto son:

- Detección de fraude con tarjetas de crédito
- Filtrado de correo *spam*
- Diagnóstico médico
- Predicción del tiempo
- Reconocimiento de emociones del habla
- Estimación de las emisiones de gas
- Análisis forense
- Reconocimiento óptico de caracteres

2.4.9. Algoritmos de reducción de dimensión

Los algoritmos de reducción de dimensión son algoritmos de AA que emplean técnicas para reducir el número de variables de entrada, restableciendo la varianza y manteniendo intacta la información relevante si es posible. Esto se consigue seleccionando un subconjunto de características óptimo, descartando aquellas características que no aportan información relevante, como se recoge en [Alzubi et al., 2018].

La **varianza** en el AA se refiere a que teniendo los conjuntos de entrenamiento y de test, no exista mucha diferencia entre las métricas obtenidas entre estos dos conjuntos, ya que si la varianza es alta, entonces es cuando se produce el sobreajuste (*overfitting*).

La varianza está relacionada con la complejidad de los modelos, ya que al aumentar la complejidad del modelo, aumenta la posibilidad de sobreajuste y por lo tanto aumenta también la varianza.

La **reducción de dimensión** se puede utilizar en el caso de que se quieran reducir los recursos necesarios, eliminar ruido de los datos y para poder interpretar mejor los resultados. La reducción de dimensionalidad se puede obtener seleccionando un conjunto de características del conjunto original, derivando características (creando nuevas características combinando algunas de las originales y sustituyéndolas) o agrupando las características en *clusters*.

Técnicas de la reducción de la dimensionalidad:

- **Lineales**
  - Análisis factorial
  - Análisis de componentes principales

  - Análisis discriminante lineal

- **No lineales**

  - Escala multidimensional
  - Mapeo de características isométricas
  - Mapas de *Hessien*

2.4.10. Algoritmos basados en árboles de decisión

Como su propio nombre indica, los algoritmos basados en árboles de decisión, usan una estructura en forma de árbol para resolver los problemas de clasificación o regresión como se puede observar en la Figura 2.13. Los algoritmos basados en árboles de decisión dividen los datos en nodos, creando bifurcaciones en base a los valores de las características y formando una estructura de árbol ([Mahesh, 2020]).

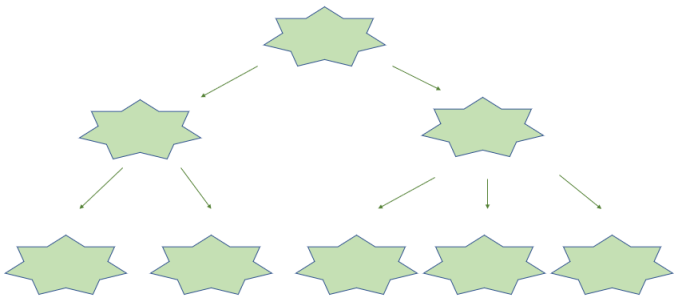

Figura 2.13: Árbol de decisión

Dentro de esta estructura, se pueden observar distintos tipos de nodo: el **nodo raíz** representa todas las muestras antes de ser divididas, el **nodo interno** representa cada característica y el **nodo hoja** es el resultado obtenido. La **rama** constituye la propia regla de decisión, y el nodo generado, es el nodo hoja.

Sobre esta estructura de árbol, se pueden realizar acciones de **división** (divide el conjunto del nodo padre en subconjuntos más pequeños) o **poda** (eliminación de nodos para reducir el tamaño de los árboles).

Los árboles de decisión pueden ser muy complejos y pueden crecer mucho pero hay una serie de parámetros que son capaces de controlar esto. Algunos de ellos son:

- ***max_depth***: este parámetro controla la profundidad del árbol, que es el recorrido más largo desde el nodo raíz hasta el último nodo hoja.
- ***min_samples_split***: controla el número mínimo de observaciones que tiene que haber en un nodo para poder hacer la división y en caso de que este número sea mayor al número de observaciones, no se divide, limitando el sobreajuste.
- ***min_samples_leaf***: controla el número mínimo de observaciones que tiene que tener un nodo para poder ser considerado un nodo hoja.
- ***max_leaf_nodes***: controla el número máximo de nodos hoja.

- ***min_impurity_split***: controla el mínimo de impureza para poder dividir un nodo.
- ***criterion***: es el criterio que se sigue para decidir la división de los datos. Algunos de los algoritmos más comunes para la selección son los siguientes. Los tres primeros son usados para problemas de clasificación con variables objetivo categóricas y el último para problemas de regresión.
  - **Índice de *Gini*** o ***gini***:

    Mide la frecuencia con la que un elemento elegido al azar es etiquetado incorrectamente. Cuanto más bajo sea, mejor.
  - ***Chi Cuadrado***:

    Mide a partir de la suma de los cuadrados de las diferencias estandarizadas entre las frecuencias observadas y esperadas de la variable objetivo.
  - **Ganancia de información**:

    Mide cómo de bueno es un atributo separando los datos de entrenamiento, de acuerdo con el valor esperado.
  - **Reducción en la varianza:** Se usa la fórmula estándar de la varianza para dividir. La división con la varianza más baja se elige para dividir la población.

Los métodos basados en árboles de decisión son modelos predictivos que tienen una precisión alta, una gran facilidad a la hora de interpretar los resultados y una gran estabilidad o tolerancia frente al ruido introducido por variables no relevantes.

Además, se pueden utilizar tanto en problemas de clasificación, como en problemas de regresión. En los problemas de clasificación, los nodos hoja representan las etiquetas de las clases y en el caso de los problemas de regresión, los nodos hojas son intervalos de valores. Todas la observaciones que están en el mismo intervalo predicen el mismo valor.

Ejemplo de árboles de decisión:

- *ID3*
- *C4.5*
- *C5.0*
- *Classification and Regression Trees (CART)*

2.4.11. Algoritmo de Random Forest

Random Forest es un algoritmo de AA que puede enfocarse tanto en tareas de regresión como de clasificación. Es un algoritmo de aprendizaje que usa varios modelos que se han creado para generar un modelo entre ellos que sea mejor.

Por lo tanto, en este algoritmo se ejecutan varios árboles de decisión sobre el conjunto de datos y cuando obtiene la predicción de cada uno de estos árboles, selecciona la mejor solución. En la Figura 2.14 se puede observar la estructura.

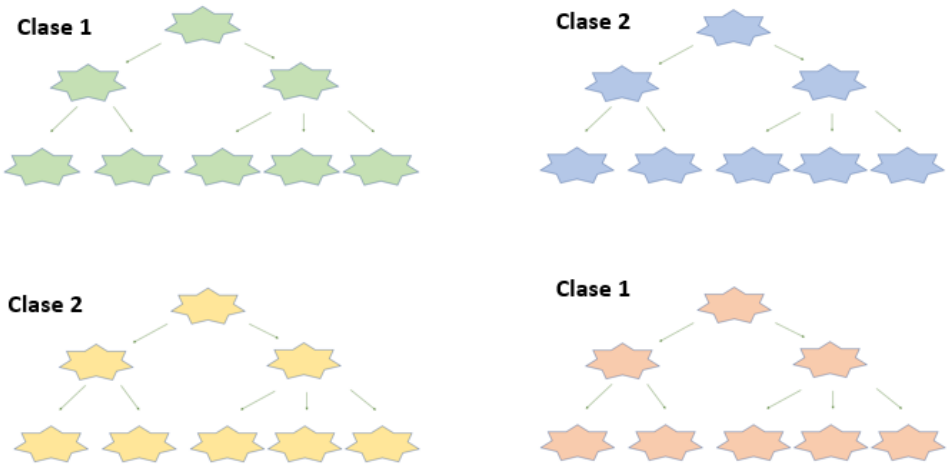

Figura 2.14: Random Forest

Dicho de otra manera, el algoritmo de RF es un ejemplo de ensamblados de *Bagging*. Un ensamblado utiliza una combinación de algoritmos débiles (es un modelo que tiene una precisión muy baja) y combina las salidas de estos para obtener un modelo robusto, como se explica en [Mahesh, 2020].

Una manera de realizar esta combinación es el *Bagging, Boosting* o *Stacking*. Esto consiste en entrenar modelos de forma independiente y después combinar los resultados, que en el caso de clasificación es la **moda** (el voto mayoritario de cada algoritmo) y en el de regresión la **media** de los resultados.

Una de las mayores **ventajas** de este algoritmo es el poder que tiene para manejar una gran cantidad de datos. También reduce el riesgo de sobreajuste al hacer el promedio de los resultados, y es estable frente a nuevos valores. Sin embargo, también tiene **desventajas** como la dificultad de interpretar los resultados, la cantidad de tiempo de entrenamiento, el coste y la baja eficacia con conjuntos de datos pequeños.

Los hiperparámetros más importantes de este algoritmo son:

- ***n_estimators***: número de árboles que se van a generar.
- ***max_features***: número máximo de características seleccionadas por cada árbol.
- ***n_jobs***: número de *cores* de la *Central Processing Unit (CPU)* (si se dispone de varios) que se van a utilizar para el entrenamiento del modelo.

Además, también presenta los hiperparámtros de los árboles de decisión mencionados en la Sección 2.4.10 como son *max_depth, min_samples_split, min_samples_leaf* y *max_leaf_nodes*.

2.4.12. Algoritmo XGBoost

El **Extreme Gradient Boosting (XGBoost)** es uno de los algoritmos basados en árboles más avanzados de la actualidad. Una evolución de este tipo de algoritmos se puede observar en la Figura 2.15.

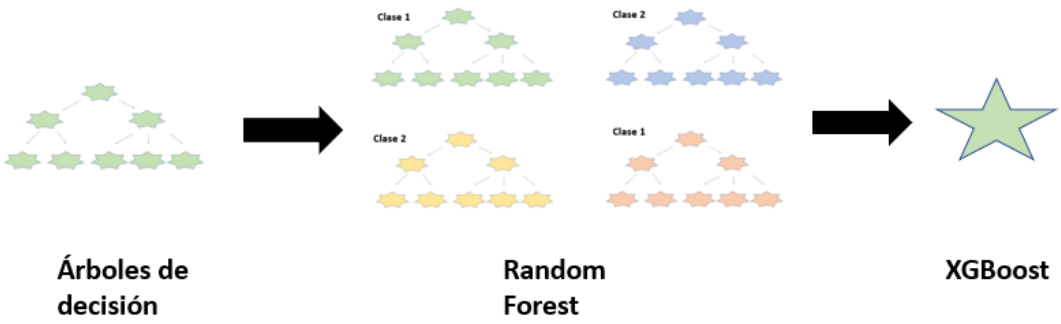

Figura 2.15: Evolución algoritmos basados en árboles de decisión

Este algoritmo consiste principalmente en un ensamblado de árboles de decisión en forma de secuencia. Es decir, los árboles que se van añadiendo en la secuencia, analizan a los árboles añadidos previamente para corregir los errores con el objetivo de regularizarlos. Los resultados de los árboles de decisión se impulsan debido al procesamiento secuencial de los datos, con una función de pérdida que minimiza el error en cada iteración. Se llega al concepto de **gradiente descendente** cuando ya no se pueden corregir más.

Este algoritmo también tiene **ventajas** como la de poder manejar una gran cantidad de datos, así como la gran velocidad con la que se realiza y la precisión de los resultados. Además, puede trabajar con valores atípicos, el procesamiento se puede realizar en paralelo y muchas más. Algunas de estas ventajas han sido mencionadas en [Ma et al., 2018]. Sin embargo, esto tiene la consecuencia de que se consumen muchos recursos computacionales y hay que ajustar los hiperparámetros para minimizar el error de precisión.

El algoritmo **XGBoost** tiene 3 tipos de parámetros que hay que establecer antes de usarlo:

- **Parámetros de refuerzo**
  - *eta*: reducción del peso de cada paso, para mejorar la robustez del modelo.
  - *gamma*: reducción mínima de pérdidas para poder realizar una división de un nodo hoja.
  - *max_depth*: profundidad máxima del árbol.
  - *min_child_weight*: suma de peso mínimo de muestra de los nodos hijos.
  - *subsample*: proporción de submuestras de las instancias de entrenamiento.
- **Parámetros generales**
  - *silent*: este parámetro mantiene sus valores por defecto como 0 que se utiliza para imprimir mensajes de ejecución y se necesita especificar explícitamente el valor 1 para el modo silencioso.
  - *num_pbuffer*: valor actualizado automáticamente por el algoritmo.
  - *num_feature*: valor actualizado automáticamente por el algoritmo.
- **Parámetros de aprendizaje**
  - *objective*: define la función de perdida que hay que minimizar (*RMSE*, *MAE*, entre otras).
  - *eval_metric*: métricas de evaluación de las predicciones.

2.4.13. Algoritmos de aprendizaje profundo

Como se ha observado anteriormente, gracias al AA las máquinas son capaces de aprender la relación oculta existente entre los datos de entrada.

El DL es un tipo de aprendizaje compuesto por varias capas o redes neuronales, capaz de extraer las características más relevantes de forma automática. Por lo tanto, en AA los algoritmos aprenden directamente de las características extraídas de forma manual, mientras que en DL los algoritmos son capaces de extraer las características más importantes de forma automática, como se puede observar en la Figura 2.16:

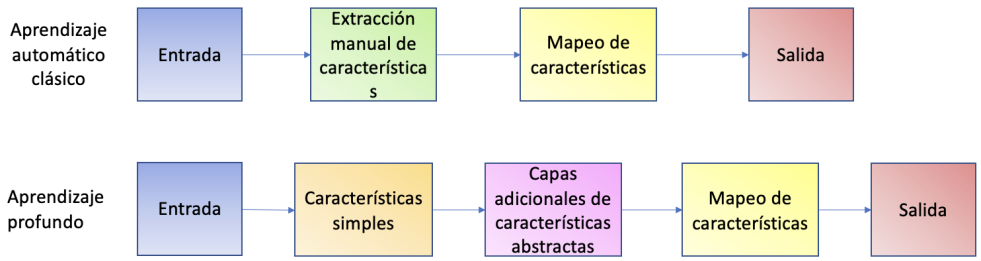

Figura 2.16: Estructura de una red neuronal simple

Los algoritmos de DL utilizan una estructura compuesta por varias capas o niveles llamada red neuronal artificial. En DL el modelo está basado en capas y cada capa está compuesta por un número de nodos llamados neuronas. Cada neurona tiene un peso según su importancia y un umbral asociados y recibe los valores obtenidos de otras neuronas.

Si la salida de una neurona está por encima de un cierto umbral entonces esa neurona se activa y envía datos a la siguiente capa, en caso contrario no se activa y no se pasan datos a la siguiente capa.

Las múltiples capas de redes neuronales procesan los datos y en cada capa se crea un representación cada vez mas simplificada de los datos y el algoritmo aprende progresivamente en cada una de estas capas. [Shrestha and Mahmood, 2019] afirma que este tipo de algoritmos pueden trabajar con una gran cantidad de datos.

Al usar varias capas y por lo tanto diferentes niveles de profundidad, el DL hace que el sistema aprenda de elementos complejos.

Los datos entran por la primera capa donde hay neuronas que se activan según los valores obtenidos tras la función de activación, luego se pasan los datos a las capas intermedias donde se procesan para que posteriormente se devuelva el resultado por la capa de salida.

La estructura de una red neuronal simple se detalla en la Figura 2.17.

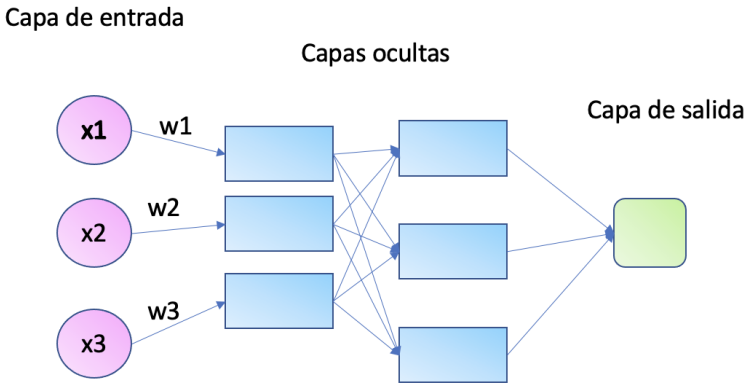

Figura 2.17: Estructura de una red neuronal simple

2.4.13.1. Redes neuronales

Las redes neuronales están formadas con elementos de procesamiento simples para procesar información. Las redes neuronales son capaces de aprender relaciones complejos de los datos de entrada (*input*) y clasificarlos. Para ello, en la etapa de entrenamiento se le aporta las variables de entrada y las etiquetas correspondientes.

El proceso de entrenamiento consiste en ir ajustando los pesos para lograr la salida correcta. En este proceso se extraen las características haciendo que la entrada a la red neuronal se represente de una forma más compacta.

La evolución y aparición de los distintos tipos de redes neuronales se muestra en la Figura 2.18:

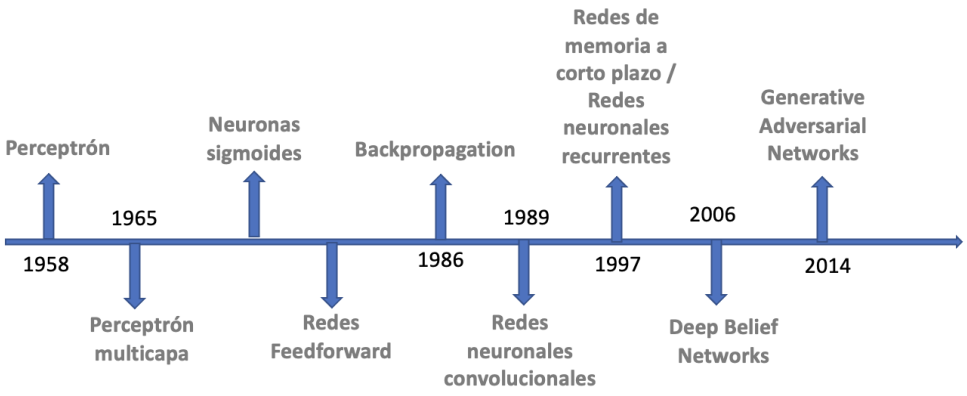

Figura 2.18: Evolución redes neuronales

2.4.13.2. Funciones de activación

Dados unos datos de entrada, la función de activación es capaz de predecir la salida de la neurona. Cada una de las capas que forman la red neuronal tiene una función de activación.

En cuanto a las funciones de activación más usadas, se pueden diferenciar 3 tipos, entre otros:

Función sigmoidal

La función sigmoidal tiene la característica principal del problema de la saturación. Se satura a los extremos del rango en el que se mapea la entrada. Es decir, se toman los valores de entrada y se mapean a la salida con valores entre 0 y 1, como se puede ver en la Figura 2.19. Si la entrada a la que se enfrenta es muy baja o es muy alta, se satura a los valores de 0 y 1, respectivamente [act, 2018]. En la zona rodeada en la figura se puede ver la saturación producida mencionada anteriormente.

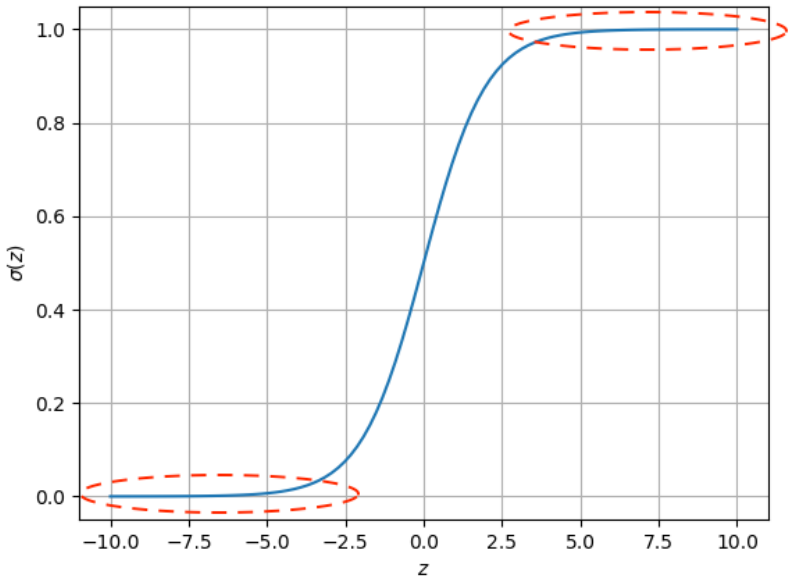

Figura 2.19: Función sigmoidal

Función tangente hiperbólica La principal diferencia entre la función Función tangente hiperbólica (tanh) y la función sigmoidal es que el rango en el que se mapean los valores de entrada se amplía entre -1 y 1, como se puede ver en la Figura 2.20. Sin embargo, sigue teniendo el problema de la función sigmoidal [act, 2018]. En la zona rodeada en la figura se puede ver la saturación producida mencionada anteriormente.

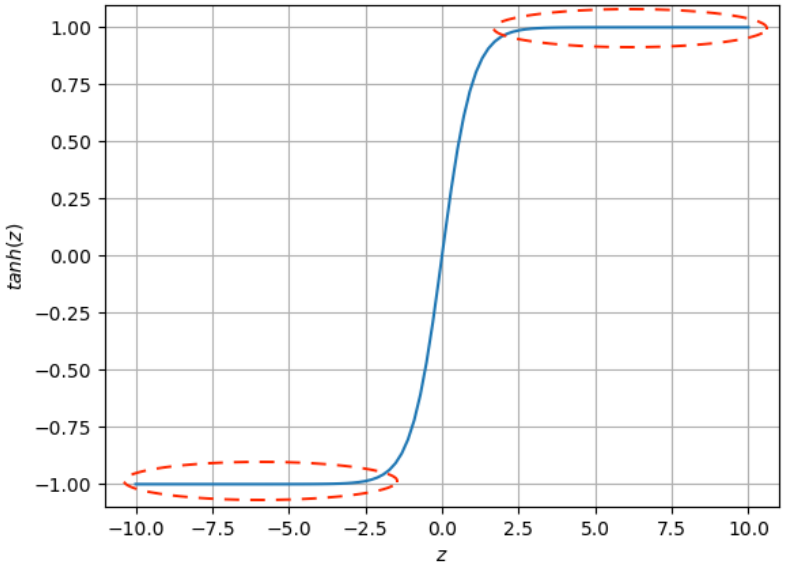

Figura 2.20: Función tanh

Función ReLU Esta función de activación, en cambio, evita el problema de saturación mencionado en las dos anteriores [act, 2018]. Como se puede ver en la Figura 2.21, los datos que entran como entrada que son positivos, no sufren cambios, sin embargo, aquellos datos de entrada que sean negativos, provoca que la salida sea 0.

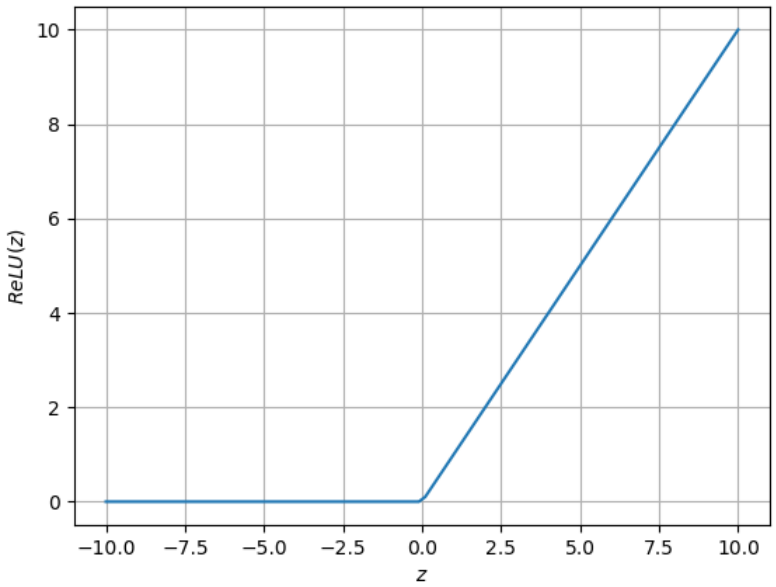

Figura 2.21: Función ReLU

Los ***autoencoders***, las CNN y las DGCNN son tres de las redes neuronales profundas más importantes que existen.

2.4.13.3. Autocodificadores

Los autocodifcadores, o su término anglosajón ***autoencoders***, son una arquitectura de DL no supervisada cuyo objetivo es reconstruir los datos de entrada. Durante el entrenamiento no se establece la clase de la entrada ya que el propósito es reconstruir la salida siendo esta el mismo dato de entrada. Lo que quiere decir que los *autoencoders* son algoritmos que extraen las características de los datos de entrada sin la necesidad de que estos sean etiquetados, como indica [Essien and Giannetti, 2020]. El *autoencoder* intenta aprender una representación en miniatura del *dataset*. Es decir, se entrena la red neuronal para que aprenda a generar a la salida el mismo dato de entrada. Se usa para la extracción automática de características.

Los *autoencoders* tienen tres elementos como se ve en la Figura 2.22:

1. ***Encoder*** Este elemento compacta los datos de entrada hasta cierto punto.
2. **Bottleneck** Este elemento separa el elemento anterior *encoder* con el elemento siguiente *decoder*. Es el elemento resultante de la compactación del *encoder* y a partir del cuál el *decoder* construye la salida. Esta región también es denominada espacio latente.

   La representación compacta que se obtiene de este elemento es el resultado del entrenamiento de los datos, es decir, es en este elemento donde la red aprende a extraer los datos relevantes del *dataset* de entrada.
3. **Decoder** Este elemento es el elemento final del *autoencoder*. Saca la salida reconstruyendo los datos compactados que le han llegado a *bottleneck* del *encoder*.

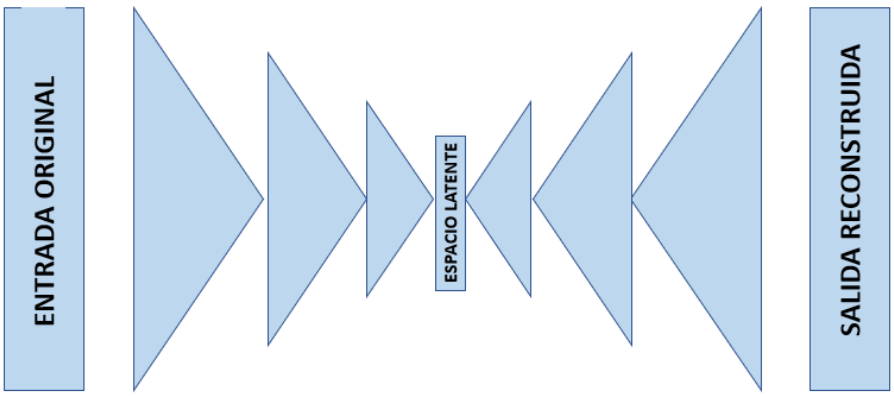

Figura 2.22: Autoencoder

2.4.13.4. Redes Neuronales Convolucionales

Las CNN son un tipo de red neuronal artificial basadas en el aprendizaje supervisado. Extraen las características del *dataset* que recibe procesando sus capas. Necesita un gran *dataset* como en el caso anterior del que pueda obtener las características. Por lo tanto, las CNN también están formadas por varias capas.

Como su propio nombre indica, su funcionamiento se basa en las llamadas convoluciones. Esto hace que se tomen los bytes con características en común y se opere

con el producto escalar contra una pequeña matriz llamada *kernel* que posteriormente generará otra matriz de salida que es la nueva capa de neuronas ocultas.

Algunos de los componentes de estas redes son los *kernels* o *filtros* y *Max-pooling* [CNN, 2018].

- ***kernels* o *filtros*** Son las matrices contra las que se opera matemáticamente. Este *filtro* recorre todas las neuronas de entrada por filas y columnas y genera una nueva matriz de salida, que se convierte en la nueva capa de neuronas ocultas. Lo que se hace realmente es aplicar un conjunto de *filtros* (*kernels*) que es un sólo *kernel*.
- ***Max-pooling*** Es un tipo de proceso de *subsampling* en el que se reduce el tamaño de las imágenes filtradas y se conservan las características más importantes que detectó cada filtro.

2.4.13.5. Redes Neuronales Convolucionales Gráficas

Las redes neuronales convolucionales gráficas funcionan como las redes neuronales convolucionales básicas pero con la característica de que las neuronas de entrada son **grafos**. Al utilizar grafos para representar los datos, la información puede codificarse para modelar las relaciones entre las características y proporcionar una mejor visión de los datos ([Zhang et al., 2019]).

Las convoluciones explicadas a continuación que realizan este tipo de redes permiten que, mediante unos *filtros* o *kernels*, las CNN aprendan características de las celdas vecinas, que son a su vez cada vez más complejas. Dentro de la misma capa, se usa el mismo *filtro* en toda la imagen, esto se conoce como *peso compartido*.

Los grafos que sirven como entrada a la red están formados por nodos y aristas como se ve en la Figura 2.23. Las aristas unen los nodos entre sí formando el grafo. En cada convolución, se aplica una operación (producto escalar) determinada a las neuronas de entrada para generar una matriz como nueva salida.

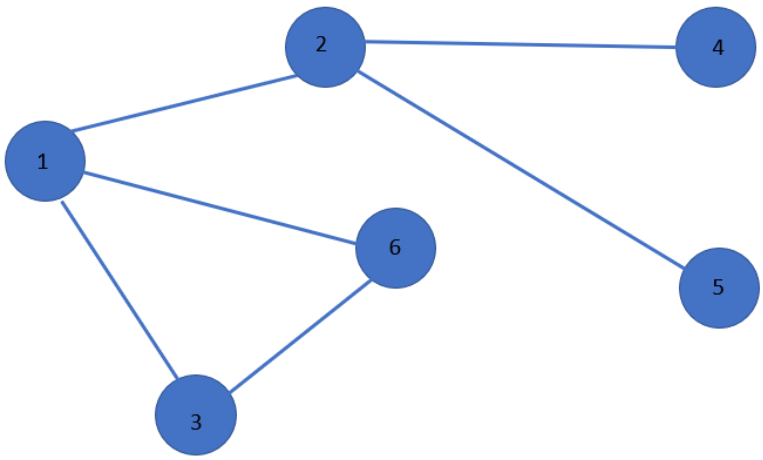

Figura 2.23: DGCNN

Dependiendo del problema que se quiera resolver o los datos que se van a manejar, los

filtros o *kernels* varían. Como salida, se obtiene el mismo número de matrices como *filtros* se hayan usado.

La diferencia más significativa es que las DGCNN son una versión de las CNN donde el número de conexiones de los nodos varía y los nodos no están ordenados. En las CNN, las conexiones tratan datos regulares y estructurados.

Las DGCNN se pueden clasificar en dos algoritmos, redes convolucionales de gráficos espaciales y redes convolucionales de gráficos espectrales.

Las **redes neuronales convolucionales gráficas** son aquellas que trabajan con grafos. Un nodo de la red analiza la información que contienen los nodos vecinos obteniendo así como resultado las características del grafo de entrada. La estructura de estas redes neuronales convolucionales gráficas son similares a las CNN pero recibiendo grafos como entrada. Este tipo de redes neuronales convolucionales gráficas se clasifican en 3 tipos:

- Red neuronal de gráficos recurrentes
- Red convolucional espacial
- Red convolucional espectral

Capítulo 3

Estado del Arte

El análisis de tráfico tiene muchas finalidades, entre ellas evaluar el rendimiento y la seguridad de las operaciones y la gestión de la red, son algunos de los objetivos, como se menciona en [Alqudah and Yaseen, 2020]. En este artículo se presentan las distintas técnicas utilizadas en diversos trabajos para realizar estas tareas. Los trabajos que recoge [Alqudah and Yaseen, 2020], las técnicas que se usan y los objetivos se detallan en la Tabla 3.1.

En la tabla mencionada, se describen diversos trabajos que se han realizado. Existen trabajos relacionados cuyo objetivo es clasificar el tráfico en la red, como [Bujlow et al., 2012] y [SE, 2013]. Además se puede hacer la comparación entre ellos debido a que se utilizan diferentes técnicas. Mientras que en [Bujlow et al., 2012] se utiliza el algoritmo *C5.0*, en el trabajo de [SE, 2013] las técnicas aplicadas son los algoritmos de árboles de decisión y bayesianos.

En la Tabla 3.1 también se puede observar trabajos que se basan en el aprendizaje supervisado, como pueden ser [Suthaharan, 2014] y [Bartos et al., 2016]. Sin embargo, el enfoque que le dan los autores de los artículos es diferente. Mientras en [Bartos et al., 2016] el objetivo es detectar amenazas de seguridad, en [Suthaharan, 2014] el objetivo es la clasificación del tráfico de intrusión en la red mediante el aprendizaje de las características de la red.

Gran parte de los trabajos que se detallan en la Tabla 3.1 basan su técnica en la AA y tienen distintos objetivos. [Sommer and Paxson, 2010], [Blowers and Williams, 2014] y [Laskov et al., 2005] son un ejemplo, aunque tienen en común la identificación de anomalías o intrusos en la red.

[Tahaei et al., 2020] y [Cao et al., 2014] presentan una taxonomía de la clasificación actual de tráfico de red. Los métodos mencionados en estos trabajos son:

- **Métodos basados en puertos**

  Este método utiliza la información de las cabeceras de TCP y UDP para extraer el número de puerto. Después de la extracción del número de puerto, se compara con el número de puerto TCP/UDP asignado de la Internet Assigned Numbers Authority (IANA) para la clasificación del tráfico. Es un proceso fácil y rápido.

  Sin embargo, los puertos, la red, la Network Address Translation (NAT), el reenvío de puertos y el protocolo, entre otros, hacen que la precisión no sea tan alta. Sólo entre el 30 % y el 70 % del tráfico actual puede usar esta clasificación basada en

Tabla 3.1: Análisis de tráfico

	Técnica aplicada	Objetivo
[Sommer and Paxson, 2010]	AA	Los métodos Machine Learning (ML) se han aplicado a la detección de *spam* con mayor eficacia que a la detección de intrusos, ya que la detección de anomalías es la más adecuada para encontrar las diferentes formas de ataques conocidos
[Zhang et al., 2012]	Redes neuronales artificiales usando el algoritmo Voting Experts (VE)	Extraer las características del protocolo a partir de las palabras extraídas por el algoritmo VE
[Furno et al., 2017]	AA usando Exploratory Factor Analysis (EFA)	Análisis de la estructura espacial
[Mirsky et al., 2018]	Redes neuronales artificiales usando *Kitsune*	Detectar tráfico malicioso que entra y sale de la red
[Suthaharan, 2014]	AA usando aprendizaje supervisado	Clasificación del tráfico de intrusión en la red mediante el aprendizaje de las características de la red
[Blowers and Williams, 2014]	AA usando *clustering*	Detección de anomalías basadas en *clustering*
[Laskov et al., 2005]	AA, SVM, KNN, *K-Means*	Comparación del aprendizaje supervisado y no supervisado al detectar actividades maliciosas
[Mukkamala et al., 2002]	SVM	Descubrir patrones o características que describan los comportamientos de los usuarios para construir clasificadores que reconozcan las anomalías
[Zamani et al., 2009]	Algoritmo artificial inmune	Detectar inclusión en sistemas distribuidos
[Bujlow et al., 2012]	AA usando el algoritmo *C5.0*	Clasificar tráfico en la red
[SE, 2013]	AA usando los algoritmos de árboles de decisión y bayesianos	Clasificar tráfico en la red
[Bartos et al., 2016]	AA usando aprendizaje supervisado	Detectar amenazas de seguridad, tanto las conocidas como las no vistas anteriormente

puertos.

- **Métodos basados en carga**

  Basados en el análisis de la información que hay en la carga útil de paquetes de la capa de aplicación.

- **Inspección profunda de paquetes**

  Identifica las aplicaciones inspeccionando las cabeceras de los paquetes o incluso la carga útil y proporciona una alta precisión con bajas tasas de FN.

- **Clasificación estadística**

  Se basan en características estadísticas o de series temporales y emplean algoritmos de AA (es decir, teorema de Bayes, SVM, arboles de decisión)

- **Clasificación del comportamiento**

  Estos métodos utilizan parámetros independientes de la carga útil, como la longitud del paquete, el tiempo entre llegadas y la duración del flujo para eludir el problema

de la codificación de la carga útil y la privacidad del usuario.

[Papadogiannaki and Ioannidis, 2021] presenta una taxonomía para los trabajos de inspección del tráfico de red categorizada por caso de uso, técnica y objetivo como se puede observar en la Figura 3.1.

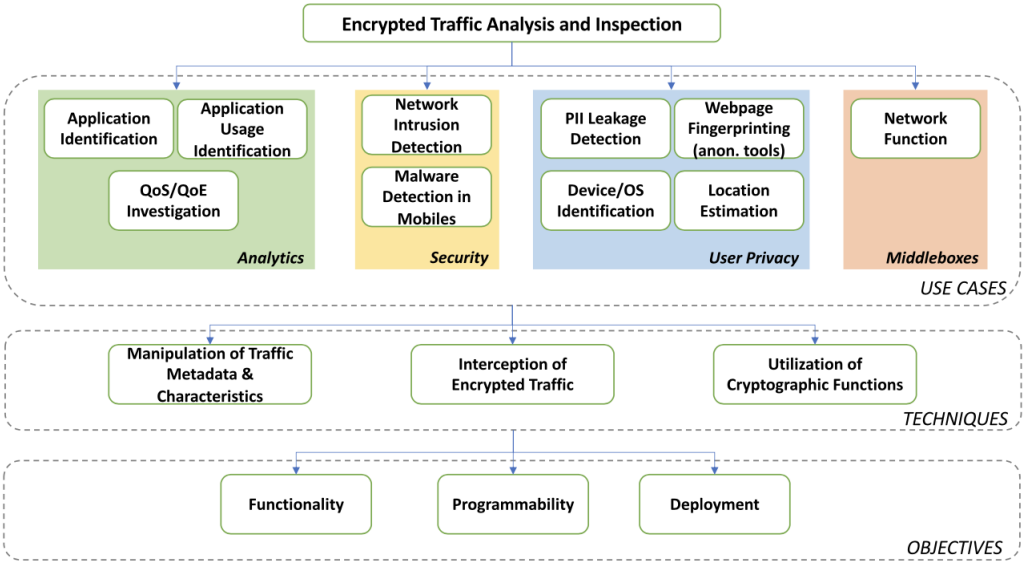

Figura 3.1: Análisis de tráfico

[Wang et al., 2019] explica también el DL y su aplicación en CNN y SAE. Aunque los enfoques clásicos de AA pueden resolver muchos problemas que los métodos basados en el puerto y en la carga útil, no pueden resolver, siguen teniendo algunas limitaciones, como la pérdida de tiempo, el coste el tiempo y el coste de la investigación y la actualización frecuente de las características.

Los estudios sobre la clasificación del tráfico móvil no solo ayudan a mejorar la asignación de recursos de la red móvil basada en servicios o aplicaciones, sino que también mejoran la seguridad de la red y las aplicaciones móviles.

[Wang et al., 2019] desarrolla que DL realiza una extracción de características sin intervención humana. Las técnicas de DL implican 3 pasos:

1. Se definen y diseñan las entradas del modelo.
2. Los modelos y los algoritmos se eligen deliberadamente en función de las características de los modelos y del objetivo del clasificador.
3. El clasificador DL se entrena para extraer automáticamente las características del tráfico y asociar las entradas con las corresspondientes etiquetas de clase.

Como el enfoque basado en DL ha resultado ser eficaz, hay trabajos que lo utilizan. Algunos de estos trabajos lo utilizan por separado o en combinación con otro método de AA. [Aceto et al., 2018] propone el DL para crear clasificadores de tráfico que extraen las características automáticamente.

Por otro lado, [Antonello et al., 2012] propone una estrategia que emplea 2 métodos: un SAE en conjunto con una CNN para integrar en una sola fase la extracción de características y la clasificación. En [Yang et al., 2018] y [Wang et al., 2020] también se emplean los *autoencoders* (simples o apilados) junto a las CNN. En [Yang et al., 2018] se obtiene una precisión del 97,9 % utilizando CNN y en [Wang et al., 2020] un *F1-score* del 98 % en los experimentos.

Otro trabajo que utiliza los *autoencoders* es [Aouedi et al., 2020] que implementa un SAE disperso acompañado de técnicas de *denoising* y *dropout* para la extracción de características evitando el problema del sobreajuste y consigue llegar a una precisión del 86,89 % en las pruebas realizadas.

Cuando hay más nodos en la capa oculta que entradas, la red corre el riesgo de aprender la llamada *función de identidad*, también llamada *función nula*, lo que significa que la salida es igual a la entrada, lo que hace que el *autoencoder* sea inútil. *Denoising Autoencoders* son aquellos que convierten aleatoriamente en cero algunos valores de entrada para eliminar este problema.

En [Radivilova et al., 2018] se presenta un análisis de los principales mecanismos de análisis del tráfico SSL/TLS.

Actualmente, existe una necesidad creciente de garantizar la privacidad de los ciudadanos frente a los ataques que vulneran la seguridad y privacidad de las personas y de las empresas.

Normalmente, se instala una pasarela o un grupo de pasarelas en el perímetro de una red corporativa o departamental. Realizan la interceptación y el descifrado de los datos dentro de la red de la empresa para protegerse de los ataques mediante SSL/TLS. [Radivilova et al., 2018]

Un Intrusion Detection System (IDS) puede detectar accesos no autorizados y generar alertas [ids, 2020], pero si el tráfico se transmite de forma cifrada, el IDS sólo puede realizar una verificación limitada basada en las cabeceras de los paquetes. Los IDS no realizan ninguna acción para bloquear estos accesos indebidos.

Cisco ha desarrollado una tecnología denominada Encrypted Traffic Analytics (ETA) que permite, basándose en la telemetría de red recibida de los equipos de red y en algoritmos de AA, clasificar el tráfico cifrado, separando el tráfico de red legítimo del tráfico malicioso. En [Radivilova et al., 2018] se explican dos mecanismos:

1. **Mecanismos básicos de detección de tráfico SSL/TLS**

   - **Re-Firma de Certificados** Se utiliza cuando no se tiene acceso a los certificados y claves privadas del servidor.
     - **La inspección basada en la re-firma de certificados**
       Se basa en un validador que tiene un certificado de Certification Authority (CA) de confianza que puede utilizarse para firmar certificados de servidor SSL que han sido interceptados y modificados (destinados a un servidor SSL externo).
     - **Certificados de servidor autofirmados**
       SSL *Visibility Appliance* puede utilizar el certificado de re-firma para procesar los certificados de servidor autofirmados o puede simplemente sustituir las claves en un certificado de servidor autofirmado, en cuyo caso

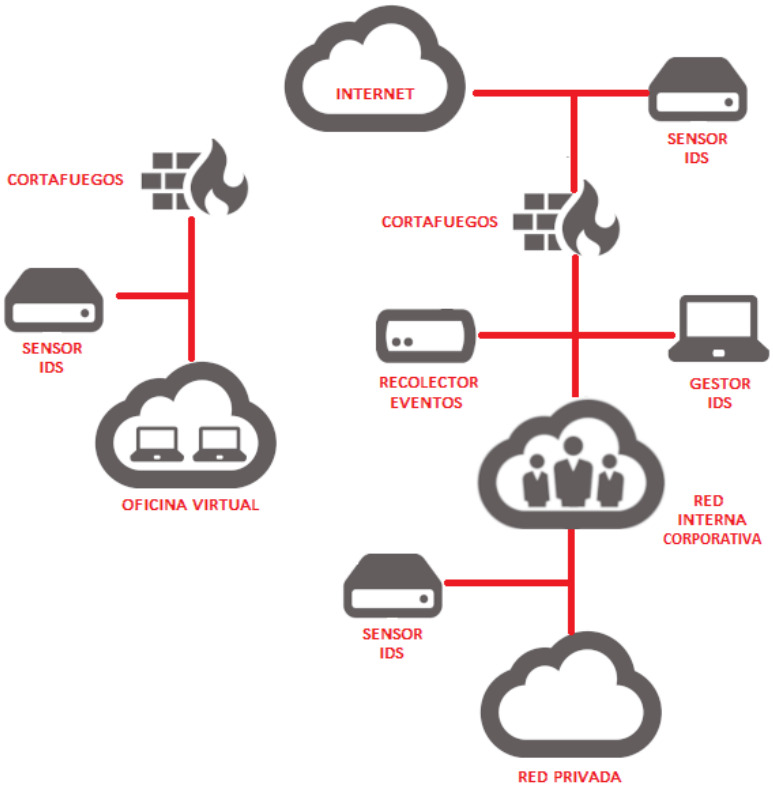

Figura 3.2: IDS

no se utiliza la CA de un dispositivo SSL/TLS y el certificado de servidor sigue siendo autofirmado.

- **Clave de servidor conocida**
  Se basa en un dispositivo *SSL Visibility Appliance* que tiene una copia de las claves privadas y los certificados de servidor, y se utiliza para los mensajes entrantes.
  - ***Cisco ETA*** Recupera cuatro elementos de datos básicos: una asignación de bytes, una función específica de TLS, una secuencia de longitudes y tiempos de paquetes y un paquete de datos de origen. Su arquitectura (*Application-Specific Integrated Circuit (ASIC)*) puede hacer esto sin ralentizar la red de datos.
  - ***Cisco Stealthwatch Enterprise*** Utiliza servidores *proxy*, telemetría de punto final, *NetFlow*, segmentación de tráfico, mecanismos de política y acceso y mucho más para establecer el comportamiento normal básico de los *hosts* y usuarios de una empresa. Soporta un mapa de riesgo global, identificando los servidores asociados a ataques que pueden ser utilizados como parte de un ataque en un futuro. En lugar de descifrar el tráfico, *Stealthwatch* utiliza algoritmos de AA para identificar patrones maliciosos en el tráfico cifrado

2. **Mecanismos básicos de desencriptado de tráfico SSL/TLS**

   1. **Realizar la inspección en el propio servidor** La forma más fácil de verificar el tráfico encriptado es utilizar un IDS basado en el *host* (Host

Intrusion Detection System (HIDS)) en el servidor, donde se descifra el tráfico perteneciente a este.

2. ***Proxy* de terminación SSL/TLS (*proxy* inverso)** Un *proxy* inverso es un servidor que actúa como intermediario entre los servidores *backend* y los clientes. El servidor de *proxy* inverso puede ser configurado para cifrar SSL/TLS, actuando como un *proxy* de terminación SSL/TLS.
3. **El IDS realiza el descifrado** Algunos IDS, como *Juniper IDP*, ofrecen la posibilidad de realizar el proceso de descifrado con una clave privada.
4. **Una herramienta independiente realiza el descifrado** *Viewssld* es una herramienta gratuita de código abierto que puede descifrar el tráfico SSL/TLS para IDS, utilizando el intercambio de claves RSA. *Symantec* utiliza la solución *Encrypted Traffic Management* para eliminar el ciego del tráfico cifrado. El componente clave de este conjunto de soluciones *SSL Visibility Appliance* es una herramienta de verificación, descifrado y gestión de SSL de alto rendimiento, que puede alcanzar los 9 *Gbps* de descifrado de SSL y es capaz de transmitir simultáneamente la información descifrada por varias herramientas de seguridad.

[Radivilova et al., 2018] concluye que SSL/TLS se ha convertido en un estándar universal para autenticar y cifrar mensajes entre clientes y servidores. Está ampliamente distribuido en las empresas y está creciendo rápidamente debido al rápido aumento de las aplicaciones móviles, en la nube y en la web. Sin embargo, SSL crea un riesgo de seguridad al introducir un "**punto ciego**", que aumenta el riesgo de que el *malware* y la actividad no autorizada entren en la organización.

La clasificación de tráfico encriptado está abordada en diferentes trabajos y se utilizan distintos métodos. En [Ji and Meng, 2020] se propone una representación de 2 redes basadas en la red convolucional de grafos para evitar un gran número de muestras etiquetadas, ya que este es uno de los inconvenientes de utilizar AA, que requiere un número alto de muestras correctamente etiquetadas.

Sin embargo, no siempre (en el caso de que sea aprendizaje profundo lo normal es que sea necesario un alto numero de ejemplos o muestras debido al alto numero de parámetros empleados por la red). Con esta representación consiguen una tasa de precisión de la clasificación del 97,35 %. Otro trabajo que emplea grafos y redes neuronales es [Zhang et al., 2018], DGCNN, donde el enfoque lee directamente los grafos que se pasan como entrada sin necesidad de preprocesamiento, junto con su clase, y el algoritmo de DL aprende a clasificarlos. DGCNN tiene 3 etapas secuenciales: las capas de convolución de los grafos (extraen las características de los vértices y define un orden coherente de estos), la capa *SortPooling* (ordena las características de los vértices en el orden definido anteriormente y además amplia el tensor de salida para que los grafos con distintos números de vértices unifiquen sus tamaños) y las capas densas y convolucionales tradicionales (hacen predicciones sobre las representaciones ordenadas de los grafos, aprendiendo patrones locales en la secuencia de nodos). Para los experimentos se han utilizado 5 datasets de datos bioinformáticos (MUTAG, PTC, NCI1, PROTEÍNAS y DD) y 3 datasets de datos de redes sociales (COLLAB, IMDB-B, IMDB-M) para medir la precisión de clasificación de grafos de DGCNN con 4 *kernels* de grafos: el *kernel* del subárbol Weisfeiler-Lehman (Weisfeiler-Lehman Subtree Kernel (WL)), el *kernel* de propagación (Propagation Kernel (PK)), el *kernel* de camino aleatorio (Random Walk kernel (RW)) y el marco del *kernel*

graphlet (Graphlet Kernel (GK)). DGCNN ha obtenido resultados más altos que los *kernels* de grafos casi con todos los *datasets*.

El método de clasificación de tráfico basado en AA requiere ingenierías de características y en general un exhaustivo preprocesamiento y [Song et al., 2019] para evitar esto utiliza un método basado en CNN que representa los datos de tráfico como vectores y a continuación utiliza CNN para extraer las características más significativas para la clasificación de tráfico.

No solo el uso de tráfico encriptado dificulta la tarea de clasificación sino también la gran cantidad de aplicaciones disponibles para descargar. *AppScanner*, [Taylor et al., 2016] es una herramienta para la identificación o clasificación de aplicaciones móviles de *Android* en tiempo real y para la identificación automática de huellas digitales. Construye las huellas digitales de las aplicaciones ejecutándolas en un dispositivo físico para que posteriormente se aplique procesamiento a esos rastros de red.

AppScanner puede trabajar tanto de forma *online*, como de forma *offline* para capturar y procesar el tráfico de red. Una vez que las capturas de tráfico han sido procesadas, se extraen las características que van a alimentar el algoritmo de AA. Se han utilizado 2 clasificadores: Support Vector Classifier (SVC) y RF Classifier y se han utilizado distintos enfoques, lo que ha permitido validar que los clasificadores multiclase pueden emplearse para identificar una multitud de aplicaciones en un único clasificador. Las aplicaciones sobre las que se han hecho las pruebas posteriormente han sido reidentificadas con una precisión superior al 99 %.

Otra dificultad a la hora de realizar la clasificación es que no es posible que el enfoque de huellas digitales cubran todas las aplicaciones existentes en la red y además hay muchas similitudes entre aplicaciones por el uso de bibliotecas comunes o de redes de distribución de contenidos.

La solución propuesta por [van Ede et al., 2020], *FlowPrint* utiliza un enfoque semi-supervisado para identificar aplicaciones móviles, combinando la agrupación basada en el destino, el aislamiento del navegador y el reconocimiento de patrones. Encuentra correlaciones temporales entre las características, que se utilizan para generar huellas digitales, obteniendo una precisión del 89,2 % en pruebas realizadas con aplicaciones de *Android* y *iOS* y un 93,5 % en la detección de aplicaciones no vistas anteriormente.

Para la realización de los experimentos se han empleado *datasets* de tráfico de red encriptado y etiquetado por aplicación que contienen tanto datos generados por usuarios como sintéticos, aplicaciones de *Android* y *iOS*, aplicaciones potencialmente maliciosas y benignas, distintas tiendas de aplicaciones y distintas versiones de las aplicaciones. Los 4 conjuntos utilizados son: *ReCon* (contiene capturas de tráfico etiquetado de 512 aplicaciones de *Android*), *CrossPlatform* (muestras generadas por usuarios para 215 aplicaciones de *Android* y 196 de *iOS*, además de otras aplicaciones de la tienda *Tencent MyApps* y 360 *Mobile Assistant*), *Andrubis* (muestras etiquetadas de aproximadamente 1 millón de aplicaciones de *Google Play Store* y de otras 15 tiendas de aplicaciones) y *Browser* (un *dataset* propio para recoger tráfico de navegadores: *Chrome, Firefox, Samsung Internet* y *UC Browser*). La métrica utilizada es Adjusted Mutual Information (AMI), una métrica utilizada en el aprendizaje no supervisado para definir la cantidad relativa de entropía obtenida al conocer una característica con respecto a la clase.

En [van Ede et al., 2020] se ha optado por un enfoque semi-supervisado y el modelo propuesto ha conseguido identificar aplicaciones no vistas anteriormente.

MAppGraph, [Pham et al., 2021] es un sistema para clasificar aplicaciones móviles que aborda el problema del tráfico encriptado, el carácter cambiante del comportamiento de los usuarios de las aplicaciones móviles y la similitud entre aplicaciones debido a las bibliotecas compartidas y las redes de distribución de contenido.

[Wang et al., 2017] consigue integrar las tareas de extracción de características, la selección óptima de estas y el clasificador en un marco unificado con la intención de aprender la relacional no lineal entre los datos.

Capítulo 4

Sistema propuesto

En este capítulo se va a explicar la metodología seguida para llevar a cabo la clasificación de aplicaciones en Android a partir de un *dataset* propio creado. Este capítulo se organiza de la siguiente manera: en la Sección 4.1 se proporciona una descripción del sistema propuesto, seguido de la obtención del conjunto de datos en la Sección 4.2. En la Sección 4.3 se presentan las características utilizadas y el preprocesamiento de los datos de entrada y el modelo y los parámetros utilizados en la Sección 4.4.

4.1. Descripción del Sistema Propuesto

El objetivo del sistema propuesto es la clasificación de aplicaciones móviles de *Android* a través del uso de algoritmos de DL. Hay un total de 16 aplicaciones para las cuales se han hecho capturas de tráfico en un entorno controlado. Para realizar las capturas se ha utilizado la aplicación *PCAPdroid* con cada una de las aplicaciones. Una vez obtenidas las capturas .PCAP se procede a transformar los datos en crudo a datos útiles que puedan ser procesados por los algoritmos de DL.

Para desarrollar el sistema propuesto, se han utilizado las bibliotecas de *Alive_Progress, Keras, Matplotlib, Numpy, Pandas, Pyshark, Scikit_Learn, Stellargraph ([Stellargraph, 2021]), Tensorflow, Tensorflow_Addons.*

A partir de este momento el sistema se puede dividir en 2 partes:

- Extracción de características y preprocesamiento mediante diferentes funciones y algoritmos desarrollados por el grupo GASS

- Creación del modelo y experimentos: se realiza en base a tres arquitecturas diferentes

A continuación, se puede ver el diagrama de flujo en la Figura 4.1, así como el sistema propuesto en la Figura 4.2:

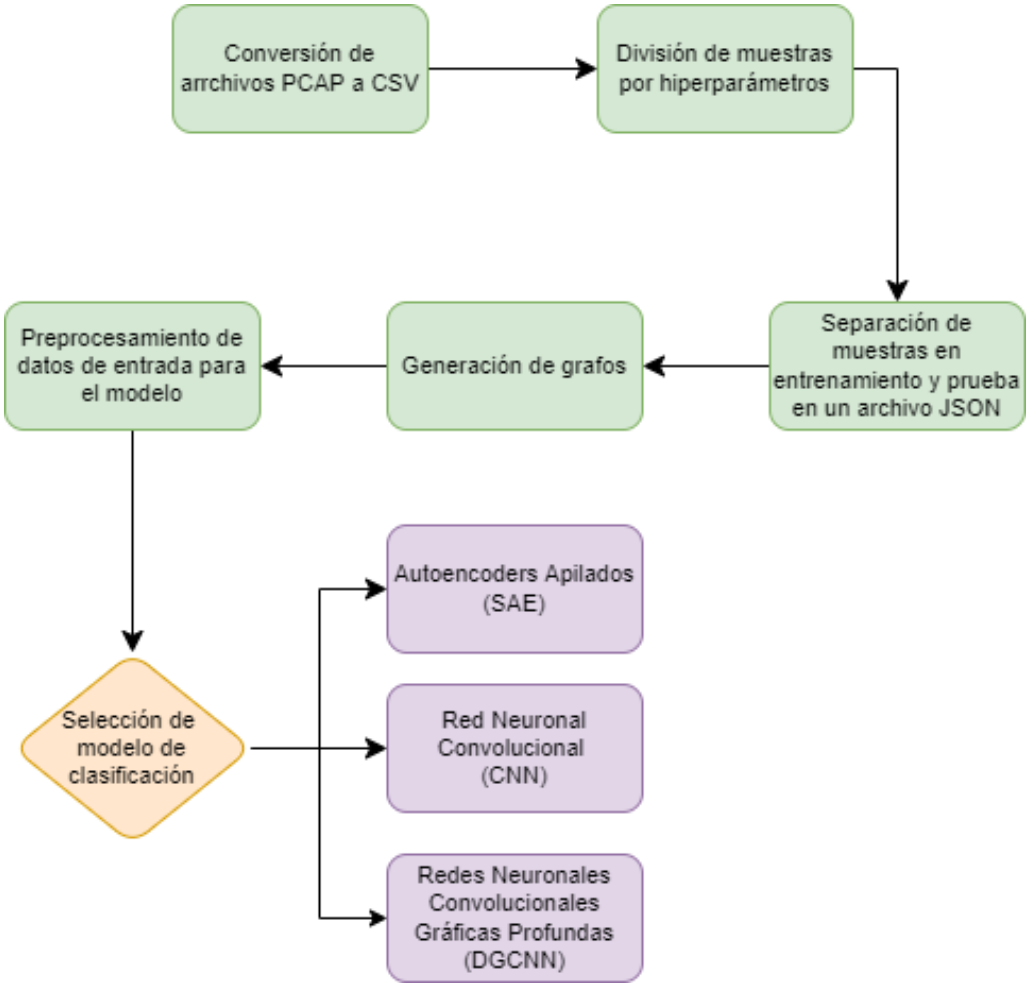

Figura 4.1: Diagrama de flujo del sistema

4.2. Creación del conjunto de datos

En esta sección se explica cómo se han recolectado los datos y el conjunto de datos final con el que se van a entrenar las diferentes arquitecturas de DL para clasificar aplicaciones *Android*. Para ello, se captura el tráfico de las aplicaciones *Android* con una aplicación también de *Android* denominada *PCAPdroid*.

Esta aplicación permite guardar los paquetes del flujo de datos en un archivo *.pcap*. [Faranda., 2020].

Los archivos PCAP que se obtienen de las aplicaciones, se renombran para poder conseguir un formato estandarizado en todos los archivos de cada aplicación.

En la Tabla 4.1, se puede visualizar el tamaño en Megabyte (MB) del conjunto de datos PCAP por cada aplicación tras haber realizado una limpieza de los archivos corruptos o dañados que no se pueden visualizar correctamente. En total, se cuenta con un *dataset* de 14602,9 MB o, lo que es lo mismo, 14,6 Gigabyte (GB).

Tabla 4.1: Tamaño del conjunto de datos inicial

	App1	App2	App3	App4	App5	App6	App7	App8
MB PCAP	16,1	987,0	757,0	233,0	464,0	905,0	1340,0	35,0

	App9	App10	App11	App12	App13	App14	App15	App16
MB PCAP	1490,0	35,8	2650,0	381,0	161,0	620,0	4310,0	218,0

Una vez que se tiene el *dataset* creado con los archivos *.pcap* mencionados anteriormente, se obtienen una serie de características estadísticas y numéricas para finalmente guardar el conjunto de datos en formato Comma Separated Values (CSV).

Tras este preprocesamiento, se cuenta con un conjunto de datos cuyo número de muestras se puede visualizar de forma detallada en la Tabla 4.2.

Tabla 4.2: Número de muestras por los parámetros y aplicación tras el preprocesamiento

	5-3		4-2		3-1		2-0		1-0	
	Total	Real	Total	Real	Total	Real	Total	Real	Total	Real
App1	480	320	440	289	390	237	330	199	450	252
App2	510	419	500	397	490	350	460	295	850	457
App3	200	73	150	58	110	38	70	26	70	26
App4	380	153	320	124	320	118	240	83	280	93
App5	820	650	780	601	730	570	640	509	1060	884
App6	370	236	340	216	320	186	280	160	460	222
App7	280	118	280	120	290	122	290	124	590	249
App8	230	179	240	182	220	163	220	144	350	204
App9	840	343	780	317	650	259	590	227	950	344
App10	130	128	140	138	140	134	150	137	290	233
App11	800	652	760	600	750	563	700	484	1340	793
App12	480	192	390	158	420	170	320	125	530	203
App13	970	677	930	613	870	564	810	485	1460	792
App14	310	241	300	230	290	205	270	166	400	199
App15	2280	901	1720	688	1220	481	700	282	620	253
App16	220	91	20	88	190	73	160	60	250	93
TOTAL	**9300**	**5373**	**8090**	**4819**	**7400**	**4233**	**6230**	**3506**	**9950**	**5297**

4.3. Preprocesamiento y características

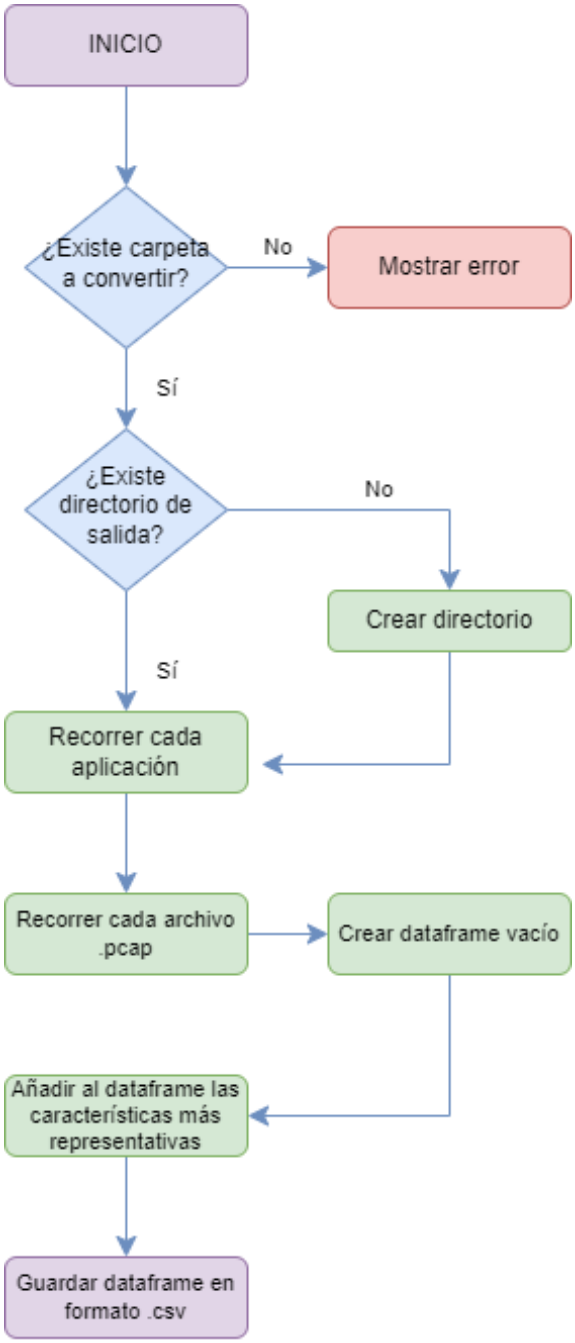

Figura 4.2: PCAP a CSV

Entre las características nombradas anteriormente, se cuenta con variables categóricas y continuas.

Todo el código desarrollado durante el trabajo está en el lenguaje de programación *Python* con una variedad de librerías usadas como pueden ser *Keras*, *TensorFlow*, *Scikit Learn*, *Pandas* o *Numpy* entre otras.

Para convertir los archivos PCAP que se han generado en la etapa anterior, se

usa el módulo **pcap_csv** como se puede ver en la Figura 4.2. En este módulo, se realizan operaciones de Entrada/Salida (E/S) en 2 carpetas. Una de ellas es *pcap_entrada* y la otra *csv_salida*, siendo ambos los valores por defecto definidos para las variables *INPUT_PCAP_FOLDER* y *OUTPUT_CSV_FOLDER*, respectivamente, del fichero *constants.py*.

En la primera de ellas están ubicados los archivos *.pcap* mencionados anteriormente y organizados a su vez por carpetas cuyo nombre es la aplicación.

En la segunda carpeta mencionada anteriormente *carpeta_salida* se guardan los archivos CSV donde se almacenan organizados por carpetas.

En la Figura 4.3 se puede ver la salida del terminal cuando se convierten los archivos PCAP en CSV.

Figura 4.3: PCAP a csv en proceso

En caso de que ya se hayan procesado anteriormente los archivos PCAP y se encuentren los CSV en la carpetas correspondientes, no se procesan y se pasa al siguiente archivo como se ve en la Figura 4.4.

```
csv_salida\whatsapp\wap_10_2.21.24.11_0112_pdr.csv already exists
csv_salida\whatsapp\wap_11_2.21.24.11_2711_dmm.csv already exists
```

Figura 4.4: PCAP a csv existente

Una vez convertidos todos los archivos con extensión *.pcap*, se ejecuta el módulo ***generating_samples***. Este módulo genera una carpeta llamada ***csv_hiperparametros*** en la que se almacena una carpeta por cada uno de los *hiperparámetros* utilizados, como se puede ver en la Figura 4.5.

El objetivo que tiene el módulo es dividir un flujo de tráfico de red en formato CSV en trozos más pequeños de la misma duración, llamados ***samples*** definidos por una serie de parámetros.

Se comprueba que no exista la carpeta con el mismo nombre de parámetro. Si es así, por cada tupla se crea una carpeta y en ella se crea otra carpeta que va a contener las muestras. Cada aplicación tiene su correspondiente carpeta de parámetros con los ficheros de muestra.

En este módulo los datos de entrada se dividen en entrenamiento y prueba. De las muestras generadas anteriormente, el 80 % representa el conjunto de entrenamiento y el 20 % restante el conjunto de prueba.

El parámetro usado y definido en el módulo ***constant.py*** llamado ***TRAIN_SIZE*** indica el porcentaje que corresponde al conjunto de entrenamiento.

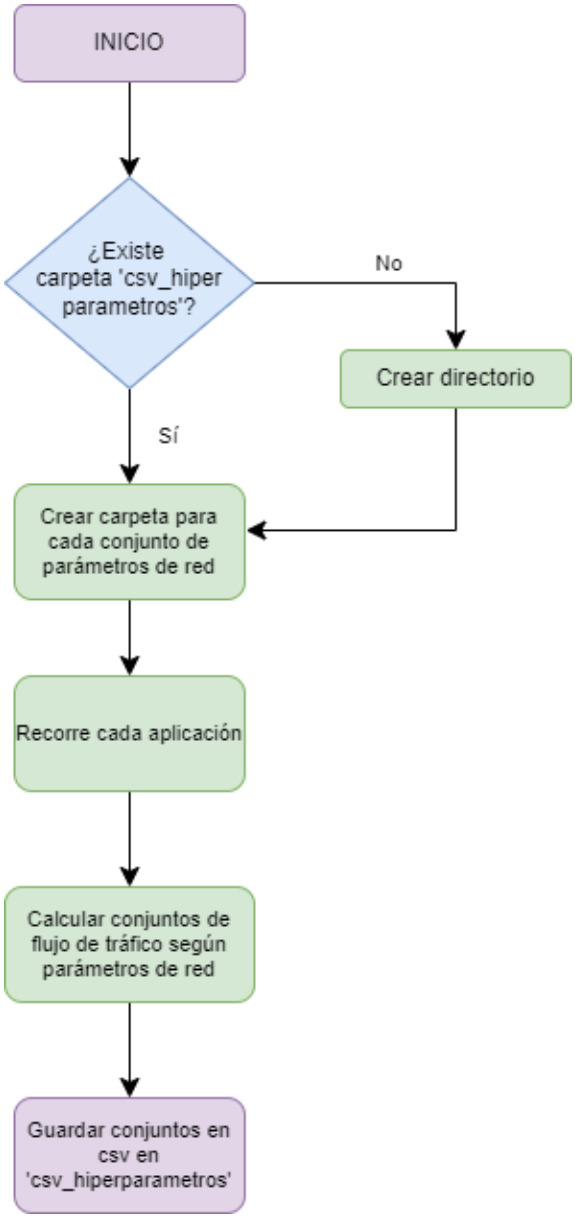

Figura 4.5: Módulo de generating_samples

Las muestras de entrenamiento y de prueba se guardan en ficheros de tipo *json*.

Como se puede observar en el diagrama de la Figura 4.6, el siguiente paso del sistema es el preprocesamiento de los datos con el módulo de ***generating_graphs***, la extracción de las características del flujo y de los paquetes y la generación de grafos.

Finalmente, las características se estandarizan y se guardan los grafos generados, como se ve en la Figura 4.7.

Se evalúan los diferentes modelos construidos con el conjunto de prueba utilizando las siguientes métricas de clasificación: *Precision, Recall, F1-Score* y *Accuracy*.

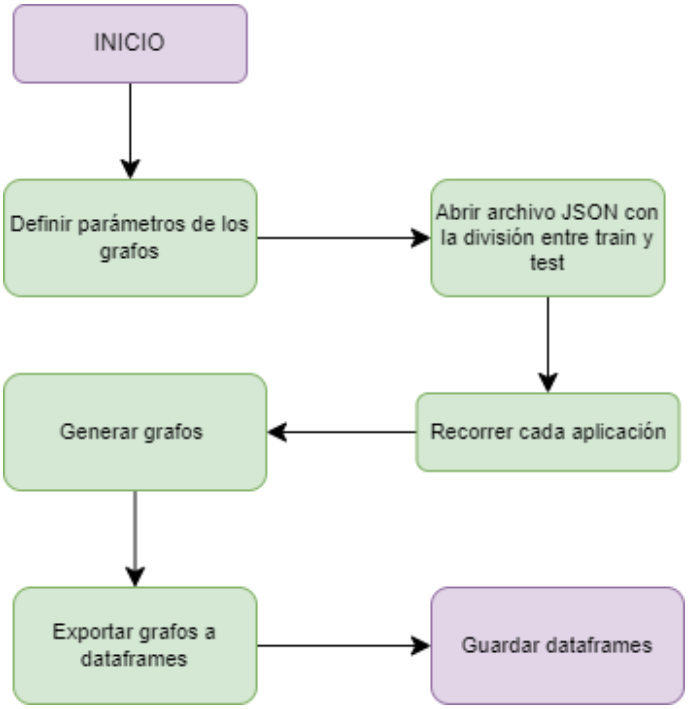

Figura 4.6: Módulo de generating_graphs

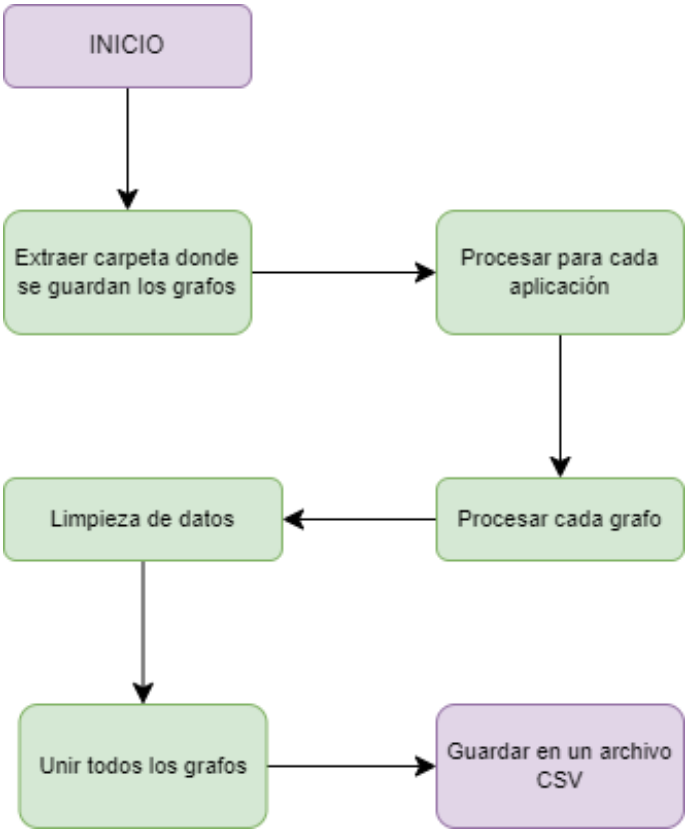

Figura 4.7: Módulo de preprocess_model

4.4. Arquitecturas de aprendizaje profundo propuestas

Tal y como se ha mencionado hasta ahora en este documento, el objetivo del sistema propuesto es la construcción de un modelo de IA para la clasificación de tráfico de red de aplicaciones móviles de *Android*.

Para realizar este modelo de clasificación, se han tenido en cuenta tres propuestas diferentes, con el fin de conocer cuál de ellas es mejor para este caso de uso. Las arquitecturas utilizadas han sido:

- Apilamiento de *Autoencoders*
- CNN
- DGCNN

El marco teórico de los algoritmos ha sido descrito en las Secciones 2.4.13.3, 2.4.13.4 y 2.4.13.5, respectivamente.

4.4.1. Arquitectura de Apilamiento de Autocodificadores

Dado que el sistema propuesto esta basado en [Lotfollahi et al., 2020], se optó por implementar un apilamiento de *autoencoders* SAE.

La arquitectura SAE presenta un conjunto de *autoencoders* apilados. Los pasos que se han seguido para conseguir el apilamiento son:

1. Crear cada uno de los *autoencoders* con N capas *Dense* totalmente conectadas.
2. Poner como función de activación para la primera capa la función *ReLU*.
3. Para la segunda capa se ha seleccionado la función sigmoidal.
4. Además, para cada uno de los *autoencoders* hay que reducir proporcionalmente el tamaño en cada una de sus capas dividiendo en partes iguales la diferencia entre el tamaño de entrada y el tamaño de salida, el cual debe ser el número de aplicaciones que van a formar parte de la clasificación. Véase la Figura 4.8.

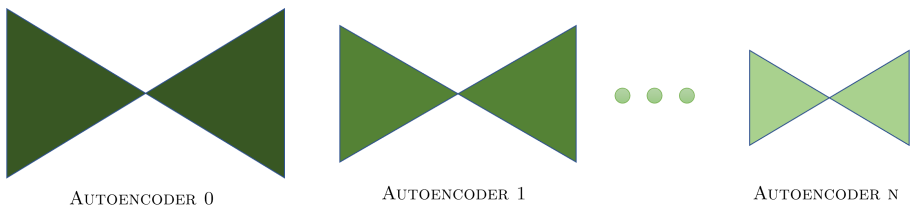

Figura 4.8: Creación inicial de autoencoders

5. Seguidamente, se apilan los codificadores de los *autoencoders* en orden de creación, es decir, de mayor a menor y, tras ellos, los decodificadores (*decoders*) en orden inverso (de menor número de neuronas a mayor), tal y como se puede visualizar en la Figura 4.9.

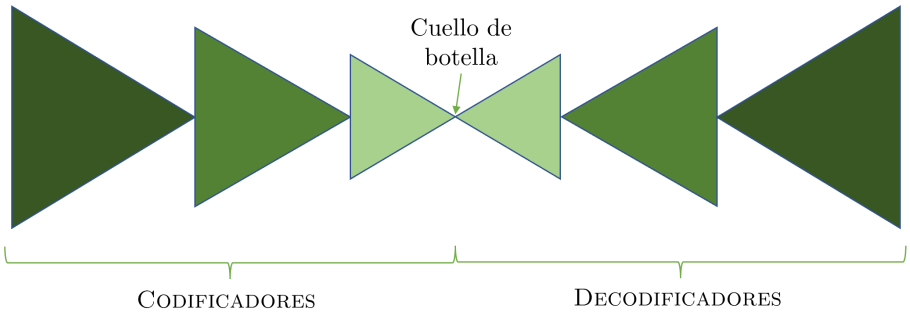

Figura 4.9: Apilamiento de autoencoders

6. En la Figura 4.10 se ve cómo en este paso, se opta por aplicar solo los codificadores (*encoders*) de cada uno de los *autoencoders*, con la información que se devuelve del cuello de botella o, su término inglés, *bottleneck*. En este punto, la salida del modelo es un vector de longitud fija que proporciona una representación comprimida de los datos de entrada, por lo tanto, no es necesario reconstruir la entrada del modelo con los decodificadores o *decoders* ampliando los datos de nuevo [Goodfellow et al., 2016].

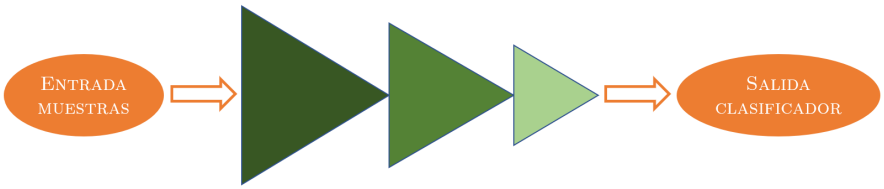

Figura 4.10: Clasificador basado en autocodificadores apilados

7. Al final, se añade una última capa con una función de activación *softmax*, que asigna probabilidades a cada clase en un problema multiclase.
8. Una vez creado el clasificador con el SAE se pasa a la fase de entrenamiento.
9. Para el entrenamiento se intenta reducir la función de perdida de entropía cruzada categórica (*categorical cross-entropy*) dado que la clasificación es multiclase.
10. Una vez entrenado el modelo, se guarda en formato Hierarchical Data Format version 5 (HDF5).
11. Para evaluar el modelo construido, se predice con el conjunto de datos *test* obteniendo las métricas nombradas anteriormente. Gracias a estas métricas, se evalúa el rendimiento del modelo.

4.4.2. Arquitectura de Redes Neuronales Convolucionales

La arquitectura de este algoritmo está basada en una CNN unidimensional. A continuación se describen los pasos seguidos para la construcción de esta arquitectura:

1. Para la construcción de un clasificador utilizando CNN, previamente se debe expandir o transformar el vector de características de entrada, dado que la convolución funciona sobre dimensiones espaciales.

2. En cuanto a la creación de la arquitectura del modelo, se crean primeramente 2 capas consecutivas *Conv1D*, con 200 y 100 neuronas respectivamente, y *ReLU* como función de activación.

3. Después de pasar el proceso de convolución, se crea una capa *MaxPooling* que se encarga de extraer las características más relevantes.

4. Una capa de aplanamiento (*Flatten*) es necesaria para transformar la salida en una matriz de una dimensión, pasando así toda la información de las capas convolucionales hacía el perceptrón multicapa, para que haga el proceso de clasificación.

5. Entre cada capa, se han realizado pruebas añadiendo otra capa de *Dropout* para evitar el sobreajuste podando el modelo.

6. El vector unidimensional que ha generado la capa *Flatten* se introduce en una red de N capas *Dense* totalmente conectadas, cada una con 300, 200, 100 y 50 neuronas respectivamente.

7. Por último, se añade una capa *softmax*, igual que se hace en la arquitectura SAE, para la tarea de clasificación.

8. Utiliza, al igual que el clasificador SAE, la entropía cruzada categórica como función de perdida.

9. El modelo se guarda para poder cargarse en la siguiente etapa, que es la de predicción sobre el conjunto de datos de prueba.

10. Para concluir, se obtienen los resultados de las distintas métricas.

4.4.3. Arquitectura de Redes Neuronales Convolucionales Gráficas

Debido a la naturaleza del problema del análisis de tráfico de red, se ha optado por emplear una nueva arquitectura, que son las Redes Neuronales Convolucionales Gráficas, cuyo desarrollo está basado en el algoritmo interno creado por el GASS.

Como se ha explicado anteriormente en la Sección 2.4.13.5, este modelo tiene como entrada grafos y se usa la biblioteca de *StellarGraph*.

En la Figura 4.11 se ve la arquitectura del modelo.

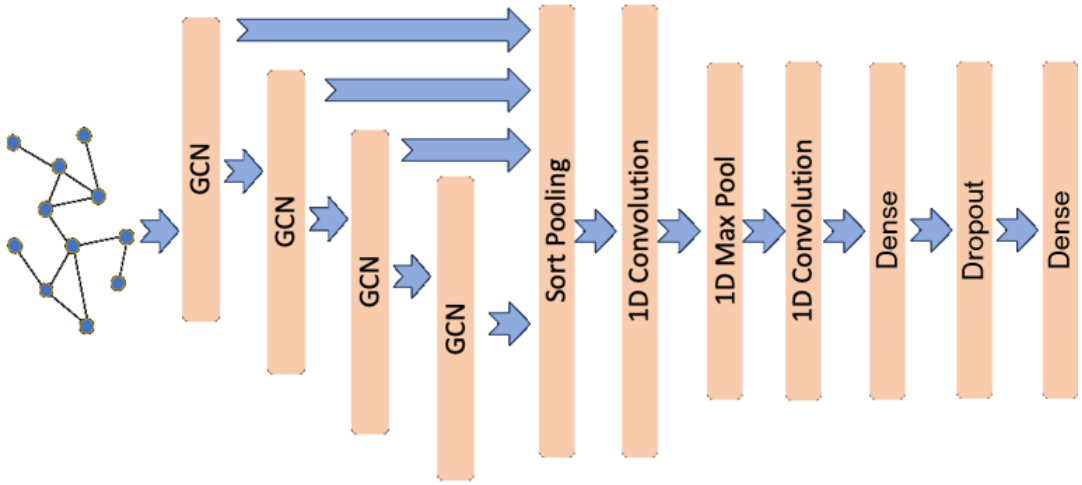

Figura 4.11: Arquitectura modelo DGCNN

A continuación se detallan los pasos seguidos para construir el modelo:

1. Inicialmente, se generan los grafos mediante la biblioteca *StellarGraph* en base al algoritmo propio de GASS para cada una de las aplicaciones.

2. Además, hay que generar las etiquetas que relacionan a cada grafo con su respectiva clase o aplicación dado que estamos creando un modelo de AA supervisado.

3. Las primeras cuatro capas de la arquitectura son de tipo CNN gráficas. Estas capas convolucionales gráficas tienen activaciones de tipo tanh.

4. La siguiente capa es una capa convolucional unidimensional, *Conv1D*, seguida de una capa de agrupación máxima, *MaxPool1D*.

5. A continuación, hay una segunda capa *Conv1D* que va seguida de dos capas de tipo *Dense*, la segunda utilizada para la clasificación binaria.

6. Las capas convolucionales y densas utilizan activación *ReLu* excepto la última capa densa que utiliza *softmax* para la clasificación.

7. Como se puede comprobar en la Figura 4.11, se añade una capa *Dropout* después de la primera capa *Dense*.

8. Dados los datos divididos en conjuntos de entrenamiento y prueba, se crea un objeto generador de datos de tipo *StellarGraph.PaddedGraphGenerator* que prepara los datos para el entrenamiento. Este generador suministra las matrices de características y las matrices de adyacencia a un modelo de clasificación de grafos de la biblioteca *Keras*. Además, realiza el relleno (*padding*) hasta que cada grafo cuente con el número de nodos indicado.

9. Se crean generadores de datos adecuados para el entrenamiento en el modelo de la biblioteca *Keras* llamando al método de flujo (*flow*) especificando los datos de entrenamiento y prueba.

10. Se entrena el modelo utilizando el método *fit* del modelo. Pero antes, se tiene que dividir el conjunto de datos en conjuntos de entrenamiento y de prueba: el 80 % de los datos para el entrenamiento y el 20 % restante para las pruebas.

11. Una vez realizado el entrenamiento, se puede utilizar para predecir mediante el método *predict*.

12. Por último, se calcula el rendimiento del modelo entrenado con el conjunto de datos de prueba.

Capítulo 5

Experimentos y Resultados

En este capítulo, se presentan los experimentos realizados con el sistema propuesto y los resultados que se han obtenido.

Cabe destacar que por motivos de confidencialidad, no es posible especificar el nombre de los parámetros con los que se han realizado las pruebas. Se usarán nombres alternativos sin ningún tipo de significado real, tanto para los parámetros de los flujos de tráfico de red como para los parámetros del algoritmo propio de grafos perteneciente a GASS.

Además, en base al número de muestras por aplicación que contiene el *dataset*, se han reducido el número de aplicaciones para los experimentos eliminando aquellas que no contaban con suficientes muestras para poder ser clasificadas.

5.1. Elección de métricas para la evaluación

Como métrica para la comparación y selección de modelos, se ha elegido el ***F1-score***, dado que se usa un conjunto de datos que está desbalanceado, tal y como se ha podido comprobar en el número de muestras por aplicación en la Tabla 4.2 del Capítulo 4.

Además, para realizar la media de la puntuación F1 de todas las clases, se ha optado por realizar la denominada como **media macro** ('*macro average*'), la cual se calcula utilizando la siguiente Fórmula 5.1:

$$MediaMacroF1score = \frac{F1_{App1} + F1_{App2} + F1_{App3} + ... + F1_{AppN}}{NúmeroApps} \tag{5.1}$$

Esta métrica realiza la media de puntuaciones F1 sin tener en cuenta el número de muestras de cada aplicación, por lo que la predicción de la clase mayoritaria no incrementa la media final, tal y como lo hace la media ponderada ('*weighted average*').

5.2. Experimentos con Autocodificadores Apilados

En esta sección, se pretende mostrar las pruebas que han sido llevadas a cabo para optimizar el modelo de clasificación basado en SAE. En los experimentos que se han realizado se modifican los siguientes parámetros o configuraciones:

- Balanceamiento del conjunto de datos (uso de *class_weight*)
- Número de *autoencoders* apilados en el SAE
- Adición de capas *Dropout*
- Número de épocas *(epochs)* en el entrenamiento
- Experimentos con parámetros del tráfico de red
- Experimentos con parámetros de los grafos

5.2.1. Balanceamiento del conjunto de datos

En unas pruebas iniciales, se pudo comprobar que los resultados obtenidos no eran nada buenos, dado que el modelo no llegaba a predecir en ningún caso algunas de las clases (aplicaciones a clasificar). Comparándolo con el número de muestras, estas clases eran las clases minoritarias, es decir, las clases que contaban con un menor número de muestras.

Para ejemplificar, se puede comprobar en la Tabla 5.1 la relación que tiene el soporte de una clase, el cual es el número de veces que aparecen muestras etiquetadas como dicha clase en el conjunto de prueba, con su puntuación F1. Esta prueba se realizó en la fase inicial de experimentación, por lo que fue utilizada la configuración de parámetros por defecto en base al Estado del Arte de este proyecto, como por ejemplo, fijando el número de *autoencoders* en tres [Antonello et al., 2012].

Tabla 5.1: Experimentos de SAE con carga desbalanceada

	f1-score	Soporte
App1	0	95
App2	0,00	105
App5	0,17	166
App7	0,00	62
App9	0,06	173
App10	0,00	26
App11	0,01	152
App13	0,27	187
App15	0,63	456
	MEDIA=0.27	TOTAL=1422

Las clases minoritarias, es decir, aquellas que cuentan con un menor soporte de muestras, tienen un *f1-score* igual a 0 lo cual significa que el modelo nunca ha clasificado correctamente estas clases.

La mayoría de los algoritmos de AA no funcionan correctamente con conjuntos de datos desbalanceados. A pesar de eso, se puede modificar el algoritmo de entrenamiento actual para tener en cuenta el número de muestras por clase. Esto se puede conseguir dando diferentes pesos a las clases, tanto mayoritarias como minoritarias. Dicha diferencia de pesos influye en la clasificación de las clases durante la fase de entrenamiento. El objetivo es penalizar la clasificación errónea de la clase minoritaria estableciendo un peso de clase más alto y, al mismo tiempo, reduciendo el peso de la clase mayoritaria.

Para intentar solucionar este problema que tanto afecta a los algoritmos de ML y DL, se ha probado añadiendo el parámetro *class_weights* a la hora de entrenar el modelo, que ha sido realizado con la biblioteca *Keras Tensorflow* en *Python*. Este parámetro es un diccionario que asigna un valor de peso a cada una de las clases (aplicaciones en este sistema), pudiendo ser calculado de forma manual o de forma automática para que sea equilibrado o, como en este caso, balanceado.

En la Tabla 5.2 se muestran los resultados tras el cálculo de los valores de este diccionario, extraídos del informe de clasificación del modelo entrenado, utilizando, de esta forma, el conjunto de datos balanceado.

Tabla 5.2: Experimentos de SAE con carga balanceada

	f1-score	Soporte
App1	0,23	95
App2	0,34	105
App5	0,17	166
App7	0,16	62
App9	0,15	173
App10	0,39	26
App11	0,33	152
App13	0,36	187
App15	0,65	456
	MEDIA=0,31	TOTAL=1422

A pesar de no contar aún con buenos resultados, no se puede evidenciar una relación lineal tan clara entre la puntuación F1 y el número de muestras de cada clase, la cual perjudica a las clases minoritarias. Además, el clasificador comienza a reconocer la mayoría de las clases lo cual es de vital importancia.

Hay que tener en cuenta que la arquitectura seleccionada (SAE) reduce el número de características de la entrada, por lo que podría ser que necesitase un mayor número de muestras que otras arquitecturas, especialmente para el entrenamiento.

Para concluir este experimento, tras los resultados obtenidos se decretó que **es necesario balancear la carga del *dataset***, por lo que para los siguientes experimentos se balancea para intentar mejorar estas métricas.

5.2.2. Número de autocodificadores apilados

Se han empezado los experimentos con el número de *autoencoders* que se apilan en la arquitectura SAE. Este valor, inicialmente fue fijado en 3, basado en las referencias descritas en el Capítulo 3.

Para comprobar si el valor tres para el número de *autoencoders* también es el mejor en este sistema, se realizaron experimentos con diferente número de autocodificadores apilados. En la Tabla 5.3 se pueden comprobar las métricas medias de *f1-score* con una configuración de los flujos de tráfico de red determinada para cada número de *autoencoders* apilados en el clasificador de SAE.

Tabla 5.3: Experimentos con el número de autocodificadores apilados

f1-score	1 AE	2 AEs	3 AEs	4 AEs	5 AEs
App1	0.39	0.30	0.30	0.31	0.15
App2	0.38	0.46	0.28	0.48	0.31
App5	0.18	0.29	0.00	0.00	0.10
App7	0.50	0.33	0.00	0.44	0.41
App9	0.29	0.23	0.24	0.18	0.02
App10	0.12	0.00	0.00	0.51	0.00
App11	0.33	0.34	0.38	0.40	0.39
App13	0.31	0.36	0.18	0.39	0.03
App15	0.69	0.70	0.00	0.77	0.58
MEDIA	0.35	0.34	0.15	0.39	0.22

En los experimentos realizados en esta etapa, se concluyó que el número de *autoencoders* que se deben apilar en el SAE para obtener los mejores resultados es **cuatro**. Aún así, la diferencia entre el uso de 1, 2 ó 4 *autoencoders* no es muy grande, así que este valor podría cambiar a lo largo de los siguientes experimentos.

5.2.3. Adición de capas Dropout

Con el objetivo de evitar el sobreajuste, se quiso poner a prueba el uso de capas *Dropout* entre cada una de las capas de los *autoencoders*, es decir, una capa *Dropout* después de la capa del codificador (*encoder*) y otra tras el decodificador (*decoder*). En la Tabla 5.4, se plasman los resultados de dos pruebas con las mismas configuraciones de parámetros, en las que en una columna se añaden las capas *Dropout* y en la otra no aparecen.

El objetivo principal de la inserción de estas capas es omitir aleatoriamente neuronas durante el proceso de entrenamiento de una red neuronal, para evitar el sobreaprendizaje del mismo con el *dataset* de entrenamiento. Esta técnica, se debería utilizar únicamente en aquellos casos en los que el modelo proporcione muy buenas métricas, debidas al sobreaprendizaje. Por el razonamiento anterior, en este clasificador basado en una arquitectura SAE **no es necesario el uso de capas *Dropout***.

Tabla 5.4: Experimentos de uso de capas Dropout en el modelo SAE

f1-score	Con Dropout	Sin Dropout
App1	0.15	0.05
App2	0.02	0.41
App5	0.12	0.32
App7	0.04	0.06
App9	0.09	0.03
App10	0.03	0.04
App11	0.25	0.37
App13	0.18	0.29
App15	0.60	0.69
MEDIA	0.17	0.24

5.2.4. Parámetros del tráfico de red

En esta sección, se detallan los experimentos realizados variando los parámetros del tráfico de red. Como se menciona anteriormente, por temas de confidencialidad no se pueden mencionar dichos parámetros, por lo que los denominaremos como *Pi*, siendo *i* cada una de las diferentes configuraciones de parámetros de red.

En la Tabla 5.5, se incluyen las pruebas más relevantes realizadas para comparar la eficacia de estos parámetros. Estas pruebas, representadas como filas, son independientes entre ellas y se han realizado tomando diferentes valores para el resto de parámetros configurables.

Tabla 5.5: Experimentos con parámetros de red en el modelo SAE

f1-score	P1	P2	P3	P4	P5
Prueba 1	0,29	0,21	0,32	0,32	0,26
Prueba 2	0,53	0,51	0,47	0,40	0,48
Prueba 3	0,43	0,41	0,38	0,31	0,34
Prueba 4	0,50	0,28	0,30	0,47	0,48
Prueba 5	0,25	0,14	0,40	0,47	0,44
Prueba 6	0,26	0,24	0,20	0,30	0,33
Prueba 7	0,28	0,20	0,27	0,27	0,28
Prueba 8	0,19	0,13	0,12	0,32	0,29
Prueba 9	0,22	0,23	0,25	0,25	0,40
MEDIA	0,33	0,26	0,30	0,35	0,37

Como se puede observar, la media más alta se obtuvo con *P5*.

5.2.5. Parámetros de los grafos

En esta sección, se detallan los experimentos realizados variando los parámetros del algoritmo interno de GASS basado en grafos.

En la Tabla 5.6 se observa cómo el mejor *f1_score* se consigue con el *Parámetro1* igual a 5 y el *Parámetro2* igual a 10.

Tabla 5.6: Experimentos con parámetros del tráfico de red

	Parámetro1	Parámetro2	f1_score
Prueba 1	10	5	0,28
Prueba 2	5	10	0,39
Prueba 3	10	10	0,29
Prueba 4	3	5	0,32

5.2.6. Número de épocas en el entrenamiento

Para este experimento, se ha tratado de plasmar en un gráfico la **curva de aprendizaje** que tiene este modelo SAE en base a la métrica *f1_score*. En el mismo,

se relaciona la puntuación F1 con cada época de entrenamiento. Un ejemplo de esta curva es la que aparece en la Figura 5.1, en la que se muestra la curva de aprendizaje que sufre el entrenamiento en base al *f1_score* en cada una de las épocas.

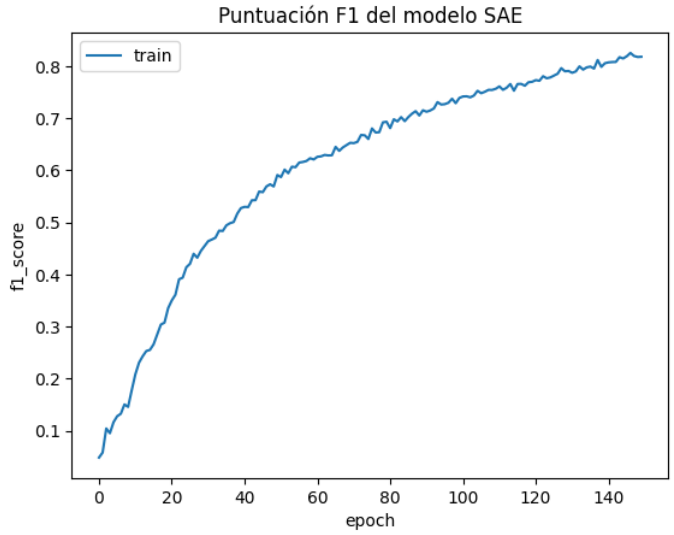

Figura 5.1: Curva de aprendizaje del modelo de SAE

Cabe destacar que hasta este punto de la fase de experimentación, las pruebas se habían realizado con un número de épocas fijado en 30, obteniendo un *f1-score* máximo de 0,39. Tras analizar este gráfico, se concluye que se debe usar **150 épocas** para entrenar este clasificador, ya que hasta ese punto la puntuación F1 presenta una curva creciente pronunciada, llegando a alcanzar un *f1-score* de 0,8.

5.2.7. Parámetros finales del modelo

Tras el estudio o análisis de los mejores valores de los parámetros anteriores, se ha obtenido el modelo final de la arquitectura basada en SAE. Como recapitulación de esta sección, para lograr optimizar el ajuste de parámetros en esta arquitectura, los valores elegidos para obtener el modelo final han sido:

- Apilamiento de cuatro *autoencoders*.
- Inserción del balanceado de carga mediante *class-weights*.
- Descarte del uso de capas *Dropout*.
- Uso del conjunto P5 de parámetros de red.
- Parámetros de los grafos igualados a 3 y 5.
- Entrenamiento mediante 150 épocas.

La Figura 5.2, representa la matriz de confusión obtenida con el *testing* del modelo. En dicha matriz, se puede visualizar tanto la buena clasificación de algunas aplicaciones

concretas (por ejemplo *App1*, *App2* o *App7*) como la peor clasificación de otras (por ejemplo las aplicaciones *App5*, *App9* o *App11*).

Además, la Tabla 5.7, representa la matriz de confusión normalizada donde se observa lo mencionado anteriormente.

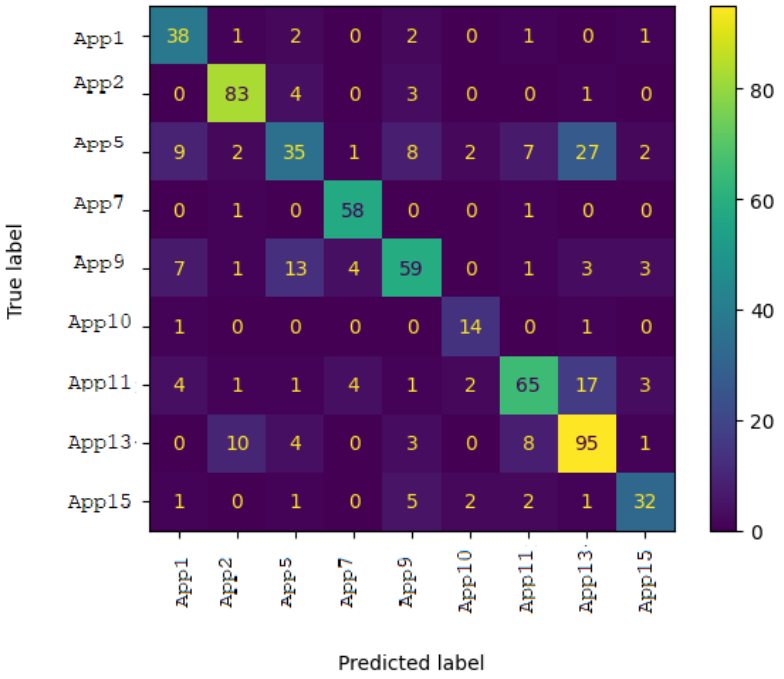

Figura 5.2: Matriz de confusión del modelo final de SAE

Tabla 5.7: Matriz de confusión normalizada del modelo final SAE

Etiqueta real \ Predicción	App1	App2	App5	App7	App9	App10	App11	App13	App15
App1	84,44 %	2,22 %	4,44 %	0,00 %	4,44 %	0,00 %	2,22 %	0,00 %	2,22 %
App2	0,00 %	91,21 %	4,40 %	0,00 %	3,30 %	0,00 %	0,00 %	1,10 %	0,00 %
App5	9,68 %	2,15 %	37,63 %	1,08 %	8,60 %	2,15 %	7,53 %	29,03 %	2,15 %
App7	0,00 %	1,67 %	0,00 %	96,67 %	0,00 %	0,00 %	1,67 %	0,00 %	0,00 %
App9	7,69 %	1,10 %	14,29 %	4,40 %	64,84 %	0,00 %	1,10 %	3,30 %	3,30 %
App10	6,25 %	0,00 %	0,00 %	0,00 %	0,00 %	87,50 %	0,00 %	6,25 %	0,00 %
App11	4,08 %	1,02 %	1,02 %	4,08 %	1,02 %	2,04 %	66,33 %	17,35 %	3,06 %
App13	0,00 %	8,26 %	3,31 %	0,00 %	2,48 %	0,00 %	6,61 %	78,51 %	0,83 %
App15	2,27 %	0,00 %	2,27 %	0,00 %	11,36 %	4,55 %	4,55 %	2,27 %	72,73 %

Finalmente, en la Tabla 5.8 se puede comprobar la métrica de *f1_score* obtenida por cada aplicación. La media de este modelo final es 0,73.

Tabla 5.8: Métricas del modelo final de SAE

	f1-score
App1	0.72
App2	0.87
App5	0.46
App7	0.91
App9	0.69
App10	0.78
App11	0.71
App13	0.71
App15	0.74
MEDIA	0.73

5.2.8. Modelo final con todas las aplicaciones

En esta sección, se utiliza el conjunto de datos en su totalidad, el cual cuenta con 16 aplicaciones y está gravemente desbalanceado. En la Tabla 5.9, se visualizan los resultados obtenidos con este *dataset*, habiendo configurado el modelo con los parámetros finales descritos en la sección anterior. Finalmente, se obtuvo una media de puntuación F1 del 56 %.

Tabla 5.9: Resultados modelo final SAE con dataset desbalanceado de 16 aplicaciones

	f1-score		*f1-score*
App1	0,52	App9	0,42
App2	0,59	App10	0,70
App3	0,42	App11	0,53
App4	0,63	App12	0,50
App5	0,08	App13	0,58
App6	0,22	App14	0,54
App7	0,94	App15	0,83
App8	0,31	App16	0,20
MEDIA		**0,56**	

5.3. Experimentos con Redes Neuronales Convolucionales

En esta sección se muestran los experimentos que se han realizado con la arquitectura de CNN. En los experimentos que se han realizado se modifican los siguientes parámetros o configuraciones:

- Balanceamiento del conjunto de datos (uso de *class_weight*).
- Número de capas densas en la CNN.
- Número de épocas en el entrenamiento

- Experimentos con parámetros del tráfico de red
- Experimentos con parámetros de los grafos

5.3.1. Balanceamiento del conjunto de datos

Con la arquitectura y parámetros propuestos en la Sección 4.4.2, en las pruebas iniciales se hicieron los experimentos con los datos sin balancear. Observando los resultados obtenidos, se decidió balancear el conjunto de datos con el parámetro *class_weight*. Podemos ver en la Tabla 5.10 los resultados obtenidos sin balancear la carga.

Tabla 5.10: Experimento de CNN con carga desbalanceada

	f1-score	Soporte
App1	0,41	95
App2	0,49	105
App5	0,64	166
App7	0,31	62
App9	0,53	173
App10	0,40	26
App11	0,60	152
App13	0,47	187
App15	0,29	456
	MEDIA=**0,46**	TOTAL=1422

En la Tabla 5.11 se puede ver una gran mejoría en cuanto a los resultados cuando la carga está balanceada.

Tabla 5.11: Experimento de CNN con carga balanceada

	f1-score	Soporte
App1	0,91	95
App2	0,94	105
App5	0,90	166
App7	0,88	62
App9	0,72	173
App10	0,90	26
App11	0,89	152
App13	0,93	187
App15	0,82	456
	MEDIA=**0,88**	TOTAL=1422

5.3.2. Número de capas densas en la CNN

Los experimentos realizados en esta sección se realizan variando el número de capas *Dense* al final de la CNN. Se ha probado en un rango desde 3 hasta 8. La siguiente Tabla 5.12 muestra los resultados obtenidos.

Tabla 5.12: Experimentos número de capas densas de CNN

	3	4	5	6	7	8
f1_score	0.87	0.85	0.83	0.89	0.89	0.89

Por lo tanto, se usarán 6 capas de tipo *Dense* para el modelo final de CNN.

5.3.3. Número de épocas en el entrenamiento

Para este experimento, se ha tratado de plasmar en un gráfico la **curva de aprendizaje** que tiene este modelo CNN en base a la métrica *f1_score*. En el mismo, se relaciona la puntuación F1 con cada época de entrenamiento. Un ejemplo de esta curva es la que aparece en la Figura 5.3, en la que se muestra la curva de aprendizaje que sufre el entrenamiento en base al *f1_score* en cada una de las épocas. El modelo deja de crecer exponencialmente a partir de la época 25, aproximadamente.

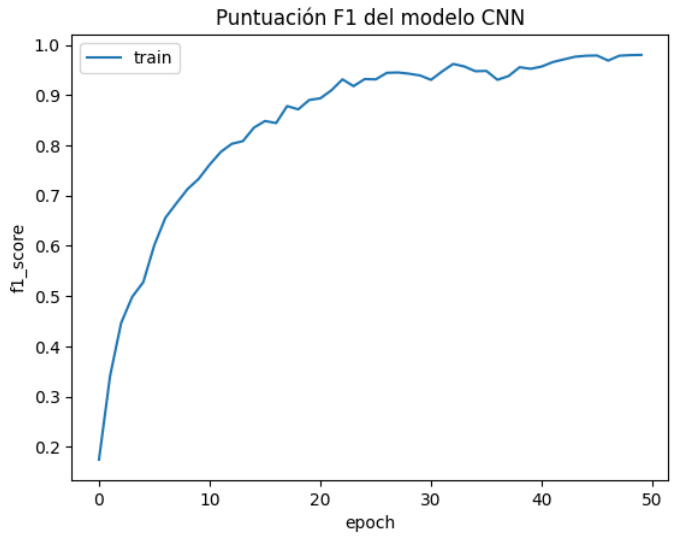

Figura 5.3: Curva de aprendizaje CNN

Por lo tanto, se usarán **25 épocas** en todos los demás experimentos.

5.3.4. Parámetros del tráfico de red

En esta sección se detallan los experimentos realizados variando los parámetros del tráfico de red. Como se menciona anteriormente, por temas de confidencialidad no se pueden mencionar dichos parámetros, por lo que usaremos *Pi*, siendo *i* cada uno de los distintos parámetros, como se puede ver en la Tabla 5.13. Sólo se mostrarán las pruebas más representativas.

Como se puede observar, el valor más alto lo produce *P5*.

Tabla 5.13: Experimentos parámetros del tráfico de red

f1-score	P1	P2	P3	P4	P5
Prueba 1	0,77	0,72	0,74	0,79	0,84
Prueba 2	0,57	0,57	0,64	0,69	0,70
Prueba 3	0,46	0,39	0,42	0,51	0,52
Prueba 4	0,27	0,25	0,33	0,54	0,53
Prueba 5	0,88	0,88	0,86	0,87	0,89
Prueba 6	0,80	0,73	0,76	0,85	0,84
Prueba 7	0,79	0,76	0,77	0,81	0,83
Prueba 8	0,89	0,85	0,87	0,88	0,88
Prueba 9	0,89	0,89	0,86	0,89	0,89
Prueba 10	0,60	0,55	0,61	0,68	0,72
MEDIA	0,69	0,66	0,69	0,75	0,76

5.3.5. Parámetros de los grafos

En esta sección se detallan los experimentos realizados variando los parámetros de los grafos. Como se menciona anteriormente, por temas de confidencialidad no se pueden mencionar dichos parámetros, por lo que se usarán los nombres de *Parámetro1* y *Parámetro2*, como se puede observar en La Tabla 5.14.

Tabla 5.14: Experimentos con parámetros de los grafos con balanceamiento

	Parámetro1	Parámetro2	f1_score
Prueba 1	5	1	0,84
Prueba 2	10	5	0,70
Prueba 3	5	10	0,89
Prueba 4	10	10	0,72
Prueba 5	3	5	0,96

Como conclusión a la tabla anterior, se observa que la mejor métrica *f1_score* se obtiene con *Parámetro1* igual a 3 y *Parámetro2* igual a 5.

5.3.6. Parámetros finales del modelo

Tras el estudio o análisis de los mejores valores de los parámetros anteriores, se ha obtenido el modelo final de la arquitectura basada en CNN. Como recapitulación de esta sección, para lograr optimizar el ajuste de parámetros en esta arquitectura, los valores elegidos para obtener el modelo final han sido:

- Uso de 6 capas de tipo *Dense* al final de la CNN.
- Uso de capas *Dropout* para evitar el sobreajuste.
- Inserción del balanceado de carga mediante *class-weights*.

- Uso del conjunto P5 de parámetros de red.
- Parámetros de los grafos igualados a 3 y 5, respectivamente.
- Entrenamiento mediante 25 épocas.

Con los valores anteriores, se obtuvo un valor del 98 % en cuanto a la métrica del *f1_score*. Se puede observar la matriz de confusión en la Figura 5.4. Además, en la Tabla 5.15 se representa la matriz de confusión normalizada, donde se ve la buena clasificación de todas las aplicaciones.

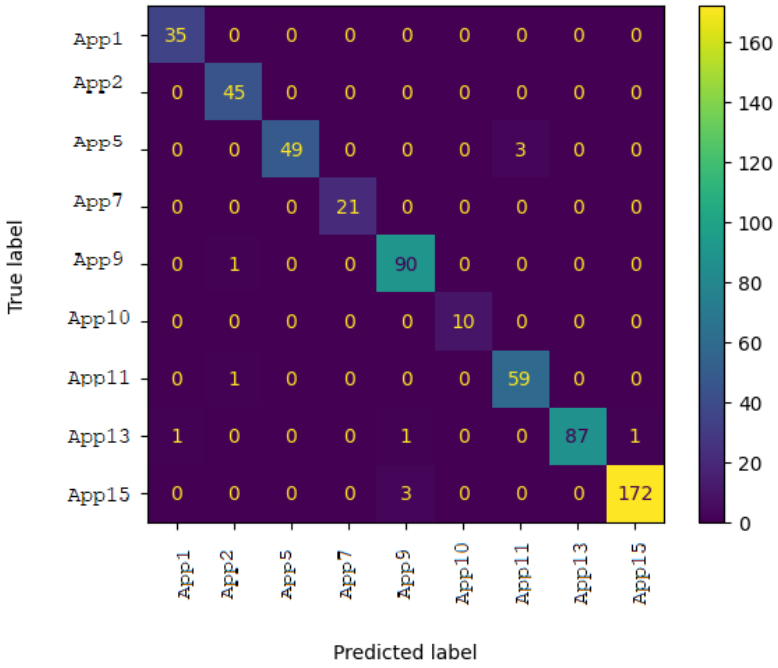

Figura 5.4: Modelo CNN

Tabla 5.15: Modelo CNN

Etiqueta real \ Predicción	App1	App2	App5	App7	App9	App10	App11	App13	App15
App1	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %
App2	0,00 %	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %
App5	0,00 %	0,00 %	94,23 %	0,00 %	0,00 %	0,00 %	5,77 %	0,00 %	0,00 %
App7	0,00 %	0,00 %	0,00 %	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %
App9	0,00 %	1,10 %	0,00 %	0,00 %	98,90 %	0,00 %	0,00 %	0,00 %	0,00 %
App10	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	100,00 %	0,00 %	0,00 %	0,00 %
App11	0,00 %	1,67 %	0,00 %	0,00 %	0,00 %	0,00 %	98,33 %	0,00 %	0,00 %
App13	1,11 %	0,00 %	0,00 %	0,00 %	1,11 %	0,00 %	0,00 %	96,67 %	1,11 %
App15	0,00 %	0,00 %	0,00 %	0,00 %	1,71 %	0,00 %	0,00 %	0,00 %	98,29 %

En la Tabla 5.16 se puede comprobar la métrica de *f1_score* obtenida por cada aplicación. La media de este modelo final es 0,98, lo que indica la buena capacidad de generalización del modelo.

Tabla 5.16: Métricas del modelo final de CNN

	f1-score
App1	0.99
App2	0.98
App5	0.97
App7	1.00
App9	0.97
App10	1.00
App11	0.97
App13	0.98
App15	0.99
MEDIA	0.98

5.3.7. Modelo final con todas las aplicaciones

En esta sección, se utiliza el conjunto de datos en su totalidad, el cual cuenta con 16 aplicaciones y está gravemente desbalanceado. En la Tabla 5.17, se visualizan los resultados obtenidos con este *dataset*, habiendo configurado el modelo con los parámetros finales descritos en la sección anterior. Finalmente, se obtuvo una media de puntuación F1 del 83 %.

Tabla 5.17: Modelo final con dataset desbalanceado de 16 aplicaciones

	f1-score		*f1-score*
App1	0,83	App9	0,85
App2	0,84	App10	0,84
App3	0,67	App11	0,85
App4	0,89	App12	0,90
App5	0,82	App13	0,89
App6	0,73	App14	0,80
App7	0,97	App15	0,88
App8	0,69	App16	0,80
MEDIA		**0,83**	

5.4. Experimentos con Redes Neuronales Convolucionales Gráficas

En esta sección se muestran los experimentos que se han realizado con la arquitectura de DGCNN.

Este es el modelo de referencia, por lo que se han utilizado los siguientes parámetros por defecto:

- Uso de una capa *Dropout* para evitar el sobreajuste.
- Uso del conjunto P5 de parámetros de red.
- Parámetros de los grafos igualados a 3 y 5, respectivamente.
- Entrenamiento mediante 150 épocas.

En la Tabla 5.18, se muestran los resultados obtenidos en el *testing* del modelo entrenado mediante el conjunto de datos reducido de 9 aplicaciones. En esta, se muestra finalmente un *f1_score* del 85 %.

Tabla 5.18: Modelo final con número de aplicaciones reducido

	f1-score
App1	0,87
App2	0,89
App5	0,90
App7	0,83
App9	0,88
App10	0,66
App11	0,83
App13	0,85
App15	0,96
MEDIA	0,85

Por último, en la Tabla 5.19 se pueden visualizar los diferentes *f1_scores* para el conjunto de datos de 16 aplicaciones al completo. Como media final se obtiene un 66 %.

Tabla 5.19: Modelo final con todas las aplicaciones

	f1-score		*f1-score*
App1	0,80	App9	0,80
App2	0,40	App10	0,44
App3	0,81	App11	0,88
App4	0,72	App12	0,73
App5	0,86	App13	0,42
App6	0,00	App14	0,83
App7	0,96	App15	0,66
App8	0,44	App16	0,78
MEDIA		**0,66**	

Capítulo 6

Capítulo de contribuciones

En esta sección se explican las contribuciones de cada integrante del grupo que ha realizado este Trabajo de Fin de Grado.

Este Trabajo de Fin de Grado surge como idea de nuestros directores de proyecto, Luis Javier García Villalba y Daniel Povedano Álvarez. Los tres alumnos estábamos interesados en formar parte de un proyecto así. El haber hecho este trabajo entre tres personas nos ha permitido ser parte de un gran proyecto, poder llevar a cabo acciones que individualmente no podríamos haber realizado y un constante aprendizaje mutuo.

Cada uno de nosotros ha aportado de forma equitativa al proyecto, poniendo en práctica todo lo aprendido durante la carrera universitaria.

El trabajo ha sido constante durante todo el desarrollo, con reuniones semanales con nuestros tutores e investigadores de GASS, lo que nos ha permitido aprender mucho de temas muy diversos. A continuación, se explica con más detalle lo que ha aportado cada miembro del equipo para realizar dicho proyecto y todos los conocimientos que se han ido adquiriendo.

6.1. Pablo Díaz Rodríguez

Comencé la carrera de ingeniería informática en 2018. A lo largo de los cuatro años he tenido asignaturas que me han gustado mucho y profesores que me han enseñado y han incrementado mis ilusiones por esta carrera. Una vez que pasaron los tres primeros años del grado en ingeniería informática, tenía que elegir el tema para el trabajo de fin de grado. Cuando ví las propuestas que nos ofrecían, me puse en contacto con los directores de este proyecto para que me explicaran cuál era la intención. Y sin ninguna duda, decidí trabajar con ellos.

Al comenzar este Trabajo de Fin de Grado, mis compañeros y yo nos dividimos la tareas y todos participamos en cada una de las fases del proyecto (investigación, desarrollo del sistema propuesto, experimentos y resultados, memoria).

La primera tarea que realicé al empezar este trabajo fue buscar información fiable y verídica en artículos académicos sobre el análisis de tráfico de la red. Utilicé *Google Académico* para la realización de esta tarea y encontré varios *papers* que hablaban sobre este tema. Leí estos artículos y algunas de las referencias que aparecían en ellos y empecé a profundizar también en las herramientas que utilizan para realizar el análisis de tráfico.

Junto con nuestros tutores, nos centrarnos en la tarea de clasificación de aplicaciones móviles de *Android* para el tráfico de red. Al leer estos artículos aprendí diversos conceptos y técnicas que se emplean actualmente para la tarea de clasificación.

Muchos de los trabajos leídos llevan a cabo esta tarea utilizando algoritmos de AA y emplean como lenguaje de programación. Primero, me centré en aprender la sintaxis de este lenguaje de programación debido a que durante la carrera solamente lo había estudiado en la asignatura de sistemas inteligentes. Posteriormente decidí investigar sobre las diferentes librerías de y sus módulos. Muchas de estas librerías nos ayudaron a la hora de implementar la solución propuesta.

Como nuestro trabajo usa algoritmos de AA decidimos utilizar algunas de las bibliotecas que investigué, como por ejemplo: *TensorFlow* (sus diagramas de flujo de datos se utilizan en AA y nosotros la utilizamos en la creación de redes neuronales), *Keras* (la utilizamos en el desarrollo del modelo de AA), *Pandas* (la empleamos para manejar y analizar nuestras estructura de datos), *Numpy* (nos ayudó en el análisis de datos y el cálculo numérico al tener un gran volumen de datos), *Scikit-Learn* (utilizada en la construcción del modelos de AA) y *Sklearn* (utilizada en los algoritmos de AA y el preprocesamiento de los datos), entre otras.

Desde el principio decidimos empezar a crear nuestro propio conjunto de datos, recogiendo muestras de tráfico entre aplicaciones móviles de *Android*. *PCAPdroid* fue la aplicación con la que realizamos las capturas y los archivos .PCAP que devuelve esta aplicación los convertimos a archivos .CSV. Cada uno de nosotros subía las capturas a para aumentar nuestro *dataset*.

Al empezar con la fase de desarrollo del sistema propuesto implementé el preprocesamiento de los datos y la extracción de características con uno de los algoritmos desarrollados por el grupo GASS de nuestra facultad. Posteriormente, mis compañeros y yo desarrollamos la herramienta para la clasificación de aplicaciones móviles de *Android* de tráfico de red. Al realizar la implementación tuvimos que corregir errores en nuestros modelos de AA que detectamos durante las revisiones periódicas que realizábamos y volver a comprobar que el funcionamiento de los modelos era el correcto.

Entre los tres integrantes, repartimos las distintas combinaciones de los parámetros para la realización de los distintos experimentos. Los experimentos los hicimos con distintos valores para el número de neuronas, número de épocas y para los parámetros de los grafos.

Seguidamente he limpiado el código, optimizándolo, quitando comentarios innecesarios o las partes donde imprimíamos por consola, ya que esto nos ayudaba a la hora de desarrollar o en las revisiones.

Una vez hechos todos los experimentos con la batería de combinaciones y se documentaron, pasé a interpretarlos y a sacar las conclusiones.

LaTeX fue la herramienta elegida para escribir la memoria. En la memoria plasmamos todos los pasos que dimos desde el comienzo del proyecto: la recopilación de información sobre el tema de la clasificación de aplicaciones móviles de *Android* de tráfico de red, la implementación de la solución planteada utilizando los algoritmos de AA (SAE, CNN y DGCNN), los experimentos realizados sobre este sistema y las conclusiones de los resultados obtenidos.

A pesar de no haber utilizado esta herramienta nunca, con los recursos encontrados en linea sobre la creación de tablas, como añadir imágenes, citar artículos o cuál es la estructura general de un documento de investigación, fui capaz de redactar la memoria

con relativa facilidad.

Los meses que duró la realización de este proyecto aprendí a trabajar en equipo, a priorizar tareas, pero sobre todo adquirí numerosos conocimientos sobre el análisis de tráfico de red (en concreto la tarea de clasificación de aplicaciones móviles), sobre el AA y los principales algoritmos que se utilizan para esta tarea y sobre la redacción de documentos de investigación.

6.2. Adina Han

Este trabajo de fin de grado lo realicé en conjunto con mis 2 compañeros, aunque durante la realización de este cada uno de nosotros tuvimos también varias tareas individuales. Al comienzo decidimos dividir este trabajo en 4 etapas principales: la investigación, el desarrollo del sistema propuesto, los experimentos y los resultados y la realización de la memoria.

La investigación la realizamos principalmente durante los primeros meses, aunque durante el desarrollo y los experimentos se siguió consultando trabajos académicos. *Google Académico* fue la herramienta principales de busca de artículos científicos y trabajos académicos. Busqué trabajos relacionados con el análisis de tráfico de red, las herramientas que se emplean para la realización del análisis y también las distintas técnicas utilizadas. Los tutores también nos proporcionaron enlaces a trabajos que tienen relevancia para el estudio de este trabajo de fin de grado. De los trabajos que leímos, hemos seleccionado los que mayor relación tienen con la temática propuesta y los resumimos para su utilización posterior a la hora de realizar la memoria, tal y como nos indicaron nuestros tutores. Los enlaces proporcionados también eran para acceder a recursos que nos podían ayudar a tener más conocimiento sobre los algoritmos empleados en estas técnicas.

Las técnicas utilizados para el análisis de tráfico de red que pueden emplearse actualmente con el cambio constante de la red son principalmente técnicas de IA. Las bases de la IA, los distintos tipos de aprendizaje (supervisado, no supervisado, semi-supervisado y por refuerzo) y los algoritmos principales los vi durante la carrera cuando cursé la asignatura de IA y la optativa *Minería de datos*. También repasé los conocimientos que tenía del lenguaje de programación *Python*. Estos fundamentos que adquirí durante la carrera los profundicé mediante la realización de cursos en linea, mediante tutoriales y vídeos. Al final me centré en el AA y sobre todo en DL debido a que los enfoques que utilizan estos algoritmos obtienen mejores resultados, en concreto las redes neuronales, los autoencoders y las redes neuronales convolucionales gráficas. La utilización del lenguaje *Python* ofrece muchas facilidades porque dispone de múltiples librerías, como *TensorFlow*, *Keras*, *Numpy*, *Pandas*, *Scikit-Learn*, *Sklearn*, *StellarGraph* que tienen módulos y funciones implementados que ayuda mucho a la hora de realizar el desarrollo.

La investigación nos ayudó a tener más conocimientos y poder ponerlos en práctica para implementar una herramienta para la clasificación de aplicaciones móviles de Android que utiliza algoritmos de AA, en concreto, las redes neuronales, autoencoders y las redes neuronales convolucionales gráficas. Los trabajos académicos que encontramos los subimos a la herramienta *Drive*, junto con los resúmenes y cualquier información que nos pareció relevante y de ayuda.

La implementación del sistema propuesto se realizó en conjunto. *PyCharm* fue el Integrated Development Environment (IDE) que utilicé y tal y como mencioné antes

el lenguaje de programación *Python* debido a las facilidades que proporciona. *Git* fue la herramienta elegida para el control de versiones porque al desarrollar juntos la herramienta necesitábamos tener en todo momento la versión de código actualizada.

La creación de nuestro propio dataset lo realizamos a la vez con la investigación y el desarrollo. Se recopiló el tráfico de varias aplicaciones móviles de Android utilizando la aplicación *PCAPdroid*. Los archivos .PCAP devueltos por esta aplicación también los guardábamos en el *Drive* para que posteriormente los 3 utilizáramos el mismo conjunto de datos para los experimentos.

El desarrollo del sistema de clasificación de aplicaciones móviles de Android que propusimos tiene 2 etapas: el preprocesamiento de los datos y la creación de un modelo de IA, que a su vez tiene 3 divisiones: modelo basado en el SAE, modelo basado en CNN y modelo basado en DGCNN.

Durante el desarrollo corregimos los errores que surgían en cada revisión y afinamos cada vez más la herramienta. Mientras mi compañero hacía pruebas con el conjunto de datos que íbamos ampliando cada vez más con nuevas capturas, busqué la combinación de los distintos valores que pueden tomar los hiperparámetros (de los grafos y de los algoritmos de AA seleccionados) para obtener los óptimos para nuestro conjunto de datos de entrenamiento.

Realicé distintas pruebas con cada uno de los algoritmos (SAE, CNN y DGCNN) con distintos valores para los parámetros: número de épocas, número de neuronas, utilizando las capas de dropout, número de autoencoders. Una vez realizados los experimentos pertinentes pasé a documentar estos resultados en forma de tablas o matrices de confusión. Esto le ha proporcionado a mi compañero una fácil interpretación de los resultados, permitiéndole sacar las conclusiones sin dificultad.

Mis compañeros y yo realizamos varias revisiones durante la fase de desarrollo y experimentos del sistema para intentar descartar cualquier fallo en la manera de plantear la solución.

Para la redacción de la memoria mis compañeros y yo utilizamos como herramienta LaTeX. En la memoria introducimos la información encontrada durante la investigación, los resúmenes de los trabajos relevantes, una descripción del sistema propuesto y los experimentos y resultados que realizamos con este sistema y las conclusiones a las que llegamos una vez realizadas las pruebas.

Tuvimos reuniones semanales con nuestros tutores mediante *Meet* para ver el progreso de las tareas siguiendo unos plazos establecidos. Además tenía otras reuniones solo con mis compañeros para estar al tanto de los avances de los demás, para ayudarnos mutuamente y resolver todas las dudas que surgían. Al realizar el trabajo en equipo y planificar las tareas nos llevó a poder terminar a tiempo todo lo relacionado con este trabajo de fin de grado.

Durante el desarrollo de este proyecto adquirí muchos conocimientos nuevos, aprendí a centrarme en las tareas importantes en cada momento del proceso y respetar el plazo de realización de estas, aprendí a trabajar en equipo y a buscar soluciones a problemas que nos surgieron durante el desarrollo y también aprendí a buscar información fiable y evaluarla.

6.3. David Merino Mayor

Una vez conocí la propuesta de este Trabajo de Fin de Grado, no dudé en postularme como candidato contactando a los directores de este trabajo. Este trabajo me llamó mucho la atención dado que en él se mezclaban los dos campos que más me habían gustado a lo largo de mi carrera universitaria y de los cuales quería profundizar mis conocimientos: la Ciberseguridad y la Ciencia de los Datos. Mi asignatura favorita de la carrera ha sido *Redes y Seguridad II*, en la cual cuento con Matrícula de Honor, y me gustó mucho la idea de usar los conocimientos que había aprendido en un proyecto real. También, aplicaría las técnicas que me habían enseñado sobre la Ciencia de Datos y la IA en las asignaturas optativas de *Sistemas Inteligentes* y *Minería de Datos*.

Tras conocer que formaba parte de este proyecto que tanto me ilusionaba en el mes de Agosto de 2021, conocí al resto de mis compañeros y tuvimos que comenzar con la primer tarea, basada en planificar por etapas el desarrollo del trabajo y establecer plazos para la realización de las tareas.

En la primera etapa, la tarea de investigar el tema propuesto fue la primera tarea en conjunto. Realizamos esta tarea sobre el análisis de tráfico de red, pero también sobre las herramientas existentes que se usan para llevar a cabo esta tarea. Descubrí que en la actualidad hay muchos enfoques que se le da a esto y toman en cuenta los protocolos de seguridad. Por este motivo, decidí investigar un poco más sobre los principales protocolos de seguridad y los usos que tienen. Leí sobre los siguientes protocolos: SSL y TLS, SSH, SET, IPsec y HTTPS. Los artículos sobre análisis de tráfico de red, protocolos de seguridad y clasificación de aplicaciones los encontré en *Google Académico*, que nos fue de mucha ayuda en esta etapa. En esta etapa empezamos a crear nuestro *dataset*, con muestras obtenidas de la aplicación *PCAPdroid*. Esta aplicación permite obtener capturas del tráfico de red entre aplicaciones móviles de *Android* en formato .PCAP. El *dataset* de las muestras lo ampliamos durante los meses que duró el proyecto.

Mientras tanto, al haber sido informado de que usaríamos el lenguaje de programación Python para la implementación del proyecto, me decidí por realizar un curso de Algoritmos y Estructuras de Datos en Python en la plataforma de aprendizaje *Codecademy*.

La segunda etapa del proyecto fue el desarrollo de una herramienta que fuera capaz de analizar el tráfico de red y clasificarlo adecuadamente como tráfico de una de las aplicaciones de las cuales se capturó el tráfico. En la tarea de investigación observamos que el enfoque que mejores resultados obtiene es el uso de técnicas basada en algoritmos de AA y sobre todo los algoritmos de DL. Por este motivo decidimos que el sistema propuesto implemente 3 de los algoritmos de AA: las CNN, SAE y DGCNN.

A lo largo de la etapa de desarrollo realicé varias pruebas con *datasets* de distintos tamaños y observé la importancia de haber creado un conjunto de datos propio, con datos reales, los cuales mejorarían nuestros resultados más adelante.

Tras investigar sobre las arquitecturas en las que teníamos que basar nuestros modelos, comenzamos a implementarlos en Python. Esta tarea, no fue nada fácil, ya que contábamos con un *dataset* propio, el cual tuvimos que preprocesar para hacer uso de los grafos y, después, procesarlos de nuevo como entrada al modelo teniendo en cuenta las diferentes arquitecturas propuestas.

Una vez implementamos la arquitectura de las capas de nuestros modelos, después tuvimos que realizar experimentos con configuraciones distintas de parámetros, tanto de

los propios algoritmos como del conjunto de datos o de parámetros de red. Tras configurar dichos valores, pasamos a entrenar el modelo y a obtener las predicciones. Esta tarea fue mucho más rápida, que no sencilla, debido a la coordinación con el resto de mis compañeros.

Para poder interpretar los resultados logrados, busqué las métricas que mejor se adapten a los algoritmos utilizados y al *dataset* que tenemos. En varias ocasiones, subsanamos errores encontrados en la creación del modelo al observar los malos resultados obtenidos en las primeras pruebas. Con la métricas que encontré, mi compañera documentó los resultados y mi compañero los interpretó. A partir de aquí, sacamos las conclusiones pertinentes y pensamos en futuras mejoras que podríamos aplicarle a nuestro sistema.

La memoria fue la última tarea que realizamos y también fue una tarea que se realizó en grupo y utilizamos el sistema de composición de textos LaTeX para la redacción de la memoria. Aprovecho para dar las gracias al grupo GASS por proporcionarnos una aplicación web con un sistema de LaTeX integrado, la cual nos permitió poder trabajar de forma colaborativa y en línea con el resto de mis compañeros.

En cada etapa de este trabajo logré adquirir nuevos conocimientos sobre distintos temas técnicos, como por ejemplo: el AA y sus principales algoritmos, recolección de flujos de tráfico de red y su posterior análisis, conocimiento en protocolos de seguridad en red, ciberseguridad o manejo de LaTeX, entre un largo etcétera. Además, obtuve muchos conocimientos relativos a la investigación y posterior documentación y redacción de dichos trabajos de investigación. Pero también aprendí a respetar los plazos establecidos de las tareas, trabajar en conjunto con un equipo de investigación y con el resto de mis compañeros.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

En este trabajo, se propuso una herramienta que analizase el tráfico de red y pudiese clasificarlo como tráfico de una aplicación concreta de *Android*. Es decir, desarrollar unos modelos de aprendizaje de los que se pudiesen obtener unos resultados con una buena precisión.

Analizando los resultados obtenidos de cada una de las arquitecturas de DL propuestas y descritas en la Sección 5, se concluye que el mejor modelo en términos de complejidad y métricas finales como el *f1_score* es el CNN. Este modelo obtiene unos resultados del **98 %** de exactitud al tener un *dataset* equilibrado con 9 clases diferentes, a pesar de que la carga no esté completamente balanceada. Sin embargo, si el *dataset* no está equilibrado y cuenta con un mayor número de clases (16 aplicaciones) se obtiene un resultado más bajo, con una métrica de *f1_score* de media del **83 %**.

El modelo utilizado del SAE se descarta por su gran complejidad y tiempo de ejecución. Su estructura de apilamiento de los *autoencoders* para obtener el clasificador ha hecho que su implementación sea muy compleja.

Además de eso, analizando la métrica de *f1_score* del SAE, esta es menor que la del modelo de CNN. Con el *dataset* de nueve aplicaciones, se obtiene un resultado de la métrica de *f1_score* de **73 %**. Las pruebas realizadas con el *dataset* completamente desequilibrado y con todas las aplicaciones existentes, se obtiene un resultado del **56 %**. Por lo tanto, estos porcentajes son bastante menores que el modelo elegido, que es el CNN.

En cuanto a las conclusiones del modelo de referencia basado en DGCNN, aunque el tiempo de ejecución debido a los grafos es muy grande, es mejor que el modelo de SAE debido a su menor complejidad.

Sin embargo, si se compara el modelo de CNN con el modelo de DGCNN, existen diversas ventajas. Si se tiene en cuenta la complejidad o el tiempo de ejecución del modelo debido al uso de grafos, el modelo de redes neuronales convolucionales gráficas sale perdiendo. Además, si tenemos también en cuenta métricas como por ejemplo la de *f1_score*, también sale perdiendo ya que es menor. Mientras con el modelo de DGCNN se obtiene un resultado del **66 %** para todas las aplicaciones del *dataset*, con el modelo de CNN se obtiene un **83 %**.

Como conclusiones generales de todos los resultados, se observa que cuanto más grande

es el *dataset*, mejores resultados se obtienen cuando están equilibrados. Además, el tiempo de ejecución de cada modelo, depende mucho de la potencia del ordenador en el que se esté ejecutando.

En este trabajo se ha reparado en que cualquier modelo de aprendizaje es diferente y cada uno tiene unos parámetros diferentes para llegar a dar los mejores resultados.

Existen parámetros que son compartidos por varios modelos. Sin embargo, cada uno de ellos se ajusta a un determinado valor para dar los mejores resultados posibles.

7.2. Trabajo Futuro

Como posibles trabajos futuros pueden señalarse los siguientes:

- Ampliar el *dataset*, tanto en número de muestras como en número de aplicaciones a clasificar. Esto permitirá, además de poder clasificar más flujos de tráfico de red, poder contar con mejores métricas en los modelos ya que aumentará las muestras.
- Tratar de balancear el *dataset* capturando más tráfico de las aplicaciones minoritarias (menos muestras actualmente).
- Realizar los experimentos sobre una Graphics Processing Unit (GPU) para reducir el tiempo de entrenamiento y poder realizar más experimentos.
- Utilizar una combinación de otros algoritmos de AA.
- Crear una aplicación con interfaz de usuario que contenga el sistema propuesto.
- Utilizar el sistema propuesto para la detección de intrusos o para la detección del tráfico de *malware*.

Capítulo 8

Introduction

8.1. Motivation

In the last decade, network traffic has increased dramatically, compromising the privacy and security of both citizens and businesses. Therefore, the need has arisen to implement new security controls that make the task of analysing the network and, in particular, the classification of mobile applications based on network traffic, more difficult.

The most well-known techniques to perform such a task are: deep packet inspection (DPI), port number based methods (*port number*), payload inspection (*payload*) and methods based on AA, among others. As the characteristics of the network environment are constantly changing, these solutions quickly become obsolete. Deep learning DL is a new approach that has been used in recent years for network traffic classification.

8.2. Context

This Final Degree Project was developed in the GASS (`https://gass.ucm.es`) of the Department of Software Engineering and Artificial Intelligence of the Faculty of Computer Science of the UCM.

8.3. Subject of the Research

The aim of this project is the study and classification of Android mobile applications. For this purpose, it is based on other research works found in the literature, using algorithms such as *Autoencoders*, CNN and DGCNN algorithms for classification.

Another objective of the research is to check which of the generated models provides a higher accuracy in the data information. For this purpose, some hyperparameters of the different DL architectures mentioned above are modified, such as, for example, the number of neurons, the number of layers, the number of epochs and the activation functions, among others.

8.4. Work Plan

The following phases have been carried out for the development of this work:

1. **Research**

   The research phase has been carried out during the first months of work, and the general topic has been the traffic analysis in Android applications (network analysis, protocols, traffic analysis techniques) and the techniques used to carry out this task (supervised, unsupervised, semi-supervised, semi-supervised, semi-supervised, semi-supervised, semi-supervised, semi-supervised, semi-supervised), unsupervised, semi-supervised, the different algorithms of AA and DL such as classification algorithms, regression algorithms, algorithms based on decision trees, clustering and neural networks, among others).

   With the information found in the works, the final topic has been gradually defined, finally focusing on the classification of mobile applications in Android. During the research, the different works that have served as a basis for the development of this project have been identified.

   The articles in scientific journals, which were found in the search engine specialised in academic documents, Google Scholar, and the references used in these articles have been the knowledge base from which we have started. During the following phases, the research continued but at a slower pace.

2. **Development**

   The initial part of the development was carried out at the same time as the research phase. The academic papers found in the research phase have been the starting point in the development of the tool.

   The models of AA have been incorporated in the implementation of the system proposed by this work. For this part, we have used the methods included in the State of the Art ([Lotfollahi et al., 2020] or [Zhang et al., 2018], among others) in Chapter 3: SAE, CNN, and DGCNN. At the same time, network traffic captures have been made in PCAP format in order to have our own dataset.

   This phase has been carried out using the Python programming language and its respective libraries, such as Numpy, Keras, Pandas, Scikit-Learn, TensorFlow, PyTorch, Sklearn or Matplotlib.

3. **Experiments and results**.

   In this last phase, experiments have been carried out with different parameters for the selected algorithms in order to obtain the combination that provides the best results. The results obtained have been analysed on the basis of different metrics and interpreted in order to draw the conclusions presented in this work.

The workflow can be seen in the Figure 8.1.

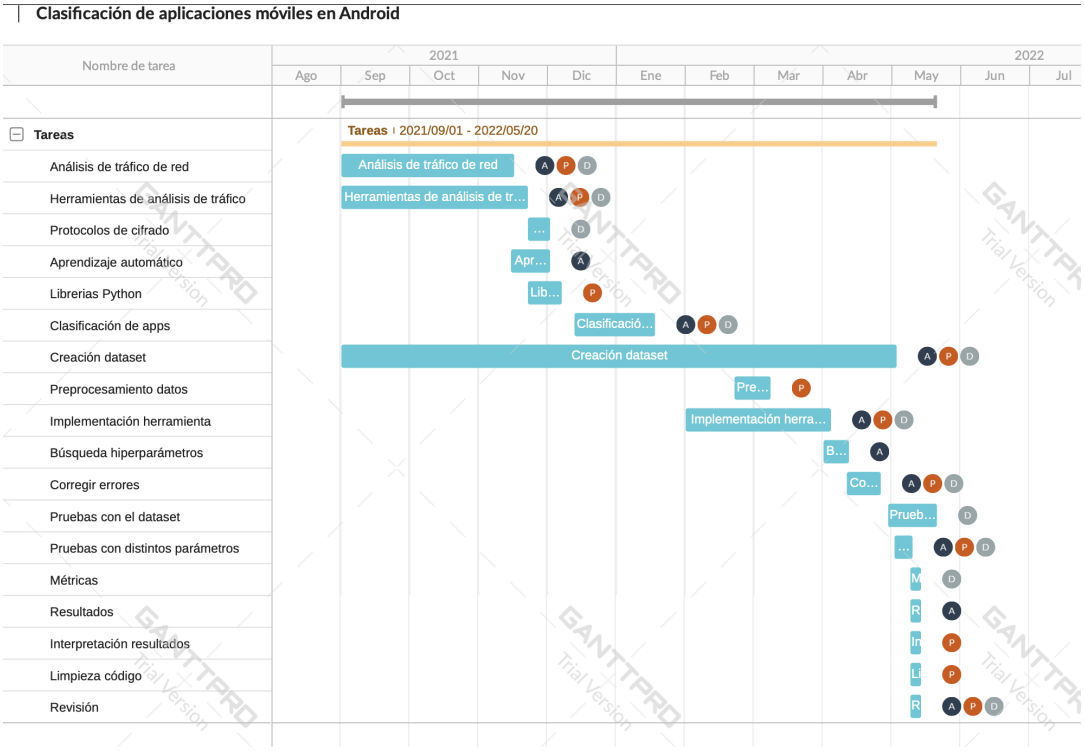

Figura 8.1: Gantt chart of the work flow

8.5. Structure of the Work

The memory of this work is organised as follows:

The Chapter 2 presents the overview of the evolution of network traffic analysis, the most popular security protocols and the algorithms of AA used to perform the analysis.

The Chapter 3 introduces the current methods used for traffic classification in Android mobile applications and their performance by comparing the different techniques in related works.

The proposed system is detailed in the Chapter 4 and the experiments and results obtained are formulated in the Chapter 5.

The contributions of each member of the group are detailed in the Chapter 6.

Finally, conclusions and future work are summarised in the Chapter 7.

The Chapter 8 and the Chapter 9 represent the English translations of the Introduction, Conclusions and Future Work.

Capítulo 9

Conclusions and Future Work

9.1. Conclusions

In this work, we proposed a tool that could analyse network traffic and classify it as traffic of a specific Android application. In other words, to develop learning models from which results could be obtained with good accuracy.

Analysing the results obtained from each of the proposed DL architectures described in Section 5, it is concluded that the best model in terms of complexity and final metrics such as f1_score is the CNN model. This model obtains **98 %** accuracy results by having a balanced dataset with 9 different classes, despite the fact that the load is not completely balanced. However, if the dataset is not balanced and has a larger number of classes (16 applications) a lower result is obtained, with an average f1_score metric of **83 %**.

The used model of the SAE is discarded due to its high complexity and execution time. Its structure of stacking the to obtain the classifier has made its implementation very complex.

Besides that, analysing the metric of f1_score, it is lower than that of the CNN model. With the dataset balanced using nine applications, a result of **73 %** is obtained for the f1_score metric. Tests performed with the unbalanced dataset and with all existing applications, a result of **56 %** is obtained. Therefore, these percentages are significantly lower than the chosen model, which is the CNN.

As general conclusions of all the results, it can be seen that the larger the dataset, the better the results obtained when they are balanced. Furthermore, the execution time of each model depends a lot on the power of the computer on which it is being executed.

In this work it has been noted that any learning model is different and each one has different parameters to achieve the best results.

There are parameters that are shared by several models. However, each of them is adjusted to a certain value to give the best possible results.

9.2. Future Work

Possible future work may include the following:

- Expand the dataset, both in number of samples and in number of applications to be classified. This will allow, in addition to being able to classify more network traffic flows, to have better metrics in the models as it will increase the samples.
- Try to balance the dataset by capturing more traffic from minority applications (currently fewer samples).
- Run the experiments on a GPU to reduce training time and be able to run more experiments.
- Use a combination of other algorithms from AA.
- Create an application with user interface containing the proposed system.
- Use the proposed system for intrusion detection or for detection of malware traffic.

Bibliografía

[Aceto et al., 2018] Aceto, G., Ciuonzo, D., Montieri, A., and Pescapé, A. (2018). Mobile encrypted traffic classification using deep learning. In *2018 Network traffic measurement and analysis conference (TMA)*, pages 1–8. IEEE.

[act, 2018] act (2018). Funciones de activación. https://www.codificandobits.com/.

[Alqudah and Yaseen, 2020] Alqudah, N. and Yaseen, Q. (2020). Machine learning for traffic analysis: a review. *Procedia Computer Science*, 170:911–916.

[Alzubi et al., 2018] Alzubi, J., Nayyar, A., and Kumar, A. (2018). Machine learning from theory to algorithms: an overview. In *Journal of physics: conference series*, volume 1142, page 012012. IOP Publishing.

[Antonello et al., 2012] Antonello, R., Fernandes, S., Kamienski, C., Sadok, D., Kelner, J., Godor, I., Szabo, G., and Westholm, T. (2012). Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. *Journal of Network and Computer Applications*, 35(6):1863–1878.

[Aouedi et al., 2020] Aouedi, O., Piamrat, K., and Bagadthey, D. (2020). A semi-supervised stacked autoencoder approach for network traffic classification. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE.

[Bartos et al., 2016] Bartos, K., Sofka, M., and Franc, V. (2016). Optimized invariant representation of network traffic for detecting unseen malware variants. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 807–822.

[Blowers and Williams, 2014] Blowers, M. and Williams, J. (2014). *Machine Learning Applied to Cyber Operations*, pages 155–175. Springer New York, New York, NY.

[Bujlow et al., 2012] Bujlow, T., Riaz, T., and Pedersen, J. M. (2012). Classification of http traffic based on c5. 0 machine learning algorithm. In *2012 IEEE Symposium on Computers and Communications (ISCC)*, pages 000882–000887. IEEE.

[Cao et al., 2014] Cao, Z., Xiong, G., Zhao, Y., Li, Z., and Guo, L. (2014). A survey on encrypted traffic classification. In *International Conference on Applications and Techniques in Information Security*, pages 73–81. Springer.

[CNN, 2018] CNN (2018). Redes neuronales. https://www.aprendemachinelearning.com/.

[eduardocollado, 2020] eduardocollado (2020). ssh. https://www.eduardocollado.com/2020/08/28/algoritmos-de-cifrado-en-ssh/.

[Encaminamiento, 2021] Encaminamiento (2021). Tipos de encaminamiento. https://docs.oracle.com/cd/E19957-01/820-2981/gdyen/index.html.

[Essien and Giannetti, 2020] Essien, A. and Giannetti, C. (2020). A deep learning model for smart manufacturing using convolutional lstm neural network autoencoders. *IEEE Transactions on Industrial Informatics*, 16(9):6069–6078.

[Faranda., 2020] Faranda., E. (2020). PCAPdroid: User Guide. https://emanuele-f.github.io/PCAPdroid/quick_start.html.

[Ferri et al., 2009] Ferri, C., Hernández-Orallo, J., and Modroiu, R. (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38.

[Furno et al., 2017] Furno, A., Fiore, M., and Stanica, R. (2017). Joint spatial and temporal classification of mobile traffic demands. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press.

[Hootsuite, 2022] Hootsuite (2022). Hootsuite. https://www.hootsuite.com/.

[IBM, 2021] IBM (2021). TLS. https://www.ibm.com/docs/es/ibm-mq/9.1?topic=tls-overview-ssltls-handshake.

[ids, 2020] ids (2020). IDS. https://www.incibe.es/protege-tu-empresa/blog/son-y-sirven-los-siem-ids-e-ips.

[Ji and Meng, 2020] Ji, X. and Meng, Q. (2020). Traffic classification based on graph convolutional network. In *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pages 596–601. IEEE.

[Laskov et al., 2005] Laskov, P., Düssel, P., Schäfer, C., and Rieck, K. (2005). Learning intrusion detection: supervised or unsupervised? In *International Conference on Image Analysis and Processing*, pages 50–57. Springer.

[Letsencrypt, 2021] Letsencrypt (2021). Let's encrypt. https://letsencrypt.org/.

[Lotfollahi et al., 2020] Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R., and Saberian, M. (2020). Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012.

[Ma et al., 2018] Ma, X., Sha, J., Wang, D., Yu, Y., Yang, Q., and Niu, X. (2018). Study on a prediction of p2p network loan default based on the machine learning lightgbm and xgboost algorithms according to different high dimensional data cleaning. *Electronic Commerce Research and Applications*, 31:24–39.

[Madhulatha, 2012] Madhulatha, T. S. (2012). An overview on clustering methods. *arXiv preprint arXiv:1205.1117*.

[Mahesh, 2020] Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386.

[Mirsky et al., 2018] Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. (2018). Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*.

[Muhammad and Yan, 2015] Muhammad, I. and Yan, Z. (2015). Supervised machine learning approaches: A survey. *ICTACT Journal on Soft Computing*, 5(3).

[Mukkamala et al., 2002] Mukkamala, S., Janoski, G., and Sung, A. (2002). Intrusion detection: support vector machines and neural networks. In *proceedings of the IEEE International Joint Conference on Neural Networks (ANNIE), St. Louis, MO*, pages 1702–1707.

[Nasteski, 2017] Nasteski, V. (2017). An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62.

[Nian et al., 2020] Nian, R., Liu, J., and Huang, B. (2020). A review on reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering*, 139:106886.

[Papadogiannaki and Ioannidis, 2021] Papadogiannaki, E. and Ioannidis, S. (2021). A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys (CSUR)*, 54(6):1–35.

[Paulson, 2002] Paulson, L. C. (2002). Verifying the set protocol: Overview. *Formal Aspects of Security*, pages 4–14.

[Pham et al., 2021] Pham, T.-D., Ho, T.-L., Truong-Huu, T., Cao, T.-D., and Truong, H.-L. (2021). MAppGraph: Mobile-App Classification on Encrypted Network Traffic using Deep Graph Convolution Neural Networks. In *Annual Computer Security Applications Conference (ACSAC 2021)*, Virtual Conference.

[PK, 1984] PK, F. A. (1984). What is artificial intelligence? *"Success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do"*., page 65.

[Prandini et al., 2010] Prandini, M., Ramilli, M., Cerroni, W., and Callegati, F. (2010). Splitting the https stream to attack secure web connections. *IEEE Security & Privacy*, 8(6):80–84.

[Radivilova et al., 2018] Radivilova, T., Kirichenko, L., Ageyev, D., Tawalbeh, M., and Bulakh, V. (2018). Decrypting ssl/tls traffic for hidden threats detection. In *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pages 143–146.

[Reddy et al., 2018] Reddy, Y., Viswanath, P., and Reddy, B. E. (2018). Semi-supervised learning: A brief review. *Int. J. Eng. Technol*, 7(1.8):81.

[SE, 2013] SE, V. E. (2013). Survey of traffic classification using machine learning. *International journal of advanced research in computer science*, 4(4).

[Shinde and Shah, 2018] Shinde, P. P. and Shah, S. (2018). A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*, pages 1–6. IEEE.

[Shrestha and Mahmood, 2019] Shrestha, A. and Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE access*, 7:53040–53065.

[Sommer and Paxson, 2010] Sommer, R. and Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316.

[Song et al., 2019] Song, M., Ran, J., and Li, S. (2019). Encrypted traffic classification based on text convolution neural networks. In *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, pages 432–436. IEEE.

[Stellargraph, 2021] Stellargraph (2021). StellarGraph - Supervised graph classification with Deep Graph CNN. https://stellargraph.readthedocs.io/en/stable/demos/graph-classification/dgcnn-graph-classification.html?highlight=paddedgraphgenerator.

[Suthaharan, 2014] Suthaharan, S. (2014). Big data classification: Problems and challenges in network intrusion prediction with machine learning. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):70–73.

[Tahaei et al., 2020] Tahaei, H., Afifi, F., Asemi, A., Zaki, F., and Anuar, N. B. (2020). The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications*, 154:102538.

[Taylor et al., 2016] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. (2016). Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454. IEEE.

[TCP/IP, 2021] TCP/IP (2021). Arquitectura protocolos. https://www.ibm.com/docs/es/aix/7.2?topic=protocol-tcpip-protocols.

[Umadevi and Marseline, 2017] Umadevi, S. and Marseline, K. J. (2017). A survey on data mining classification algorithms. In *2017 International Conference on Signal Processing and Communication (ICSPC)*, pages 264–268. IEEE.

[van Ede et al., 2020] van Ede, T., Bortolameotti, R., Continella, A., Ren, J., Dubois, D. J., Lindorfer, M., Choffnes, D., van Steen, M., and Peter, A. (2020). Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and Distributed System Security Symposium (NDSS)*, volume 27.

[Velan et al., 2015] Velan, P., Čermák, M., Čeleda, P., and Drašar, M. (2015). A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5):355–374.

[Vieira et al., 2020] Vieira, S., Pinaya, W. H. L., and Mechelli, A. (2020). Introduction to machine learning. In *Machine Learning*, pages 1–20. Elsevier.

[Wang et al., 2020] Wang, M., Zheng, K., Luo, D., Yang, Y., and Wang, X. (2020). An encrypted traffic classification framework based on convolutional neural networks and stacked autoencoders. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 634–641. IEEE.

[Wang et al., 2019] Wang, P., Chen, X., Ye, F., and Sun, Z. (2019). A survey of techniques for mobile service encrypted traffic classification using deep learning. *IEEE Access*, 7:54024–54033.

[Wang et al., 2017] Wang, W., Zhu, M., Wang, J., Zeng, X., and Yang, Z. (2017). End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE international conference on intelligence and security informatics (ISI)*, pages 43–48. IEEE.

[Yang et al., 2018] Yang, Y., Kang, C., Gou, G., Li, Z., and Xiong, G. (2018). Tls/ssl encrypted traffic classification with autoencoder and convolutional neural network. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 362–369. IEEE.

[Zamani et al., 2009] Zamani, M., Movahedi, M., Ebadzadeh, M., and Pedram, H. (2009). A ddos-aware ids model based on danger theory and mobile agents. In *2009 International Conference on Computational Intelligence and Security*, volume 1, pages 516–520. IEEE.

[Zhang et al., 2012] Zhang, H., Banick, W., Yao, D., and Ramakrishnan, N. (2012). User intention-based traffic dependence analysis for anomaly detection. In *2012 IEEE Symposium on Security and Privacy Workshops*, pages 104–112.

[Zhang et al., 2018] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence*.

[Zhang et al., 2019] Zhang, S., Tong, H., Xu, J., and Maciejewski, R. (2019). Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23.