Detección de ataques en redes de datos utilizando métodos de Deep Learning

Jorge Buzzio García
Universidad de Antioquia, Medellín, Colombia
Facultad de Ingeniería
jbuzzio410@gmail.com

***Abstract*—Este trabajo es el trabajo final del curso Deep Learning del semestre 2024-2. Consta de la creación de un modelo de Red Neuronal Convolucional para detectar ataques de red. Para esto se usó el dataset CICIDS2017, que contiene ataque DDoS, Dos, Escaneo y trafico Normal.**

***Index Terms*—Redes neuronales, Ataque de denegación de servicios, Redes de datos, Ciberseguridad.**

I. Contexto de aplicación

Dado el constante crecimiento de las redes de datos originado por la demanda de las personas de estar cada vez mas conectadas y a mayores velocidades, ha hecho que tecnologias como 5G, Internet de las cosas y la inteligencia artificial, se encuentren en consante evolución a pasos agigantados. En un mundo que cada vez es mas dependiente de la tecnología, es importante que la seguridad de la información como de los dispositivos sea cada vez mas robusta. Constantemente estamos expuestos a multiples tipos de ciberataques, del cual el que esta tomando mayor relevancia son los ataques de denegación de servicio.

Los ataques de denegación de servicio (DoS) y ataques de denegación de servicio distribuido (DDoS) son aquellos que buscan saturar los recursos de computo disponibles en un servicio, imposibilitando que usuarios legitimos puedan acceder a los servicios. Los ataques de denegación de servicio han ido en aumento en los ultimos años y esto se refleja en el reporte de la empresa Vercara, que afirma que hubo un incremento del 184% de DoS en la segunda mitad del 2024. Además de haber detectado alrededor de 10157 ataques DDoS en agosto del 2024, lo que significa para ellos un incremento del 56% en comparación con el mes de julio [1].

Es por este motivo que este trabajo del curso de Deep Learning es de detectar ataques DoS, DDoS y escaneo utilizando modelos de Deep Learning utilizando datasets que describan un comportamiento anómalo en tráfico de redes de datos.

II. Descripción de la estructura de los notebooks entregados

El trabajo consta de seis notebooks. Tres de ellos en formato ipynb y los otros tres en formato de archivo python. X0_dividir_dataset.py, X1_dataset_to_image.py y X2_gendataset.py se ejecutan para la creación del dataset. El dataset en la primera entrega fue el CICDDoS2019, el cual tenia un desbalance muy alto entre los ataques y el trafico normal. Por lo que se considero utilizar el dataset CICIDS2017, creado por la misma institución que creo el dataset CICDDoS2019. El archivo X0_dividir_dataset.py, divide el dataset entre los registros considerados normales y los registros considerados ataques. Despues de dividir estos registros, se eliminan variables redundantes o que no aportan al entrenamiento quedando un total de 60 variables. Seguido de eso se toman 180 registros por las 60 variables de tal forma que se puede crear una imagen de 60x60x3. Con el archivo X1_dataset_to_image.py se transforman los registros del dataset a imagenes. Y por ultimo, X2_gendataset.py etiqueta las imagenes y genera un archivo npz que contiene el dataset.

Cabe considerar que de los ataques presentes en el dataset CICIDS2017, la cantidad de imagenes que se pudieron crear fueron significativas solamente para la clase DDoS, DoS, PortScan y Benign. Por lo que, solamente se utilizaron estas clases. El notebook 01_exploración_de_datos.ipynb muestra la distribución de estas clases. Se ploteo la distribución de las clases en un diagrama de barras tal como se muestra en la figura 1.

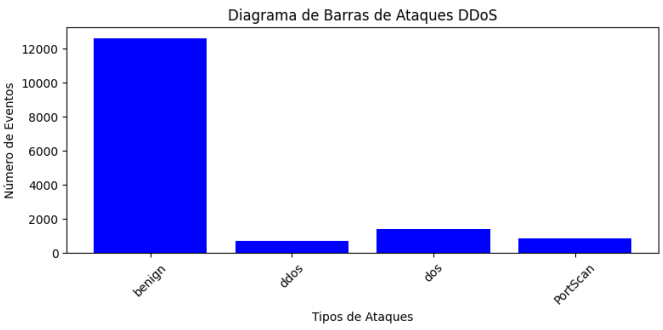

Fig. 1. Diagrama de barras de la distribución de imágenes.

El dataset constó de 15609 imagenes de las cuales se distribuyeron para la clase benign en 12618, ddos en 711, dos en 1398 y PortScan 882. Se muestra algunas imagenes de cada clase en la figura 2.

El notebook 02_preprocesado.ipynb busca balancear los datos presentes en el dataset. El dataset tiene una predominancia de la clase benign con 12618 y la siguiente es DoS con solo 1398. Por lo que, en este notebook, se hace un submuestreo

Identify applicable funding agency here. If none, delete this.

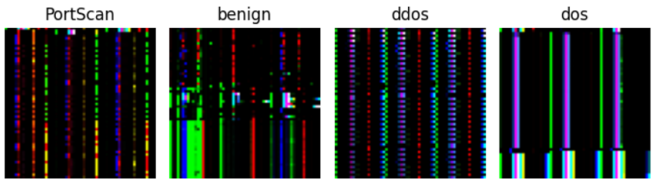

Fig. 2. Muestras de imagenes por clase.

de la clase benign para reducirlo a 3000 imagenes y no tenga una diferencia tan grande con las demas clases.

Por último el notebook 03_arquitectura_de_linea_base.ipynb, muestra una arquitectura convolucional que busca identificar patrones en las imagenes que ayuden a clasificar correctamente las imagenes que puedan alertarnos de la presencia de un ataque de red.

III. Descripción de la solución

Como se mencionó anteriormente, el dataset elegido para esta tarea fue el CICIDS2017. Despues de realizar el balanceo de las clases en el dataset de imagenes,se carga el dataset teniendo un total de 5991 imagenes. Teniendo los datos la forma (5991,60,60,3) y las etiquetas (5991,). Dado que las etiquetas constan de variables categoricas como Benign, DoS, DDoS y PortScan, se realiza un label encoder transformando las variables categoricas a variables numéricas mapeando PortScan:0, benign:1, ddos:2 y dos:3.

Del dataset se divide los conjuntos de entrenamiento y el conjunto de pruebas en una relacion de 0.25, quedando el entrenamiento en 4493 muestras y 1498 muestras en las pruebas. La arquitectura constó de una red convolucional que acepta como entrada imagenes de 60x60x3, seguida de una capa convolucional con filtros de 6x6 y funcion de activacion relu. Seguido de eso se realiza tambien un maxpooling con un pooling size de 2x2. Seguido nuevamente se encuentra una capa convolucional pero ahora de 60 filtros y de tamaño 4x4. Después de esto, se realiza un flatten para que el resultado ingrese a una capa dense y para evitar que se genere overfitting, la salida de la capa densa va a un dropout del 0.2. Finalmente, la salida del dropout ingresa a la capa densa de salida con 4 clases, que son las clases correspondientes de nuetro dataset. En la figura 3 se observan los parametros de la arquitectura.

La capa de entrada no cuenta con parametros por lo que preenta un valor de 0. La capa de convolución consta de 1635 parametros que resultan de los 15 filtros de 6x6x3 mas 15 valores del bias dando (6x6x3x15 +15 = 1635). El max pooling no cuenta con parametros por eso que se muestra el 0. Para la segunda capa convolucional se muestran 14460 parametros que provienen de los 60 filtros de 4x4x15 (15 filtros de la capa anterior), ademas de los 60 valores del bias (4*4*15*60+60 = 14460). El flatten aplana las neuronas de la red sin agregar parametros, por eso la cantida 0 de parametros. La capa densa consta de las 34560 neuronas de la capa anterior por las 16 neuronas de la capa densa dando un total de 552976 que consta del producto de 34560*16 + 16 del vias. Por ultimo en la capa

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 60, 60, 3)	0
conv2d (Conv2D)	(None, 55, 55, 15)	1,635
max_pooling2d (MaxPooling2D)	(None, 27, 27, 15)	0
conv2d_1 (Conv2D)	(None, 24, 24, 60)	14,460
flatten (Flatten)	(None, 34560)	0
dense (Dense)	(None, 16)	552,976
dropout (Dropout)	(None, 16)	0
output_1 (Dense)	(None, 4)	68

Total params: 569,139 (2.17 MB)
Trainable params: 569,139 (2.17 MB)
Non-trainable params: 0 (0.00 B)

Fig. 3. Modelo de red convolucional

de salida, se presentan 68 parametros que provienen de los 16 *4 + 4 del bias de la capa de salida. Esto da un total de 569,139 parametros en la red.

IV. Descripción de las iteraciones

Para validar el modelo se realizaron 10 y 20 iteraciones. Cuando se realiza 10 iteraciones en un inicio el accuracy tiene valores bajos ya que el modelo recien esta aprendiendo, pero a medida que se avanza en los epochs, el accuracy va aumentando junto con el validation accuracy, mostrando que no llega a presentar algun tipo de overfitting como se muestra en la figura 4. De la misma manera al realizar 20 epochs el modelo logra buenos resultados, sin embargo el accuracy a partir del epoch 14 se estanca y no avanza mas, esto se puede mostrar en la figura 5.

V. Descripción de los resultados

Al evaluar los resultados del entrenamiento del modelo con el conjunto de datos creado, vemos que tiene un buen desempeño tomando como referencia el valor del accuracy que consta de aproximadamente un 95.17% tal como se muestra en la tabla 1.

En las figuras 6 y 7 vemos tambien la matriz de confusión resultante del entrenamiento, mostrando resultados bastante buenos. A partir de la matriz de confusión de la figura 6 podemos obtener la tabla 1 con los resultados de accuracy, precision, recall y F1-score para los resultados.

Accuracy (Acc): Es la métrica más básica y mide el porcentaje de imágenes correctamente clasificadas sobre el total de predicciones.

Precision (Pr): Es el ratio de flujos de imágenes correctamente clasificadas (TP), frente a todas las imágenes clasificadas (TP+FP).

Recall (Rc): Es el ratio de imágenes correctamente clasificadas (TP), frente a todas las imágenes clasificadas (TP+FN).

F1 Score: Es una combinación armónica de precisión y recall en una sola medida.

Al ser este un problema de clasificación las métricas mas utilizads son Accuracy, Precision, Recall, Factor 1 y la matriz de confusión. Recordando que PortScan:0, benign:1, ddos:2 y dos:3. Notamos que en el caso de ddos, en los resultados al entrenar con 10 epochs y con 20 epochs, el modelo clasifica correctamente todas las imagenes de esta clase. Luego en

```
Epoch 1/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 7s 30ms/step - accuracy: 0.4836 - loss: 5.5225 - val_accuracy: 0.6542 - val_loss: 0.5736
Epoch 2/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6387 - loss: 0.6941 - val_accuracy: 0.5915 - val_loss: 0.7050
Epoch 3/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6555 - loss: 0.6930 - val_accuracy: 0.6896 - val_loss: 0.6503
Epoch 4/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6774 - loss: 0.6540 - val_accuracy: 0.8278 - val_loss: 0.4187
Epoch 5/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7038 - loss: 0.5628 - val_accuracy: 0.8465 - val_loss: 0.4302
Epoch 6/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7348 - loss: 0.5535 - val_accuracy: 0.8451 - val_loss: 0.4053
Epoch 7/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7830 - loss: 0.4476 - val_accuracy: 0.8505 - val_loss: 0.3326
Epoch 8/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8605 - loss: 0.3793 - val_accuracy: 0.8318 - val_loss: 0.3885
Epoch 9/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8202 - loss: 0.4211 - val_accuracy: 0.9840 - val_loss: 0.2615
Epoch 10/10
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9130 - loss: 0.3363 - val_accuracy: 0.9740 - val_loss: 0.2623
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9822 - loss: 0.2588
{'loss': 0.2623068690299988, 'compile_metrics': 0.9739652872085571}
```

Fig. 4. Entrenamiento con 10 epochs

```
Epoch 6/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9105 - loss: 0.2825 - val_accuracy: 0.9746 - val_loss: 0.2612
Epoch 7/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8986 - loss: 0.3665 - val_accuracy: 0.9720 - val_loss: 0.2287
Epoch 8/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9108 - loss: 0.2628 - val_accuracy: 0.9793 - val_loss: 0.1949
Epoch 9/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9304 - loss: 0.2306 - val_accuracy: 0.9813 - val_loss: 0.2136
Epoch 10/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9322 - loss: 0.1946 - val_accuracy: 0.9840 - val_loss: 0.2316
Epoch 11/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9374 - loss: 0.2066 - val_accuracy: 0.9800 - val_loss: 0.2163
Epoch 12/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9368 - loss: 0.1862 - val_accuracy: 0.9826 - val_loss: 0.1866
Epoch 13/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9497 - loss: 0.1580 - val_accuracy: 0.9806 - val_loss: 0.1670
Epoch 14/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9751 - loss: 0.1329 - val_accuracy: 0.9833 - val_loss: 0.1716
Epoch 15/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9691 - loss: 0.1371 - val_accuracy: 0.9846 - val_loss: 0.1527
Epoch 16/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9710 - loss: 0.1207 - val_accuracy: 0.9780 - val_loss: 0.1785
Epoch 17/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9649 - loss: 0.1403 - val_accuracy: 0.9860 - val_loss: 0.1530
Epoch 18/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9784 - loss: 0.0904 - val_accuracy: 0.9860 - val_loss: 0.1758
Epoch 19/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9776 - loss: 0.1004 - val_accuracy: 0.9873 - val_loss: 0.1983
Epoch 20/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9736 - loss: 0.1066 - val_accuracy: 0.9786 - val_loss: 0.1911
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9842 - loss: 0.1397
{'loss': 0.19111862778663635, 'compile_metrics': 0.9786381721496582}
```

Fig. 5. Entrenamiento con 20 epochs

las demas clases la relacion entre los aciertos con los fallos presenta una diferencia muy significativa.

VI. Observaciones y recomendaciones

Para este trabajo fue importante el uso de GPU en el google colab ya que con solo CPU el tiempo que demora el modelo en entrenarse aumente considerablemente. Aun mas cuando se quizo implementar transfer learning. La cantidad de imagenes en el conjunto de datos tambien juega un rol importante en el tiempo y uso de recursos. Fue necesario reducir la cantidad de imagenes porque el google colab quedaba sin memoria rapidamente. En comparacion con CICDDoS2019, CICIDS2017 presenta patrones mas notorios entre clases, lo que hace mas facil a los modelos poder discernir entre una clase y otra. CICDDoS2019, a pesar que presenta mayor canti-

TABLE I
Métricas de Evaluación del Modelo

Métrica	Valor
Accuracy	95.17%
Precision (Clase 1)	96.23%
Precision (Clase 2)	96.33%
Precision (Clase 3)	99.46%
Precision (Clase 4)	99.42%
Recall (Clase 1)	87.93%
Recall (Clase 2)	98.89%
Recall (Clase 3)	98.98%
Recall (Clase 4)	98.01%
F1-Score (Clase 1)	91.94%
F1-Score (Clase 2)	97.60%
F1-Score (Clase 3)	99.22%
F1-Score (Clase 4)	98.70%

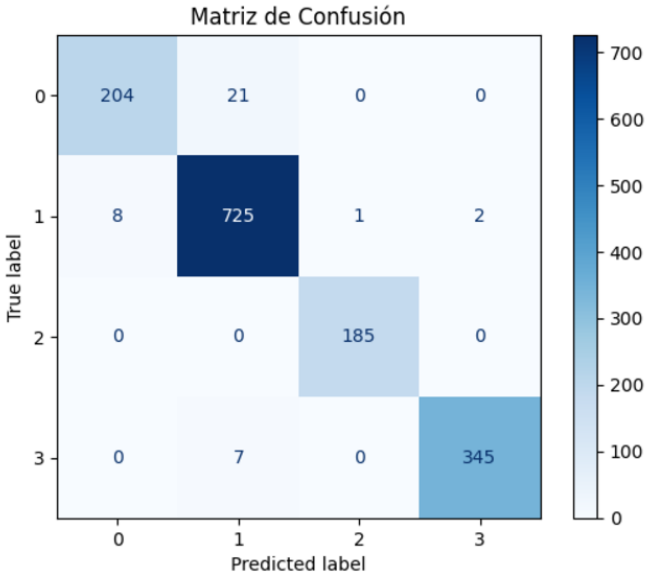

Fig. 6. Matriz de confusion para 10 epochs

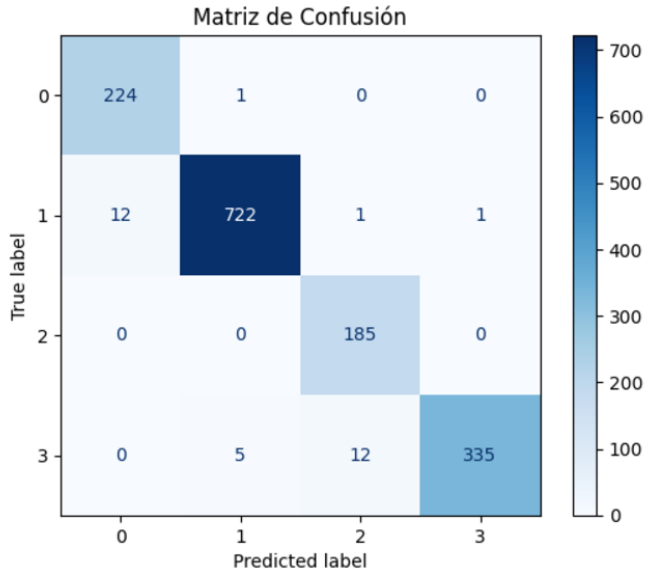

Fig. 7. Matriz de confusion para 20 epochs

dad de clases, todas son DDoS, por lo que el comportamiento puede asemejarse mucho, haciendo que los modelos tengan mayor dificultad en clasificarlo.

Los datos utilizados para realizar los notebooks y las pruebas se encuentran en el github del proyecto y en el google drive. Estos datos están públicos.

REFERENCES

[1] A. González, ”Los ataques DDoS registran un aumento del 186% en la primera mitad de 2024,” IT Digital Security, Sep. 20, 2024. [Online]. Available: https://www.itdigitalsecurity.es/endpoint/2024/09/los-ataques-ddos-registran-un-aumento-del-186-en-la-primera-mitad-de-2024. [Accessed: Sep. 28, 2024].

[2] I. Sharafaldin, A. H. Lashkari, S. Hakak and A. A. Ghorbani, ”Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy,” 2019 International Carnahan Conference on Security Technology (ICCST), Chennai, India, 2019, pp. 1-8, doi: 10.1109/CCST.2019.8888419.

[3] F. Hussain, S. G. Abbas, M. Husnain, U. U. Fayyaz, F. Shahzad and G. A. Shah, ”IoT DoS and DDoS Attack Detection using ResNet,” 2020 IEEE 23rd International Multitopic Conference (INMIC), Bahawalpur, Pakistan, 2020, pp. 1-6, doi: 10.1109/INMIC50486.2020.9318216.

[4] A. A. Freitas Junior, F. Lima Filho, A. Brito Júnior, and L. F. Silveira, ”Detecção de Ataques DDoS com Base em Métricas de Tráfego usando Redes Convolucionais,” in *Anais do XVI Congresso Brasileiro de Inteligência Computacional*, 2023.

[5] X. Liu, Z. Tang and B. Yang, ”Predicting Network Attacks with CNN by Constructing Images from NetFlow Data,” 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Washington, DC, USA, 2019, pp. 61-66, doi: 10.1109/BigDataSecurity-HPSC-IDS.2019.00022.

[6] J. Buzzio, ”DeepLearning,” GitHub repository, 2024. [Online]. Available: https://github.com/jbuzzio/DeepLearning/tree/main. [Accessed: Sep. 28, 2024].