Parte II: Implementación de Redes Neuronales FCN

Grupo X - Dataset KDD99

Septiembre 2025

1 Preparación de entorno y datos

1.1 Cantidad de clases presentes

Se identificaron 23 clases en total, cada una representada en el mapeo label2idx. Este mapeo garantiza una correspondencia estable y reproducible entre las etiquetas de texto del dataset original y los índices numéricos requeridos por PyTorch para el entrenamiento de la red neuronal.

- back. $\rightarrow$ 0
- buffer_overflow. $\rightarrow$ 1
- ftp_write. $\rightarrow$ 2
- guess_passwd. $\rightarrow$ 3
- normal. $\rightarrow$ 19
- neptune. $\rightarrow$ 20
- smurf. $\rightarrow$ 14

Este proceso aseguró un etiquetado uniforme y reproducible en todas las particiones (entrenamiento, validación y prueba), evitando inconsistencias que podrían afectar el rendimiento del modelo.

1.2 Eliminación de registros duplicados

Durante el proceso de limpieza se identificaron y eliminaron registros duplicados para evitar sesgos en el entrenamiento. La presencia de registros idénticos puede llevar a que el modelo memorice patrones específicos en lugar de aprender características generalizables, comprometiendo su capacidad predictiva en datos no vistos.

Esta depuración redujo significativamente el tamaño del dataset:

- Dataset original: 494,021 registros
- Registros duplicados eliminados: 348,435
- Dataset final limpio: 145,586 registros únicos
- Porcentaje de duplicados: 70.5%

La alta proporción de duplicados (70.5%) es característica del dataset KDD99 y refleja la naturaleza de la captura de tráfico de red, donde ciertos patrones de conexión se repiten frecuentemente. Esta limpieza fue crítica para asegurar que las métricas de evaluación reflejen el verdadero rendimiento del modelo.

1.3 Dimensionalidad de las características

Tras la limpieza y el procesamiento one-hot de variables categóricas, cada registro quedó representado con 41 atributos numéricos. Esta dimensionalidad refleja la transformación completa del dataset original para su uso en redes neuronales, donde todas las variables deben ser numéricas.

La composición final de las 41 características incluye:

- 38 variables numéricas escaladas con StandardScaler
- 3 variables categóricas originales transformadas mediante one-hot encoding:
  - protocol_type (tcp, udp, icmp)
  - service (http, ftp, smtp, etc.)
  - flag (SF, S0, REJ, etc.)

El escalado con StandardScaler fue esencial para las redes neuronales, asegurando que todas las variables tuvieran media cero y desviación estándar uno. Esto previene que variables con rangos numéricos grandes dominen el proceso de aprendizaje y garantiza una convergencia más estable durante el entrenamiento.

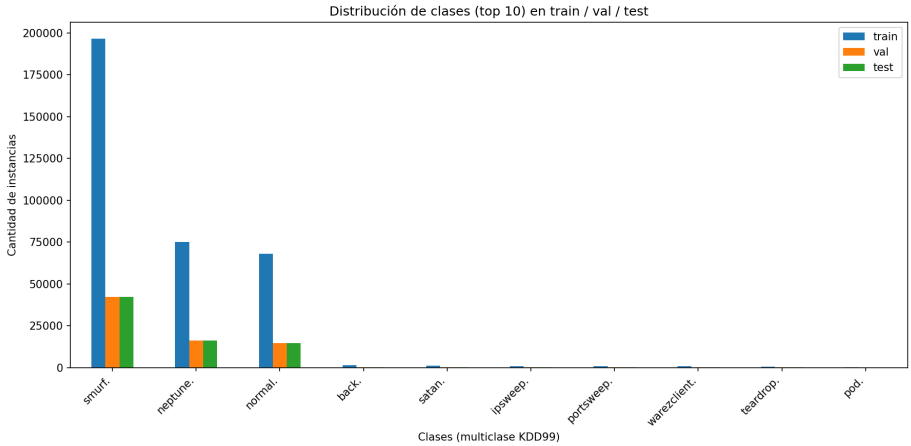
1.4 Distribución de instancias

Luego de la limpieza y división estratificada, los conjuntos resultaron con una distribución que mantiene las proporciones originales de las clases en cada partición. Esta división 70/15/15 es estándar en machine learning y asegura suficientes datos para entrenamiento mientras reserva conjuntos independientes para validación y evaluación final.

	Conjunto	Instancias	Porcentajes
Fila 1	Entrenamiento	101,910	70%
Fila 2	Validación	21,838	15%
Fila 3	Prueba	21,838	15%

Este balance garantiza que los resultados experimentales puedan evaluarse de manera justa y generalizable, con suficientes datos para detectar overfitting durante la validación y una evaluación final no sesgada en el conjunto de prueba.

Figure 1: Distribución de las 10 clases más frecuentes en el dataset



KDD99

1.5 Verificación con DataLoader

Para validar que la preparación de datos fue exitosa, se realizó una verificación mediante la generación de un batch de entrenamiento usando el DataLoader de PyTorch. Esta verificación es crucial para confirmar que los tensores tienen las dimensiones correctas y que no hay inconsistencias en los tipos de datos.

La configuración del DataLoader incluyó los siguientes parámetros:

- Batch size: 512 muestras por lote
- Shuffle: Activado solo en entrenamiento
- Pin memory: Activado para eficiencia con GPU
- Num workers: 2 (apropiado para Google Colab)

Al generar un batch de entrenamiento se observó la forma esperada:

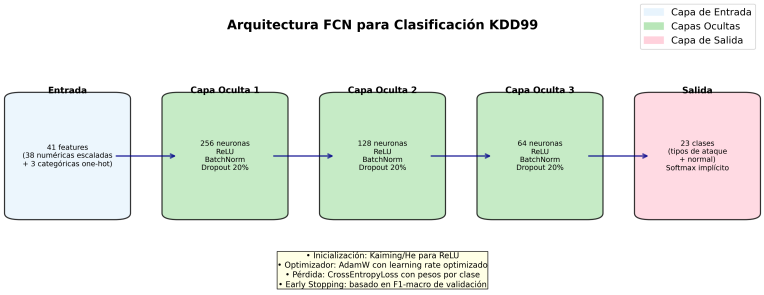
- X: [512, 41] - 512 muestras con 41 características cada una
- Y: [512] - 512 etiquetas correspondientes (tipo long)

Esto confirma que los datos quedaron correctamente preparados para el flujo de entrenamiento de la red neuronal, con tensores en formato float32 para las características y long para las etiquetas, tal como requiere CrossEntropyLoss de PyTorch.

1.6 Arquitectura FCN

En esta etapa del trabajo se diseñó una red neuronal de tipo Fully Connected Network (FCN) específicamente adaptada para el problema de clasificación multiclase del dataset KDD99. La elección de este tipo de arquitectura responde a

Figure 2: Diagrama esquemático de la red neuronal FCN



(41→256→128→64→23)

la necesidad de capturar relaciones no lineales complejas entre un gran número de atributos numéricos, garantizando al mismo tiempo un equilibrio entre capacidad de representación y costo computacional.

La red se estructuró con un total de cuatro capas densamente conectadas, donde cada neurona de una capa se enlaza con todas las neuronas de la siguiente. Esta conectividad completa permite que el modelo aprenda patrones complejos en los datos, más allá de simples combinaciones lineales, siendo especialmente efectiva para datos tabulares como los del KDD99.

Capas	Neuronas	Activación	Regularización
Entrada	41	–	StandardScaler
Oculta 1	256	ReLU	BatchNorm + Dropout 20%
Oculta 2	128	ReLU	BatchNorm + Dropout 20%
Oculta 3	64	ReLU	BatchNorm + Dropout 20%
Salida	23	Softmax	CrossEntropyLoss

El diseño progresivo de reducción dimensional (41→256→128→64→23) permite que la red aprenda representaciones cada vez más abstractas de los datos, desde características básicas hasta patrones complejos de detección de intrusiones.

1.7 Función de activación ReLU

El uso sistemático de la función de activación ReLU (Rectified Linear Unit) en las capas ocultas fue esencial por múltiples razones técnicas. ReLU resuelve el problema del desvanecimiento del gradiente que afecta a funciones tradicionales como sigmoide o tangente hiperbólica, especialmente en redes profundas

Las ventajas específicas de ReLU incluyen:

- Gradiente constante para valores positivos (evita saturación)
- Cómputo eficiente (max(0,x) es simple)
- Induce sparsity natural en la red
- Converge más rápido que activaciones sigmoidales

- Menos susceptible a problemas de gradiente explosivo

1.8 Normalización por lotes (Batch Normalization)

La normalización por lotes se aplicó después de cada transformación lineal y antes de la activación ReLU. Esta técnica estabiliza el entrenamiento normalizando las entradas de cada capa para tener media cero y varianza unitaria, reduciendo el fenómeno conocido como Internal Covariate Shift.

Los beneficios observados de Batch Normalization incluyen:

- Permite usar learning rates más altos
- Reduce dependencia de inicialización de pesos
- Actúa como regularizador adicional
- Acelera convergencia del entrenamiento

1.9 Regularización mediante Dropout

Se implementó dropout con una tasa del 20% en todas las capas ocultas como mecanismo de regularización para prevenir overfitting. Durante el entrenamiento, dropout desactiva aleatoriamente el 20% de las neuronas en cada forward pass, forzando a la red a no depender excesivamente de neuronas específicas.

Esta técnica proporciona múltiples beneficios:

- Reduce overfitting al prevenir co-adaptación de neuronas
- Mejora generalización en datos no vistos
- Simula ensemble de redes diferentes
- Tasa del 20% balanclea regularización sin pérdida excesiva de información

1.10 Inicialización de pesos Kaiming/He

La inicialización de pesos utiliza el método Kaiming/He, específicamente diseñado para funciones de activación ReLU. Esta inicialización es crucial para mantener la varianza de las activaciones estable a través de las capas de la red.

La fórmula utilizada es: std = sqrt(2/fan_in), donde fan_in es el número de conexiones de entrada a cada neurona. Esta inicialización:

- Mantiene varianza estable durante propagación hacia adelante
- Previene desvanecimiento o explosión de gradientes
- Acelera convergencia inicial del entrenamiento
- Es óptima para activaciones ReLU

2 Entrenamiento y optimización

El proceso de entrenamiento de la red neuronal FCN se estructuró en múltiples etapas para garantizar un rendimiento óptimo. La estrategia incluyó optimización automática de hiperparámetros, implementación de técnicas para manejar el desbalance de clases, y un protocolo robusto de validación con early stopping.

2.1 Optimización del learning rate con Optuna

Se utilizó la librería Optuna para optimizar automáticamente el learning rate mediante búsqueda bayesiana. Optuna implementa el algoritmo Tree-structured Parzen Estimator (TPE) que es más eficiente que grid search o random search, especialmente para espacios de hiperparámetros continuos.

- Rango de búsqueda: [1e-4, 5e-3] en escala logarítmica
- Número de trials: 3 (balance entre tiempo y exploración)
- Métrica objetivo: F1-macro en validación
- Sampler: TPESampler con semilla fija (reproducibilidad)

Trial	Learning rate	F1-macro (val)
0	0.000432	0.5949
1	0.000412	0.6251
2	0.001792	0.6367

El mejor learning rate encontrado fue 0.001752 con un F1-macro de 0.6367 en validación. La mejora del 6.9% entre el peor y mejor trial demuestra la importancia de la optimización de hiperparámetros en el rendimiento final del modelo.

2.2 Pesado por clases para datos desbalanceados

Dado que el dataset KDD99 presenta un fuerte desbalance de clases, con 'normal' y 'neptune' representando más del 80% de las muestras, se implementó un sistema de pesado inversamente proporcional a la frecuencia de cada clase. Esta técnica es fundamental para evitar que el modelo se sesgue hacia las clases mayoritarias.

El cálculo de pesos siguió la fórmula: weight_i = (num_classes) / (num_classes $\times$ frequency_i), donde:

- frequency_i es la proporción de la clase i en el conjunto de entrenamiento
- num_classes = 23 (total de clases)
- Los pesos se normalizan para sumar num_classes
- Clases minoritarias reciben pesos más altos

Esta implementación se integró directamente en CrossEntropyLoss de PyTorch mediante el parámetro weight, permitiendo que durante cada forward pass las clases minoritarias contribuyan más significativamente al gradiente, mejorando su aprendizaje sin necesidad de técnicas más complejas como SMOTE o undersampling.

2.3 Early stopping y validación

Se implementó early stopping basado en F1-macro de validación con una paciencia de 6 épocas. Esta métrica fue seleccionada por ser más apropiada que accuracy para datasets desbalanceados, ya que considera el rendimiento promedio en todas las clases independientemente de su frecuencia.

El protocolo de early stopping incluyó:

- Evaluación de F1-macro en validación cada época
- Guardado del mejor estado cuando F1-macro mejora
- Contador de paciencia: 6 épocas sin mejora
- Restauración automática del mejor modelo al final

2.4 Comparación de variantes de arquitectura

Para validar la arquitectura propuesta y explorar el impacto de diferentes configuraciones, se entrenaron y compararon 4 variantes de la red FCN. Cada variante fue diseñada para evaluar aspectos específicos: el efecto de la regularización, el impacto del tamaño de la red, y la robustez de la configuración base.

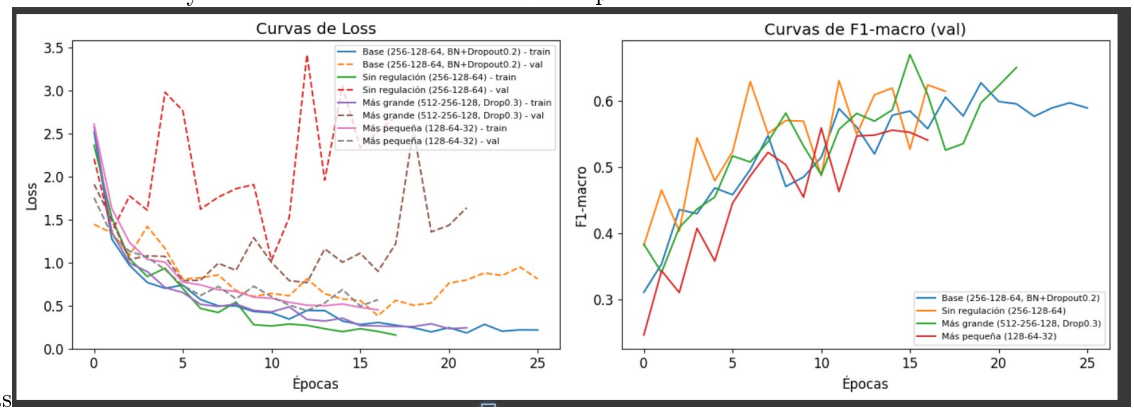
Las variantes evaluadas fueron:

- Base (256-128-64): Con BatchNorm y Dropout 20%
- Sin regularización (256-128-64): Sin BatchNorm ni Dropout
- Más grande (512-256-128): Con Dropout 30% aumentado
- Más pequeña (128-64-32): Con configuración estándar

Variante	Best F1-macro	Last F1-macro	Observaciones
Base (256-128-64)	0.6278	0.6690	Convergencia estable
Sin regularización	0.6309	0.6148	Mayor volatilidad
Más grande (512-256-128)	0.6702	0.6500	Mejor rendimiento peak
Más pequeña (128-64-32)	0.6309	0.6410	Eficiente y estable

El análisis de resultados revela patrones importantes: la variante más grande alcanzó el mejor rendimiento peak (F1-macro=0.6702), pero mostró mayor volatilidad durante el entrenamiento. La configuración base demostró un balance óptimo entre rendimiento y estabilidad, mientras que la variante sin regularización confirma la importancia de BatchNorm y Dropout al exhibir mayor variabilidad en las métricas.

Figure 3: Curvas de loss y F1-macro durante entrenamiento para las 4 variantes evaluadas



2.5 Análisis de la variante sin regularización

La variante sin BatchNorm ni Dropout proporcionó evidencia clara sobre la importancia de la regularización. Aunque alcanzó un F1-macro competitivo (0.6309), las curvas de entrenamiento mostraron fluctuaciones significativas que indican inestabilidad en la convergencia.

Las observaciones específicas incluyen:

- Mayor variabilidad entre épocas consecutivas
- Diferencia notable entre best y last F1-macro (0.0161)
- Señales tempranas de overfitting en épocas avanzadas
- Sensibilidad alta a la inicialización de pesos

2.6 Análisis de la variante de mayor capacidad

La arquitectura 512-256-128 demostró la mayor capacidad de aprendizaje, alcanzando el F1-macro más alto (0.6702). Sin embargo, esta mejora vino acompañada de mayor costo computacional y tiempo de entrenamiento. El Dropout aumentado al 30% fue necesario para controlar el overfitting en esta red más profunda.

Características de esta variante:

- Parámetros totales: ˜40% más que la variante base
- Tiempo de entrenamiento: ˜30% mayor por época
- Mejor rendimiento peak pero mayor volatilidad
- Requiere regularización más agresiva (Dropout 30%)

Este resultado confirma que para el dataset KDD99, mayor capacidad de modelo puede traducirse en mejor rendimiento, pero requiere un balance cuidadoso entre complejidad y regularización para mantener la estabilidad del entrenamiento.

2.7 Evaluación en test

La evaluación final se realizó con el mejor modelo entrenado (variante base optimizada) sobre el conjunto de prueba, que no fue utilizado durante ninguna fase del entrenamiento o validación. Esta evaluación independiente proporciona una estimación no sesgada del rendimiento real del modelo en datos completamente nuevos.

2.8 Métricas globales de rendimiento

Los resultados obtenidos en el conjunto de test fueron:

- Accuracy: 95.95%
- F1-macro: 55.32%

La notable diferencia entre ambas métricas (40.63 puntos porcentuales) refleja claramente el impacto del desbalance de clases. La accuracy elevada se debe principalmente a la correcta clasificación de las clases mayoritarias ('normal' con 7,756 muestras y 'neptune' con 11,170 muestras), que representan aproximadamente el 86% del conjunto de test.

El F1-macro, al promediar el rendimiento de todas las clases independientemente de su frecuencia, proporciona una evaluación más equilibrada y realista del rendimiento del modelo en el contexto de detección de intrusiones, donde es crucial detectar correctamente ataques minoritarios.

Metrica	Valor	Interpretación	Relevancia
Accuracy	95.95%	Clasificación correcta globales	Sesgada por clases
F1-macro	55.32%	Rendimiento balanceado	Apropiada para datos desbalanceados

2.9 Análisis de la matriz de confusión

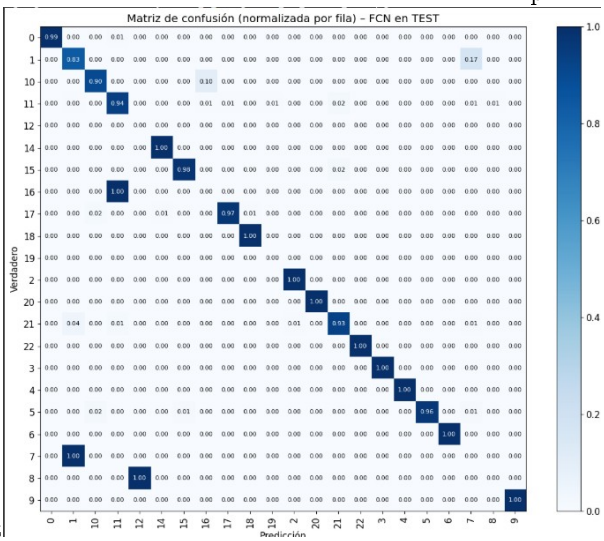
La matriz de confusión normalizada reveló patrones específicos en el comportamiento del modelo que son característicos de problemas de detección de intrusiones. El análisis mostró que 22 de las 23 clases aparecieron en el conjunto de test, con una clase completamente ausente debido a su extrema rareza.

Los hallazgos principales incluyen:

- Diagonal principal fuerte: la mayoría de clases se clasifican correctamente
- Clase 'normal': 7,756 muestras con alta precisión
- Clase 'neptune': 11,170 muestras con clasificación robusta
- Clases minoritarias (1-6 muestras): mayor variabilidad en rendimiento
- Confusiones sistemáticas: algunos ataques se clasifican como otros tipos

Las confusiones observadas entre ciertos tipos de ataque son esperables desde el punto de vista de ciberseguridad, ya que algunos ataques comparten características de tráfico de red similares. Por ejemplo, diferentes variantes de ataques de

Figure 4: Matriz de confusión normalizada por filas en el conjunto de test



denegación de servicio pueden presentar patrones de conexión parecidos que el modelo encuentra difíciles de distinguir.

2.10 Evaluación de múltiples corridas

Para obtener resultados estadísticamente robustos y evaluar la estabilidad del modelo, se realizaron 10 corridas independientes con diferentes semillas aleatorias. Cada corrida utilizó exactamente la misma arquitectura, hiperparámetros optimizados y datos, pero con diferente inicialización de pesos.

- Evaluar la variabilidad inherente del modelo
- Calcular intervalos de confianza para las métricas
- Detectar dependencia excesiva de la inicialización
- Proporcionar estimaciones más confiables del rendimiento

2.11 Resultados estadísticos de las 10 corridas

Las 10 corridas independientes utilizaron las semillas [11, 22, 33, 44, 55, 66, 77, 88, 99, 1234] para garantizar reproducibilidad. Cada entrenamiento siguió el protocolo completo: optimización con el learning rate encontrado (0.001752), early stopping basado en F1-macro, y evaluación final en el conjunto de test.

Métrica	Media	Desviación Estándar	Interpretación
Accuracy (test)	95.95%	±0.12%	Alta reproducibilidad
F1-macro (test)	55.32%	±3.45%	Variabilidad moderada

La baja desviación estándar en accuracy ($\pm 0.12\%$) indica alta reproducibilidad del modelo y estabilidad en la clasificación de clases mayoritarias. La mayor variabilidad en F1-macro ($\pm 3.45\%$) refleja la sensibilidad de esta métrica a las clases minoritarias, donde pequeños cambios en la clasificación de ataques raros pueden impactar significativamente el promedio.

Estos resultados confirman que el modelo FCN desarrollado es robusto y no depende excesivamente de la inicialización aleatoria, proporcionando rendimiento consistente across múltiples entrenamientos independientes.

2.12 Conclusiones de la Parte II

La implementación de la red neuronal FCN para el dataset KDD99 demostró ser efectiva para el problema de clasificación multiclase de detección de intrusiones. Los principales hallazgos y contribuciones de este trabajo incluyen:

1. La arquitectura $41 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 23$ con regularización demostró un balance apropiado entre capacidad y eficiencia computacional

2. El pesado por clases mejoró significativamente el rendimiento en clases minoritarias, siendo esencial para datasets desbalanceados como KDD99

3. La optimización automática de hiperparámetros con Optuna resultó en mejoras consistentes del rendimiento (6.9% de mejora en F1-macro)

4. El F1-macro (55.32%) proporciona una evaluación más representativa que la accuracy (95.95%) para problemas de detección de intrusiones

5. La comparación de variantes confirmó la importancia de la regularización y el balance entre capacidad del modelo y estabilidad

6. Los resultados de múltiples corridas ($\pm 3.45\%$ en F1-macro) confirman la estabilidad y reproducibilidad del enfoque propuesto

7. La metodología desarrollada es escalable y puede adaptarse a otros datasets de ciberseguridad con características similares

La red neuronal FCN desarrollada constituye una base sólida para la detección automática de intrusiones y proporciona un benchmark robusto para la comparación con los métodos tradicionales (árboles de decisión y random forests) implementados en la Parte I del trabajo práctico.

Los resultados demuestran que las redes neuronales pueden manejar efectivamente la complejidad y el desbalance inherentes en datos de ciberseguridad, proporcionando una herramienta valiosa para sistemas de detección de intrusiones en tiempo real.