UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TRABAJO DE FIN DE GRADO

Algoritmos de Aprendizaje Automático aplicados a problemas de Ciberseguridad

Presentado por: Pablo Jiménez Poyatos

Dirigido por: Luis Fernando Llana Diaz

Grado en Matemáticas

Curso académico 2023-24

Agradecimientos

Resumen

Palabras clave:

Abstract

Keywords:

Índice general

1. Introducción **1**

1.1. Motivación y objetivos del trabajo 1

1.2. Contexto y antecedentes del trabajo 1

1.2.1. Redes neuronales 1

1.2.2. Importancia de la detección y prevención de ataques 1

1.2.3. Evolución de las amenazas cibernéticas 1

1.2.4. Avances en el aprendizaje automático para ciberseguridad 1

1.3. Estructura de la memoria 1

1.4. Contribuciones 1

2. Fundamentos de las redes neuronales **3**

2.1. Revisión teórica 3

2.2. Arquitecturas relevantes 3

2.2.1. Autoencoder 3

2.2.2. Deep Belief Networks 4

2.2.3. Red Neuronal Convolucional 4

2.2.4. Red Neuronal Recurrente 4

2.3. Implementación en Python 4

2.3.1. Frameworks 4

2.4. Principales Deep Learning Frameworks. Keras. 4

3. Aplicación en la ciberseguridad **7**

3.1. Clasificación de Malware 7

3.1.1. Microsoft Malware Classification Challenge 7
3.2. Resumen del Conjunto de Datos 10
3.2.1. Autoencoder 12
3.2.2. Red Neuronal Convolucional 12
3.2.3. Resultados 12
3.3. Detección de intrusiones 12
3.3.1. KDD Cup 1999 12
3.3.2. Autoencoder 12
3.3.3. Red Neuronal Convolucional 12
3.3.4. Red Neuronal Profunda 13
3.3.5. Red Neuronal Recurrente 13
3.3.6. Restricted Boltzmann Machine 13
3.3.7. Resultados 13

4. Conclusiones y Trabajo Futuro **15**
4.1. Conclusiones 15
4.2. Trabajo futuro 15

Bibliografía **17**

Capítulo 1

Introducción

1.1. Motivación y objetivos del trabajo

1.2. Contexto y antecedentes del trabajo

1.2.1. Redes neuronales

1.2.2. Importancia de la detección y prevención de ataques

Destaca la importancia crítica de la detección y prevención de ataques cibernéticos en entornos empresariales y gubernamentales, así como en la protección de datos sensibles y la infraestructura crítica.

1.2.3. Evolución de las amenazas cibernéticas

Describe brevemente cómo han evolucionado las amenazas en el ámbito de la ciberseguridad a lo largo del tiempo, desde virus simples hasta ataques sofisticados como el ransomware y el phishing.

1.2.4. Avances en el aprendizaje automático para ciberseguridad

Proporciona una visión general de cómo los algoritmos de aprendizaje automático han revolucionado el campo de la ciberseguridad, permitiendo la detección temprana de amenazas, el análisis de comportamiento anómalo y la automatización de respuestas.

1.3. Estructura de la memoria

1.4. Contribuciones

Capítulo 2

Fundamentos de las redes neuronales

2.1. Revisión teórica

Puedo introducir los tipos de funciones de activavion. Esta bien explicado en el TFG wuolah o en el articulo de KDD cup 199 de DNN network intrusion. Puedo añadir overfitting y underfitting. lo que es aprendizaje supervisado y no supervisao Partes de una neurona y como trabaja(bias, pesos...)

2.2. Arquitecturas relevantes

Mini tabla resumen en Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective y miniresumen de todos los tipos en review Deep Cybersecurity: A Comprehensive Overview from Neural Networkand Deep Learning Perspective y review

2.2.1. Autoencoder

Leer y sacar la información del word autoencoders.

Los autoencoders son una clase de redes neuronales artificiales utilizadas en aprendizaje no supervisado para aprender representaciones eficientes de datos. Su objetivo principal es codificar la entrada en una representación comprimida y significativa, y luego decodificarla de manera que la reconstrucción sea lo más similar posible a la entrada original.

La arquitectura básica de un autoencoder consta de dos partes: el encoder y el decoder. El encoder mapea los datos de entrada a una representación oculta de menor dimensión utilizando funciones principalmente no lineales, mientras que el decoder reconstruye los datos de entrada a partir de esta representación oculta. Durante el entrenamiento, los parámetros del autoencoder se optimizan para minimizar la diferencia entre la entrada y la salida reconstruida, utilizando una función de pérdida que mide esta discrepancia.

Los autoencoders se han utilizado en una amplia variedad de aplicaciones, incluida la reducción de dimensionalidad, la extracción de características, la eliminación de ruido en los datos de entrada y la detección de anomalías. Su versatilidad y capacidad para aprender representaciones

útiles de los datos los hacen herramientas poderosas en el campo del aprendizaje automático y la inteligencia artificial.

2.2.2. Deep Belief Networks

Red Neuronal Profunda

2.2.3. Red Neuronal Convolucional

2.2.4. Red Neuronal Recurrente

Restricted Boltzmann Machine

2.3. Implementación en Python

2.3.1. Frameworks

numpy matplotlib sklearn

Deep neural network

2.4. Principales Deep Learning Frameworks. Keras.

A medida que las técnicas de aprendizaje profundo han ido ganando popularidad, muchas organizaciones académicas e industriales se han centrado en desarrollar marcos de trabajo para facilitar la experimentación con redes neuronales profundas de manera sencilla para el usuario. En esta sección, ofrecemos una visión general de los bibliotecas más conocidss: TensorFlow, PyTorch y Keras.

TensorFlow [10] es una biblioteca de código abierto que fue desarrollada en 2016 por el equipo de Google Brain del departamento de IA de Google. Es una plataforma que se centra principalmente en el aprendizaje automático y la computación de alto rendimiento. Su modelo de programación permite que los datos fluyan de una operación a otra de manera flexible gracias a su estructura de grafo computacional y al concepto de flujo de datos. Los gráficos son estructuras de datos que contienen un conjunto de objetos tf.Operation (nodos), que representan unidades de cálculo y objetos tf.Tensor [1], que representan las unidades de datos que fluyen entre esas operaciones (Figura 2.1). Esto permite una ejecución eficiente y paralela de operaciones en diferentes dispositivos de hardware, como en una CPU, GPU, dispositivos móviles y sistemas distribuidos a

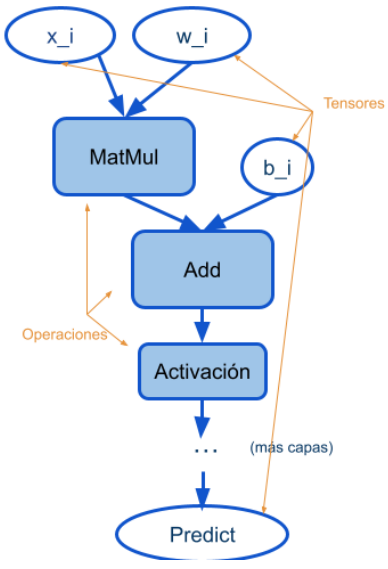

Figura 2.1: Gráfico de una red neuronal.

[1] arreglos multidimensionales que pueden contener datos de cualquier tipo y forma

gran escala con cientos de nodos. Esta flexibilidad y además su
capacidad de diferenciación automática lo hacen una biblioteca útil para una gran variedad de aplicaciones, desde la investigación hasta el desarrollo y la producción de modelos de aprendizaje automático.

PyTorch [7], presentado por el equipo de investigación en IA de Facebook en 2016, es una biblioteca de aprendizaje profundo desarrollado en Python que simplifica la creación de modelos complejos a través de una interfaz de programación sencilla. A diferencia de otros marcos populares que emplean gráficos de computación estáticos, PyTorch se basa en la computación dinámica (su topología puede variar durante la ejecución del programa), lo que permite una mayor flexibilidad para diseñar arquitecturas complejas. Cambiar el comportamiento de una red neuronal típicamente implica reiniciar desde cero, pero PyTorch emplea una técnica llamada auto-diferenciación en modo inverso, que permite realizar cambios en el comportamiento de la red con poco esfuerzo. PyTorch se ha vuelto popular tanto en la comunidad científica como en la industria debido a su facilidad de uso y su capacidad para crear modelos complejos de manera eficiente. Además, ha sido adoptado por varias organizaciones importantes, incluidas Facebook, Twitter y NVIDIA, lo que garantiza su continuo desarrollo y soporte.

Keras [9] es un marco de redes neuronales de código abierto desarrollado por François Cholle, un miembro del equipo de IA de Google. Se considera un meta-marco de trabajo que interactúa con otros marcos. En particular puede ejecutarse en la parte superior de TensorFlow y Theano. Está implementado en Python y proporciona APIs de redes neuronales de alto nivel para desarrollar modelos de aprendizaje profundo. En lugar de manejar operaciones de bajo nivel (diferenciación y manipulación de tensores), Keras depende de una biblioteca especializada que sirve como su motor de backend. Keras minimiza el número de acciones requeridas por un usuario para una acción específica. Una característica importante de Keras es su facilidad de uso sin sacrificar la flexibilidad. Keras permite a los usuarios implementar sus modelos como si estuvieran implementados en los marcos base (como TensorFlow o Theano[2]). Proporciona un rendimiento y escalabilidad de nivel industrial, siendo utilizado por organizaciones de renombre como NASA, YouTube y Waymo para una amplia gama de aplicaciones en inteligencia artificial y aprendizaje automático.

Analizando los resultados de [17], podemos observar que usando la CPU, Keras destaca por encima de las demás. No solo logra el mejor accuracy en los tres datasets (MNIST, CIFAR-10, CIFAR-100), sino que además también tiene los tiempos de ejecución más bajos y una de las mejores tasas de convergencia. Estos resultados junto con su facilidad de uso, accesibilidad y documentación bien estructurada, me hacen llegar a la conclusión de que utilizar Keras en mi investigación es la mejor opción. Según Matthew Carrigan,

> The best thing you can say about any software library is that the abstractions it chooses feel completely natural, such that there is zero friction between thinking about what you want to do and thinking about how you want to code it. That's exactly what you get with Keras.

Aakash Nain

> Keras is that sweet spot where you get flexibility for research and consistency for deployment. Keras is to Deep Learning what Ubuntu is to Operating Systems.

[2]Otro framework de Python para aprendizaje automático.

Capítulo 3

Aplicación en la ciberseguridad

3.1. Clasificación de Malware

Hoy en día, uno de los principales retos que enfrenta el software anti-malware es la enorme cantidad de datos y archivos que se requieren evaluar en busca de posibles amenazas maliciosas. Una de las razones principales de este volumen tan elevado de archivos diferentes es que los creadores de malware introducen variaciones en los componentes maliciosos para evadir la detección. Esto implica que los archivos maliciosos pertenecientes a la misma "familia" de malware (con patrones de comportamiento similares), se modifican constantemente utilizando diversas tácticas, lo que hace que parezcan ser múltiples archivos distintos.

Para poder analizar y clasificar eficazmente estas cantidades masivas de archivos, es necesario agruparlos e identificar sus respectivas familias. Además, estos criterios de agrupación pueden aplicarse a nuevos archivos encontrados en computadoras para detectarlos como maliciosos y asociarlos a una familia específica.

3.1.1. Microsoft Malware Classification Challenge

El conjunto de datos utilizado en este estudio proviene del Microsoft Malware Classification Challenge (BIG 2015) [1], una competición dirigida a la comunidad científica. Su objetivo era promover el desarrollo de técnicas efectivas para agrupar diferentes variantes de malware en sus respectivas familias. Se puede descargar desde su página web `https://www.kaggle.com/c/malware-classification`, pero hay que tener en cuenta su espacio, 0.5 TB sin comprimir. Para usarla, me descargué la carpeta comprimida (7z) con todo el dataset. Después, la subí al servidor Simba de la facultad de informática y finalmente, usando el comando *7zz x file_ name.7z*, la descomprimí.

Este dataset contiene 5 archivos:

- dataSample.7z - Una carpeta comprida 7z con una muestra de los datos disponibles.
- train.7z - Una carpeta comprida 7z con los datos sin procesar para el conjunto de entrenamiento.
- trainLabels.csv - Un archivo csv con las etiquetas asociadas a cada archivo del conjunto de entrenamiento.

- test.7z - Una carpeta comprida 7z con los datos sin procesar para el conjunto de prueba.
- sampleSubmission.csv - Un archivo csv que muestra el formato de envío válido de las soluciones.

Para nuestro estudio, nos enfocaremos exclusivamente en el conjunto de datos de entrenamiento, que consta de los archivos "train.7z" y "trainLabels.csv". Los archivos 'test.7z' y 'sampleSubmission.csv' están destinados específicamente para la competición. Nosotros no los utilizaremos debido a que son programas de malware sin etiquetar y para entrenar nuestras redes neuronales en este problema de clasificación de malware, es necesario conocer sus clases. Además, la carpeta 'dataSample.7z' proporciona dos programas que se encuentran también en la carpeta train.7z, por lo que tampoco la utilizaremos en nuestro experimento.

Cada programa malicioso tiene un identificador, un valor hash de 20 caracteres que identifica de forma única el archivo, y una etiqueta de clase, que es un número entero que representa uno de los 9 nombres de familias a los que el malware puede pertenecer 3.1. Cada programa contiene dos archivos. Uno asm con el código extraído por la herramienta de desensamblado IDA y otro bytes con la representación hexadecimal del contenido binario del programa sin los encabezados ejecutables (para garantizar esterilidad). Para nuestro estudio vamos a utilizar únicamente este ultimo archivo. Vamos a previsualizar su estructura.

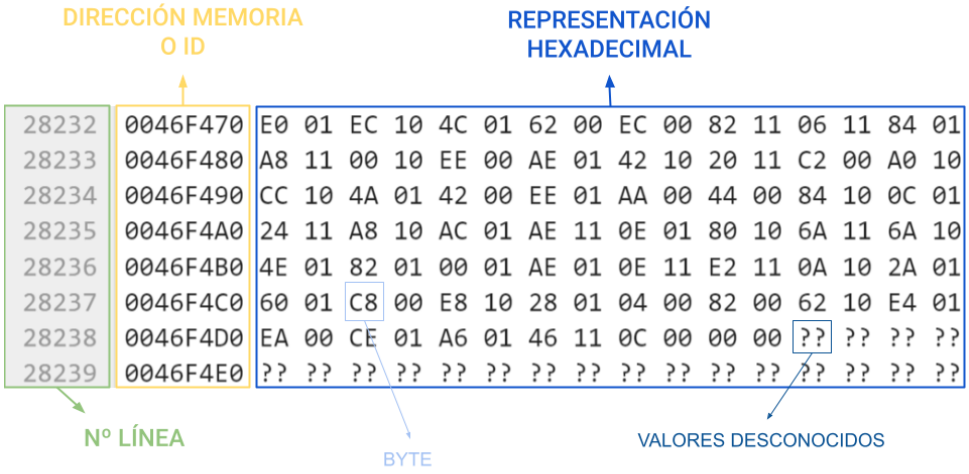

Figura 3.1: Previsualización del archivo"0ACDbR5M3ZhBJajygTuf.bytes" de la carpeta train.

Como aparece en 3.1, los ocho primeros caracteres son direcciones de memoria, seguido de la representación hexadecimal del contenido binario del programa, que contiene 16 bytes (cada uno dos caracteres). A veces nos podemos encontrar con "??" en el lugar de un byte. Este símbolo se utiliza en estos archivos para representar que se desconoce su información porque su memoria no se puede leer [5]. También en `https://www.kaggle.com/code/dheemanthbhat/malware-classification-with-multiprocessing`.

Hay un total de 21.741 programas de malware, pero nosotros tan solo usaremos los 10868 pertenecientes al entrenamiento. Estos programas pertenecen a una de estas 9 familias de malware: Rammit, Lollipop, Kelihos_ ver3, Vundo, Simda, Tracur, Kelihos_ ver1, Obfuscator y Gatak. Cada una de estas familias se clasifican en una de estas 6 variadades:

1. Worm (Gusano)- Los gusanos informáticos son un tipo de malware que puede propagarse a través de redes sin la intervención de un programa huésped o de un humano. Son caba-

llos de Troya maliciosos que pueden replicarse y propagarse de una computadora a otra. Los gusanos infectan a sus anfitriones mediante el engaño y la astucia, pudiendo causar graves daños a las computadoras comprometidas al consumir ancho de banda, sobrecargar sistemas, eliminar o modificar archivos e instalar virus adicionales. Además, las vulnerabilidades en el software, los archivos adjuntos de correo electrónico y las conexiones de red pueden facilitar su propagación.

2. Adware (programa publicitario) - El adware es una variedad de malware que muestra anuncios no deseados a los usuarios, típicamente como ventanas emergentes o banners. A menudo se distribuye como parte de descargas de software gratuitas, pero también se puede obtener a través de un ciberataque o una vulnerabilidad. El adware tiene como objetivo generar ingresos para sus desarrolladores mostrando anuncios a los consumidores. Sin embargo, puede causar un daño significativo a una computadora infectada al degradar su rendimiento, reducir su disponibilidad y violar la privacidad del usuario al recopilar información sobre sus actividades de navegación.

3. Backdoor (Puerta trasera) - Un backdoor permite que una entidad no autorizada tome el control completo del sistema de una víctima sin su consentimiento. Un troyano de puerta trasera siempre se presenta como una herramienta de software legítima esencialmente requerida por el usuario. Otras opciones pueden ser visitar un sitio web malicioso o hacer clic en un enlace no deseado. Al ejecutarse, se añade a sí mismo en la rutina de inicio del sistema y busca una conexión a Internet. Una vez que el sistema está en línea, se conecta con su autor, quien luego toma el control del sistema para realizar diferentes tareas, como descargar/cargar archivos, registrar pulsaciones de teclas, enviar correos electrónicos no deseados o robar contraseñas, entre otras cosas.

4. Trojan (Troyano)- Los troyanos son programas maliciosos que se disfrazan como programas o archivos legítimos y pueden tomar el control de una computadora para ejecutar operaciones maliciosas como robar datos, dañar el sistema o abrir puertas traseras para otros virus. El término "Troyano"proviene de la leyenda griega del caballo de Troya, una táctica engañosa utilizada para conquistar Troya. En el ámbito digital, los troyanos son parásitos digitales hostiles capaces de leer contraseñas, grabar pulsaciones de teclas y propagar otros virus. Se utilizan técnicas de ingeniería social, como correos electrónicos de phishing o descargas maliciosas, para propagarlos. A diferencia de los virus informáticos y los gusanos, los troyanos no pueden replicarse y deben ser instalados o ejecutados por los usuarios.

5. Trojan downloader (Descargador troyano) - Es un programa malicioso que se descarga e instala en una computadora infectada. Puede abrir conexiones de red ilícitas, mutarse a sí mismo, deshabilitar herramientas de seguridad y transferir información personal del usuario sin permiso. Además, su función principal es la de descargar e instalar otro malware dañino en el sistema infectado.

6. Obfuscated malaware (Malware obfuscado) - La obfuscación es una técnica utilizada para hacer que el código de un programa sea más difícil de entender o de analizar. Esto implica modificar el código del programa malicioso de manera que sea más complicado para los investigadores de seguridad o los programas antivirus detectarlo y comprender cómo funciona. Su objetivo principal es eludir la detección y análisis por parte de los programas antivirus y otros sistemas de seguridad.

Analizando 3.2, podemos observar como la distribución entre las clases de los datos de entrenamiento no es uniforme. Mientras que de la clase Simbda 42, de la clase Kelihos_ ver3 hay 2942, un 98 % más. En [12] deciden prescindir de esta clase, pero nosotros hemos decidido hacer el análisis con las 9 clases.

Cuadro 3.1: Family, Type, y Total Samples

Class ID	Familia	Tipo	Muestras	Porcentaje
1	Ramnit	Worm	1541	14.18
2	Lollipop	Adware	2478	22.8
3	Kelihos_ ver3	Backdoor	2942	27.07
4	Vundo	Trojan	475	4.37
5	Simda	Backdoor	42	00.39
6	Tracur	Trojan Downloader	751	6.91
7	Kelihos_ ver1	Backdoor	398	3.66
8	Obfuscator.ACY	Malware obfuscado	1228	11.30
9	Gatak	Backdoor	1013	9.32

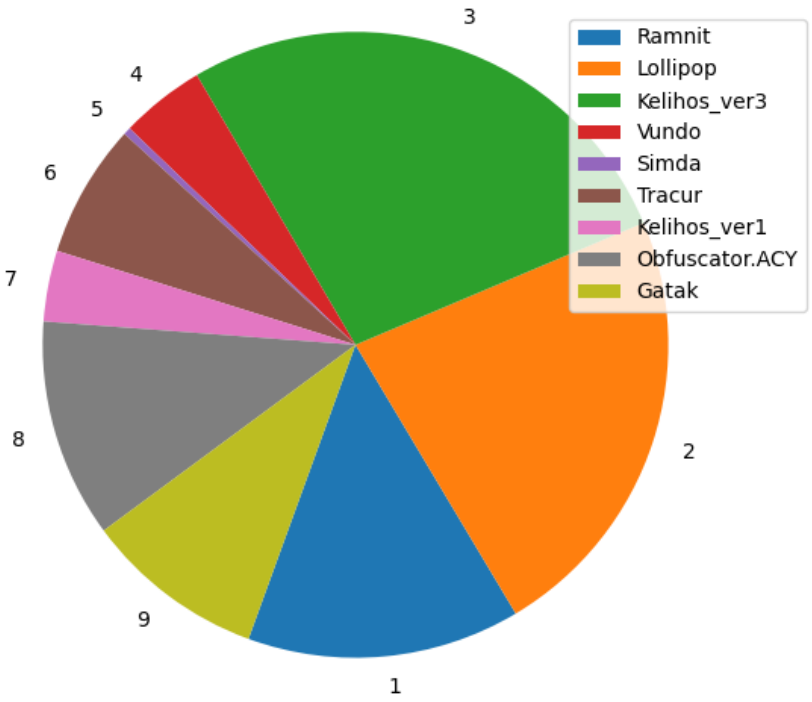

Figura 3.2: Distribución del BIG 2015 training dataset.

A la hora de crear nuestros modelos, hemos dividido el conjunto de datos aleatoriamente en grupos con el 75 %, 15 % y 10 % para entrenamiento, test y validación respectivamente. La 3.3 muestra como quedarían distribuidas las clases en los diferentes grupos.

3.2. Resumen del Conjunto de Datos

El conjunto de datos utilizado en este estudio proviene del Desafío de Clasificación de Malware de Microsoft (BIG 2015), accesible a través de Kaggle. Este conjunto de datos consta de programas de malware pertenecientes a 9 categorías diferentes. Se proporcionan conjuntos de datos de entrenamiento y prueba, pero solo se utilizan los datos de entrenamiento debido a la falta de etiquetas para el conjunto de prueba para su entrega. Además

El conjunto de entrenamiento contiene 10,868 muestras de malware, cada una con un identifi-

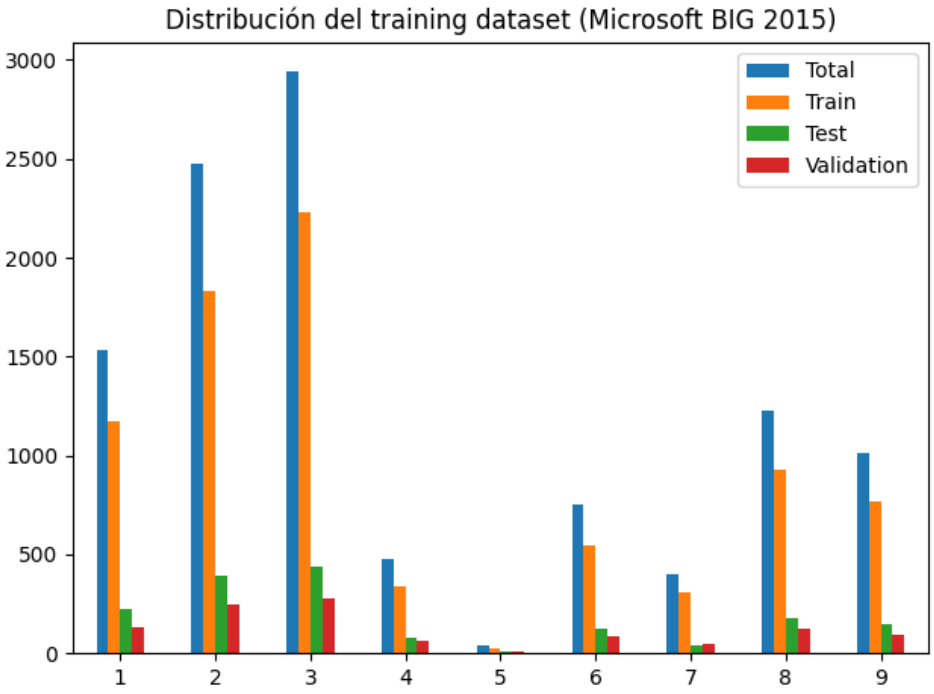

Figura 3.3: Distribución de las clases en cada grupo.

cador único, un valor hash de 20 caracteres y una etiqueta de clase que representa una de las 9 familias de malware. Cada archivo de malware tiene una representación hexadecimal de su contenido binario, junto con un archivo 'bytes' que contiene esta información. No se excluye ninguna categoría de malware en este estudio.

La distribución de los datos se realiza con un 72 % para entrenamiento, un 8 % para validación y un 20 % para pruebas. Se realizan varias divisiones y análisis de los conjuntos de datos para ajustar y evaluar diferentes modelos y enfoques de aprendizaje profundo para la clasificación de malware.

Este conjunto de datos, con su amplia gama de muestras etiquetadas y la diversidad de familias de malware representadas, proporciona una base sólida para la investigación y desarrollo de métodos efectivos de detección y clasificación de amenazas informáticas.

Se decidió escoger este dataset y no otro porque el objetivo que tenemos en este trabajo de investigación es el de aprender y desarrollar diferentes métodos de aprendizaje automático y este dataset nos permite utilizar tanto una CNN como un Autoencoder según [21].

Como podemos observar, todos los programas proporcionados son malware, lo que nos indica que con este dataset no podemos crear un modelo que nos prediga si un programa es malware o no, sino que tan solo podemos hacer un modelo de clasificación. Para ello vamos a abordar este experimento de dos formas diferentes. Una de ellas es utilizando una CNN y la otra es usando un autoencoder. utilizar dos redes neuronales

3.2.1. Autoencoder

3.2.2. Red Neuronal Convolucional

3.2.3. Resultados

El entorno de hardware en el que he realizado todos los experimentos es un sistema operativo Linux 6.1.0-17 de 64 bits con glibc2.36, funcionando en una arquitectura X86_ 64. La CPU utilizada tiene 12 núcleos. En cuanto a la GPU, es un VGA de Intel Corporation 200 Series/Z370 Chipset Family SPI Controller con el identificador [8086:a2a4]. La memoria RAM disponible es de 126GB.

3.3. Detección de intrusiones

3.3.1. KDD Cup 1999

3.3.2. Autoencoder

Para la clasificacion binaria usar autoencoder con el entrenamiento de las imágenes (buenas o malas) y según el error que den, se clasifica. Para la multiclasificación, tenemos dos opciones:

- Usamos autoencoder para comprimir la información de entrada y despues esa informacion la usamos para clasificarla usando una DNN [15]
- Usamos una cadena de autoencoders en el cual la salida de h es la entrada del autoencoder h+1. Utilizo el articulo [8] donde se desarrolla todo el modelo y explicacion y ademas se hace referencia al artículo [4] porque se basa en él (lo de salida de h es la entrada de h+1). Ver tambien:
  - Asymmetric Stacked Autoencoder
  - Constrained Nonlinear Control Allocation based on Deep Auto-Encoder Neural Networks.

  El algoritmo consiste en entrenar las capas por separado en la que el input del autoencoder es la salida del autoencoder anterior. Lo que de verdad nos interesa es la capa oculta, que tiene una representación comprimida de los datos de entrada y sus pesos. Estos pesos son con los que se inicializa el entrenamiento de la stacked autoencoder acabando en softmax. He usado el url para enterlo `https://amiralavi.com/tied-autoencoders/`. Además en [3] explica bastante bien la diferencia entre capa autoencoder y un autoencoder.

3.3.3. Red Neuronal Convolucional

Para clasificar los datos del dataset KDD 1999 usando las Convolutional Neural Network (CNN) vamos a seguir los siguientes articulos [13, 25, 20, 14]. Prácticamente todo el cuerpo del experimento se encuentra en el artículo [13], pero en el artículo [14] aparece la parte de normalización de los datos y algunos hiperparametros de inicio.

3.3.4. Red Neuronal Profunda

Por otro lado, el método Deep Neural Network (DNN) utiliza una arquitectura muy parecida a una CNN. Podemos ver todo el procesamiento de los datos y el modelo en el artículo [18]. Además, hay buena explicacion del experimento en [24]. Por último, en el articulo [6] están los experimentos con DNN, RNN, RBM que puedo tomar también como referencia porque está muy bien explicado las capas e hiperparametros que utiliza.

3.3.5. Red Neuronal Recurrente

En el articulo [6] están los experimentos con DNN, RNN, RBM que puedo tomar también como referencia porque está muy bien explicado las capas e hiperparametros que utiliza.

3.3.6. Restricted Boltzmann Machine

En el articulo [6] están los experimentos con DNN, RNN, RBM que puedo tomar también como referencia porque está muy bien explicado las capas e hiperparametros que utiliza.

3.3.7. Resultados

El entorno de hardware en el que he realizado todos los experimentos es un sistema operativo Linux 6.1.0-17 de 64 bits con glibc2.36, funcionando en una arquitectura X86_ 64. La CPU utilizada tiene 12 núcleos. En cuanto a la GPU, es un VGA de Intel Corporation 200 Series/Z370 Chipset Family SPI Controller con el identificador [8086:a2a4]. La memoria RAM disponible es de 126GB.

Capítulo 4

Conclusiones y Trabajo Futuro

4.1. Conclusiones

4.2. Trabajo futuro

Bibliografía

[1] Microsoft malware classification challenge (big 2015), 2015.

[2] Apache Software Foundation. Apache mxnet, 2015.

[3] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006.

[5] Niken Dwi Wahyu Cahyani, Erwid M Jadied, Nurul Hidayah Ab Rahman, and Endro Ariyanto. The influence of virtual secure mode (vsm) on memory acquisition. *International Journal of Advanced Computer Science and Applications*, 13(11), 2022.

[6] Wisam Elmasry, Akhan Akbulut, and Abdul Halim Zaim. Empirical study on multiclass classification-based network intrusion detection. *Computational Intelligence*, 35(4):919–954, 2019.

[7] Facebook AI Research. Pytorch, 2017.

[8] Fahimeh Farahnakian and Jukka Heikkonen. A deep auto-encoder based approach for intrusion detection system. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 178–183. IEEE, 2018.

[9] Google AI Team. Keras, 2015.

[10] Google Brain Team. Tensorflow, 2015.

[11] Sanchit Gupta, Harshit Sharma, and Sarvjeet Kaur. Malware characterization using windows api call sequences. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, pages 271–280. Springer, 2016.

[12] Temesguen Messay Kebede, Ouboti Djaneye-Boundjou, Barath Narayanan Narayanan, Anca Ralescu, and David Kapp. Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft malware classification challenge (big 2015) dataset. In *2017 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 70–75. IEEE, 2017.

[13] Jiyeon Kim, Jiwon Kim, Hyunjung Kim, Minsun Shim, and Eunjung Choi. Cnn-based network intrusion detection against denial-of-service attacks. *Electronics*, 9(6):916, 2020.

[14] Taejoon Kim, Sang C Suh, Hyunjoo Kim, Jonghyun Kim, and Jinoh Kim. An encoding technique for cnn-based network anomaly detection. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2960–2965. IEEE, 2018.

[15] Ivandro O Lopes, Deqing Zou, Ihsan H Abdulqadder, Francis A Ruambo, Bin Yuan, and Hai Jin. Effective network intrusion detection via representation learning: A denoising autoencoder approach. *Computer Communications*, 194:55–65, 2022.

[16] Mika Luoma-aho. Analysis of modern malware: obfuscation techniques. 2023.

[17] Nesma Mahmoud, Youssef Essam, Radwa Elshawi, and Sherif Sakr. Dlbench: an experimental evaluation of deep learning frameworks. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 149–156. IEEE, 2019.

[18] Mohammed Maithem and Ghadaa A Al-Sultany. Network intrusion detection system using deep neural networks. In *Journal of Physics: Conference Series*, volume 1804, page 012138. IOP Publishing, 2021.

[19] Montreal University. Theano, 2010.

[20] Sinh-Ngoc Nguyen, Van-Quyet Nguyen, Jintae Choi, and Kyungbaek Kim. Design and implementation of intrusion detection system using convolutional neural network for dos detection. In *Proceedings of the 2nd international conference on machine learning and soft computing*, pages 34–38, 2018.

[21] Prajoy Podder, Subrato Bharati, M Mondal, Pinto Kumar Paul, and Utku Kose. Artificial neural network for cybersecurity: A comprehensive review. *arXiv preprint arXiv:2107.01185*, 2021.

[22] Sajedul Talukder. Tools and techniques for malware detection and analysis. *arXiv preprint arXiv:2002.06819*, 2020.

[23] Tokyo University. Chainer, 2015.

[24] Rahul K Vigneswaran, R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security. In *2018 9th International conference on computing, communication and networking technologies (ICCCNT)*, pages 1–6. IEEE, 2018.

[25] Zhongxue Yang and Adem Karahoca. An anomaly intrusion detection approach using cellular neural networks. In *Computer and Information Sciences–ISCIS 2006: 21th International Symposium, Istanbul, Turkey, November 1-3, 2006. Proceedings 21*, pages 908–917. Springer, 2006.

[26] Ilsun You and Kangbin Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.

[27] Mohamad Fadli Zolkipli and Aman Jantan. Malware behavior analysis: Learning and understanding current malware threats. In *2010 Second International Conference on Network Applications, Protocols and Services*, pages 218–221. IEEE, 2010.