

Detección de fraude en movimientos transaccionales con Deep Learning

17 de febrero de 2025



**Universidad
Internacional
de Valencia**

Titulación:

Master en Big Data & Data Science

Curso académico

2024 – 2025

Alumno/a:

Guerrero Balber, Cristian

D.N.I: 76086500Q

Director/a de TFM: Camilo
Andrés Pulzara Mora

Convocatoria:

Primera

De:

Planeta Formación y Universidades

Índice

Resumen	5
1. Introducción.....	7
2. Objetivos.....	11
2.1. Criterios de éxito. Alta detección de fraudes a costa de una baja precisión. .	11
2.2. Objetivos	12
3. Desafíos y Estrategias en la Detección de Fraude Bancario en Línea	14
4. Marco Teórico.....	17
4.1. Introducción a Redes Neuronales.....	17
4.2. Redes Neuronales Convolucionales (CNN)	19
4.3. Métricas	20
4.4. Herramientas de Software	22
5. Desarrollo del proyecto y resultados	24
5.1. Metodología.....	24
5.2. Planteamiento del problema: detección de fraudes bancarios	26
5.3. Identificación y selección del dataset.....	27
5.3.1. Descripción del dataset	28
5.4. Análisis Exploratorio de Datos (EDA, por sus siglas en inglés).....	30
5.5. Redes Neuronales Profundas.....	42
5.5.1. Red Neuronal Densa (DNN)	46
5.5.2. Red Neuronal Convolucional (CNN)	57
5.5.3. Modelo Combinado: DNN + CNN	66
5.6. Resultados generales	68
6. Conclusiones	70
6.1. Evaluación del cumplimiento de los objetivos	71
6.2. Trabajos futuros.....	73
7. Referencias	74
Apéndice I.....	77

Índice de figuras

Figura 1. Red Neuronal Profunda. Elaboración propia a través de Canvas.....	17
Figura 2. Componentes de una red neuronal. (A) Perceptrón, unidad básica de una ANN. (B) Red Neuronal conectada a múltiples unidades de perceptrón. (C) Ejemplos de cuatro funciones de activación diferentes: sigmoide, tangente hiperbólica, identidad y unidad lineal rectificada. Fuente: Choi et al., (2020)	18
Figura 3. Las salidas (no los filtros) de cada capa (en horizontal) de una arquitectura típica de red convolucional aplicada a la imagen de un perro Samoyedo (abajo a la izquierda; y entradas RGB (rojo, verde, azul), abajo a la derecha). Cada imagen rectangular es un mapa de características. Fuente: (LeCun et al., 2015).....	19
Figura 4. Cronograma de las tareas definidas. Elaboración propia.	25
Figura 5. Diagrama de las fases del desarrollo. Elaboración propia.	25
Figura 6. Desbalanceo de clases entre transacciones legítimas (0) y fraudulentas (1). Elab. Propia.	31
Figura 7. Análisis KDE de las características numéricas. Elaboración propia.	33
Figura 8. Matriz de Correlaciones. Elaboración propia.	35
Figura 9. Distribución de clases en los diferentes conjuntos tras submuestreo. Elaboración propia.	37
Figura 10. Distribución de los datos en los distintos conjuntos: Entrenamiento, validación y test. Elaboración propia.....	37
Figura 11. Silhouette Score en función de épsilon y min_samples. Elab. Propia.....	40
Figura 12. Reducción de hiperdimensiones mediante PCA y t-SNE. Clusters y Clases. Elab. Propia	41
Figura 13. Arquitectura de red neuronal densa desarrollada. Elab. Propia.....	47
Figura 14. DNN entrenada sin balanceo de clases en red neuronal ni dataset. Elab. Propia	49
Figura 15. DNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal pero no en el dataset. Elab. Propia	50
Figura 16. Distribución de clases en el dataset de entrenamiento para la CTGAN. Elab. Propia	52
Figura 17. Distribución de clases en el dataset de entrenamiento tras la generación de datos sintéticos de la clase 1 con CTGAN. Elab. Propia	52
Figura 18. DNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal y también en el dataset mediante una CTGAN. Elab. Propia	53
Figura 19. Distribución de clases en el dataset de entrenamiento tras la generación de datos sintéticos de la clase 1 con SMOTE. Elab. Propia	54
Figura 20. DNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal y también en el dataset mediante SMOTE. Elab. Propia	55
Figura 21. Imágenes sintéticas a partir de vectores de transacciones legítimas. Elab. Propia	60
Figura 22. Imágenes sintéticas a partir de vectores de transacciones fraudulentas. Elab. Propia	61

Figura 23. Imágenes sintéticas a partir de vectores de transacciones fraudulentas. Elab. Propia	61
Figura 24. Arquitectura de red neuronal convolucional típica. En este caso se trata de una CNN compuesta por dos bloques convolucionales. (Adrán C. & Gabriel M., s.f., Figura 18.).	62
Figura 25. Arquitectura de red neuronal convolucional desarrollada. Elab. Propia.....	64
Figura 26. CNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal pero no en el dataset de imágenes. Elab. Propia	65
Figura 27. DNN + CNN combinados vs DNN + CNN promediados. Elab. Propia	66

Índice de tablas

Tabla 1. Principales técnicas de Machine Learning para detección de fraude en movimientos bancarios	15
Tabla 2. Comparación de los modelos de tipo DNN.....	56
Tabla 3. Comparación de los modelos de tipo CNN.....	66
Tabla 4. Comparación de los modelos de tipo CNN Combinada.....	67
Tabla 5. Resultados totales.....	69

Resumen

En el presente trabajo se aborda el desafío de la detección de fraudes bancarios en línea, un problema crítico en el contexto actual de la banca digital. Se establecen criterios de éxito enfocados en lograr una alta tasa de detección de fraudes, aunque a costa de una precisión más baja siempre y cuando se mantenga una exactitud alta, lo que implica una mayor tolerancia a los falsos positivos.

El objetivo general del proyecto es aplicar técnicas de deep learning para detectar transacciones fraudulentas en el ámbito financiero, contribuyendo a la seguridad en este sector. Los objetivos específicos incluyen seleccionar un dataset adecuado, eliminar variables redundantes, transformar los datos para el entrenamiento de redes neuronales, y evaluar la utilidad de la clusterización. Además, se busca implementar modelos de deep learning, como redes neuronales densas (DNN) y convolucionales (CNN), para lograr una tasa de detección superior al 80% sin comprometer la exactitud, comparando los resultados para elegir el modelo más robusto.

Tras una identificación y selección cuidadosa del dataset, se ha realizado un **Análisis Exploratorio de Datos (EDA)** para comprender las características del conjunto de datos y preparar la información para los modelos. El núcleo del trabajo se centra en el desarrollo y prueba de diversas arquitecturas de redes neuronales, incluyendo la **Red Neuronal Densa (DNN)**, la **Red Neuronal Convolucional (CNN)**, y un modelo combinado **DNN + CNN**. Para el entrenamiento de la CNN, se ha llevado a cabo una conversión de las instancias tabulares de vector a pseudo-imagen mediante transformaciones matriciales. Cada uno de estos enfoques se ha evaluado para determinar su capacidad e identificar fraudes con alta efectividad, superando los retos inherentes al desbalanceo de clases y la variabilidad de los patrones fraudulentos.

Los resultados obtenidos indican que los modelos DNN y CNN combinados en sus predicciones promediadas han cumplido con los criterios establecidos, alcanzando un AUC-ROC y una sensibilidad superior al 80%. Esto permite una detección eficiente de fraudes sin afectar la exactitud general del sistema.

Palabras clave: *Detección de fraudes bancarios, redes neuronales densas (DNN), redes neuronales convolucionales (CNN), AUC-ROC, sensibilidad, balanceo de clases, transformación matricial, pseudo-imágenes, análisis exploratorio de datos (EDA), clusterización en detección de fraudes, modelos combinados*

Abstract

This work addresses the challenge of online bank fraud detection, a critical issue in the current context of digital banking. Success criteria are established with a focus on achieving a high fraud detection rate, even at the cost of lower precision, as long as high accuracy is maintained, which implies a greater tolerance for false positives.

The general objective of the project is to apply deep learning techniques to detect fraudulent transactions in the financial sector, contributing to security in this domain. Specific objectives include selecting an appropriate dataset, eliminating redundant variables, transforming the data for training neural networks, and evaluating the utility of clustering. Additionally, the goal is to implement deep learning models, such as deep neural networks (DNN) and convolutional neural networks (CNN), to achieve a fraud detection rate above 80% without compromising accuracy, comparing the results to select the most robust model.

After careful identification and selection of the dataset, an Exploratory Data Analysis (EDA) was conducted to understand the dataset's characteristics and prepare the information for the models. The core of the work focuses on the development and testing of various neural network architectures, including the Deep Neural Network (DNN), Convolutional Neural Network (CNN), and a combined DNN + CNN model. For CNN training, a conversion of tabular vector instances into pseudo-images was carried out through matrix transformations. Each of these approaches was evaluated to determine their ability to identify fraud effectively, overcoming challenges related to class imbalance and the variability of fraudulent patterns.

The results obtained indicate that the combined DNN and CNN models in their averaged predictions have met the established criteria, achieving an AUC-ROC and sensitivity above 80%. This enables efficient fraud detection without compromising the overall accuracy of the system.

Keywords: *bank fraud detection, deep neural networks (DNN), convolutional neural networks (CNN), AUC-ROC, recall, class balancing, matrix transformation, pseudo-images, exploratory data analysis (EDA), clustering in fraud detection, combined models.*

1. Introducción

El fraude bancario en línea es una preocupación creciente a nivel mundial, debido a la expansión de los servicios financieros digitales y la globalización de las transacciones. Según la Organización de las Naciones Unidas (ONU), los delitos financieros y el fraude cibernético representan una amenaza significativa a nivel global. Como organización, resalta la preocupación por el aumento de la tasa y diversidad de los delitos cometido en el mundo digital, así como el reconocimiento de que diversos delincuentes se están sirviendo de las tecnologías de la información y las comunicaciones para llevar a cabo actividades delictivas (Asamblea General de las Naciones Unidas, 2019).

Las regulaciones y leyes a nivel internacional buscan mitigar estos riesgos. Un ejemplo es el Reglamento General de Protección de Datos (RGPD) de la Unión Europea, que exige a las instituciones financieras implementar medidas robustas de seguridad y protección de datos para prevenir fraudes (Reglamento (UE) 2016/679).

Las instituciones financieras han comenzado a adoptar tecnologías avanzadas para hacer frente a este desafío. Entre ellas, la inteligencia artificial (IA) y el aprendizaje automático (machine learning, ML) juegan un papel crucial en la detección y prevención del fraude. Estas tecnologías permiten analizar grandes volúmenes de datos en tiempo real, identificando patrones y comportamientos sospechosos que podrían indicar actividad fraudulenta.

Una de las aplicaciones más prometedoras es el aprendizaje profundo (deep learning), que utiliza redes neuronales profundas para mejorar la precisión en la identificación de fraudes. Esta tecnología se ha demostrado efectiva en el análisis de transacciones financieras y la detección de anomalías que podrían pasar desapercibidas con métodos tradicionales (A. Roy et. Al., 2018).

En la actualidad, el uso de la computación y la IA ha cobrado una relevancia significativa en diversas áreas del conocimiento y la industria. Estas herramientas permiten la automatización de procesos, la toma de decisiones basadas en datos y la resolución de problemas complejos en campos como la medicina, la economía, la seguridad y la generación de contenido. En particular, el ML, se ha convertido en una disciplina fundamental para el desarrollo de sistemas capaces de realizar tareas predictivas y adaptativas con alta eficiencia.

El aprendizaje automático se divide en dos grandes categorías: supervisado y no supervisado. En el aprendizaje supervisado, el modelo se entrena con un conjunto de datos etiquetados, donde cada entrada está asociada a una salida esperada. Por otro lado, en el aprendizaje no supervisado, el modelo trabaja con datos sin etiquetar, identificando patrones o estructuras subyacentes en la información. Ambos enfoques tienen aplicaciones en diversos problemas, dependiendo de la naturaleza de los datos disponibles y del objetivo del análisis.

Dentro del aprendizaje supervisado, uno de los problemas más comunes es el de clasificación, donde el objetivo es asignar una etiqueta a cada instancia en función de sus características. En particular, los problemas de clasificación binaria presentan retos adicionales cuando los datos están desbalanceados, es decir, cuando una de las clases es considerablemente más frecuente que la otra. Este desbalanceo puede provocar que los modelos prioricen la clase mayoritaria, lo que reduce la capacidad de detección de la clase minoritaria y compromete la efectividad del modelo.

Para abordar estos desafíos, los modelos de aprendizaje automático clásicos, como los árboles de decisión, los bosques aleatorios y las máquinas de soporte vectorial, han demostrado ser herramientas eficaces en la resolución de problemas de clasificación binaria. Además, diversas técnicas de balanceo de datos, como el sobremuestreo de la clase minoritaria, el submuestreo de la clase mayoritaria y la generación de datos sintéticos, han sido implementadas para mitigar el impacto del desbalanceo y mejorar el rendimiento de los modelos.

Sin embargo, en los últimos años, el uso de redes neuronales ha cobrado una importancia creciente en la resolución de problemas de clasificación con desbalanceo de clases. Las redes neuronales profundas (*deep learning*) ofrecen ventajas significativas, como la capacidad de aprender representaciones complejas de los datos y la posibilidad de automatizar la extracción de características sin necesidad de una ingeniería de atributos explícita. Estas propiedades permiten mejorar la generalización y la capacidad de detección de patrones ocultos en conjuntos de datos masivos y heterogéneos.

En el contexto de la detección de fraudes bancarios, el uso de redes neuronales profundas se justifica por la naturaleza del problema, caracterizado por un alto desbalanceo en los datos y la necesidad de identificar transacciones fraudulentas con alta precisión. La aplicación de modelos de *deep learning* permite mejorar la capacidad de predicción, reducir los falsos positivos y minimizar las pérdidas económicas asociadas a actividades fraudulentas. Por tanto, el presente trabajo se centra en el estudio y aplicación de técnicas de *deep learning* para la detección de fraudes en transacciones bancarias.

El objetivo general del proyecto es aplicar técnicas avanzadas de *deep learning* para detectar transacciones fraudulentas en el ámbito financiero, con el fin de mejorar la seguridad y la confianza en los servicios bancarios en línea. Este enfoque pretende contribuir significativamente al desarrollo de soluciones más eficientes en la detección temprana de fraudes, lo que podría prevenir pérdidas económicas considerables y proteger la integridad de las transacciones bancarias. Para lograr este objetivo, se han planteado varios objetivos específicos que guían el desarrollo del proyecto.

Para superar los objetivos propuestos, se ha **seleccionado un dataset** adecuado que contenga transacciones bancarias etiquetadas como legítimas o fraudulentas. El dataset seleccionado es la suite Bank Account Fraud (BAF) (Jesus et al., s. f.), publicada en NeurIPS 2022, que incluye seis conjuntos de datos sintéticos para la detección de

fraudes en cuentas bancarias. Es un conjunto realista y robusto, con datos desequilibrados, tipos de sesgo controlados y dinámicos, lo que lo convierte en un recurso adecuado para evaluar métodos de ML y ML justo. Además, preserva la privacidad mediante técnicas de privacidad diferencial y codificación de características.

Como parte del **preprocesamiento de los datos**, se han identificado y eliminado variables redundantes o con nula varianza, lo que mejora la eficiencia del modelo y la calidad de los resultados. Además, se ha realizado una transformación de los datos para adaptarlos de forma óptima al entrenamiento de redes neuronales, lo que incluye la conversión de variables categóricas a numéricas a través del método *OneHotEncoder*, el escalado de los datos en un rango entre 0 y 1 con *MinMaxScaler*, así como la conversión de las instancias tabulares en arreglos matriciales de tres canales de *Numpy* generando imágenes sintéticas. Cada transacción se ha convertido en una imagen RGB mediante operaciones matemáticas específicas que resaltan las relaciones entre características. Así, una CNN puede ser entrenada con datos tabulares permitiendo la integración de datos anonimizados con sistemas de visión por computadora, aspecto que no es posible con redes neuronales densas.

Otro aspecto clave ha sido la **evaluación de la utilidad de la clusterización**, en particular mediante el algoritmo *DBSCAN*, con el objetivo de identificar patrones subyacentes en las transacciones fraudulentas. Sin embargo, aunque se ha explorado esta técnica, no se han logrado formar clusters interesantes, ya que solo se obtuvo un cluster y un pequeño porcentaje de ruido, lo que sugiere que este enfoque no aportó información relevante para la detección de fraudes en este caso.

Se han implementado redes neuronales profundas (DNN) y redes neuronales convolucionales (CNN), tanto de manera individual como combinada, para evaluar su rendimiento en cuanto a la identificación de transacciones fraudulentas. Además, **se ha trabajado con técnicas de balanceo de datos como SMOTE y CTGAN** para mejorar el desempeño del modelo ante el desbalanceo típico en este tipo de datasets. El objetivo principal ha sido lograr una alta sensibilidad, reduciendo al mismo tiempo los falsos positivos, mientras se mantenía un nivel aceptable de precisión y exactitud.

Seguidamente, **se han comparado los resultados obtenidos** de cada modelo para seleccionar el que ofrezca los mejores resultados en términos de cumplimiento de los criterios de éxito definidos y la robustez frente a variaciones en los datos, asegurando que el modelo elegido sea capaz de identificar los fraudes de manera precisa y eficiente en diferentes escenarios.

Los resultados obtenidos han demostrado que, en conjunto, **los modelos DNN y CNN han superado los criterios establecidos**, con un AUC-ROC superior a 0,8 y una sensibilidad mayor al 80%, lo que permite detectar una gran proporción de fraudes sin comprometer la exactitud global del sistema. Entre los modelos evaluados, la combinación de CNN y DNN ha mostrado el mejor desempeño, logrando un buen equilibrio entre las métricas de evaluación. Estos resultados sugieren que las redes



neuronales profundas, especialmente cuando se combinan diferentes arquitecturas, son una solución efectiva para la detección de fraudes en el ámbito bancario.

Estos resultados sugieren que las redes neuronales profundas, y en particular la estrategia de combinar arquitecturas como las CNN y DNN, pueden ofrecer una solución altamente efectiva para abordar el desafío de la detección de fraudes en el ámbito bancario, siendo capaces de captar patrones complejos en los datos y mejorar el desempeño general del sistema en condiciones prácticas.

2. Objetivos

Dado el carácter desbalanceado de los datos y la criticidad de identificar correctamente las transacciones fraudulentas, métricas como la exactitud, precisión, sensibilidad y AUC-ROC deben ser evaluadas y priorizadas según el contexto del problema. Estas métricas no solo permiten cuantificar el rendimiento del modelo, sino que también guían la toma de decisiones en la optimización del mismo, asegurando un equilibrio entre la detección efectiva de fraudes y la minimización de falsos positivos. Por ello, la definición de estos criterios de éxito es un paso previo indispensable para establecer objetivos realistas y alineados con las necesidades del negocio.

2.1. Criterios de éxito. Alta detección de fraudes a costa de una baja precisión.

En escenarios de detección de fraudes donde se prioriza una alta sensibilidad a costa de una menor precisión, es fundamental establecer umbrales adecuados para las métricas de evaluación del modelo. A continuación, se definen los valores objetivo para cada métrica junto con su justificación:

- **Exactitud (Accuracy):** Dado el desbalance típico en los datos de fraude, donde las transacciones fraudulentas representan una minoría, la exactitud puede ser engañosa. Un modelo que clasifica todas las transacciones como no fraudulentas podría mostrar una alta exactitud debido a la predominancia de la clase mayoritaria. Sin embargo, situaciones donde se regula el umbral de decisión para favorecer la sensibilidad, la exactitud puede verse comprometida al, por ejemplo, hacer todo lo contrario a lo anterior: clasificar todas las transacciones como fraudulentas. Por lo tanto, la exactitud deberá ser, al menos, de un 90%.
- **Precisión (Precision):** La precisión mide la proporción de transacciones identificadas como fraudulentas que realmente lo son. En este escenario, se acepta una menor precisión para asegurar una mayor detección de fraudes. **Un valor de precisión muy baja en torno al 5%** podría ser aceptable siempre y cuando se mantenga una exactitud global por encima de lo marcado en el punto anterior. Esto es, aunque el modelo se equivoca en el 95% de sus predicciones fraudulentas dando falsos positivos, en global el modelo solo se equivoca en un 20% de los casos.
- **Sensibilidad (Recall o Tasa de Verdaderos Positivos):** La sensibilidad es crucial en este contexto, ya que indica la capacidad del modelo para identificar transacciones fraudulentas. Se busca una sensibilidad alta, **idealmente por encima del 80%**, para garantizar que la mayoría de los fraudes sean detectados.
- **Área Bajo la Curva ROC (AUC-ROC):** El AUC-ROC proporciona una medida general de la capacidad del modelo para distinguir entre clases. Un valor de

AUC-ROC superior a 0,8 se considera aceptable en este contexto, indicando un buen equilibrio entre sensibilidad y especificidad.

Estos valores se basan en prácticas comunes en la detección de fraudes, donde la prioridad es identificar la mayor cantidad posible de actividades fraudulentas, incluso si esto implica una mayor tasa de falsos positivos.

Este escenario, podría darse donde el costo de los falsos positivos no sea muy alto y esto depende, en gran medida, de la estrategia de la empresa; si un banco neutraliza una transacción con una precisión tan baja, el costo podría repercutir en gran medida sobre la reputación de la empresa con repercusiones económicas a medio y largo plazo. Sin embargo, mediante una estrategia conservadora donde solo se neutralice la operación si la predicción del modelo es de transacción fraudulenta con un umbral de decisión alto donde la precisión crezca, este impacto podría reducirse. También, se podría considerar la apertura de una cuenta de crédito limitada en función de la predicción y de la seguridad de esta, haciendo que el fraude cause menos pérdidas en caso de realmente serlo.

2.2. Objetivos

Este proyecto tiene como objetivo general contribuir al avance de la ciencia de datos, específicamente en el campo del *machine learning* mediante el uso de técnicas de *deep learning*, aplicadas a la detección de transacciones fraudulentas. A través de este enfoque, se busca prevenir acciones delictivas en el ámbito financiero, promoviendo así la seguridad y el bienestar global. Para lograr esto, se plantean los siguientes objetivos específicos:

- Identificar y seleccionar un dataset representativo de transacciones bancarias que incluya etiquetas de clase para instancias legítimas y fraudulentas, asegurando que sea adecuado para el entrenamiento y validación de modelos de deep learning.
- Detectar si existe alguna variable redundante debido a multicolinealidad o nula varianza de la misma.
- Transformar el *Dataset* de forma que sea relativamente óptimo para alimentar el entrenamiento de una red neuronal profunda.
- Verificar si mediante la clusterización se puede obtener información relevante para la detección de fraudes.
- Implementar un modelo de *Deep learning* capaz de detectar, al menos, el 80% de las transacciones fraudulentas sin comprometer la exactitud (*accuracy*) del modelo en general. Esto es, que la exactitud sea mayor o igual al 80%. Además, la métrica AUC-ROC debe también ser mayor o igual a 0,8.
 - Diseñar una red neuronal profunda (DNN por sus siglas en inglés) para clasificación binaria (transacción legítima o fraudulenta).
 - Diseñar una red neuronal convolucional (CNN por sus siglas en inglés) para clasificación binaria alimentada por imágenes sintéticas creadas a

- partir de vectores unidimensionales (instancias individuales normalizadas).
- Combinar ambos modelos.
- Evaluar los resultados, verificar si uno o más modelos cumple las condiciones de éxito y escoger el mejor en términos de resultados y robustez frente a variaciones de datos externos así como de cumplimiento de los criterios de éxito.

3. Desafíos y Estrategias en la Detección de Fraude Bancario en Línea

Con el uso generalizado de tecnologías avanzadas de Internet, la banca en línea (también conocida como banca por Internet) se ha convertido en un canal principal para las operaciones bancarias tanto minoristas como empresariales. Sin embargo, las actividades fraudulentas en la banca en línea están evolucionando y volviéndose cada vez más sofisticadas, lo que supone una amenaza grave para la seguridad y confianza en este tipo de servicios. Este tipo de fraude se ha convertido en un problema crítico dentro de la gestión de delitos financieros para los bancos, generando pérdidas significativas debido a la aparición y evolución de estrategias complejas, como estafas de phishing, infecciones por malware y sitios web falsos.

Theobald, M. (2022), publicó en su *Informe Estado Global de Fraude e Identidad de LexisNexis Risk Solutions* los siguientes hallazgos clave:

- Incremento en ataques de bots: Las empresas han experimentado un aumento del 38% en ataques de bots maliciosos en 2021, siendo el comercio electrónico especialmente vulnerable con un aumento del 155% en estos ataques.
- Crecimiento de ataques por humanos: A nivel mundial, los ataques iniciados por humanos han aumentado un 32% desde 2021, destacándose América del Norte con un incremento del 52%.
- Evolución del fraude con nuevos métodos de pago: La adopción y demanda de métodos de pago sin contacto en Asia-Pacífico ha impulsado el aumento del fraude con códigos QR. Los pagos con código QR y las transferencias "peer-to-peer" están ganando popularidad en el sudeste asiático e India. La modalidad "compra ahora y paga después" está incrementando el fraude en la apertura de nuevas cuentas, especialmente en Europa, Medio Oriente y África.
- Riesgos en la experiencia del cliente: Las redes de fraude son cada vez más sofisticadas y presentes en el ecosistema digital, lo que ha llevado a un aumento significativo de estafas, como la ingeniería social, el robo de identidad y el fraude de adquisición de cuentas, con un incremento del 211% en los ataques de inicio de sesión en aplicaciones móviles.
- Desafíos en la verificación de identidad: Verificar la identidad del cliente sigue siendo un reto importante para las empresas, debido a la falta de datos de terceros en tiempo real y el seguimiento limitado de transacciones en línea.

Detectar de manera efectiva y eficiente estos tipos de fraude es uno de los principales desafíos que enfrentan las instituciones bancarias debido a los siguientes motivos (Wei et al., 2013):

- El fraude en la banca en línea presenta características muy complejas.
- Los actores sospechosos son hábiles e inteligentes en la realización de actividades fraudulentas.

- Los comportamientos ilícitos son altamente dinámicos.
- El fraude se oculta dentro de comportamientos diversificados de los clientes.
- Las transacciones fraudulentas están dispersas en grandes volúmenes de datos desequilibrados.
- Las actividades fraudulentas ocurren en períodos de tiempo muy breves, lo que exige una detección en tiempo real.

La detección inmediata es crucial, ya que recuperar las pérdidas es sumamente difícil si el fraude no se identifica a tiempo. Además, la mayoría de los clientes no revisa regularmente el historial de sus cuentas bancarias, lo que retrasa la identificación y notificación de transacciones fraudulentas. Esto disminuye las posibilidades de recuperar los fondos perdidos. Adicionalmente, cada alerta generada por los sistemas de detección debe ser investigada manualmente, lo cual consume mucho tiempo. Por esta razón, los sistemas de detección de fraude en la banca en línea deben ser altamente precisos, contar con una elevada tasa de detección y generar un número reducido de falsos positivos, permitiendo una gestión eficaz de las alertas en entornos bancarios complejos.

Con el fin de evidenciar las principales estrategias empleadas para la detección de fraudes bancarios, en Zabala et al., (2022) se exponen evidencias de concordancia en cuanto al uso de técnicas de *Machine Learning* (ML) aplicadas a este tipo de problema. De acuerdo con este estudio, las 5 metodologías principales son las siguientes:

Técnicas	Porcentaje de técnicas principales en la revisión de literatura
Redes neuronales	7 (32%)
Bosques aleatorios	5 (23%)
Naive Bayes	4 (18%)
Máquinas Vectoriales de soporte	4 (18%)
Modelos lineales generalizados (modelo logit, probit, log-log)	2 (9%)
Total	22 (100%)

Tabla 1. Principales técnicas de Machine Learning para detección de fraude en movimientos bancarios

Esta revisión documental permitió identificar las principales técnicas de ML aplicadas a la seguridad y prevención de fraudes financieros, destacando el uso de redes neuronales, Random Forest y Naive Bayes. Las redes neuronales, aunque muy versátiles, no siempre son las más precisas, mientras que Naive Bayes, por su simplicidad y análisis probabilístico, ofrece mayor precisión. A pesar de estas conclusiones, el estudio presenta limitaciones, como la falta de enfoques locales y el uso de investigaciones desactualizadas, lo que sugiere la necesidad de futuras investigaciones centradas en el análisis de ML y su efectividad en contextos nacionales.

La clasificación de datos desbalanceados es un problema en el que las proporciones de las clases de un conjunto de datos varían considerablemente. En este caso, una clase está representada por un número reducido de muestras (llamada clase minoritaria),

mientras que la otra clase está compuesta por la mayoría de las muestras (llamada clase mayoritaria). Este problema provoca que el rendimiento de un clasificador se incline hacia la clase mayoritaria, lo que genera un sesgo en el desempeño, ya que las soluciones tienden a tener mejor precisión para la clase mayoritaria y una precisión deficiente para la clase minoritaria.

Es relevante destacar que el problema de detección de fraudes de movimientos bancarios es un problema con clases desbalanceadas. De acuerdo con Kaur et al., (2020), las distribuciones de datos desbalanceadas son comunes en aplicaciones reales como la detección de fallos, la detección de fraudes y el diagnóstico médico. Las técnicas más populares para abordar este problema incluyen la regla de limpieza de vecindario, el método SMOTE con nivel seguro, algoritmos sensibles al costo y redes neuronales. Existen tres enfoques principales para tratar el desbalanceo en la clasificación de datos:

- Métodos de preprocessamiento.
- Enfoques centrados en algoritmos.
- Enfoques híbridos.

Según este autor, en el preprocessamiento se utilizan métodos de remuestreo como el sobremuestreo o submuestreo aleatorio (conocido en inglés como *oversampling* y *undersampling* respectivamente), o una combinación de ambos para equilibrar la cantidad de muestras en cada clase. Sin embargo, este enfoque solo afecta el conjunto de entrenamiento sin modificar el algoritmo de aprendizaje. El enfoque centrado en algoritmos ajusta las suposiciones para favorecer a la clase minoritaria y cambia los costos para equilibrar las clases.

En cuanto a los algoritmos de ML que propone el artículo anteriormente mencionado según la aplicación concreta de detección de fraude, destaca Redes Neuronales (*Neural Networks*, NN por sus siglas en inglés), máquinas de soporte vectorial (*Support Vector Machine*, SVM por sus siglas en inglés), árboles de decisión, vecinos más cercanos (*K-Nearest Neighbors*, KNN por sus siglas en inglés).

Dos de estos algoritmos propuestos coinciden con el estudio realizado por Kaur et al., (2020), NN y SVM. Además, bosques aleatorios es un modelo de ensamblado basado en un conjunto de árboles de decisión, por lo que se puede tomar como tercera coincidencia.

Para el desarrollo de este proyecto, se utilizarán algoritmos de *Deep Learning*. Los algoritmos de deep learning son una categoría de algoritmos de machine learning donde se utilizan múltiples capas ocultas (Figura 1) para mejorar los resultados. Se ha demostrado que el deep learning es una solución muy prometedora para abordar el fraude en las transacciones financieras, aprovechando al máximo los grandes volúmenes de datos de los bancos (A. Roy et. Al., 2018). El término deep learning se refiere de manera general al machine learning mediante una red neuronal artificial (ANN) profunda y de múltiples capas.

4. Marco Teórico

4.1. Introducción a Redes Neuronales

Las redes neuronales son un conjunto de modelos inspirados biológicamente en las neuronas humanas. Particularmente, las redes neuronales densas están compuestas por capas ocultas de unidades de procesamiento no lineales, donde cada neurona puede enviar datos a otra neurona conectada dentro de las capas ocultas. Estas unidades de procesamiento descubren representaciones intermedias de manera jerárquica. Las características descubiertas en una capa forman la base para el procesamiento de la capa siguiente. De esta manera, los algoritmos de *deep learning* aprenden conceptos intermedios entre la entrada en bruto y el conocimiento objetivo A. Roy et. Al (2018).

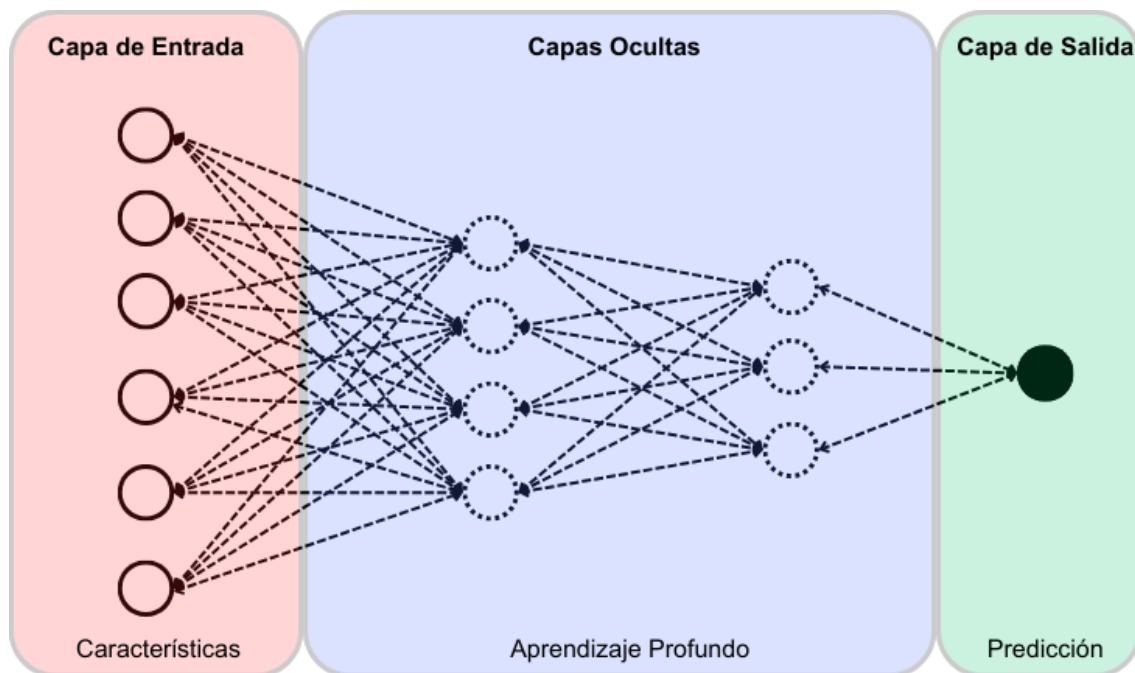


Figura 1. Red Neuronal Profunda. Elaboración propia a través de Canvas.

De acuerdo con Choi et al., (2020), el perceptrón (Figura 2A), es un algoritmo de aprendizaje automático que recibe como entrada un conjunto de características junto con sus respectivas etiquetas y busca definir una frontera de decisión, ya sea una línea, un plano o un hiperplano, dependiendo de la dimensión del espacio en el que se representen los datos. Para ello, transforma las características mediante la función sigmoide (Figura 2C).

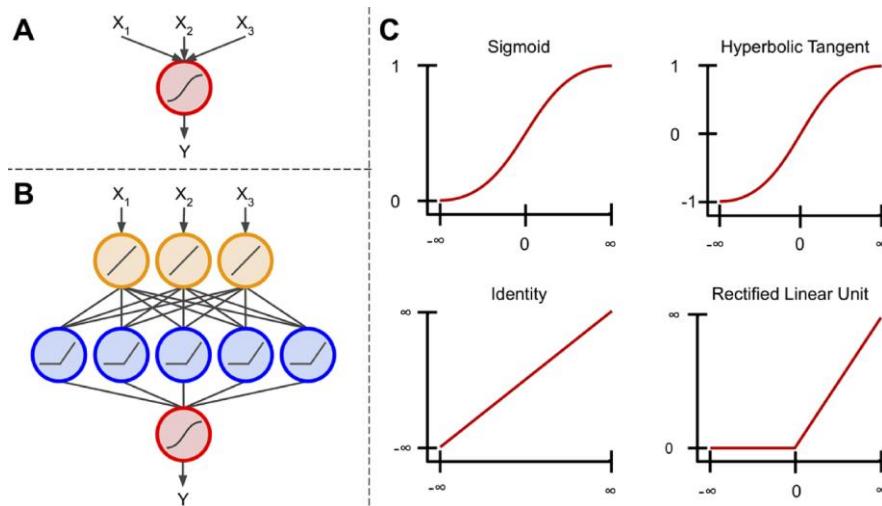


Figura 2. Componentes de una red neuronal. (A) Perceptrón, unidad básica de una ANN. (B) Red Neuronal conectada a múltiples unidades de perceptrón. (C) Ejemplos de cuatro funciones de activación diferentes: sigmoide, tangente hiperbólica, identidad y unidad lineal rectificada. Fuente: Choi et al., (2020)

Cuando se combinan varios perceptrones, el modelo resultante se conoce como perceptrón multicapa o red neuronal artificial (ANN). Estas redes suelen estar compuestas por una capa de nodos de entrada, una o varias capas ocultas intermedias y una capa de nodos de salida. En las ANN más básicas, el número de capas ocultas suele variar entre cero y tres, mientras que en las redes neuronales profundas este número puede aumentar hasta decenas o incluso cientos de capas Choi et al., (2020).

En la mayoría de los casos, las ANN funcionan propagando la información en una sola dirección, desde la entrada hasta la salida. Este tipo de arquitectura se denomina red de propagación hacia adelante (feedforward neural network), donde los datos pasan de cada nodo en una capa a todos los nodos de la siguiente capa, transformándose en cada paso (Figura 2B) Choi et al., (2020).

Durante el proceso de aprendizaje, la red realiza un pase hacia adelante (forward pass), donde calcula las salidas de cada capa aplicando funciones no lineales, como la unidad lineal rectificada (ReLU) o la función sigmoide. Luego, en el pase hacia atrás (backward pass), se calculan los gradientes del error con respecto a los pesos de la red, utilizando la regla de la cadena para derivadas. Estos gradientes se usan para actualizar los pesos, minimizando así el error entre las predicciones de la red y los valores reales. Este proceso se repite iterativamente hasta que la red aprende a realizar la tarea asignada (LeCun et al., 2015).

En este proyecto, se utilizará una red densa de tan solo 5 capas que se describirá en el apartado 5.5.1. Idealmente se utilizarían más capas para alcanzar los mejores resultados según lo expuesto. Sin embargo, el costo computacional sería demasiado alto como para poder realizar las pruebas oportunas.

4.2. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son un tipo de red neuronal diseñada para procesar datos estructurados en forma de múltiples arreglos, como imágenes, audio o video. Estas redes son especialmente eficaces en tareas de visión por computadora, reconocimiento de voz y procesamiento de lenguaje natural, gracias a su capacidad para capturar patrones locales y jerárquicos en los datos. Las CNN se basan en cuatro principios clave: conexiones locales, pesos compartidos, capas de pooling y el uso de múltiples capas para extraer características cada vez más abstractas (LeCun et al., 2015).

La arquitectura típica de una CNN consta de varias etapas. Las primeras etapas incluyen capas convolucionales y capas de pooling. En una capa convolucional, las unidades están organizadas en mapas de características, donde cada unidad se conecta a regiones locales de la capa anterior mediante filtros. Estos filtros detectan patrones locales, como bordes o texturas, y aplican una función no lineal, como la ReLU (Unidad Lineal Rectificada). La idea de compartir pesos en los filtros permite que la red detecte los mismos patrones en diferentes ubicaciones, lo que es especialmente útil en imágenes, donde los motivos pueden aparecer en cualquier parte (LeCun et al., 2015).

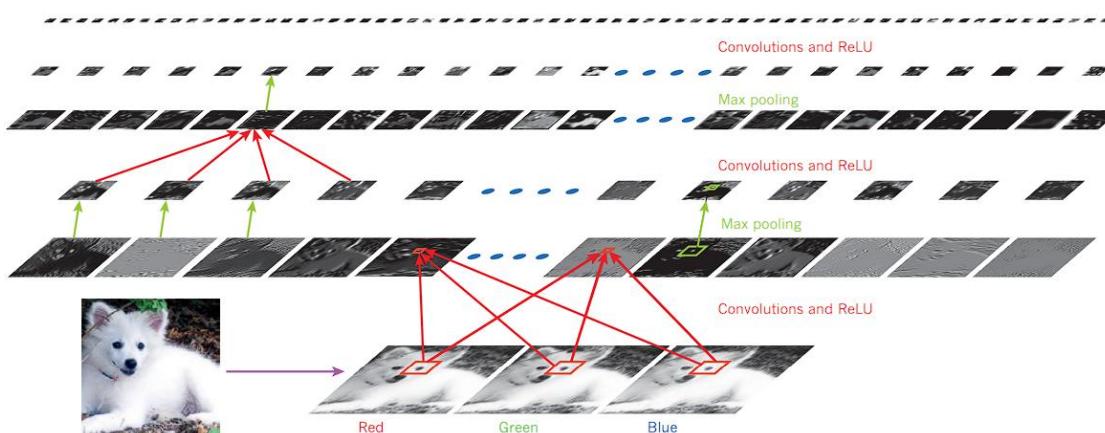


Figura 3. Las salidas (no los filtros) de cada capa (en horizontal) de una arquitectura típica de red convolucional aplicada a la imagen de un perro Samoyedo (abajo a la izquierda; y entradas RGB (rojo, verde, azul), abajo a la derecha). Cada imagen rectangular es un mapa de características. Fuente: (LeCun et al., 2015).

Las capas de pooling reducen la dimensionalidad de los mapas de características al resumir regiones locales, lo que introduce invariancia a pequeñas traslaciones o distorsiones en los datos. Estas capas, combinadas con las convolucionales, permiten a la CNN construir una jerarquía de características, desde patrones simples (como bordes) hasta conceptos más complejos (como objetos completos) (LeCun et al., 2015).

En el campo del procesamiento de imágenes, las convoluciones son una herramienta común para realizar operaciones como suavizar, resaltar detalles o detectar bordes en una imagen. Una imagen digital en escala de grises se representa como una sola matriz, mientras que una imagen a color se compone de tres matrices superpuestas,

correspondientes a los canales rojo, verde y azul. Cada valor en estas matrices, que generalmente oscila entre 0 y 255, indica la intensidad de un píxel y su color en la imagen (Choi et al., 2020).

En este proyecto, se utilizarán las ventajas expuestas desarrollando una red convolucional cuya arquitectura se detalla en el apartado 5.5.2. Además, dentro de dicho apartado en la subsección De Vector unidimensional a Imagen RGB, se explicará el procedimiento seguido para convertir vectores extraídos de las instancias del dataset a un array RGB para crear pseudo-imágenes y así poder entrenar a la CNN.

4.3. Métricas

La exactitud ha sido históricamente la métrica más empleada para evaluar el rendimiento de los modelos. No obstante, en casos donde los conjuntos de datos están desbalanceados, esta métrica pierde validez, ya que no tiene en cuenta la distribución de aciertos entre las distintas clases. Esto puede resultar en interpretaciones incorrectas; por ejemplo, un modelo con un 90% de exactitud en un dataset con un alto desbalanceo ($IR = 9$) no es realmente efectivo si predice todos los casos como pertenecientes a la clase mayoritaria (López et al., 2013).

La Curva ROC es una herramienta ampliamente utilizada para evaluar el rendimiento de los clasificadores, ya que permite visualizar la relación entre la tasa de verdaderos positivos (TPRate) y la tasa de falsos positivos (FPRate). Esta gráfica muestra que no es posible aumentar los aciertos sin incrementar también los errores. El Área Bajo la Curva ROC (AUC) mide la probabilidad de que el clasificador distinga correctamente entre dos estímulos (señal y ruido), ofreciendo una métrica única para comparar modelos. En el espacio ROC, los puntos $(0, 0)$ y $(1, 1)$ representan clasificadores triviales, mientras que $(0, 1)$ corresponde al clasificador perfecto. El AUC se calcula como el área bajo esta curva (López et al., 2013).

Estos métodos gráficos son especialmente útiles para evaluar el rendimiento en problemas de clasificación, ya que permiten analizar compensaciones entre diferentes aspectos de manera multidimensional, evitando depender de una única métrica escalar que puede ser sesgada. En contextos con datos desbalanceados, donde la clase positiva es de interés, se recomienda el uso de gráficos de precisión-sensibilidad (precision-recall). Además, las curvas de costo y ROI son herramientas valiosas para analizar el costo o beneficio asociado a cada modelo (López et al., 2013).

Por lo tanto, las métricas que se van a usar para evaluar los modelos desarrollados en este proyecto se definen con más precisión a continuación.

Recall (Sensibilidad)

Mide la proporción de verdaderos positivos identificados correctamente respecto al total de positivos reales.

$$Recall = \frac{TP}{TP + FN} \quad Ecuación 1$$

Donde TP (True Positives) y FN (False Negatives).

Accuracy (Exactitud)

Mide la proporción de predicciones correctas (tanto positivas como negativas) respecto al total de predicciones.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad Ecuación 2$$

Donde TN (True Negatives) y FP (False Positives).

Precision (Precisión)

Mide la proporción de verdaderos positivos respecto al total de predicciones positivas.

$$Precision = \frac{TP}{TP + FP} \quad Ecuación 3$$

ROC-AUC (Receiver Operating Characteristic - Area Under Curve)

Mide la capacidad del modelo para distinguir entre clases. El ROC-AUC se calcula a partir de la curva ROC, que muestra la tasa de verdaderos positivos (TPR, recall o sensibilidad) frente a la tasa de falsos positivos (FPR o precision) para diferentes umbrales de decisión.

El AUC (área bajo la curva ROC) se calcula como la integral bajo la curva ROC y varía entre 0 y 1, donde 1 indica un clasificador perfecto. Este valor equivale a la probabilidad de que un clasificador asigne una puntuación más alta a una muestra positiva que a una negativa.

$$AUC = \sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot \frac{y_i + y_{i+1}}{2} \quad Ecuación 4$$

Donde (x_i, y_i) son los puntos en la curva ROC.

4.4. Herramientas de Software

Python

Python es un lenguaje de programación versátil y de fácil uso, que cuenta con una gran variedad de librerías, módulos y frameworks completos para el desarrollo de aplicaciones. Esto lo convierte en una herramienta ideal para aplicarse en diversos campos, especialmente en ciencia de datos y aprendizaje automático. Librerías como Pandas y NumPy ofrecen soporte para matrices multidimensionales, incluyen funciones matemáticas avanzadas y facilitan el manejo y modificación de estructuras de datos, lo que las hace especialmente útiles para el análisis y procesamiento de datos en el ámbito de la ciencia de datos (*Welcome to Python.Org*, 2025).

Matplotlib

Matplotlib es una librería de Python para crear visualizaciones estáticas, animadas e interactivas, ideal para gráficos 2D y 3D. Ofrece un control detallado sobre cada elemento del gráfico, permitiendo personalizar títulos, ejes, leyendas y más, lo que la hace muy versátil para representar datos científicos (*Matplotlib — Visualization with Python*, s. f.).

Seaborn

Seaborn es una librería basada en Matplotlib que simplifica la creación de gráficos estadísticos atractivos y informativos. Incluye funciones integradas para visualizar distribuciones, matrices de correlación y relaciones entre variables, con estilos y temas predefinidos que mejoran la presentación de los datos (Waskom, 2021).

Scikit-learn

Scikit-learn es una librería esencial para machine learning en Python, que ofrece algoritmos para clasificación, regresión, clustering y reducción de dimensionalidad. Además, incluye herramientas para preprocessamiento de datos, selección de modelos y evaluación, siendo una de las más utilizadas en ciencia de datos (*scikit-learn: machine learning in Python — scikit-learn 1.6.1 documentation*, s. f.).

Pandas

Pandas es una librería diseñada para la manipulación y análisis de datos, especialmente útil para trabajar con datos estructurados en forma de DataFrames y Series. Proporciona funciones para limpieza, transformación, filtrado y agregación de datos, facilitando el manejo de grandes volúmenes de información (*pandas - Python Data Analysis Library*, s. f.).

Keras

Keras es una API de alto nivel para construir y entrenar redes neuronales, compatible con TensorFlow. Simplifica la creación de modelos de deep learning, ofreciendo soporte para redes convolucionales, recurrentes y capas personalizables, lo que la hace ideal

para tareas complejas como visión por computadora y procesamiento de lenguaje natural (*Keras: Deep Learning for humans*, s. f.).

Imblearn

Imblearn es una extensión de Scikit-learn enfocada en manejar datasets desbalanceados. Incluye técnicas como sobremuestreo (SMOTE), submuestreo y combinaciones de ambas, permitiendo mejorar el rendimiento de los modelos en problemas donde una clase es significativamente más frecuente que otra (*imbalanced-learn documentation — Version 0.13.0*, s. f.).

CTGAN

CTGAN es una librería basada en redes generativas adversarias (GANs) para generar datos sintéticos que imitan la distribución de datos reales. Es especialmente útil para crear datasets balanceados o aumentar conjuntos de datos existentes, manteniendo las relaciones estadísticas entre las variables (*sdv-dev/CTGAN*, 2019/2025).

Visual Studio Code

Visual Studio Code es un editor de código ligero y potente con soporte para múltiples lenguajes, incluyendo Python. Ofrece integración con Git, herramientas de depuración, extensiones personalizables y soporte para entornos virtuales, lo que lo convierte en una opción popular para el desarrollo de software (*Visual Studio Code - Code Editing. Redefined*, s. f.).

GitHub

GitHub es una plataforma de alojamiento de código basada en Git, utilizada para control de versiones y colaboración en proyectos de software. Facilita la gestión de repositorios, issues, pull requests y automatización de flujos de trabajo con GitHub Actions, siendo una herramienta clave para desarrolladores y equipos (*GitHub · Build and Ship Software on a Single, Collaborative Platform*, 2025).

5. Desarrollo del proyecto y resultados

Habla sobre la estructura de este apartado. No todos los apartados serán necesarios, dependerá de cada trabajo. Tampoco es necesario que los títulos sean exactamente los que aparecen abajo.

5.1. Metodología

El desarrollo de este proyecto sigue una planificación estructurada para garantizar una ejecución eficiente y ordenada de cada fase. El cronograma contempla desde la identificación y selección del dataset hasta la elaboración de la memoria y la presentación final. Inicialmente, se realiza un análisis exploratorio de datos (EDA) para comprender las características del conjunto de datos y su idoneidad para el problema de detección de fraude. En esta fase, se llevan a cabo las transformaciones necesarias del dataset para que, posteriormente, se lleva a cabo el desarrollo, entrenamiento, pruebas y análisis de redes neuronales profundas para la clasificación de transacciones fraudulentas. Una vez obtenidos los resultados, se procede a su evaluación general, lo que permitirá extraer conclusiones relevantes. Finalmente, se dedica tiempo a la redacción de la memoria y a la preparación de la presentación, con el fin de comunicar de manera clara y estructurada los hallazgos y aportes del trabajo. A continuación, se presentan las tareas generales descritas, así como un diagrama de Gantt en la Figura 4.

- Búsqueda y selección del dataset
- Análisis Exploratorio de Datos (EDA) & Transformaciones
- Separación de conjuntos y preparación de datos para entrenamiento de DNNs & CNNs.
- Desarrollo, Evaluación y Análisis de resultados de las Redes Neuronales
- Análisis de Resultados Generales
- Desarrollo de la memoria
- Desarrollo de la presentación

Planificación del Proyecto (01/01/2025 - 03/03/2025)

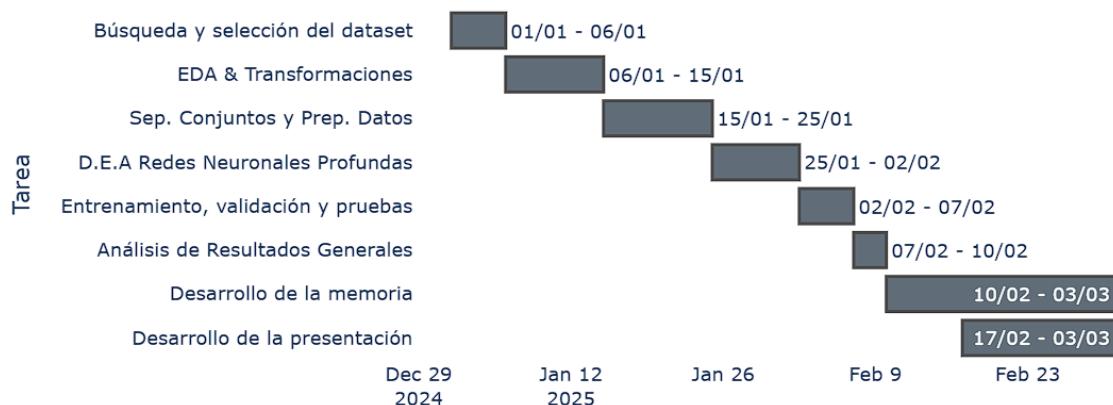


Figura 4. Cronograma de las tareas definidas. Elaboración propia.

Para entender mejor el desarrollo del proyecto, en la Figura 5 se muestra el diagrama de las distintas fases del desarrollo:

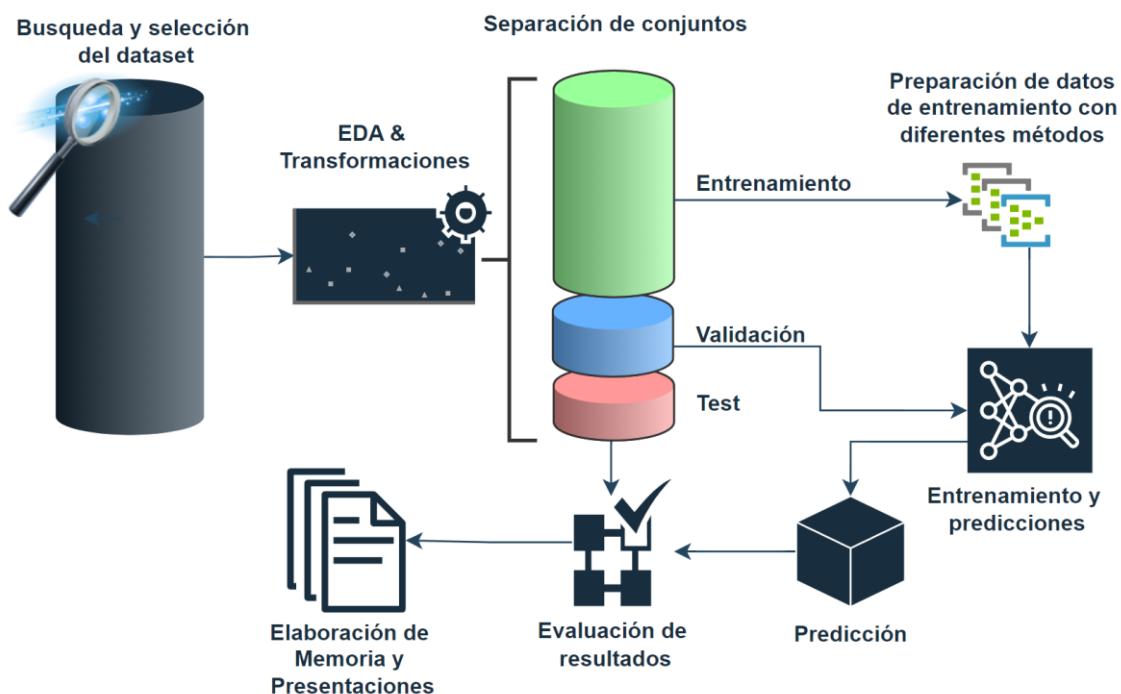


Figura 5. Diagrama de las fases del desarrollo. Elaboración propia.

5.2. Planteamiento del problema: detección de fraudes bancarios

El fraude bancario es un problema crítico que afecta a instituciones financieras, empresas y usuarios a nivel mundial, generando pérdidas económicas significativas y erosionando la confianza en los sistemas de pago y transacciones electrónicas. Con el crecimiento exponencial de las transacciones digitales, los métodos tradicionales de detección de fraude, basados en reglas predefinidas y análisis manuales, se han vuelto insuficientes para abordar la complejidad y el volumen de los datos actuales. Además, el carácter desbalanceado de los datos, donde las transacciones fraudulentas representan una fracción mínima del total, añade una capa adicional de dificultad al desarrollo de soluciones efectivas.

En este contexto, surge la necesidad de implementar técnicas avanzadas de ciencia de datos y *machine learning*, específicamente mediante el uso de *deep learning*, para identificar patrones complejos y sutiles que puedan indicar actividades fraudulentas. Sin embargo, el desarrollo de modelos de detección de fraude enfrenta varios desafíos:

- Desbalance de clases: Las transacciones fraudulentas son extremadamente raras en comparación con las legítimas, lo que dificulta el entrenamiento de modelos precisos y robustos.
- Evolución de las tácticas fraudulentas: Los delincuentes adaptan constantemente sus métodos, lo que requiere que los modelos sean capaces de generalizar y detectar nuevas formas de fraude.
- Falsos positivos: Un exceso de transacciones legítimas identificadas como fraudulentas puede generar costos operativos y afectar la experiencia del usuario.

Este proyecto se enfoca en abordar estos desafíos mediante la aplicación de técnicas de *deep learning*, como redes neuronales profundas (DNN) y redes neuronales convolucionales (CNN), para desarrollar un sistema de detección de fraudes que sea preciso, robusto y capaz de adaptarse a las dinámicas cambiantes del fraude bancario. Además, se busca contribuir al avance de la ciencia de datos en este campo, demostrando cómo el *deep learning* puede ser una herramienta efectiva para la prevención de actividades delictivas en el sector financiero, beneficiando no solo a las instituciones bancarias, sino también a la sociedad en general.

5.3. Identificación y selección del dataset

En la búsqueda de un dataset adecuado para la detección de fraudes bancarios, es esencial contar con datos que no solo sean representativos del problema, sino que también ofrezcan un reto significativo para los modelos de aprendizaje automático. En este caso, se busca un conjunto de datos de alta dimensionalidad que permita explorar patrones complejos y no triviales, con el fin de mejorar las capacidades predictivas y reducir los falsos positivos, un desafío clave en la detección de fraudes.

El dataset debe incluir múltiples características de las transacciones o cuentas, como información temporal, montos, ubicaciones, frecuencia de transacciones y otros atributos que puedan reflejar el comportamiento habitual de los usuarios y permitir identificar patrones inusuales. La presencia de atributos protegidos como edad, ingresos o historial crediticio es también importante, ya que estos elementos deben ser considerados para evaluar el sesgo y la justicia de los modelos entrenados.

Además, es crucial que el dataset contemple distribuciones dinámicas de datos, reflejando cómo los patrones de fraude pueden cambiar con el tiempo, así como posibles desbalances de clases entre transacciones fraudulentas y no fraudulentas. La existencia de ruido en los datos y la presencia de anomalías también contribuirán a hacer del trabajo un reto significativo, ya que los modelos deberán ser robustos y capaces de identificar los fraudes en condiciones complejas y cambiantes.

El *BAF Dataset Suite* (Jesus et al., s. f.) es un conjunto de seis datasets diseñados para abordar la evaluación de rendimiento y equidad de modelos de aprendizaje automático (ML) en entornos dinámicos y con datos tabulares. Estos conjuntos de datos fueron generados a partir de un dataset real de detección de fraude en la apertura de cuentas bancarias en línea, lo cual tiene una gran relevancia en el contexto de Fair ML, ya que las decisiones tomadas por los modelos de ML pueden tener un impacto significativo en la vida de los individuos, afectando, por ejemplo, el acceso a servicios financieros.

Las características principales de los datasets en el BAF Suite incluyen:

- Datos sesgados y controlados a lo largo de múltiples períodos de tiempo, lo que permite evaluar el desempeño de los modelos en entornos dinámicos, donde existen cambios en la distribución temporal de los datos, desbalance de clases y otros fenómenos típicos de entornos reales.
- Generación con redes generativas adversariales (GAN) para preservar la privacidad de los solicitantes, lo que es especialmente importante en el contexto actual de preocupaciones sociales y legislativas sobre la protección de datos personales.
- Cada conjunto de datos contiene un millón de instancias de solicitudes y treinta características diferentes, que representan propiedades observadas de las solicitudes, como el estado de empleo, ingresos personales y la validez de información proporcionada.

- Se incluyen atributos protegidos como edad, ingreso personal y estado de empleo, y se incorpora ruido en los datos para garantizar cierto nivel de privacidad diferencial.
- El BAF Dataset Suite tiene como objetivo superar varias limitaciones comunes de datasets existentes en Fair ML, tales como la falta de representación de fenómenos como el sesgo temporal, la distribución dinámica de datos y los desequilibrios de clases. Este conjunto de datos es útil para evaluar la justicia en los modelos de ML, especialmente en dominios de alto impacto, como la justicia criminal, contratación y servicios financieros, donde las decisiones algorítmicas pueden afectar significativamente a grupos históricamente desfavorecidos.

Aunque los datasets del BAF Suite tienen algunas limitaciones, como la etiquetación selectiva (solo se conoce la etiqueta verdadera de los solicitantes aceptados), se considera un avance importante en la creación de herramientas que puedan ser utilizadas para probar modelos justos en entornos complejos y en constante cambio.

5.3.1. Descripción del dataset

De entre los seis dataset que componen en *BAF*, solo se ha escogido uno (el primero, llamado *Base.csv*) debido a la ingente cantidad de datos que cada uno de ellos contiene.

Este dataset contiene 1.000.000 de registros y 32 variables, con una combinación de características numéricas y categóricas orientadas a la detección de fraudes en transacciones bancarias. La variable objetivo es *fraud_bool*, una variable binaria que indica si una transacción es fraudulenta (1) o no (0).

Las variables incluyen información sobre datos personales y financieros del cliente, como ingresos (*income*), edad (*customer_age*), historial bancario (*bank_months_count*, *credit_risk_score*) y estado laboral (*employment_status*). También hay características relacionadas con el comportamiento del usuario, como la longitud de la sesión (*session_length_in_minutes*), la cantidad de direcciones de correo asociadas a un dispositivo (*device_distinct_emails_8w*) y la frecuencia de cambios de domicilio (*prev_address_months_count*, *current_address_months_count*). Además, el dataset incorpora variables sobre patrones de transacción y seguridad, como la velocidad de operaciones en distintos períodos (*velocity_6h*, *velocity_24h*, *velocity_4w*), la validez de los números telefónicos (*phone_home_valid*, *phone_mobile_valid*) y el número de fraudes detectados en un mismo dispositivo (*device_fraud_count*).

Según Jesus et al. (s. f.), cada instancia es una solicitud de cuenta bancaria sintética con ingeniería de características. A continuación, se describen cada uno de los campos de acuerdo a la publicación mencionada:

- **income (numérico):** Ingreso anual del solicitante (en forma de decil). Rango entre [0, 1, 0,9].

- **name_email_similarity (numérico):** Métrica de similitud entre el correo electrónico y el nombre del solicitante. Valores más altos indican mayor similitud. Rango entre [0, 1].
- **prev_address_months_count (numérico):** Número de meses en la dirección registrada anterior del solicitante, es decir, su residencia previa, si aplica. Rango entre [-1, 380] meses (-1 indica un valor faltante).
- **current_address_months_count (numérico):** Número de meses en la dirección actual registrada del solicitante. Rango entre [-1, 429] meses (-1 indica un valor faltante).
- **customer_age (numérico):** Edad del solicitante en años, redondeada a la década. Rango entre [10, 90] años.
- **days_since_request (numérico):** Número de días transcurridos desde que se realizó la solicitud. Rango entre [0, 79] días.
- **intended_balcon_amount (numérico):** Monto inicial transferido para la solicitud. Rango entre [-16, 114] (valores negativos indican valores faltantes).
- **payment_type (categórico):** Tipo de plan de pago con crédito. 5 valores posibles (anonimizados).
- **zip_count_4w (numérico):** Número de solicitudes dentro del mismo código postal en las últimas 4 semanas. Rango entre [1, 6830].
- **velocity_6h (numérico):** Velocidad total de solicitudes realizadas en las últimas 6 horas, es decir, el número promedio de solicitudes por hora en ese período. Rango entre [-175, 16818].
- **velocity_24h (numérico):** Velocidad total de solicitudes realizadas en las últimas 24 horas, es decir, el número promedio de solicitudes por hora en ese período. Rango entre [1297, 9586].
- **velocity_4w (numérico):** Velocidad total de solicitudes realizadas en las últimas 4 semanas, es decir, el número promedio de solicitudes por hora en ese período. Rango entre [2825, 7020].
- **bank_branch_count_8w (numérico):** Número total de solicitudes en la sucursal bancaria seleccionada en las últimas 8 semanas. Rango entre [0, 2404].
- **date_of_birth_distinct_emails_4w (numérico):** Número de correos electrónicos asociados a solicitantes con la misma fecha de nacimiento en las últimas 4 semanas. Rango entre [0, 39].
- **employment_status (categórico):** Estado laboral del solicitante. 7 valores posibles (anonimizados).
- **credit_risk_score (numérico):** Puntuación interna de riesgo de la solicitud. Rango entre [-191, 389].
- **email_is_free (binario):** Tipo de dominio del correo electrónico de la solicitud (ya sea gratuito o de pago).
- **housing_status (categórico):** Estado de residencia actual del solicitante. 7 valores posibles (anonimizados).
- **phone_home_valid (binario):** Validez del número de teléfono fijo proporcionado.

- **phone_mobile_valid (binario):** Validez del número de teléfono móvil proporcionado.
- **bank_months_count (numérico):** Antigüedad en meses de la cuenta bancaria previa (si se posee). Rango entre [-1, 32] meses (-1 indica un valor faltante).
- **has_other_cards (binario):** Indica si el solicitante posee otras tarjetas de la misma entidad bancaria.
- **proposed_credit_limit (numérico):** Límite de crédito propuesto por el solicitante. Rango entre [200, 2000].
- **foreign_request (binario):** Indica si el país de origen de la solicitud es diferente del país del banco.
- **source (categórico):** Fuente en línea de la solicitud. Puede ser navegador web (INTERNET) o aplicación móvil (TELEAPP).
- **session_length_in_minutes (numérico):** Duración de la sesión del usuario en la web bancaria en minutos. Rango entre [-1, 107] minutos (-1 indica un valor faltante).
- **device_os (categórico):** Sistema operativo del dispositivo que realizó la solicitud. Valores posibles: Windows, macOS, Linux, X11 u otro.
- **keep_alive_session (binario):** Opción del usuario sobre el cierre de sesión.
- **device_distinct_emails (numérico):** Número de correos electrónicos distintos utilizados en la web bancaria desde el mismo dispositivo en las últimas 8 semanas. Rango entre [-1, 2] (-1 indica un valor faltante).
- **device_fraud_count (numérico):** Número de solicitudes fraudulentas realizadas desde el mismo dispositivo. Rango entre [0, 1].
- **month (numérico):** Mes en el que se realizó la solicitud. Rango entre [0, 7].
- **fraud_bool (binario):** Indica si la solicitud es fraudulenta o no.

(Jesus et al., s. f.) [Traducción propia].

5.4. Análisis Exploratorio de Datos (EDA, por sus siglas en inglés)

Inicialización del código: importaciones de librerías y definición de variables generales

En esta sección, se importan las librerías necesarias para el manejo de datos, visualización y ML, como Pandas, NumPy, Matplotlib, Seaborn y Scikit-learn. Se fija una semilla para asegurar la reproducibilidad de los procesos aleatorios y se configuran estilos y colores para las visualizaciones. Se define una ruta base para organizar la lectura y guardado de archivos, y se asigna una versión al modelo o archivo en desarrollo, facilitando el control y seguimiento del proyecto. Esto prepara el entorno para el análisis de datos, la construcción de modelos y su evaluación.

Carga e inspección visual del dataset

Se tiene un *dataset* de 1 millón de instancias. Cada instancia (fila) del conjunto de datos representa una solicitud individual realizada vía online. La etiqueta de cada instancia se encuentra en la columna "*fraud_bool*". Una instancia positiva representa un intento fraudulento, mientras que una instancia negativa representa una solicitud legítima (Jesus et al., s. f.).

Además, se tienen 32 características distintas, de las cuales:

- 4 son categóricas
- 7 son binarias
- 21 son numéricas

Debido a que la detección de fraudes bancarios se trata de un problema desbalanceado, se genera una primera observación del ratio de desbalanceo. Para ello, se visualiza el porcentaje de datos de la clase 0 (transacción legítima) y de la clase 1 (transacción fraudulenta) de la característica "*fraud_bool*".

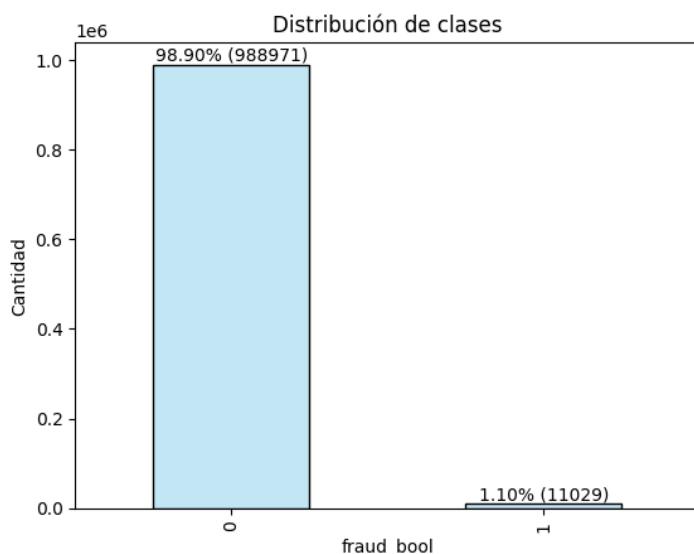


Figura 6. Desbalanceo de clases entre transacciones legítimas (0) y fraudulentas (1). Elab. Propia.

Claramente, se trata de un problema extremadamente desbalanceado, por lo que probablemente aplicar técnicas de balanceo de clases para poder obtener buenas métricas al aplicar los modelos de *deep learning* sea beneficioso.

Tratamiento de valores faltantes

Una vez descritas todas las variables intervencientes, se observa que en la información general del *dataset* no se diferencian por el tipo de datos los numéricos de los binarios. Sin embargo, es importante diferenciarlos ya que se va a aplicar un tratamiento distinto a cada tipo.

Para ello, se crea un diccionario con cada variable como clave y el tipo como valor de acuerdo con la descripción. A continuación, se crea una función que realizará una transformación a los datos faltantes (negativos) en función del tipo de variable.

La estrategia que se lleva a cabo es la siguiente:

- Variables numéricas: se transforman todos los valores negativos en ceros. Esto es importante ya que, en futuros pasos, se normalizarán los datos. Al normalizar, si una variable con rangos, por ejemplo, entre 400 y 1600, tiene un valor negativo como -200, este último modificará drásticamente el comportamiento de la normalización de la variable. Sin embargo, tampoco es conveniente que los valores "faltantes" se conviertan en valores habituales (haciendo transformaciones habituales como la media o la mediana), ya que la ausencia de datos puede ser una pista crucial para identificar transacciones fraudulentas. Por lo tanto, se transforman a 0 y esto marcaría siempre el valor más bajo (en caso de que haya valores faltantes para esa variable) en la normalización.
- Variables binarias o categóricas: se transforman todos los valores negativos a la clase menos representada. Por el mismo motivo que expuesto anteriormente, no conviene convertirlos en la clase más representada para no perder pistas importantes.

Visualización de Estimaciones de Densidad por Núcleo (KDE) en variables numéricas diferenciada por clase

Comparar las distribuciones de densidad KDE para cada característica en un dataset, diferenciando entre clases (0 y 1), es fundamental para evaluar su capacidad discriminativa en un problema de clasificación. Si una variable tiene distribuciones bien separadas entre ambas clases, significa que es un buen predictor y puede ayudar a mejorar el modelo. En cambio, si las distribuciones son casi idénticas, la variable podría no aportar información relevante o, en caso de que lo sea, necesitar otras herramientas más sofisticadas y complejas para desgranar los patrones de dicha variable en conjunto con las demás. Este tipo de herramientas se utiliza exclusivamente para las características numéricas al basarse en distribuciones continuas de probabilidad.

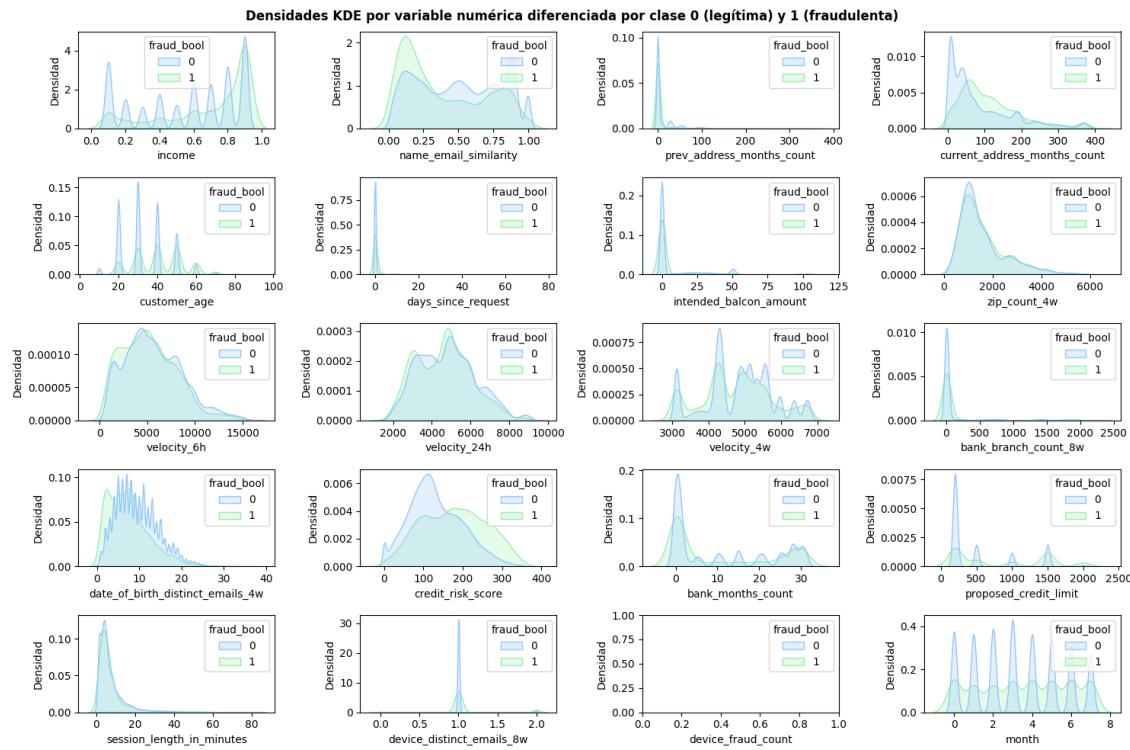


Figura 7. Análisis KDE de las características numéricas. Elaboración propia.

En la Figura 7 se muestran las distribuciones de diferentes variables numéricas en función de la clase de fraude ($\text{fraud_bool} = 0$, $\text{fraud_bool} = 1$). A partir de ellas, se pueden destacar las siguientes observaciones clave:

- Solapamiento entre clases: En la mayoría de las variables, las distribuciones de transacciones legítimas (azul) y fraudulentas (verde) se superponen considerablemente, lo que indica que no hay un único umbral claro para separar las clases.
- Algunas variables como "*name_email_similarity*" y "*credit_risk_score*" muestran diferencias en la distribución entre clases, sugiriendo que podrían aportar información valiosa para la clasificación.
- Otras variables, como "*session_length_in_minutes*" o "*customer_age*", parecen tener una distribución muy similar entre ambas clases, lo que sugiere que podrían tener menor relevancia predictiva.
- Distribuciones multimodales: Algunas variables muestran múltiples picos en su distribución (por ejemplo, "*velocity_6h*", "*velocity_24h*"), lo que sugiere que ciertos patrones de comportamiento podrían ser útiles para identificar fraude.
- Valores extremos y colas largas: Variables como "*proposed_credit_limit*" y "*zip_count_4w*" presentan valores extremos que pueden influir en la clasificación. Modelos más avanzados como redes neuronales pueden manejar mejor estos casos que modelos lineales.

- La variable "*device_fraud_count*" no presenta distribución alguna. Esto es debido a que todas las instancias de la variable contienen el mismo valor y, por lo tanto, se puede eliminar del dataset debido a que no aporta valor.

Visualización de la Matriz de Correlaciones

La matriz de correlaciones es una herramienta clave para analizar las relaciones entre las variables numéricas de un dataset. Su principal utilidad radica en detectar qué variables están fuertemente relacionadas entre sí, lo que permite reducir redundancias y evitar problemas como la **multicolinealidad**, que puede afectar el rendimiento de ciertos modelos de ML. En particular, en la detección de fraude, examinar la correlación con la variable objetivo (*fraud_bool*) ayuda a identificar qué características tienen mayor impacto en la clasificación de transacciones fraudulentas.

Las variables categóricas, al estar representadas como texto o etiquetas, no pueden ser evaluadas directamente con coeficientes de correlación como **Pearson** (usado en la matriz de correlaciones), por lo que es necesario transformarlas en una representación numérica adecuada. El método **OneHotEncoding** convierte cada categoría en una serie de variables binarias (0 o 1), lo que permite calcular su correlación con otras características del dataset. De esta manera, se pueden analizar posibles relaciones entre categorías y la variable objetivo (*fraud_bool*), así como identificar redundancias entre las categorías. Sin esta transformación, las variables categóricas quedarían excluidas del análisis de correlación, limitando la comprensión de su impacto en el modelo.

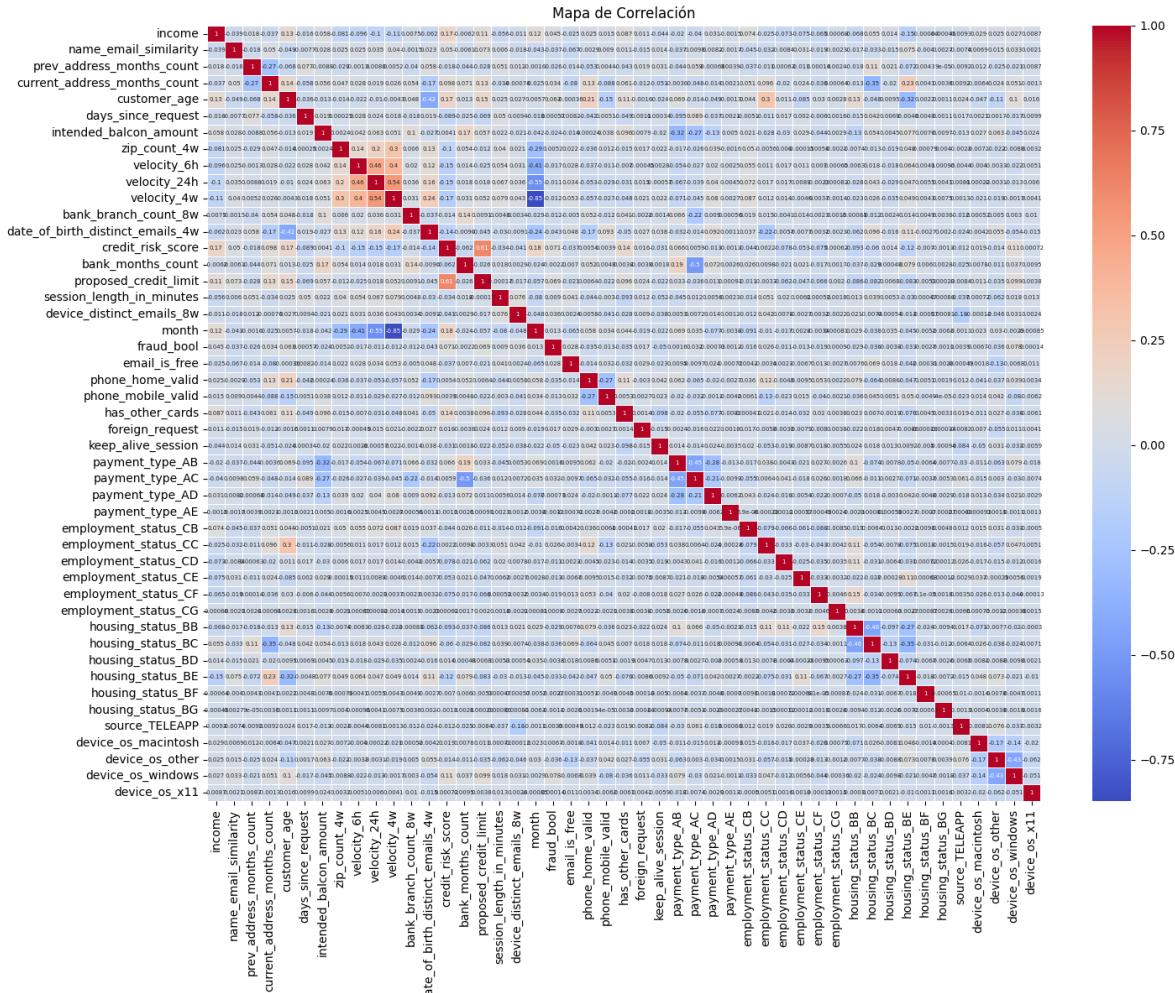


Figura 8. Matriz de Correlaciones. Elaboración propia.

Como se puede observar en la Figura 8, no existen variables altamente correlacionadas (coeficiente de Pearson mayor > 0,9). No obstante, existe una correlación moderadamente alta entre la variable *month* y *velocity_4w*. Sin embargo, debido a que la columna *month* se usará para dividir el dataset en próximos pasos, será eliminada del entrenamiento y esta correlación quedará mitigada.

Por otra parte, no existe ninguna variable que esté altamente correlacionada con *fraud_bool*, por lo que se puede inferir que las características no lineales o interacciones complejas entre variables podrían ser más relevantes para la clasificación del fraude, y el modelo probablemente necesitará explorar relaciones más complejas para identificar patrones en los datos.

División entrenamiento y validación

De acuerdo con Jesus et al., (s. f., Question 11), se recomienda usar la columna *month* para dividir los datos de entrenamiento, validación y test. Además, en dicha publicación se explica que cada instancia es independiente de cualquier otra sin que haya una secuencialidad que condicione los resultados en función del orden de las filas en la tabla.

Por lo tanto, se dividirán los datos de la siguiente manera:

- Entrenamiento: instancias anteriores al mes 6.
- Validación y Test: instancias posteriores al mes 6 inclusive.
 - Validación: 50% de los datos reservados para validación y test
 - Test: 50% de los datos reservados para validación y test

Submuestreo

Hay dos motivos por los que realizar un submuestreo de las instancias:

- Límites de rendimiento computacional: actualmente se tiene 1 millón de instancias. Entrenar modelos de aprendizaje profundo con esta cantidad de datos requiere un costo computacional muy elevado.
- Reducir el desbalanceo: aunque se aplicarán otras técnicas más avanzadas de balanceo de datos, el submuestreo puede ser una herramienta fundamental para garantizar que los modelos no se sesguen hacia la clase predominante.

Por contra, esto puede afectar a los resultados al perder variabilidad de los datos originales. Además, se deben de considerar dos aspectos muy importantes de esta operación:

- Con el fin de balancear, el submuestreo debe ejecutarse exclusivamente sobre la clase 0 (transacciones legítimas), que es la clase mayoritaria.
- El submuestreo exclusivo de la clase 0, no debe aplicarse sobre los datos de validación y test. Estos grupos deben reducirse equitativamente en ambas clases. De lo contrario, generaría una distribución desequilibrada en la producción, lo que llevaría a obtener resultados poco realistas al evaluar los modelos en esos conjuntos.
- Si se submuestra los datos de entrenamiento con un ratio de reducción, se deben submuestrear con el mismo ratio los datos de validación y test (teniendo en cuenta el punto anterior). Si esto no se cumple, se podría obtener un gran desequilibrio entre los datos de entrenamiento y los datos de validación y test (los datos de entrenamiento deben representar una mayor parte, habitualmente, entre el 70% y el 80%).

Consecuentemente, se ha aplicado un proceso de submuestreo para reducir la cantidad de muestras en cada clase, permitiendo ajustar si la reducción afecta solo a una clase o a ambas. Este procedimiento se basa en un criterio predefinido y se ha considerado un factor de 0,7 (reducción del 70% de los datos) que determina el porcentaje de instancias a eliminar en cada conjunto de datos.

De esta manera, el conjunto de entrenamiento se ajusta para reducir el desequilibrio entre clases, mientras que en los conjuntos de validación y test se mantiene una distribución más representativa de la realidad.

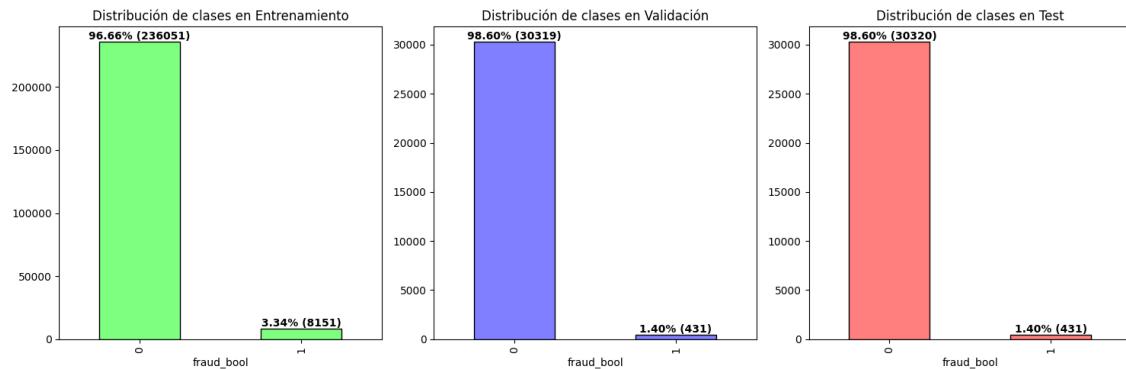


Figura 9. Distribución de clases en los diferentes conjuntos tras submuestreo. Elaboración propia.

Como puede apreciarse en la Figura 9, a pesar de haberse reducido la cantidad total de instancias, en los conjuntos de validación y test se mantiene el desbalanceo inicial (representación de la realidad). En cambio, la distribución de clases del conjunto de entrenamiento ha variado reduciéndose, no solo el número total de instancias sino, además, el desbalanceo de clases (la clase 1 pasa de 1.40% a un 3.34%). Esto puede beneficiar al aprendizaje de los modelos ya que, aunque sigue habiendo una cantidad alta de muestras legítimas, la diferencia con respecto a las fraudulentas se ha reducido.

Téngase en cuenta que los modelos de aprendizaje, ya sean redes neuronales profundas o tradicionales de ML como Bosques Aleatorios, Máquinas de Soporte Vectorial, etc, tienden a sesgarse a la clase mayoritaria en el entrenamiento (aprenden más de una clase que de otra).

En la Figura 10 se muestra la distribución del conjunto de datos global entre los tres subconjuntos.

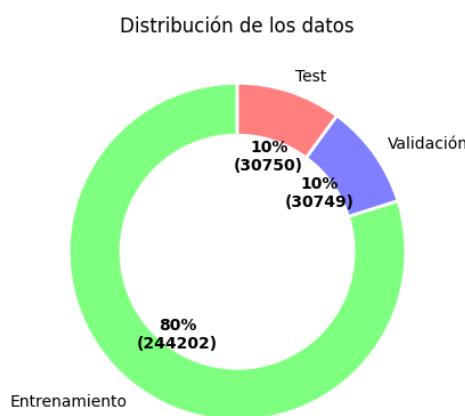


Figura 10. Distribución de los datos en los distintos conjuntos: Entrenamiento, validación y test. Elaboración propia.

Con el apoyo visual de la Figura 9 y de la Figura 10 se observa que:

- El 80% de los datos pertenecen al conjunto de entrenamiento con 244.202 instancias, de las cuales, el 3,34% (8.151) son fraudulentas y el 96,66% (236.051) legítimas.
- El 10% de los datos pertenecen al conjunto de validación con 30.750 instancias, de las cuales, el 1,40% (431) son fraudulentas y el 98,60% (30.319) legítimas.
- El 10% de los datos pertenecen al conjunto de validación con 30.750 instancias, de las cuales, el 1,40% (431) son fraudulentas y el 98,60% (30.320) legítimas.

En total, se ha reducido el dataset de 1 millón de instancias a 305.702, es decir, una reducción de aproximadamente un 70%.

Escalado de datos

Habitualmente se usa *MinMaxScaler* debido a que transforma los datos del dataset a un rango específico, generalmente [0, 1], lo que facilita el entrenamiento de redes neuronales. Este escalado permite que las características tengan magnitudes similares, evitando que aquellas con valores más grandes dominen el proceso de aprendizaje. Además, contribuye a una convergencia más rápida y estable durante el entrenamiento, ya que los algoritmos basados en gradientes funcionan mejor con datos normalizados. También preserva la distribución original de los datos, lo que es útil para mantener la interpretabilidad y evitar distorsiones en el modelo.

Adicionalmente, el escalado es necesario para que ciertos algoritmos de sobremuestreo, como SMOTE (que se usará más adelante), que se basan en cálculos de distancias, funcionen de manera eficaz, ya que garantiza que todas las características contribuyan equitativamente a la medición de distancias.

Es importante elegir el conjunto de datos sobre los que se entrenará el escalador, pues cualquier nueva transacción se escalarán en base a este. Por ello, se ha entrenado la instancia de clase de *MinMaxScaler* con el dataset previo a la división de entrenamiento, validación y test. De esta forma, se garantiza que los máximos y los mínimos cubren la mayor variabilidad posible.

Clustering

En este apartado, se busca la clusterización de las instancias con el fin de identificar patrones subyacentes en los datos, agrupar observaciones con características similares y explorar posibles segmentos dentro de las transacciones que ayuden a mejorar la detección de fraudes. Esto permitirá analizar si existen perfiles diferenciados de transacciones o comportamientos anómalos que puedan asociarse a actividades fraudulentas.

El algoritmo que se ha escogido para clusterizar es *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* por su capacidad para encontrar un número de clústeres sin necesidad de predefinirlo. Además, presenta la significante ventaja del

manejo de ruido y outliers. Cuando *DBSCAN* no es capaz de clusterizar a un individuo, lo etiqueta como -1.

Como cualquier algoritmo, este tiene sus limitaciones. Por ejemplo, si los datos presentan una distribución no globular o si la densidad de puntos es muy distinta en diferentes zonas del espacio hiperdimensional, *DBSCAN* tendrá dificultades para clusterizar correctamente. Además, es sensible a sus dos principales hiperparámetros: ϵ (épsilon) y *min_samples*.

Por ello, para paliar los efectos de escoger de forma aleatoria dichos hiperparámetros, se ha diseñado una función iterativa para, dada una serie de valores de ϵ y *min_samples*, se clustericen las instancias y se evalúe su resultado mediante la métrica *silhouette score*, o métrica de la *silueta*.

La métrica de la silueta evalúa la calidad de los clusters en algoritmos no supervisados, midiendo cuán similar es un objeto a su propio cluster en comparación con otros clusters. El valor de la silueta oscila entre -1 y 1, donde valores cercanos a 1 indican que el objeto está bien asignado a su cluster, valores cercanos a 0 sugieren que el objeto podría estar en el límite entre dos clusters, y valores negativos indican una posible asignación incorrecta. Se busca maximizar esta métrica para lograr agrupamientos más coherentes y separados.

Para la lista de valores posibles de ϵ , se ha de tener en cuenta que la mayor distancia posible de un espacio N dimensional donde todos los valores están escalados en el rango [0, 1], la mayor distancia entre dos puntos cualesquiera es \sqrt{N} . Tras el preprocesamiento llevado a cabo hasta el momento se tienen 45 características (tras el *OneHotEncoder* se aumentan), es decir, 45 dimensiones. Por lo tanto, la máxima distancia es de $\sqrt{45}$, esto es, 6,71.

De esta manera, tiene sentido definir la siguiente lista de valores: [0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5].

Para el hiperparámetro *min_samples* no existe una forma de saber cuál es un número óptimo. Por defecto es 5. Sin embargo, dada la abrumadora cantidad de datos en el dataset se definirá una lista que abarque un mayor rango de posibilidades: [3, 5, 10, 100, 1000, 50000, 100000].

Tras combinar cada posibilidad y evaluarla, se grafica en 3D obteniendo la Figura 11.

DBSCAN - Silhouette Score en función de eps y min_samples

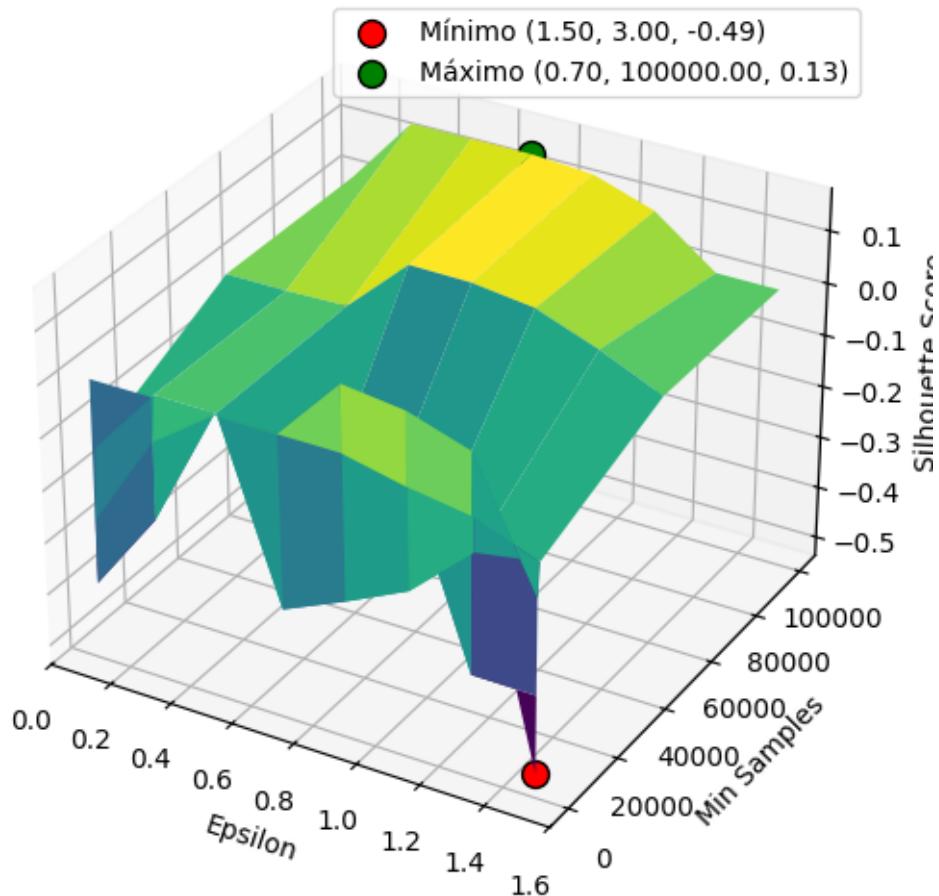


Figura 11. Silhouette Score en función de épsilon y min_samples. Elab. Propia

De la visualización se desprende que, según la métrica de la silueta, la combinación de valores sub-óptimas para (ϵ (épsilon), $min_samples$) es (0,7, 100.000). Sin embargo, el valor obtenido es muy bajo, lo que indica una clusterización difusa entre componentes del mismo cluster y poco distinguida con el resto.

Aunque se tengan 45 dimensiones, es posible hacer una representación visual sobre un plano 2D tanto de los clusters como de la distribución de las clases 0 y 1. Para pasar de 45 dimensiones a 2 dimensiones, se utilizarán dos técnicas de reducción de dimensionalidad, PCA y t-SNE.

Reducción de hiperdimensiones: Visualización en 2D mediante PCA y t-SNE

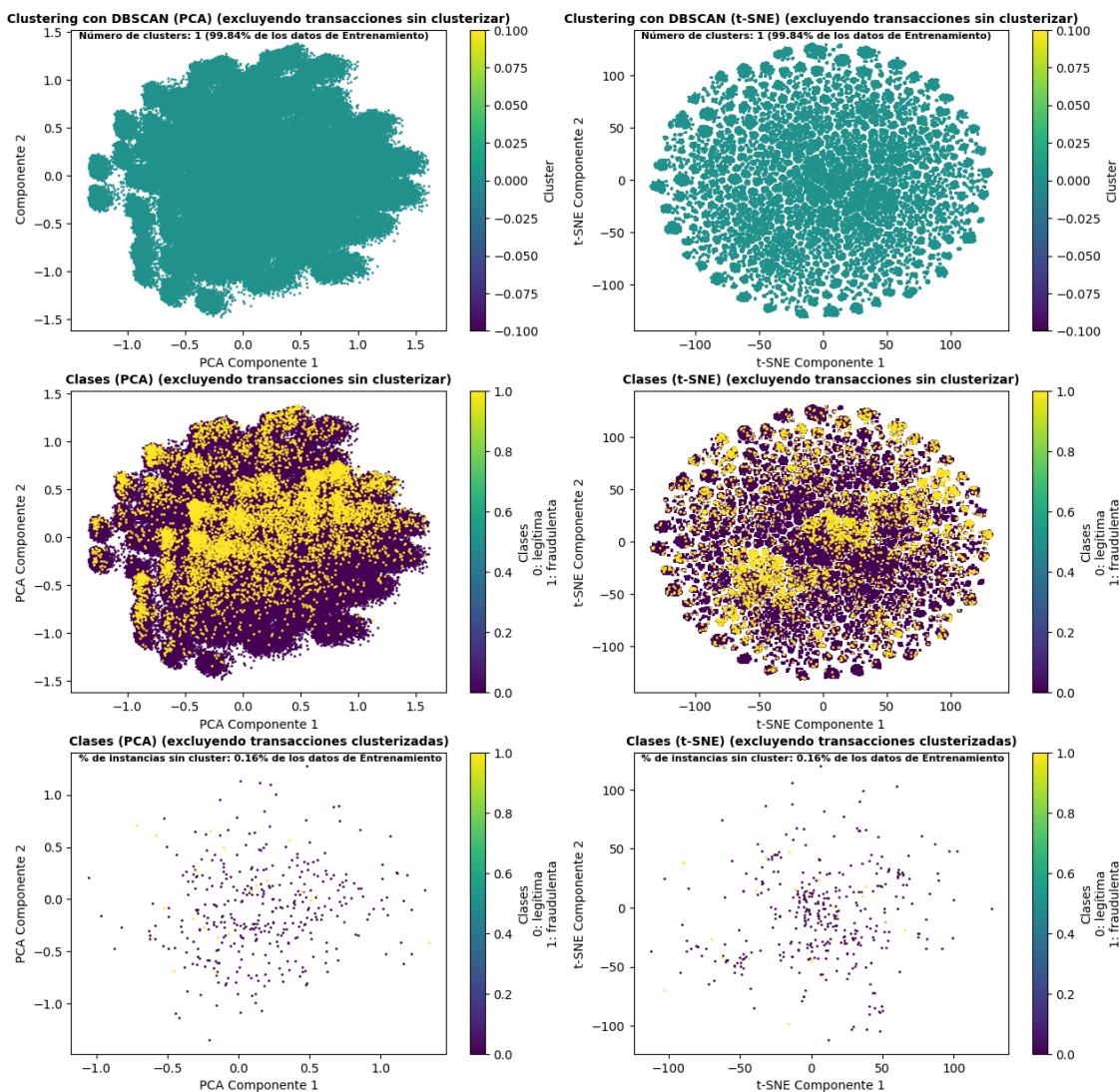


Figura 12. Reducción de hiperdimensiones mediante PCA y t-SNE. Clusters y Clases. Elab. Propia

En la Figura 12 se muestra la proyección en 2 dimensiones de los datos mediante PCA (columna izquierda) y t-SNE (columna derecha). A partir de esta figura, se pueden destacar los siguientes puntos:

- El número de clusters óptimo según el algoritmo DBSCAN, dentro de las pruebas realizadas (30 combinaciones diferentes de epsilon y min_samples) y evaluadas con la métrica de la Silueta, es 1 sola agrupación válida comprendiendo el 99,84% de los datos de entrenamiento y el resto pertenece al grupo de los no categorizados o ruido, que representan el 0,16% de los datos de entrenamiento.
- Aunque existen ciertas zonas con menos densidad de transacciones fraudulentas, en general, la dispersión y superposición de las clases en ambas

representaciones (PCA y t-SNE) indica que no hay una clara separación entre transacciones fraudulentas y no fraudulentas, lo que sugiere que las características utilizadas podrían no ser suficientes para diferenciar de manera efectiva ambos tipos de transacciones tras la reducción de dimensionalidad.

- La visualización mediante t-SNE muestra una distribución más esférica de los datos, mientras que PCA preserva una estructura más lineal. Sin embargo, en ambos casos, los fraudes no forman conglomerados diferenciados, sino que están dispersos entre las transacciones normales.
- La nula creación de variedad de clústeres y el bajo valor de la Silueta muestran que DBSCAN no ha sido útil para una clasificación efectiva mediante clusterización.

Considérese que la visualización 2D tras la reducción de dimensionalidad es una forma de visualizar hiperdimensiones, es decir, los patrones podrían existir pero no visualizarse correctamente. Sin embargo, en conjunto con los cálculos y evaluaciones (como la cantidad de clústeres y la métrica de la Silueta), se demuestra que, al menos, los datos no presentan agrupaciones globulares claras y uniformemente densas, ya que, si fuera así, DBSCAN habría desprendido mejores resultados.

Aunque podrían evaluarse múltiples formas de clusterización con otros algoritmos como *k-means* o agrupaciones no esféricas como *Modelos Mixtos Gaussianos*, queda demostrada la suficiente complejidad del problema como justificación del uso de redes neuronales para la detección de fraudes del dataset presentado.

5.5. Redes Neuronales Profundas

Descripción del experimento

Se van a diseñar dos redes neuronales de distinta naturaleza para resolver el mismo problema:

- Red Neuronal Profunda (DNN)
- Red Neuronal Convolutacional (CNN):

Una vez configurada la arquitectura de la DNN, se crearán distintos modelos compilándolos con distintos parámetros y entrenándolos con distintos datos de entrenamiento (sin aumentar y aumentados). Como el poder de computación es limitado, no se utilizará un algoritmo de optimización como *algoritmos genético* o *RandomSearchCV* y variantes para la búsqueda de los mejores parámetros debido a la gran cantidad de datos a entrenar (a pesar del submuestreo realizado en pasos anteriores), sino que se seguirá la siguiente estrategia:

- Entrenamiento de la DNN sin balanceo de clases en la red neuronal ni en el dataset: mediante el argumento *class_weight* del método *fit* de un modelo compilado, se puede pasar un diccionario con distintos pesos según la clase y los parámetros de la red se actualizarán durante el *backpropagation* de forma no igualitaria, sino según los pesos proporcionados para cada clase. En este

escenario, este diccionario tendrá un peso 1:1 para ambas clases. Es decir, es como si no se usara. Además, se utilizarán los datos de entrenamiento sin balancear con la descompensación vista.

- Entrenamiento de la DNN con balanceo de clases en la red neuronal pero no en el dataset: en este caso, se calcularán los valores para el *class_weight* mediante *compute_class_weight* de *sklearn* para entrenar la red considerando los pesos de las clases.
- Entrenamiento de la DNN con balanceo de clases en la red neuronal y en el dataset:
 - Balanceo del dataset mediante una *Red Neuronal Generativa Adversarial Condicional Tabular* (*CTGAN* por sus siglas en inglés): gracias a las redes neuronales adversariales se pueden crear datos sintéticos a partir de una muestra data. Popularmente, se utiliza para la generación de imágenes y vídeos. Sin embargo, en este escenario se utilizará para crear instancias sintéticas tabulares del dataset, concretamente, del tipo minoritario.
 - Balanceo del dataset mediante el algoritmo *SMOTE*.
- Para cada uno de los casos expuestos, se crearán tres modelos distintos variando el *dropout* para seleccionar el modelo que mejor resultados obtenga tratando de evitar el sobreajuste. Hasta el momento, esto hace el total de 9 modelos distintos, los cuales se someterán a una evaluación visual de las pérdidas de entrenamiento y las métricas clásicas de entrenamiento, validación y test.
- De las tres pruebas expuestas, se escogerá la que genere el modelo más sencillo que cumpla con los criterios de éxito y se repetirá la misma prueba pero la CNN (es decir, si se aumentan los datos de entrenamiento o si se utiliza el balanceo de clases en el propio entrenamiento de la red o ambas cosas).
- Para el entrenamiento de la CNN se realizará una transformación de vector de instancia de transacción en una imagen mediante un procedimiento explicado próximamente.
- Por último, se desarrollará un modelo combinado DNN+CNN, usando los modelos ya entrenados, pero combinando la predicción en la última capa de salida.

Una vez se ha llevado a cabo todo este proceso, se evaluarán los resultados finales y se sacarán conclusiones.

Criterios de Comparación de Modelos: métricas y umbrales

Dado que el fraude es un evento poco frecuente y de alto costo, la métrica más relevante no suele ser la exactitud, sino otras que capturen el equilibrio entre detección de fraudes y falsos positivos. A continuación, se describirán las métricas usadas para la comparación de los distintos modelos:

- Sensibilidad (*Recall*): Es la métrica más importante en detección de fraudes, ya que mide la proporción de fraudes reales que el modelo detecta correctamente. Un modelo con alta sensibilidad es preferible, incluso si tiene una precisión ligeramente menor.
- Precisión: Mide la proporción de transacciones identificadas como fraudulentas que realmente lo son. Una alta precisión reduce los falsos positivos, lo que es importante para evitar molestias a los clientes.
- AUC-ROC: Esta métrica es útil para comparar modelos en términos de su capacidad para distinguir entre clases (fraude vs no fraude) a través de todos los umbrales de decisión. Un AUC más alto indica un mejor rendimiento general.
- Exactitud (*Accuracy*): Es menos relevante en problemas con clases desbalanceadas, como el fraude, donde la clase minoritaria (fraude) es la más importante.

Además, se va a usar la *Matriz de confusión*, la cual da una visión detallada de los falsos positivos (FP) y los falsos negativos (FN), prestando especial atención a estos últimos ya que representan los fraudes no detectados.

Por otro lado, en el entrenamiento de las redes neuronales profundas, las métricas de pérdida (*loss*) son fundamentales para evaluar cómo el modelo está aprendiendo y generalizando. Estas métricas miden la discrepancia entre las predicciones del modelo y los valores reales. Un buen modelo debe tener una pérdida de validación que disminuya y se estabilice con las iteraciones de entrenamiento (*epochs*). Para elegir bien el modelo, se debe prestar atención a las siguientes características:

- Pérdida de validación estable.
- Mínimo sobreajuste.
- Comparación de pérdidas entre los distintos modelos.

Carga de librerías para redes neuronales profundas

Se emplearán las librerías de *Keras* para la construcción y entrenamiento de redes neuronales profundas, aprovechando su integración con *TensorFlow*. Se utilizarán modelos *Sequential* y *Functional* para diseñar tanto redes neuronales densas (DNN) como convolucionales (CNN). Las capas incluirán neuronas densas (*Dense*), normalización por lotes (*BatchNormalization*), y mecanismos de regularización como *Dropout* para reducir el sobreajuste. En el caso de las CNN, se implementarán capas convolucionales (*Conv2D*), de agrupamiento (*MaxPooling2D*) y aplanamiento (*Flatten*) para extraer características relevantes de los datos.

Para optimizar el entrenamiento, se empleará el optimizador *Adam* con una tasa de aprendizaje (*learning rate*) de 0,0002 y se integrarán métricas clave como *Recall*, *Precision* y *AUC* para evaluar el desempeño de los modelos.

Además, se utilizará *EarlyStopping* para detener el entrenamiento si la pérdida de validación deja de mejorar, evitando el sobreajuste.

Funciones personalizadas

Se han desarrollado varias funciones personalizadas para facilitar el entrenamiento, evaluación y análisis de los modelos de detección de fraudes. Estas funciones permiten automatizar tareas clave, optimizar el flujo de trabajo y mejorar la interpretación de los resultados.

El proceso comienza con la función `train_load_model()`, encargada de entrenar un modelo definido o, si este ya ha sido previamente generado, cargarlo junto con su historial de entrenamiento desde el directorio correspondiente. Para garantizar que el modelo se entrene con parámetros adecuados, se ha creado una función previa, `test_model()` que calcula métricas fundamentales como el balanceo de clases, define los valores de `dropout` a evaluar y organiza los distintos modelos en diccionarios según sus configuraciones. Esto permite estructurar el entrenamiento y probar diferentes variantes de la arquitectura de red.

Una vez entrenado el modelo, es esencial analizar su desempeño en distintos aspectos. Para ello, se han implementado funciones de visualización. La función `plot_model_hist()` permite representar gráficamente tanto las pérdidas como las métricas del entrenamiento y validación, facilitando la detección de problemas como el sobreajuste. De manera complementaria, `plot_metrics()` evalúa las métricas del modelo sobre el conjunto de prueba (test), iterando sobre distintos umbrales de decisión (`thresholds`) para comprender mejor su comportamiento bajo diferentes criterios de clasificación.

Para analizar el rendimiento en términos de falsos positivos y falsos negativos, se utiliza `plot_cm()`, que grafica la matriz de confusión y permite visualizar cómo el modelo está clasificando las transacciones fraudulentas y legítimas. Finalmente, la función `plot_fig_metrics()` genera un gráfico combinado que integra las anteriores, proporcionando una visión global del comportamiento del modelo.

Este conjunto de funciones permite un entrenamiento estructurado, una evaluación detallada y una interpretación visual clara del rendimiento del modelo, lo que resulta esencial para optimizar la detección de fraudes y mejorar la toma de decisiones en el proceso de modelado.

5.5.1. Red Neuronal Densa (DNN)

Desarrollo de la arquitectura de la DNN

La arquitectura desarrollada para la DNN se ha orientado para la resolución de un problema de clasificación binaria, donde el objetivo es predecir una salida entre 0 y 1. La red está compuesta por varias capas densas, comenzando con una capa de 512 neuronas, seguida por capas de 256, 128, 64 y 32 neuronas (Figura 13), en ese orden, antes de llegar a la capa de salida. Todas las capas ocultas utilizan la función de activación *ReLU* para introducir no linealidad y permitir que el modelo aprenda representaciones complejas. La capa final usa la activación sigmoide, adecuada para tareas de clasificación binaria, ya que transforma la salida en un valor entre 0 y 1, que puede interpretarse como una probabilidad.

Para evitar el sobreajuste y mejorar la capacidad generalizadora del modelo, se incorpora *Batch Normalization* después de cada capa densa. Esta técnica normaliza las activaciones de las capas, ayudando a estabilizar el entrenamiento y a acelerar la convergencia. Además, se utiliza *Dropout* en cada capa, lo que implica que, durante cada iteración de entrenamiento, una fracción de las neuronas se desactiva de manera aleatoria, forzando al modelo a aprender representaciones más robustas y evitar el sobreajuste.

El modelo se optimiza utilizando el optimizador *Adam*, que es conocido por ser eficiente y adaptativo, con una tasa de aprendizaje controlada por el parámetro *lr* (learning rate o tasa de aprendizaje). La función de pérdida empleada es *binary_crossentropy*, la cual es comúnmente utilizada en problemas de clasificación binaria, ya que mide la discrepancia entre las predicciones y las verdaderas etiquetas.

En cuanto a las métricas, el modelo evalúa el rendimiento utilizando precisión y sensibilidad (*recall*), que son esenciales cuando se trabaja con conjuntos de datos desbalanceados.

Finalmente, se incluye también el Área Bajo la Curva (AUC), una métrica adicional que evalúa la capacidad del modelo para distinguir correctamente entre las dos clases.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	23,552
batch_normalization (BatchNormalization)	(None, 512)	2,048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
batch_normalization_2 (BatchNormalization)	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8,256
batch_normalization_3 (BatchNormalization)	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 1)	33

Total params: 201,985 (789.00 KB)

Trainable params: 200,065 (781.50 KB)

Non-trainable params: 1,920 (7.50 KB)

Figura 13. Arquitectura de red neuronal densa desarrollada. Elab. Propia.

DNN *Tests*

Tras haber llevado a cabo las pruebas con la DNN descritas en el apartado Descripción del experimento, se obtienen los resultados mostrados en la Figura 14, Figura 15, Figura 18 y Figura 20.

Entrenamiento sin Balanceo de clases en red neuronal ni dataset

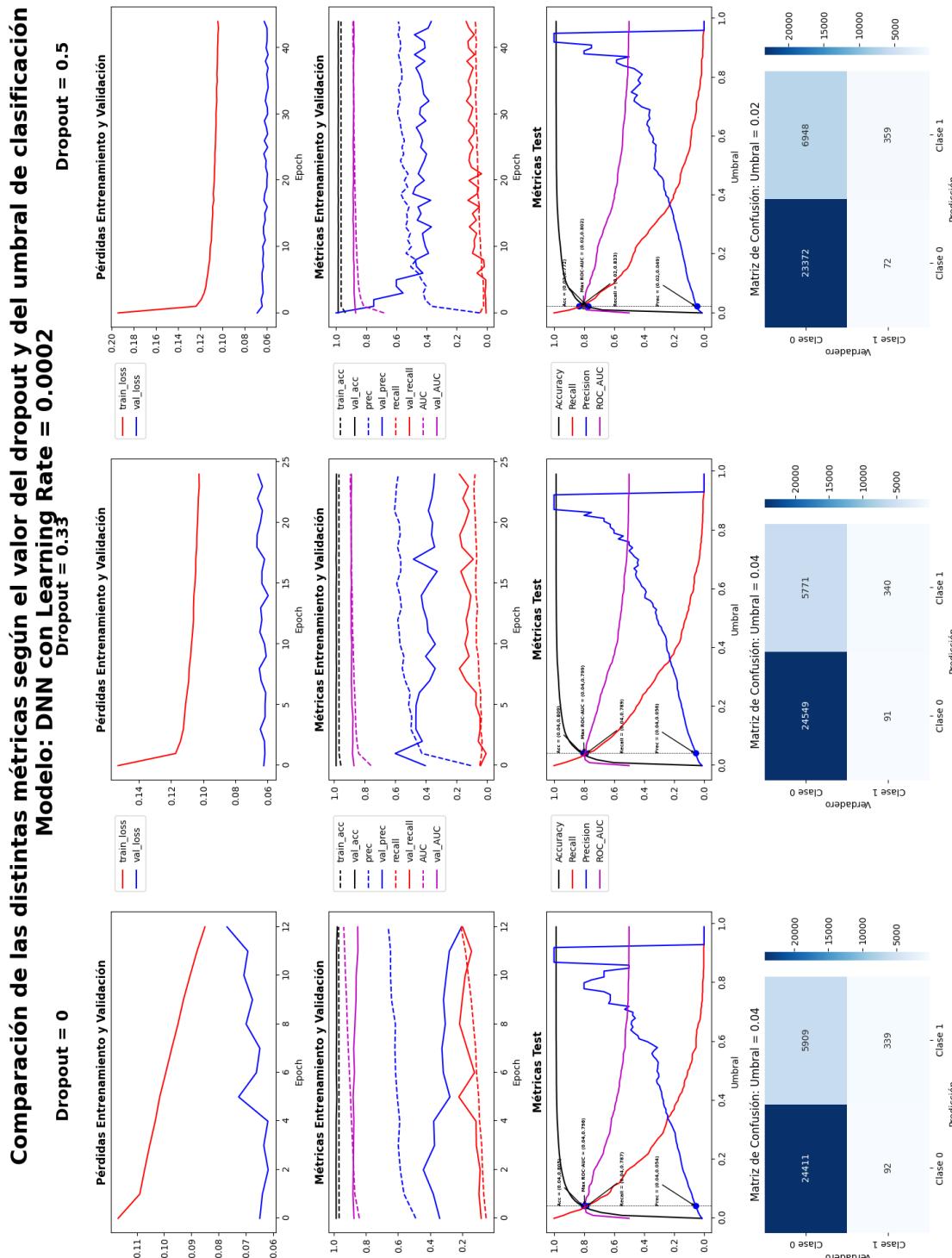


Figura 14. DNN entrenada sin balanceo de clases en red neuronal ni dataset. Elab. Propia

Entrenamiento con Balanceo de clases en red neuronal

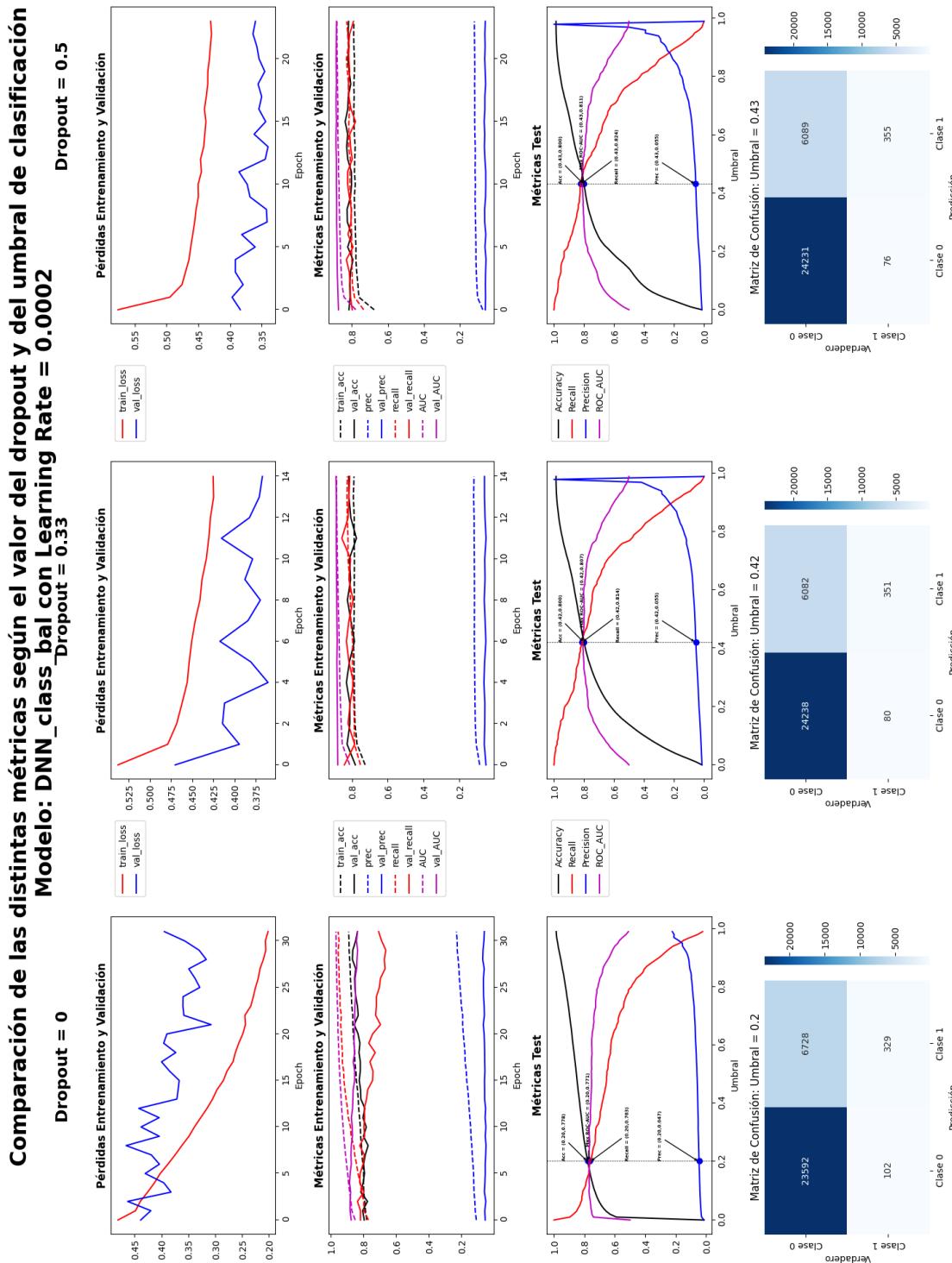


Figura 15. DNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal pero no en el dataset. Elab. Propia

Balanceo de clases del dataset mediante CTGAN

Debido a limitaciones computacionales, se ha de realizar un gran submuestreo del dataset de entrenamiento con el que entrenar a la CTGAN. Esto puede causar problemas en el reconocimiento de patrones de la clase 0 así como sus diferencias con la clase 1, ya que la variabilidad y sesgos controlados en la creación del dataset original se han ido perdiendo al submuestrear por la falta de capacidad de procesar semejante cantidad de datos.

Para que este procesamiento sea posible con recursos locales, se ha de submuestrear la clase 0 reduciéndose un 95%. De esta forma, la CTGAN será entrenada con la distribución de clases mostrada en la Figura 16.

Se han mantenido las 8.151 instancias de la clase 1 pero se ha reducido a 11.802 instancias la clase 0. Una vez entrenada la red neuronal generativa, se define una estrategia de incremento de la clase 1 sobre el dataset de entrenamiento previo a este último submuestreo, es decir, el dataset sobre el que se quieren aumentar las transacciones fraudulentas. Como debido a la poca representación de la clase 0 puede ser que los datos sintéticos que se generen contengan mucho ruido, no se debe abusar de la creación de estos. Por lo tanto, la estrategia definida ha sido que la clase 1 pase a un 5% del total, Figura 17.

Merece especial mención que se ha probado distintas estrategias de balanceo con porcentaje mayores (hasta un 30%). Sin embargo, a pesar de haber comprometido el costo computacional, los resultados no fueron superiores a los obtenidos (ver siguiente apartado, Balanceo de clases del dataset mediante CTGAN).

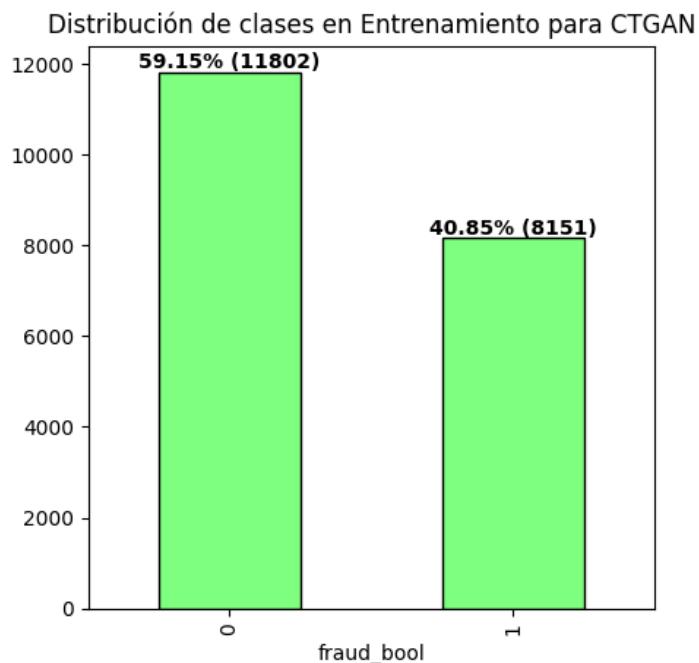


Figura 16. Distribución de clases en el dataset de entrenamiento para la CTGAN. Elab. Propia

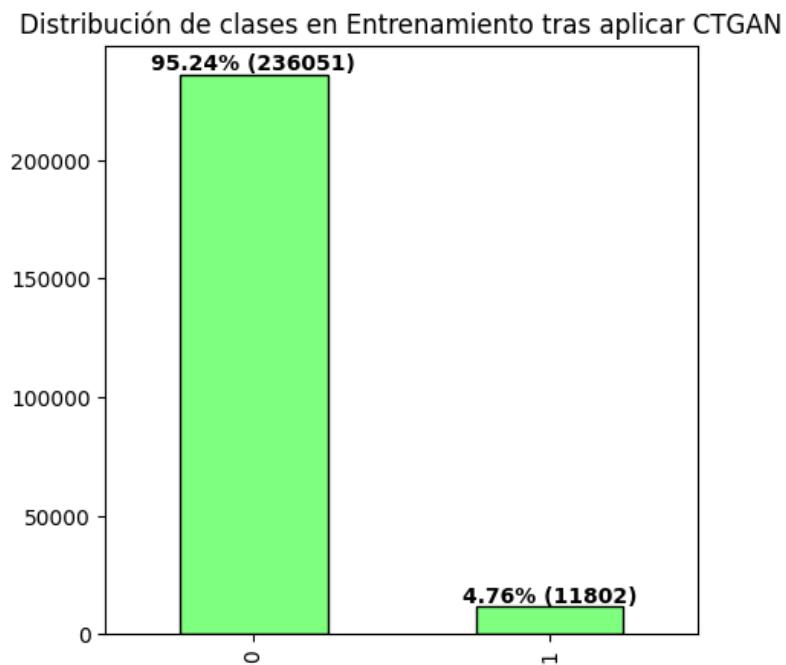


Figura 17. Distribución de clases en el dataset de entrenamiento tras la generación de datos sintéticos de la clase 1 con CTGAN. Elab. Propia

Entrenamiento con Balanceo de clases en red neuronal y dataset mediante CTGAN

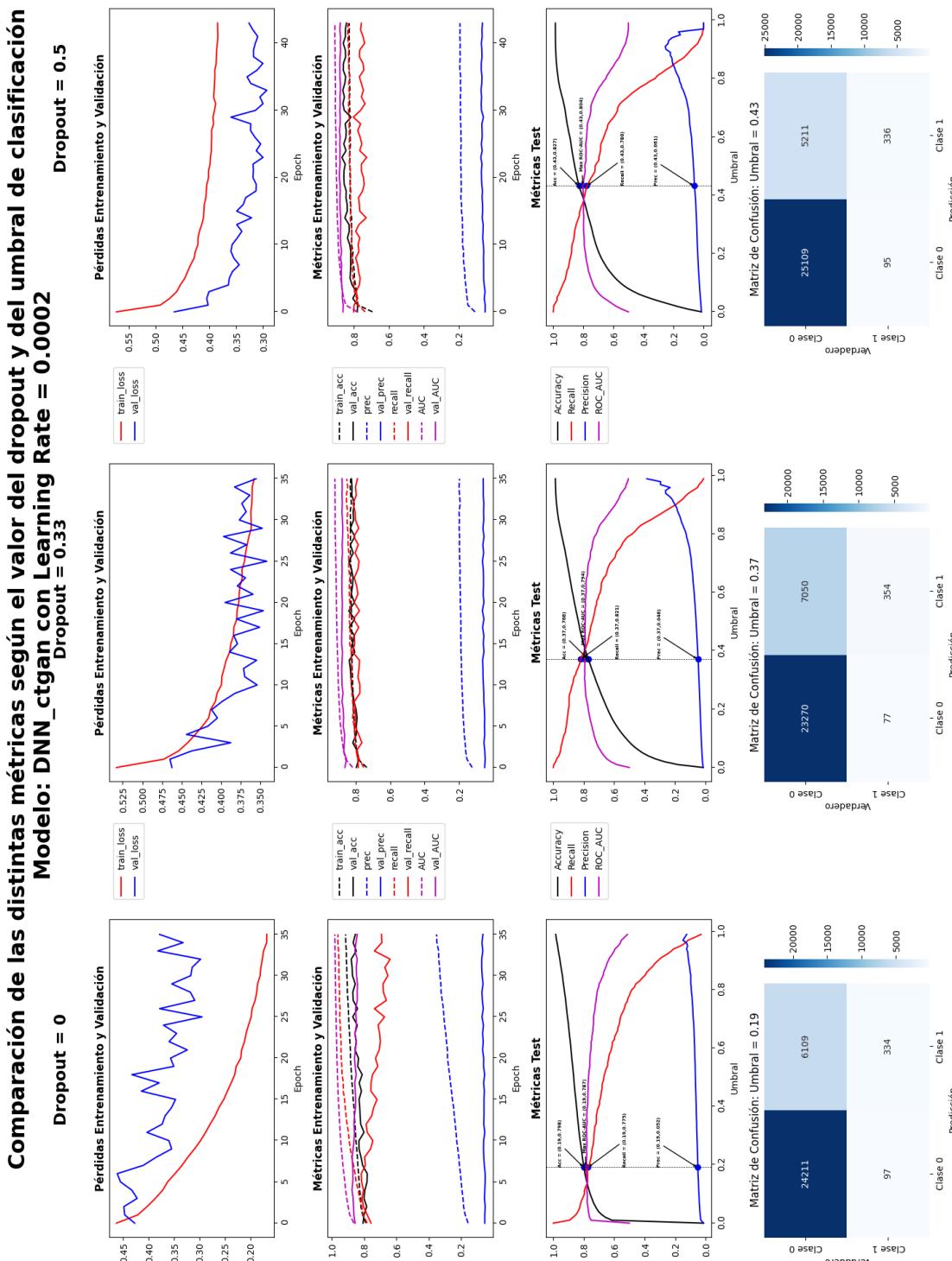


Figura 18. DNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal y también en el dataset mediante una CTGAN. Elab. Propia

Balanceo de clases del dataset mediante SMOTE

Con el fin de poder ser adecuadamente comparable con la técnica de balanceo CTGAN, se ha optado por seguir con la misma estrategia de sobremuestreo de la clase 1 con SMOTE.

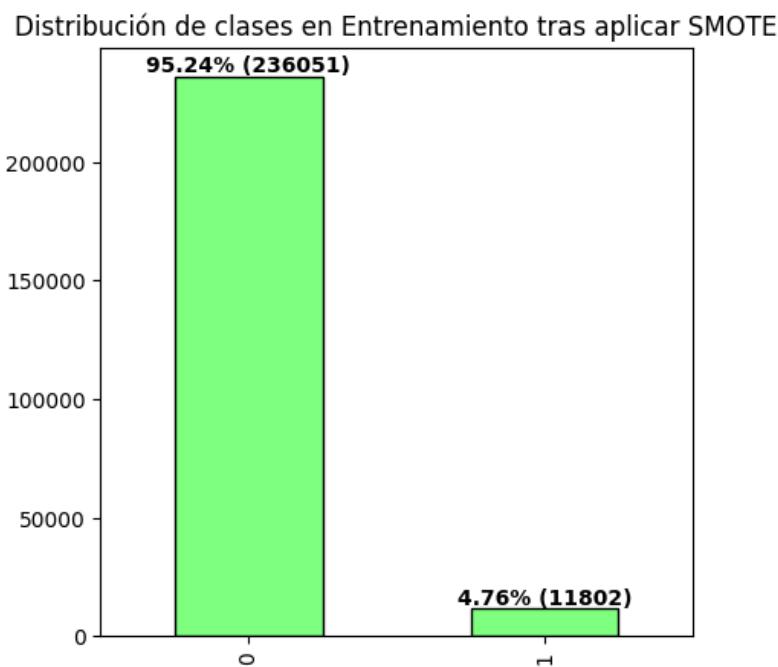


Figura 19. Distribución de clases en el dataset de entrenamiento tras la generación de datos sintéticos de la clase 1 con SMOTE. Elab. Propia

La principal diferencia, además de la manera en la que este algoritmo crea los datos sintéticos con respecto a una CTGAN, es que el dataset de entrenamiento con el que se ha entrenado no está submuestreado, ya que SMOTE no tiene un costo computacional tan elevado de entrenamiento como la CTGAN.

Entrenamiento con Balanceo de clases en red neuronal y dataset mediante SMOTE

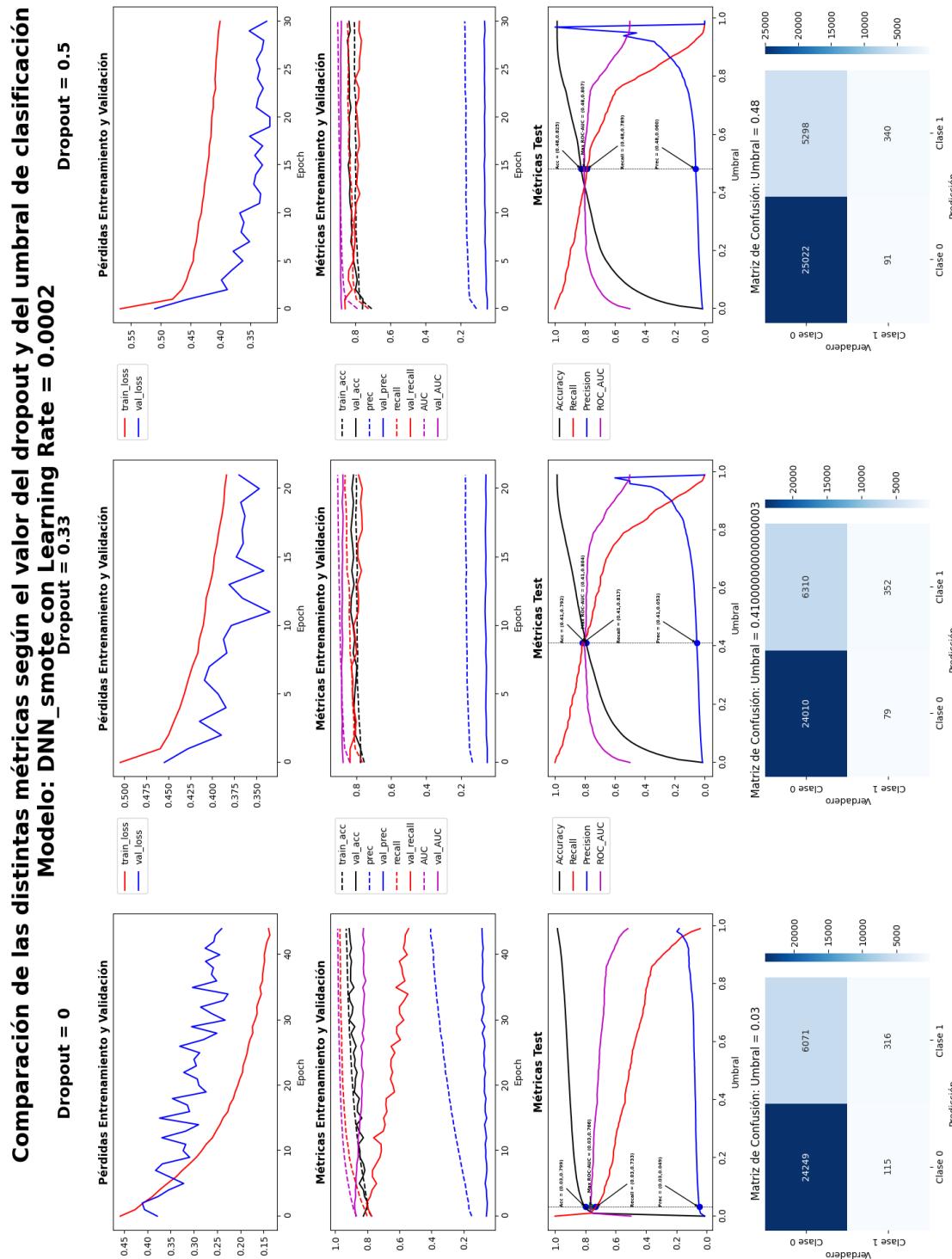


Figura 20. DNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal y también en el dataset mediante SMOTE. Elab. Propia

- Observando los resultados de los verdaderos positivos de la clase fraudulenta (clase 1) en las matrices de confusión, el **modelo 3** alcanza un valor de 359, que representa un 83,3% de sensibilidad. Este modelo se entrenó sin ningún tipo de balanceo y con un *dropout* de 0,5. Aunque, por este resultado, pueda parecer el mejor modelo, el umbral al que se alcanzan estos valores es de 0,02, muy cercano al cero y con una curva de ROC-AUC, sensibilidad y exactitud muy acusadas. Frente a una pequeña variación de los datos de test, el umbral para este resultado podría moverse un poco, causando que los resultados sean completamente diferentes (disminuyendo el umbral tan solo 0,01, la exactitud caería al 40% aproximadamente). Por lo tanto, no se considera el mejor modelo.
- El segundo modelo con el mejor resultado de verdaderos positivos de la clase fraudulenta es el **modelo 6** con 355 VP, es decir, un 82,4% de sensibilidad. Este resultado se ha obtenido con balanceo de pesos del entrenamiento de la red neuronal, pero sin balanceo de clases en el dataset y con un *dropout* de 0,5. A diferencia del anterior, las curvas de las métricas son mucho más suaves, demostrando ser un modelo más robusto frente a variaciones de los datos de test, siendo el umbral igual a 0,43.
- Poniendo el foco en el máximo valor de la curva ROC-AUC, precisamente el **modelo 6** es el que presenta el mayor valor de todos los modelos: 0,811. En cuanto a la precisión y FP de la clase 0, se mantiene en valores medios en comparación con el resto, 5,5% y 6.089 respectivamente, manteniendo una exactitud general de un 80%.

Por lo tanto, para la creación de la CNN no se usará el balanceo de clases en el dataset, pero sí se usará el balanceo de pesos en el entrenamiento.

5.5.2. Red Neuronal Convolutacional (CNN)

De Vector unidimensional a Imagen RGB

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) son un tipo de arquitectura de redes neuronales especialmente diseñada para procesar datos con estructura espacial, como imágenes. A diferencia de las redes densas, que conectan todas las neuronas entre capas, las CNN utilizan operaciones de convolución para extraer características locales y jerárquicas, lo que las hace más eficientes en términos de parámetros y computación. Esta capacidad les permite capturar patrones como bordes, texturas y formas, lo que las convierte en una herramienta poderosa para tareas de visión por computadora, como clasificación de imágenes, detección de objetos y segmentación. Además, su diseño las hace menos propensas al sobreajuste en comparación con las redes densas, especialmente cuando se trabaja con datos de alta dimensionalidad.

El uso de redes neuronales convolucionales (CNN) como alternativa a una red neuronal densa (DNN) para la clasificación binaria de transacciones fraudulentas o legítimas en

el ámbito bancario se justifica por su capacidad para capturar relaciones complejas y patrones no lineales en los datos, especialmente cuando estos se transforman en una representación visual. La idea principal consiste en convertir cada instancia tabular del dataset en una imagen de tres canales RGB, lo que permite aprovechar la potencia de las CNN para extraer características espaciales y jerárquicas que podrían pasar desapercibidas en una DNN.

En este enfoque, cada transacción se representa como un vector unidimensional escalado entre 0 y 1, que se transforma en una matriz cuadrada de tamaño $y \times y$, donde y es el número de características. La diagonal principal de la matriz se llena con los valores del vector, mientras que los triángulos superior e inferior se completan de manera simétrica utilizando operaciones matemáticas entre pares de características. Esta simetría garantiza que la estructura de la matriz sea coherente y aproveche las relaciones entre las características. Al tratarse de una imagen RGB, se generan tres matrices (canales) por transacción:

1. **Canal 1:** Se calcula la media de cada par de características, lo que proporciona información sobre la tendencia central de las relaciones entre ellas.
2. **Canal 2:** Se utiliza la diferencia entre cada par de características, lo que resalta las disparidades o contrastes entre ellas.
3. **Canal 3:** Se calcula el producto de cada par de características, lo que enfatiza las interacciones multiplicativas entre ellas.

Véase el siguiente sencillo ejemplo para entender mejor este proceso. Si se tiene una instancia del dataset cuyos valores son a , b y c , ésta puede ser representada matemáticamente por el siguiente vector, Ecuación 5):

$$V = (a, b, c) \quad \text{Ecuación 5}$$

Siguiendo la metodología, se crea una matriz cuadrada de $y \times y$, donde para este caso $y = 3$ (número de características del vector de la Ecuación 5).

$$M = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \quad \text{Ecuación 6}$$

La Ecuación 6 representa la matriz base sobre la que se crearán 3 matrices simétricas (una para cada canal). Los triángulos superiores e inferiores serán simétricos, cumpliendo con la propiedad de simetría de la Ecuación 7, con respecto a la diagonal principal, variando la forma de calcular sus parámetros.

$$\forall i, j, \quad M_{i,j} = M_{j,i} \quad \text{Ecuación 7}$$

Canal 1, se utiliza la operación promedio de los pares de valores, Ecuación 8. Estos valores se utilizarán para representar el color rojo (R).

$$M = \begin{bmatrix} a & AVG(a, b) & AVG(a, c) \\ AVG(b, a) & b & AVG(b, c) \\ AVG(c, a) & AVG(c, b) & c \end{bmatrix} \quad Ecuación\ 8$$

Canal 2, se utiliza la operación valor absoluto de la diferencia de los pares de valores, Ecuación 9. Estos valores se utilizarán para representar el color verde (G).

$$M = \begin{bmatrix} a & |diff(a, b)| & |diff(a, c)| \\ |diff(b, a)| & b & |diff(b, c)| \\ |diff(c, a)| & |diff(c, b)| & c \end{bmatrix} \quad Ecuación\ 9$$

Canal 3, se utiliza la operación producto de los pares de valores, Ecuación 10. Estos valores se utilizarán para representar el color azul (B).

$$M = \begin{bmatrix} a & prod(a, b) & prod(a, c) \\ prod(b, a) & b & prod(b, c) \\ prod(c, a) & prod(c, b) & c \end{bmatrix} \quad Ecuación\ 10$$

Como en el preprocesado de datos todas las características se convirtieron a valores numéricos acotados en [0, 1], cada una de estas matrices contendrá valores, también, acotados en ese rango. Al superponerse las tres matrices, se obtiene una representación tridimensional de cada transacción en forma de una matriz de dimensiones (x, y, y, c), donde x es el número de instancias, y es el número de características y c es el número de canales (3 en este caso). Como cada canal es interpretado como un color RGB (rojo, verde y azul), el resultado final es una seudooImagen donde puede mostrarse cualquier color como la suma de cada capa, Figura 23.

Esta transformación permite a la CNN analizar las transacciones como si fueran imágenes, capturando patrones locales y globales que podrían ser críticos para identificar fraudes. Además, al utilizar operaciones matemáticas específicas para generar los canales, se enriquece la información disponible para el modelo, lo que puede mejorar su capacidad de discriminación entre transacciones fraudulentas y legítimas. Este enfoque es especialmente útil cuando las relaciones entre características no son evidentes en el formato tabular tradicional, pero pueden ser explotadas eficientemente mediante convoluciones en el espacio de la imagen.

Además de mejorar el rendimiento en la clasificación de transacciones fraudulentas, el enfoque de transformar datos tabulares en imágenes para su procesamiento con redes neuronales convolucionales (CNN) ofrece ventajas únicas que no son factibles con redes neuronales densas (DNN). Una de estas aplicaciones es la anonimización de datos mediante la codificación en imágenes, lo que permite preservar la privacidad de la información sensible mientras se mantiene la utilidad para el análisis. Al convertir las transacciones en imágenes, los datos originales (como nombres, números de cuenta o identificadores personales) pueden ser ofuscados o eliminados, ya que la representación visual no requiere conservar la estructura tabular original. Esto es especialmente útil en entornos donde la privacidad es crítica, como en el sector bancario o financiero.

Otra aplicación interesante es la integración con sistemas de visión por computadora existentes. Una vez que las transacciones se codifican como imágenes, pueden ser procesadas por sistemas diseñados para analizar imágenes, como escáneres o herramientas de procesamiento visual. Por ejemplo, las transacciones podrían ser "escaneadas" o analizadas mediante técnicas de visión artificial, lo que permitiría su clasificación o detección de anomalías utilizando infraestructura ya desplegada para otros fines, como la inspección de documentos o la verificación de firmas. Esto no sería posible con una DNN, ya que estas redes no están diseñadas para trabajar directamente con datos visuales.

Además, la representación en imágenes permite la compresión eficiente de datos. Las técnicas de compresión de imágenes, como JPEG o PNG, pueden aplicarse a las matrices generadas, reduciendo el tamaño de los datos sin perder información crítica. Esto es especialmente útil cuando se trabaja con grandes volúmenes de transacciones, ya que facilita el almacenamiento y la transmisión de datos.

Por último, este enfoque permite la interpretación visual de los datos. Las imágenes generadas pueden ser inspeccionadas visualmente por expertos para identificar patrones o anomalías, lo que no es posible con una DNN que opera directamente sobre datos tabulares. Esto abre la puerta a un análisis híbrido, donde tanto el modelo de CNN como los expertos humanos pueden colaborar para mejorar la detección de fraudes.

A continuación, se exponen imágenes de transacciones legítimas y fraudulentas tras haber aplicado la metodología explicada.

Imágenes de transacciones no fraudulentas (valid_data)

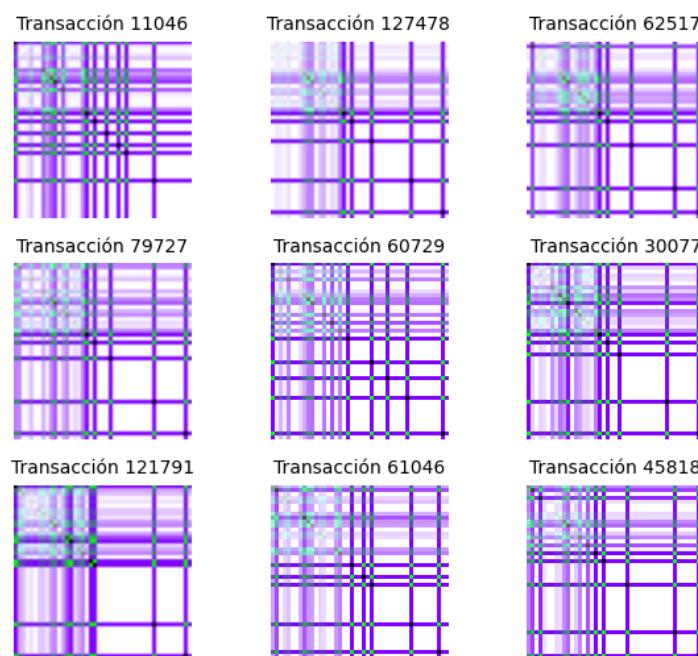


Figura 21. Imágenes sintéticas a partir de vectores de transacciones legítimas. Elab. Propia

Imágenes de transacciones fraudulentas (fraud_data)

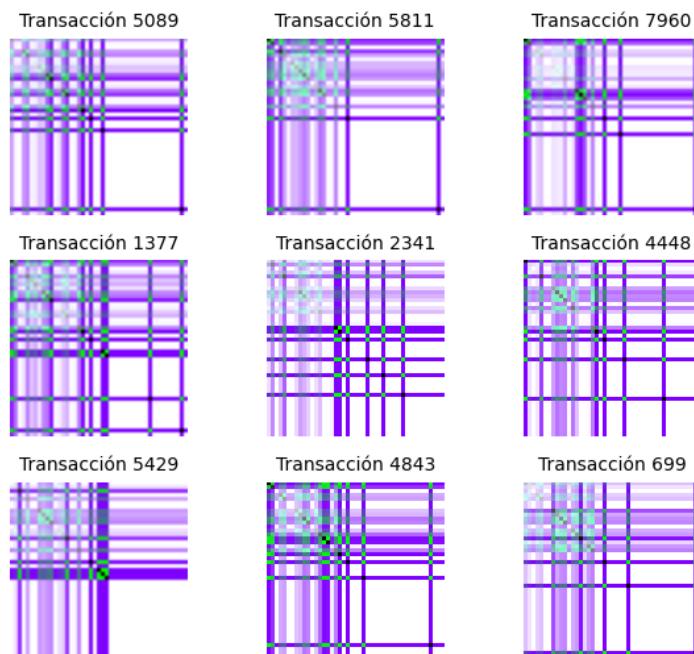


Figura 22. Imágenes sintéticas a partir de vectores de transacciones fraudulentas. Elab. Propia

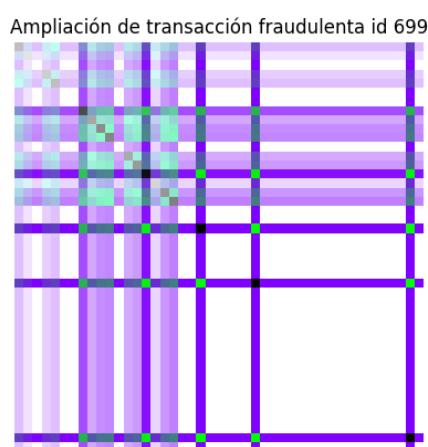


Figura 23. Imágenes sintéticas a partir de vectores de transacciones fraudulentas. Elab. Propia

Desarrollo de la arquitectura de la CNN

El desarrollo de la arquitectura usada para la CNN se ha basado en la Figura 24 (Adrán C. & Gabriel M., s.f., Figura 18).

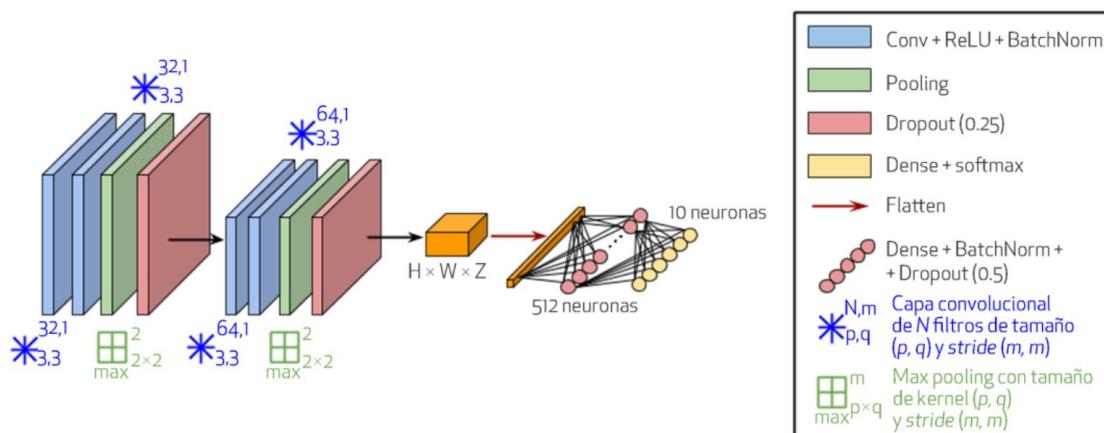


Figura 24. Arquitectura de red neuronal convolucional típica. En este caso se trata de una CNN compuesta por dos bloques convolucionales. (Adrán C. & Gabriel M., s.f., Figura 18.).

La arquitectura de la red convolucional (CNN) propuesta se basa en bloques convolucionales y capas densas, con una estructura diseñada para manejar imágenes de entrada y realizar una clasificación binaria. Comienza con la definición de las entradas de la red, que son imágenes de tamaño (*resol, resol, channels*), donde *resol* es la resolución de las imágenes (número de características del vector original) y *channels* hace referencia al número de canales de color (3 para imágenes RGB).

En el primer bloque de capas convolucionales, se emplean dos capas convolucionales con filtros de tamaño 3x3 y activación *ReLU*, seguidas de *BatchNormalization* para estabilizar el entrenamiento, lo que mejora la velocidad de convergencia. Después, se aplica una capa de *MaxPooling2D* para reducir la dimensionalidad de las activaciones y una capa de *Dropout* para evitar el sobreajuste. Esta combinación de operaciones es repetida en los siguientes bloques con una expansión progresiva en el número de filtros: 32, 64 y 128, lo que permite a la red aprender representaciones más complejas y abstractas de las imágenes. Cada bloque sigue el mismo patrón: dos capas convolucionales, *BatchNormalization*, *MaxPooling* y *Dropout*.

Después de la parte convolucional, la red pasa a una sección densa, donde las características extraídas por las capas convolucionales se aplanan mediante *Flatten*, convirtiendo la salida en un vector unidimensional. A continuación, se pasan a través de varias capas densas con activación *ReLU* y *BatchNormalization*, seguidas nuevamente de *Dropout* para prevenir el sobreajuste. Este enfoque de capas densas y de regularización permite que la red aprenda patrones más complejos y generalice mejor en datos no vistos.

Finalmente, la capa de salida tiene una sola neurona con activación sigmoide, adecuada para tareas de clasificación binaria. La función de pérdida utilizada es *binary_crossentropy*, ya que el objetivo es predecir una etiqueta de clase (fraude o no fraude) para cada imagen.

El modelo es entrenado utilizando el optimizador Adam con una tasa de aprendizaje definida por *lr* (learning rate), y se monitorean varias métricas de rendimiento: precisión, recuperación y AUC. Además, se implementa un mecanismo de *early stopping* para detener el entrenamiento si no se observa mejora en la validación, ayudando a prevenir el sobreajuste. También se emplea *class_weight* para balancear el entrenamiento como ya se usó previamente en la DNN.

Esta arquitectura (Figura 25) es más profunda y compleja en comparación con otras implementaciones más sencillas (Figura 24), permitiendo capturar una mayor cantidad de patrones y mejorar el rendimiento en tareas de clasificación complejas, como la detección de fraudes en imágenes.

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 45, 45, 3)	0
conv2d_12 (Conv2D)	(None, 45, 45, 32)	896
batch_normalization_18 (BatchNormalization)	(None, 45, 45, 32)	128
conv2d_13 (Conv2D)	(None, 45, 45, 32)	9,248
batch_normalization_19 (BatchNormalization)	(None, 45, 45, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 22, 22, 32)	0
dropout_12 (Dropout)	(None, 22, 22, 32)	0
conv2d_14 (Conv2D)	(None, 22, 22, 64)	18,496
batch_normalization_20 (BatchNormalization)	(None, 22, 22, 64)	256
conv2d_15 (Conv2D)	(None, 22, 22, 64)	36,928
batch_normalization_21 (BatchNormalization)	(None, 22, 22, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 11, 11, 64)	0
dropout_13 (Dropout)	(None, 11, 11, 64)	0
conv2d_16 (Conv2D)	(None, 11, 11, 128)	73,856
batch_normalization_22 (BatchNormalization)	(None, 11, 11, 128)	512
conv2d_17 (Conv2D)	(None, 11, 11, 128)	147,584
batch_normalization_23 (BatchNormalization)	(None, 11, 11, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 5, 5, 128)	0
dropout_14 (Dropout)	(None, 5, 5, 128)	0
flatten_2 (Flatten)	(None, 3200)	0
dense_8 (Dense)	(None, 512)	1,638,912
batch_normalization_24 (BatchNormalization)	(None, 512)	2,048
dropout_15 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 256)	131,328
batch_normalization_25 (BatchNormalization)	(None, 256)	1,024
dropout_16 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 128)	32,896
batch_normalization_26 (BatchNormalization)	(None, 128)	512
dropout_17 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 1)	129


```
Total params: 2,095,649 (7.99 MB)
Trainable params: 2,092,961 (7.98 MB)
Non-trainable params: 2,688 (10.50 KB)
```

Figura 25. Arquitectura de red neuronal convolucional desarrollada. Elab. Propia.

CNN Tests

Entrenamiento con Balanceo de clases en CNN

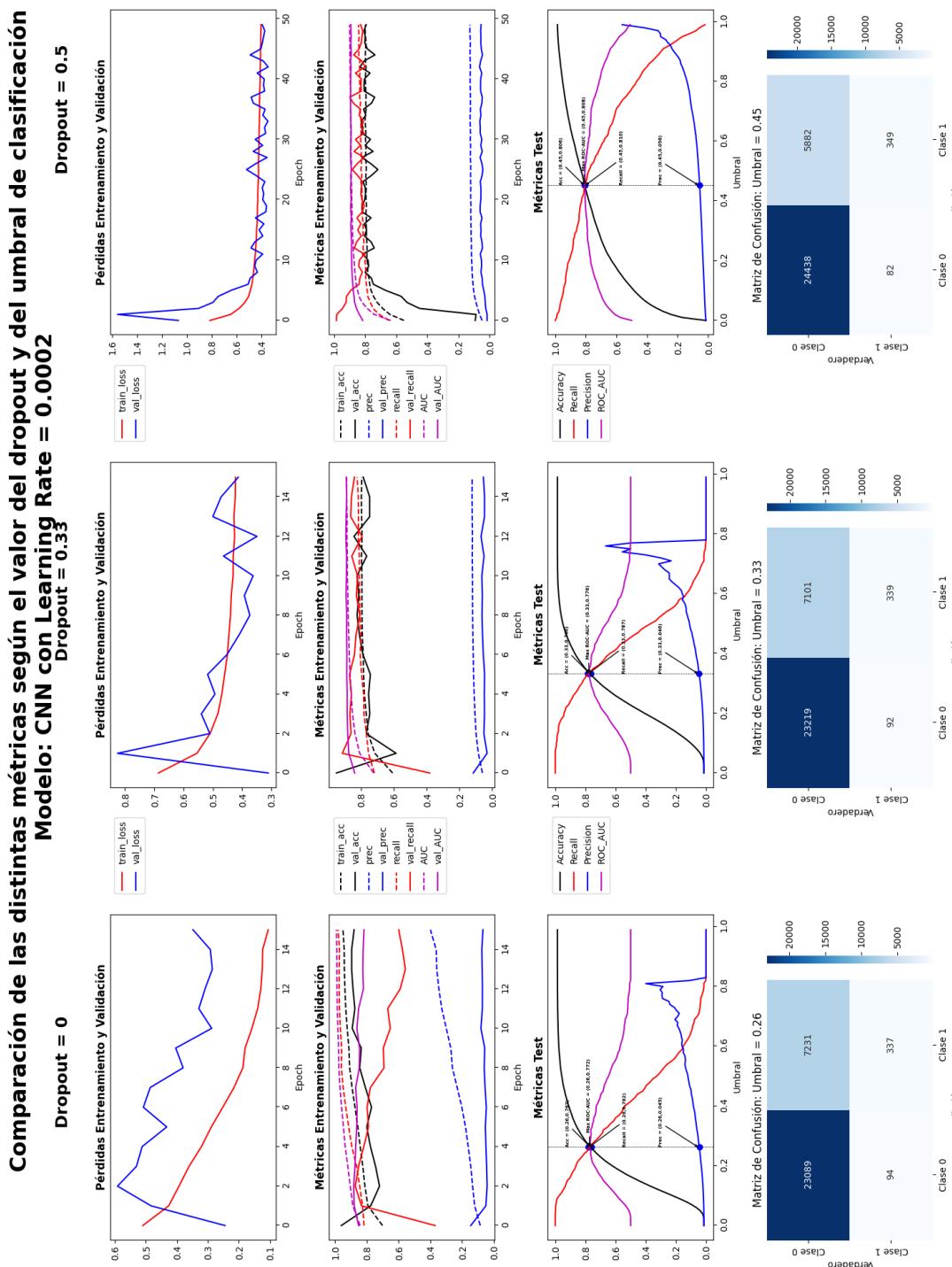


Figura 26. CNN entrenada con balanceo de clases (pesos ajustados por clase) en red neuronal pero no en el dataset de imágenes. Elab. Propia

Resultados de la CNN

Nuevamente, el dropout de 0,5 es el que mejor funciona como se puede observar en la Tabla 3, **modelo 15**, obteniendo resultados superiores a las otras dos iteraciones en todas las métricas y condiciones, así como unas curvas mucho más suaves y estables tanto en el entrenamiento de la red neuronal como en las métricas de test con un umbral más centrado.

Modelo	Tipo	Aj. Pesos	Bal. Data-Bal.	Alg. Bal.	Dropout	Max ROC-AUC	Umbral Mejor AUC-ROC	Prec. En Umbral	Recall En Umbral	Acc. En Umbral	VP (1)	FP (0)
13	CNN	Sí	No	-	0	0,772	0,26	0,045	0,782	0,762	337	7231
14	CNN	Sí	No	-	0,33	0,776	0,33	0,046	0,787	0,708	339	7101
15	CNN	Sí	No	-	0,5	0,808	0,45	0,056	0,81	0,806	349	5882

Tabla 3. Comparación de los modelos de tipo CNN

5.5.3. Modelo Combinado: DNN + CNN

DNN + CNN Tests

Entrenamiento DNN + CNN combinados y DNN + CNN aislados y predicciones promediadas

Comparación de las distintas métricas según el valor del umbral de clasificación

Modelo: DNN + CNN con dropout = 0.5 y lr = 0.0002

Promedio predicciones DNN + CNN aisladas

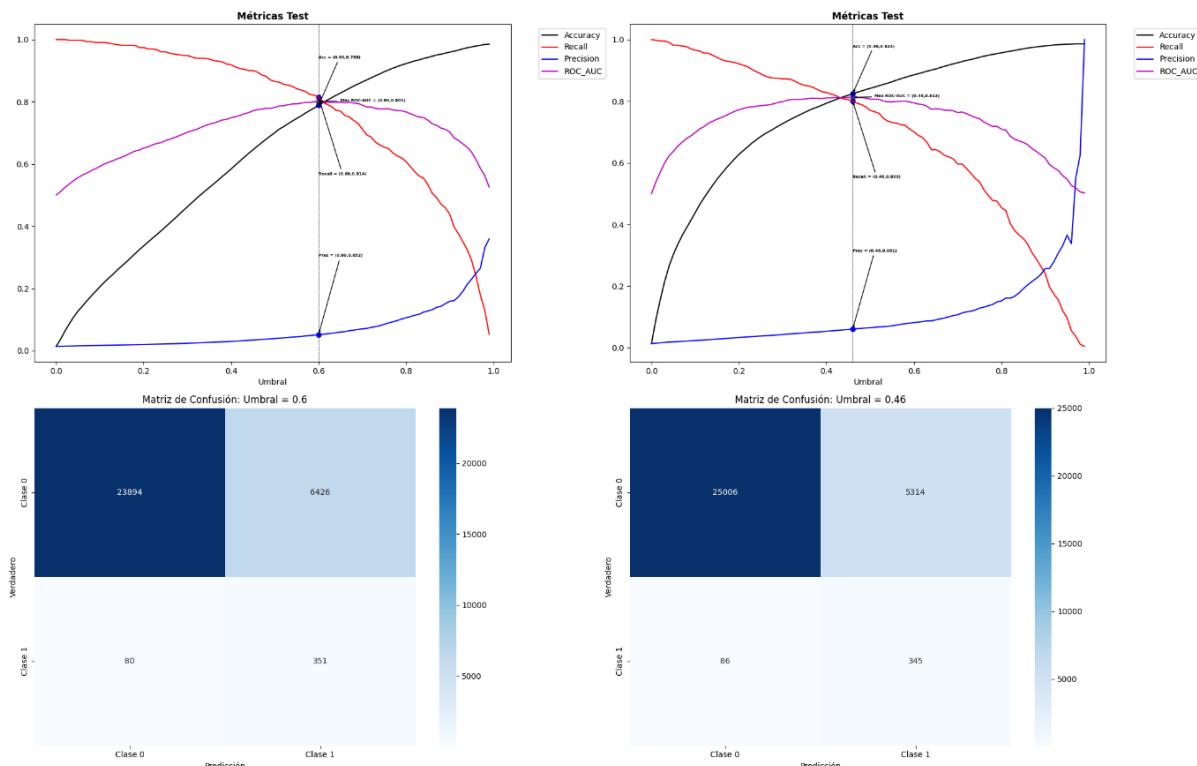


Figura 27. DNN + CNN combinados vs DNN + CNN promediados. Elab. Propia

Resultados de la CNN combinada

De acuerdo con la Figura 27 y la Tabla 4, los resultados muestran que, a pesar de que el **modelo 18** alcanza una mayor sensibilidad y, por lo tanto, mayor número de VP (1), en general el **modelo 19** obtiene mejores resultados en términos de distinción de clases en general, obteniendo una mayor métrica del ROC-AUC.

Modelo	Tipo	Aj. Pesos	Bal. Data-	Alg. Bal.	Dropout	Max ROC-AUC	Umbral Mejor AUC-ROC	Prec. En Umbral	Recall. En Umbral	Acc. En Umbral	VP (1)	FP (0)
18	CNN+DNN comb	Sí	No	-	0,5	0,801	0,6	0,052	0,814	0,788	351	6426
21	CNN+DNN prom	Sí	No	-	0,5	0,813	0,46	0,061	0,8	0,824	345	5314

Tabla 4. Comparación de los modelos de tipo CNN Combinada

5.6. Resultados generales

La Tabla 5 presenta los resultados de distintos modelos de redes neuronales, incluyendo DNN, CNN y combinaciones de ambas, evaluados con diferentes configuraciones de ajuste de pesos, balanceo de datos y dropout. Se analizan métricas clave como la ROC-AUC, precisión, recall, accuracy, y la cantidad de verdaderos positivos (VP) y falsos positivos (FP), lo que permite comparar su desempeño en la detección de fraudes.

Uno de los modelos destacados es el Modelo 6 (DNN con *dropout* 0,5, sin balanceo de datos). Este modelo muestra una de las mejores puntuaciones en ROC-AUC con 0,811, indicando una buena capacidad de clasificación. Su sensibilidad de 0,824 es uno de los más altos de la tabla, lo que significa que detecta un alto porcentaje de fraudes. Además, logra la mayor cantidad de verdaderos positivos (355), lo que refuerza su capacidad para identificar transacciones fraudulentas. Sin embargo, este modelo también tiene un número significativo de falsos positivos (6.089), lo que implica que genera más alertas erróneas que otros modelos. Aun así, si la prioridad es maximizar la detección de fraudes, este modelo es una excelente opción.

El segundo modelo a destacar es el Modelo 21 (Combinación CNN + DNN en promedio, *dropout* 0,5). Este modelo logra la mayor ROC-AUC de la tabla (0,813), lo que indica un rendimiento global superior en la clasificación. Además, presenta una precisión de 0,061, la mejor en la tabla, lo que significa que cuando predice un fraude, tiene una mayor probabilidad de ser correcto. Su exactitud de 0,824 también es destacable, mostrando un buen balance entre detección y errores. Un aspecto importante es que este modelo reduce los falsos positivos a 5.314, lo que lo hace una opción más equilibrada en comparación con otros modelos.

En términos de aplicación, la elección entre estos dos modelos depende de la estrategia deseada. Si el objetivo principal es detectar la mayor cantidad de fraudes posible sin importar el costo de generar más falsas alarmas, el Modelo 6 es la mejor opción. En cambio, si se busca un equilibrio entre detección y reducción de falsos positivos, el Modelo 21 sería más adecuado.

En conclusión, ambos modelos ofrecen ventajas distintas. El Modelo 6 es ideal si se prioriza la detección de fraudes con alta sensibilidad y más verdaderos positivos, aunque con más falsos positivos. El Modelo 21, por otro lado, ofrece el mejor balance global al combinar buena detección con menor cantidad de falsas alarmas, lo que lo hace más confiable en términos operativos.

6. Conclusiones

Este estudio ha permitido comparar distintos enfoques en la detección de fraude bancario utilizando redes neuronales profundas (DNN), convolucionales (CNN) y combinaciones de ambas. Se han evaluado diversos parámetros como el *dropout*, el balanceo de datos y el ajuste de pesos, permitiendo obtener resultados clave.

Uno de los hallazgos más importantes es que el *dropout* influye significativamente en el rendimiento de los modelos. Se ha observado que un valor de 0,5 produce los mejores resultados en términos de ROC-AUC y sensibilidad, lo que sugiere que puede ayudar a mejorar la capacidad generalizadora del modelo. Sin embargo, dado que 0,5 está en el límite evaluado, sería conveniente explorar valores más altos, como 0,6, para determinar si aún se pueden obtener mejoras.

Una contribución relevante de este estudio es la aplicación de la transformación de datos tabulares en imágenes para su procesamiento con redes neuronales convolucionales (CNN). Este enfoque ha resultado ser una alternativa prometedora a las redes neuronales densas (DNN), ya que permite capturar relaciones complejas entre las características mediante la extracción de patrones espaciales y jerárquicos. En combinación con una DNN, esta estrategia ha permitido obtener un modelo más balanceado, logrando una mejor métrica de ROC-AUC con una reducción en la cantidad de falsos positivos. Además, este enfoque ofrece beneficios adicionales, como la anonimización de datos, la integración con sistemas de visión por computadora y la compresión eficiente de la información. Dado su potencial, futuras investigaciones podrían explorar mejoras en la generación de imágenes a partir de datos tabulares y evaluar su impacto en distintos conjuntos de datos financieros.

En cuanto a la arquitectura de los modelos, las combinaciones de CNN y DNN han demostrado ser las más equilibradas, particularmente el modelo CNN+DNN en promedio (Modelo 21), que obtuvo la mejor puntuación en ROC-AUC (0,813) y logró un buen balance entre detección de fraudes y falsos positivos. Este resultado indica que la combinación de ambas arquitecturas puede capturar patrones más complejos en los datos, y sugiere que seguir explorando esta estrategia podría ser clave para mejorar aún más el desempeño del sistema.

También se ha encontrado que el uso de técnicas de balanceo de datos debe manejarse con cuidado. En particular, el sobremuestreo con CTGAN ha demostrado perder demasiada información, lo que ha afectado negativamente a los modelos que lo utilizaron. Por otro lado, el balanceo con SMOTE no ha resultado ser tampoco demasiado efectivo a pesar de haber entrenado con todos los datos de entrenamiento de la clase fraudulenta. Dado que este algoritmo se basa en la interpolación lineal de las instancias de la clase minoritaria, si los datos tienen una estructura compleja y patrones no lineales, esta interpolación puede no reflejar adecuadamente la distribución real, lo que potencialmente introduce ruido o ejemplos poco representativos. Esto resalta

la importancia de probar otros enfoques de balanceo más avanzadas o métodos híbridos que permitan mantener la información crítica para la detección de fraudes.

Otro punto relevante es que el algoritmo de clustering utilizado, DBSCAN, no ha dado buenos resultados, lo que sugiere que se necesitan enfoques más adecuados para segmentar los datos y facilitar su análisis. Explorar algoritmos no globulares más sofisticados podría ser una vía para mejorar la representación de los datos antes de entrenar los modelos.

Por último, un aspecto que ha limitado el estudio es la capacidad computacional disponible. Hasta ahora, solo se ha podido utilizar uno de los seis datasets del BAF (Jesus et al., s. f.), cada uno con un millón de transacciones. Esto implica que el modelo ha sido entrenado con una porción reducida de la información total disponible, lo que podría afectar su capacidad de generalización. Contar con mayor potencia computacional permitiría entrenar los modelos con los seis datasets, ampliando el rango de estudio y mejorando la robustez de las conclusiones.

6.1. Evaluación del cumplimiento de los objetivos

Durante el desarrollo del proyecto, se han alcanzado todos los objetivos planteados, demostrando la eficacia de las técnicas implementadas.

- Se ha identificado y seleccionado un dataset adecuado de transacciones bancarias con etiquetas de clase para instancias legítimas y fraudulentas. Este dataset es el *BAF Dataset Suite* (Jesus et al., s. f.) presentado en la conferencia NeurIPS 2022 y se centra en la evaluación de métodos de aprendizaje automático en el contexto de fraude bancario y otros escenarios relacionados con actividad anómala.
- Se ha realizado un análisis exhaustivo de las variables, detectando y eliminando aquellas que presentaban redundancia debido a multicolinealidad o nula varianza. En concreto, solo hubo una variable redundante, *device_fraud_count*, que solo presentaba un valor en todas las instancias. Además, del entrenamiento se eliminó la variable *month*, pues fue la utilizada para separar los conjuntos de entrenamiento, validación y test tal y como se recomienda en el *paper* de Jesus et al. (s.f.).
- El dataset se transformó adecuadamente, realizando las modificaciones necesarias para optimizarlo y alimentarlo de manera efectiva en las redes neuronales profundas mediante:
 - Aumento de valores negativos a cero de todos los encontrados en el dataset ya que estos representaban valores faltantes.
 - Uso de *OneHotEncoder* para transformar las variables categóricas en binarias y así poder usar algoritmos numéricos.
 - Escalado de datos con *MinMaxScaler* para el uso de algoritmos de clusterización así como mejorar el rendimiento de las redes neuronales

- Transformación de las instancias vectoriales a matrices de tres canales representando pseudo-imágenes para poder entrenar una CNN.
- Aunque se exploró la clusterización utilizando el algoritmo DBSCAN, no se logró obtener clusters significativos, solo un único cluster con algo de ruido. Esto indicó que la clusterización no aportó información relevante adicional para la detección de fraudes.
- Se implementaron modelos de deep learning que cumplieron con los requisitos establecidos, logrando una tasa de detección de fraudes superior al 80%, sin comprometer la exactitud global del sistema. La exactitud y la métrica AUC-ROC fueron superiores a 0,8, alcanzando el equilibrio deseado entre la detección de fraudes y la precisión general.
- Se diseñó una red neuronal profunda (DNN) que cumplió con la tarea de clasificación binaria de transacciones fraudulentas y legítimas.
- También se diseñó una red neuronal convolucional (CNN) que utilizó imágenes sintéticas creadas a partir de vectores unidimensionales, lo cual permitió mejorar la capacidad del modelo para detectar fraudes.
- Finalmente, se combinó el modelo DNN con el CNN, lo que permitió aprovechar las fortalezas de ambas arquitecturas.
- Despues de evaluar los resultados, se verificó que el modelo combinado de DNN y CNN cumplió con los criterios de éxito, alcanzando los valores deseados de AUC-ROC y sensibilidad. Se seleccionó este modelo como el más robusto debido a su capacidad para adaptarse a variaciones de datos externos y su rendimiento general, cumpliendo con los objetivos del proyecto.

Para completar este apartado de una forma más exhaustiva, de acuerdo con los resultados obtenidos y considerando los criterios de éxito establecidos —incluyendo un AUC-ROC, una sensibilidad (*recall*) y una exactitud (*accuracy*) mayor al 80%—, los modelos que han cumplido con todos los requisitos son los siguientes:

- Modelo 6 (DNN con ajuste de pesos, sin balanceo, dropout 0,5)
 - AUC-ROC: 0,811
 - Recall: 82,4%
 - Accuracy: 80%
- Modelo 21 (Combinación CNN+DNN por promedio, ajuste de pesos, sin balanceo, dropout 0,5)
 - AUC-ROC: 0,813
 - Recall: 80%
 - Accuracy: 82,4%

Estos modelos han logrado el mejor equilibrio entre detección de fraudes y reducción de falsos positivos, asegurando un rendimiento óptimo sin comprometer la estabilidad del sistema. En particular, la combinación de CNN y DNN (Modelo 21) ha demostrado ser la opción más robusta, al maximizar el AUC-ROC y mejorar la exactitud general.

Por otro lado, aunque otros modelos han mostrado buenos valores en una o más métricas, no han cumplido todos los criterios simultáneamente. En especial, los modelos

con técnicas de balanceo como SMOTE y CTGAN no han logrado mejorar el rendimiento global, lo que sugiere que el uso de datos sintéticos no ha sido completamente beneficioso en este caso.

Así, se logró cumplir exitosamente con todos los objetivos, contribuyendo de manera efectiva al avance de la ciencia de datos y al uso de técnicas de deep learning para la detección de fraudes bancarios, con el fin de prevenir delitos en el ámbito financiero y promover la seguridad y el bienestar global.

6.2. Trabajos futuros

A partir de estos hallazgos, se proponen las siguientes líneas de trabajo para futuras mejoras:

- Explorar valores de *dropout* superiores a 0,5 (por ejemplo, 0,6 o 0,7) para verificar si pueden mejorar aún más la capacidad de generalización del modelo.
- Probar con diferentes *learning rates*, ya que no se han explorado múltiples configuraciones y esto podría afectar la estabilidad del entrenamiento.
- Experimentar con algoritmos de clustering no globulares que puedan ofrecer una segmentación más efectiva de los datos y mejorar la detección de fraudes.
- Optimizar las estrategias de balanceo de datos, evitando la pérdida de información observada con CTGAN y probando técnicas alternativas como combinaciones de sobremuestreo y submuestreo.
- Ampliar los experimentos con más potencia computacional, lo que permitiría trabajar con los seis datasets del BAF y reducir los límites del estudio actual.
- Explorar arquitecturas híbridas más avanzadas basadas en la combinación de CNN y DNN, dado que han mostrado los mejores resultados en este estudio.
- Evaluar modelos con datos en tiempo real para analizar su rendimiento en condiciones más cercanas a un entorno de producción.

En conclusión, este estudio ha permitido identificar modelos prometedores para la detección de fraude bancario, destacando el impacto del *dropout*, la importancia de combinar arquitecturas y la necesidad de estrategias más efectivas de preprocesamiento. Siguiendo estas líneas de trabajo, se podrán lograr mejoras significativas en la detección de fraudes, reduciendo falsos positivos y mejorando la eficiencia de los modelos.

7. Referencias

Asamblea General de las Naciones Unidas. (2019). Resolución 74/247. Prevención y lucha contra los delitos cometidos en el ciberespacio.

Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos).

Theobald, M. (2022). Informe sobre el Estado Global del Fraude y la Identidad. LexisNexis Risk Solutions. <https://risk.lexisnexis.com/global/es/about-us/press-room/press-release/20221205-global-state-of-fraud-and-identity-report-reveals?form=MG0AV3>

A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, “Deep learning detecting fraud in credit card transactions,” in Proc. Syst. Inf. Eng. Design Symp. (SIEDS), Apr. 2018, pp. 129–134.

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Campbell, J. P. (2020). Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science & Technology*, 9(2), 14. <https://doi.org/10.1167/tvst.9.2.14>

GitHub · Build and ship software on a single, collaborative platform. (2025). GitHub. <https://github.com/>

imbalanced-learn documentation—Version 0.13.0. (s. f.). Recuperado 12 de febrero de 2025, de <https://imbalanced-learn.org/stable/>
Jesus, S., Pombal, J., Alves, D., Cruz, A. F., Saleiro, P., Ribeiro, R. P., Gama, J., & Bizarro, P. (s. f.). *BAF Dataset Suite Datasheet*.

Kaur, H., Pannu, H. S., & Malhi, A. K. (2020). A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions. *ACM Computing Surveys*, 52(4), 1-36. <https://doi.org/10.1145/3343440>

Keras: Deep Learning for humans. (s. f.). Recuperado 12 de febrero de 2025, de

<https://keras.io/>

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

<https://doi.org/10.1038/nature14539>

López, V., Fernández, A., García, S., Palade, V., & Herrera, F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250, 113-141.
<https://doi.org/10.1016/j.ins.2013.07.007>

Matplotlib—Visualization with Python. (s. f.). Recuperado 12 de febrero de 2025, de

<https://matplotlib.org/>

pandas—Python Data Analysis Library. (s. f.). Recuperado 12 de febrero de 2025, de

<https://pandas.pydata.org/>

Scikit-learn: Machine learning in Python—Scikit-learn 1.6.1 documentation. (s. f.).

Recuperado 12 de febrero de 2025, de <https://scikit-learn.org/stable/>

Sdv-dev/CTGAN. (2025). [Python]. The Synthetic Data Vault Project.

<https://github.com/sdv-dev/CTGAN> (Obra original publicada en 2019)

Visual Studio Code—Code Editing. Redefined. (s. f.). Recuperado 12 de febrero de

2025, de <https://code.visualstudio.com/>

Waskom, M. (2021). seaborn: Statistical data visualization. *Journal of Open Source Software*, 6(60), 3021. <https://doi.org/10.21105/joss.03021>

Wei, W., Li, J., Cao, L., Ou, Y., & Chen, J. (2013). Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, 16(4), 449-475. <https://doi.org/10.1007/s11280-012-0178-0>

Welcome to Python.org. (2025, febrero 11). Python.Org. <https://www.python.org/>

Zabala, J. A., Alchundia, I. M., & Seraquive, G. G. (2022). Revisión de literatura sobre las técnicas de Machine Learning en la detección de fraudes bancarios.



Sapienza: International Journal of Interdisciplinary Studies, 3(1), Article 1.

<https://doi.org/10.51798/sijis.v3i1.257>

Adrián Colomer Granero y Gabriel Enrique Muñoz Ríos (s. f.). Universidad Internacional de Valencia. *Redes Neuronales y Aprendizaje Profundo*.



Apéndice I

Enlace al repositorio del proyecto con el código en GitHub

<https://github.com/CrstnGB/deteccionFraudeBancario/tree/main>