BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE MEDICINA

LICENCIATURA EN MEDICINA

Desarrollo de un algoritmo de Machine Learning para identificar y clasificar hemorragias intracraneales con tomografías en el Hospital de Especialidades del CMN La Raza

Tesis presentada para obtener el grado de:
Licenciatura en Medicina

Director experto
Diego Escarramán Martínez

Candidato a titulación
Edwin Zoquiapa Galaviz

Director metodológico
Juan Carlos Pérez García

Heroica Puebla de Zaragoza, octubre 2025

ÍNDICE

1. RESUMEN

Introducción: La hemorragia intracraneal se considera una urgencia médica con una alta mortalidad y morbilidad neurológica, la detección temprana mediante tomografía computarizada es esencial, sin embargo, la disponibilidad del personal capacitado para su adecuada interpretación es limitada. La implementación de algoritmos de aprendizaje automático, en particular, redes neuronales convolucionales, se plantea como una herramienta para el apoyo diagnóstico de los médicos no entrenados o con escasa experiencia en la interpretación de imágenes.

Objetivo: Desarrollar y validar un algoritmo de machine learning basado en redes neuronales convolucionales para identificar y clasificar hemorragias intracraneales con un nivel del 95% de confianza y un margen de error máximo estimado del 5%, en imágenes de tomografía computarizada de pacientes atendidos en el Hospital de Especialidades “Dr. Antonio Fraga Mouret” del Centro Médico Nacional “La Raza”.

Material y métodos: Para el presente estudio se utilizaron imágenes provenientes del archivo clínico del Hospital de Especialidades del Centro Médico Nacional La Raza. Las imágenes se procesaron, etiquetaron y dividieron en conjuntos de entrenamiento y validación para desarrollar múltiples modelos de red neuronal convolucional, de cada modelo se evaluó su desempeño con métricas de precisión, sensibilidad, especificidad y área bajo la curva.

Resultados: El modelo con mejor rendimiento alcanzó una sensibilidad del 74%, especificidad de 44% y AUC: 0.59, aunque los valores no son óptimos para el uso clínico de forma autónoma, demuestran capacidad de detección por encima del azar y potencial para mejora, empleando un mayor volumen de datos con mejor calidad.

Conclusión: El modelo desarrollado representa una posible herramienta de apoyo diagnóstico que podría aplicarse en entornos con alta demanda, como sistema de triaje automatizado, sentando las bases para la implementación de modelos de aprendizaje automático en el flujo de trabajo y la toma de decisiones sin sustituir el criterio médico.

Palabras clave: hemorragia intracraneal, tomografía computarizada, aprendizaje automático, redes neuronales convolucionales, inteligencia artificial.

ABSTRACT

Background: Intracranial hemorrhage is a neurological emergency with high risk of death and disability. In daily practice, early recognition on CT scans is essential, but access to expert interpretation can be limited. Artificial intelligence (AI), especially convolutional neural networks (CNNs), is being explored as a support tool for diagnosis.

Objective: To design and test a CNN model able to identify intracranial hemorrhage in CT images of patients from the Specialties Hospital "Dr. Antonio Fraga Mouret", National Medical Center "La Raza".

Materials and Methods: A dataset of head CT slices from the institutional archive was processed and divided into training and validation groups. Several CNN architectures were evaluated using accuracy, sensitivity, specificity, and the area under the ROC curve.

Results: The best-performing model reached 74% sensitivity, 44% specificity, and an AUC of 0.59. Although these values are below the threshold for clinical use, they suggest detection above chance and the possibility of improvement with larger datasets.

Conclusion: The proposed model could be considered as a triage or support tool in high-demand environments. It represents a step toward future use of AI-based models in clinical workflows, always as a complement and not a replacement for medical judgment.

Keywords: intracranial hemorrhage; CT imaging; machine learning; convolutional neural networks; artificial intelligence.

2. INTRODUCCIÓN

La hemorragia intracraneal ya sea de origen traumático o espontáneo es una urgencia médica que posee una elevada mortalidad con secuelas neurológicas importantes, la detección oportuna y la clasificación precisa del tipo de hemorragia representan factores tanto para el pronóstico clínico como para la planificación terapéutica y la toma de decisiones a nivel quirúrgico, la tomografía computarizada sin contraste se mantiene como una herramienta diagnóstica de referencia por su rapidez, disponibilidad y la sensibilidad para detectar sangrados agudos, pero el proceso de interpretación de las imágenes es susceptible al error humano, la variabilidad interobservador y los retrasos que derivan de la sobrecarga laboral en los servicios de radiología de alta demanda, frente a este panorama, las tecnologías de aprendizaje automático aplicadas al análisis de las imágenes de tomografía de emergencia representan una solución viable, potencialmente precisa y escalable para la identificación y clasificación automatizada de las hemorragias intracraneales. (1)

El aprendizaje automático es un campo de la inteligencia artificial que permite que los sistemas informáticos aprendan a partir de datos sin necesidad de programarse de forma explícita como lo haría un software tradicional, este principio es útil en las tareas que implican la detección de patrones complejos y sutiles en las imágenes médicas, en la neuroimagen los algoritmos de aprendizaje automático se han desarrollado y entrenado para reconocer signos radiológicos de hemorragia intracraneal, segmentar las zonas afectadas, cuantificar el volumen del hematoma y clasificar el tipo de acuerdo a la ubicación siendo epidural, subdural, intraparenquimatoso, intraventricular o subaracnoidea mostrando un desempeño comparable a los observados por médicos especialistas en neuroimagen, dentro de los diversos modelos con algoritmos que se suelen emplear se destacan las redes neuronales convolucionales ya que las arquitecturas de estas redes han demostrado una alta efectividad para segmentar de manera automatizada en las regiones hiperdensas correspondientes a un sangrado intracraneal, las redes neuronales convolucionales son capaces de realizar detección voxel a voxel generando máscaras de segmentación precisas y clasificando patrones morfológicos en las imágenes con base en las características radiológicas que han aprendido durante el entrenamiento supervisado. (1)

3. ANTECEDENTES

3.1 Antecedentes específicos

Un aspecto importante en el desarrollo y la validación de este tipo de modelos es la calidad del conjunto de los datos que se utilizan ya que la efectividad del algoritmo depende directamente de la cantidad, diversidad, y la anotación precisa de las imágenes de entrenamiento, las bases de datos para este tipo de modelos suelen contener decenas de miles de estudios de tomografía con anotaciones realizadas por radiólogos expertos y presentando distintos balances entre estudios normales y con patología lo que puede llegar a afectar directamente la capacidad de un modelo para generalizar sin caer en sobreajuste, no sólo esto los problemas como el bias de espectro o la presencia de variables de confusión como las marcas laterales que suelen ser visibles en las imágenes con patología pueden llegar a deteriorar la capacidad del algoritmo para mantener un rendimiento constante en poblaciones diferentes a la cual se desarrolla el algoritmo. De igual forma el grosor de corte de las imágenes siendo éste de 0.5 a 5 mm no ha demostrado ser una barrera para el desarrollo de este tipo de algoritmos siempre que se mantenga una coherencia estructural entre la segmentación de los volúmenes, la segmentación automatizada no sólo mejora la velocidad diagnóstica sino que además permite cuantificar con precisión los parámetros como el volumen total de la hemorragia, su intensidad en unidades Hounsfield, la localización anatómica y la relación con las estructuras como el sistema ventricular o el mesencéfalo, el integrar los algoritmos de aprendizaje automático en el flujo de trabajo hospitalario permite reducir el tiempo de respuesta diagnóstica en los estudios positivos para hemorragia intracraneal, con disminuciones de hasta el 89.6% en pacientes ambulatorios y 34.7% en pacientes hospitalizados, funcionando estos algoritmos como herramientas de triaje automatizado reorganizando una lista de lecturas del radiólogo en función de la prioridad determinada por la detección de los signos hemorrágicos. (1)

En cuanto al rendimiento de diagnóstico estos algoritmos han demostrado sensibilidades y especificidades superiores al 90% con un área bajo la curva (AUC) cercana al 0.99 lo que los posiciona como una herramienta confiable para la detección primaria, algunos estudios evidencian que el algoritmo puede incluso superar a radiólogos con varios años de experiencia en particular con las condiciones de carga de trabajo elevada o con imágenes que suelen ser desafiantes para la interpretación, una ventaja de estos modelos es

la capacidad de brindar resultados explicables y visualmente capaces de interpretarse a través de las interfaces gráficas, los algoritmos muestran no sólo la localización y el volumen del sangrado sino también la probabilidad estimada de hemorragia, el número de cortes afectados y el tipo de sangrado que se identificó, esto permite aportar transparencia a la información radiológica y en la toma de decisiones. (1)

La hemorragia es una de las complicaciones intraoperatorias con mayor relevancia en términos de morbilidad, mortalidad y deterioro clínico, el impacto que ejerce es transversal a las diversas especialidades representando un riesgo latente incluso en los procedimientos considerados de bajo perfil, dentro de las consecuencias más frecuentes se encuentra el aumento de la duración de la intervención y la necesidad de transfusiones lo que incrementa a su vez la probabilidad de complicaciones cardiovasculares, isquémicas y renales. El desarrollo de herramientas predictivas que permitan anticipar la probabilidad de sangrado se convierte en una prioridad que puede mejorar la preparación quirúrgica y reducir la exposición a eventos adversos. En el estudio realizado por Shi et al. se utilizaron técnicas de aprendizaje automático o machine learning, con una base de datos de 48543 pacientes de los cuales 9728 presentaron una hemorragia durante el procedimiento, esta proporción revela una frecuencia alta del evento, cuya detección anticipada permitiría estratificar el riesgo con precisión, a diferencia de los estudios que emplean modelos estadísticos lineales su estudio abordó directamente las limitaciones del sobreajuste, la multicolinealidad y la no linealidad de los datos aplicando modelos robustos de machine learning. Se evaluaron siete modelos distintos Light Gradient Boosting, Extreme Gradient Boosting, CatBoost, AdaBoost, regresión logística, redes neuronales multicapa y redes neuronales recurrentes de memoria a largo plazo. Entre los modelos aplicados el Light Gradient Boosting se destacó como el más eficiente alcanzando un área bajo la curva de 0.933, sensibilidad del 87%, especificidad de 85% y una precisión del 87%, su capacidad discriminativa sugirió que el modelo es apto para identificar pacientes con un alto riesgo de hemorragia incluso dentro de las poblaciones quirúrgicas heterogéneas, uno de los aspectos relevantes de este enfoque es la incorporación del método Shapley Additive Explanations (SHAP) que permite interpretar de forma individual la influencia de cada variable sobre la predicción realizada por el modelo mostrando que el tiempo quirúrgico, los niveles de dímero D y la edad fueron los tres predictores con mayor significancia, el tiempo operatorio en particular muestra una relación

directamente proporcional con el riesgo de sangrado con un umbral alrededor de los 80 minutos, el dímero D es reconocido por el rol en la activación fibrinolítica y su asociación con los eventos traumáticos, en este estudio se ha presentado como un marcador predictivo relacionado con el sangrado cuando sus niveles suelen ser menores. Este modelo no sólo demostró ser el más preciso sino además el más eficiente desde el punto de vista computacional ya que su estructura que se basa en árboles de decisión optimizando tanto la velocidad de entrenamiento como el uso de memoria lo que lo hace adecuado para entornos hospitalarios donde se requiere un análisis en tiempo real, su capacidad para transformar automáticamente las características de entrada y manejar los valores perdidos sin requerir una imputación adicional permitió generar métricas por variable convirtiéndolo en una herramienta ideal para aplicaciones clínicas. (2)

El desarrollo de modelos para el diagnóstico médico en patologías como las hemorragias intracraneales requiere una comprensión precisa de los fundamentos que sustentan el aprendizaje supervisado, la clasificación como una técnica central del aprendizaje automatizado es el enfoque apropiado para las tareas donde se requiere asignar a categorías discretas. El aprendizaje supervisado implica el empleo de conjuntos de datos etiquetados en los que una variable objetivo se relaciona con múltiples variables independientes como las características radiológicas que se extraen de la imagen de tomografía computarizada, las técnicas de clasificación buscan establecer funciones discriminantes capaces de predecir con precisión la pertenencia de nuevas imágenes u observaciones a una clase especial, estas funciones se construyen a partir del entrenamiento con ejemplos previos permitiendo al modelo aprender patrones representativos incluso en presencia de datos incompletos o con ruido lo que resulta útil en estudios médicos donde existe amplia heterogeneidad. (3)

El árbol de decisión se destaca por la capacidad para comprender las decisiones del modelo por parte del personal médico, estos algoritmos construyen una estructura jerárquica que se basa en el principio de ganancia de información dividiendo recursivamente el conjunto de datos hasta lograr nodos homogéneos en términos de la variable de clase, las redes bayesianas ofrecen un enfoque más probabilístico que modela las dependencias entre las variables mediante grafos dirigidos acíclicos, esta estructura es ideal para incorporar

incertidumbre al diagnóstico médico como la presencia de datos faltantes o la ambigüedad de la interpretación radiológica, la posibilidad de estimar las distribuciones condicionales y realizar inferencias bayesianas permite no sólo clasificar la imagen o la observación sino también predecir la evolución clínica del paciente a partir de variables como edad, antecedentes patológicos y la inclusión del núcleo gaussiano en la estimación de las densidades mejora el rendimiento predictivo del modelo ante las variables como el volumen del hematoma o el índice de densidad en unidades Hounsfield. Otro algoritmo ampliamente utilizado es el k-nearest neighbors que se basa en la comparación de similitud entre la observación y sus vecinos más cercanos en el espacio de características, esta técnica resulta eficaz en los escenarios donde las clases no se separan linealmente y la frontera de decisión es compleja, en el caso de las hemorragias intracraneales se puede utilizar para comparar mapas de densidad o morfología de la hemorragia asignando la clase más frecuente entre los vecinos cercanos, la sencillez y la capacidad de este algoritmo para adaptarse a nuevos datos lo hacen una opción viable especialmente si se implementan mejoras como la reducción de espacio con índices estructurados tipo R-tree llegando a disminuir los requerimientos de software con bases de datos amplias. Las máquinas de soporte vectoriales son reconocidas por la capacidad de generalización en los espacios de alta dimensionalidad al representar observaciones médicas mediante vectores de textura, intensidad y forma. La máquina de soporte vectorial encuentra el hiperplano óptimo para maximizar la separación entre las clases permitiendo clasificar con alta precisión incluso con distribuciones complejas, este enfoque es útil para distinguir entre esos tipos de hemorragia con patrones radiológicos similares como lo suele ser la subaracnoidea y la intraparenquimatosa. (3)

Se debe destacar que cada técnica presenta ventajas y limitaciones ya que mientras los árboles de decisión son interpretables pero susceptibles al sobreajuste, las máquinas de soporte vectoriales son potentes pero requieren una calibración cuidadosa de los parámetros; las redes bayesianas en cambio ofrecen flexibilidad y tolerancia a datos faltantes pero pueden ser computacionalmente exigentes y el k-nearest neighbors es intuitivo pero menos escalable. (3)

El concepto de radiómica permite extraer de forma automatizada y reproducible una gran cantidad de características de las imágenes médicas superando la apreciación visual

subjetiva del médico radiólogo, estas características incluyen las propiedades estadísticas de primer orden, texturas, matrices de coocurrencia de niveles de gris y transformaciones multiescalares mediante la descomposición wavelet. En el estudio realizado por Lyu y colaboradores se extrajeron un total de 1702 características radiológicas de imágenes de tomografía craneales de pacientes con hemorragia intracraneal utilizando dos métodos de segmentación: el volumen completo del hematoma y un recorte de tres capas axiales centradas en la región de mayor extensión de la lesión, este enfoque permitió capturar la variabilidad espacial tridimensional como las características locales más representativas del hematoma, posteriormente se aplicó un proceso de selección de variables en dos etapas primero utilizando el método univariado SelectKBest y luego aplicando regresión logística con penalización LASSO para reducir la dimensionalidad y evitar el sobreajuste del modelo, finalmente se entrenó un clasificador mediante máquinas de soporte vectorial que se validó con la estrategia de validación cruzada de diez pliegues, mostrando resultados de que el modelo radiómico supera ampliamente la capacidad diagnóstica de los radiólogos humanos en la discriminación entre hemorragia intracraneal primaria y secundaria con un AUC de 0.90 en el conjunto de prueba para el volumen completo del hematoma y AUC: 0.81 para la estrategia de tres capas, en cambio las evaluaciones realizadas por médicos radiólogos presentaron un AUC entre 0.66 y 0.70. Dentro de las características radiológicas que fueron más relevantes se identificaron las matrices de tamaño de zona de niveles de gris y de dependencia de niveles de gris, el parámetro wavelet-LLL_GLSZM_ZoneEntropy demostró un mayor peso discriminativo en las imágenes por volumen de hematomas reflejando la heterogeneidad de las zonas grises dentro de la lesión por un indicador de la complejidad morfológica y texturas del hematoma, en cambio la segmentación por tres capas capturó dependencias de bajos niveles de gris en áreas extensas posiblemente asociados a las diferencias microestructurales entre los sangrados de etiología neoplásica, vascular o inflamatoria frente a los de origen hipertensivo, este estudio permitió evidenciar la superioridad del enfoque automatizado frente al diagnóstico subjetivo que suele ser convencional y el más aceptado. (4)

El uso de la tomografía computarizada como una herramienta diagnóstica primaria permite la detección temprana de hemorragias intracraneales y sus complicaciones, la presencia de variables como hemorragia intraventricular, hemorragia intracerebral,

hematoma subdural, desviación de la línea media y el grosor de la sangre subaracnoidea juegan un rol central en la evaluación inicial, además de esto las características clínicas como la edad del paciente, el estado general y la puntuación en la escala de coma de Glasgow ofrecen información sobre la reserva neurológica y la probabilidad de un deterioro progresivo, la combinación de estas variables en modelos computacionales supervisados permite un análisis multidimensional que permite aplicar nuevas técnicas para su evaluación. Las técnicas de aprendizaje automático como redes neuronales multicapa, máquinas de soporte vectorial y modelos de regresión generalizada pueden ser aplicadas a los datos clínicos y radiográficos disponibles en un centro hospitalario, el desempeño predictivo de cada una de estas técnicas se puede comparar y no necesariamente son superiores a las escalas clínicas ya consolidadas. En todos los modelos analizados la escala de coma de Glasgow y la edad fueron consistentes con los predictores más relevan seguidos como el índice de masa corporal y la presencia de hemorragia intracraneal, no obstante el empleo de machine learning permite asignar los pesos diferenciados a cada variable mediante la interpretabilidad basada en SHAP values permitiendo ofrecer un panorama detallado del impacto de cada factor en la predicción del desenlace funcional; la severidad de la hemorragia, el compromiso de estructuras cerebrales y el desarrollo de complicaciones como el vasoespasmo, infarto cerebral o hidrocefalia condicionan el pronóstico del paciente estos fenómenos suelen ser sutiles o dinámicos y no siempre se reflejan adecuadamente por escalas clínicas, los modelos de aprendizaje automático podrían captar las interacciones no lineales entre las variables que los métodos estadísticos tradicionales no suelen detectar en particular cuando el espectro de variables suele ser amplio. (5)

La hemorragia intracraneal es una de las principales emergencias neurológicas en el entorno hospitalario que se caracteriza por una alta tasa de mortalidad y discapacidad, su aparición abrupta en ausencia de traumatismo, plantea un desafío diagnóstico y terapéutico que exige rapidez precisión y coordinación multidisciplinaria, la tomografía computarizada suele ser el pilar diagnóstico inicial dado que permite una evaluación rápida y confiable además de disponible para confirmar el diagnóstico, localizar la hemorragia y estimar su volumen. La fisiopatología de la hemorragia intracraneal se basa en la ruptura de vasos a nivel del encéfalo que se encuentran previamente alterados por una hipertensión arterial mal controlada, angiopatía amiloide, malformaciones vasculares o trastornos de la coagulación,

esta ruptura provoca una extravasación sanguínea en el parénquima encefálico lo que a su vez desencadena un efecto de masa con el incremento de la presión intracraneal, desplazamiento de las estructuras en el encéfalo, edema vasogénico y eventual deterioro del nivel de conciencia. Estos cambios se reflejan en las imágenes de tomografía como áreas hiperdensas y bien delimitadas que varían de acuerdo con la localización, el tiempo de evolución y el compromiso anatómico. En el servicio de urgencias el tiempo para la interpretación de este tipo de estudios es limitado y en ocasiones puede verse comprometido por factores como la falta de médicos radiólogos, la presencia de múltiples estudios simultáneos y la variabilidad en la experiencia del personal médico. La implementación de un sistema automatizado que se base en algoritmos de aprendizaje automático podría ofrecer una solución eficiente del problema de disponibilidad diagnóstica inmediata. Durante la hemorragia intracraneal son importantes los hallazgos de imagen y la condición neurológica del paciente, la escala de coma de Glasgow, el reflejo pupilar, la presión arterial sistólica y la necesidad de una intubación forman parte del conjunto de variables que combinadas con los parámetros evidenciados en la imagen de tomografía permiten estratificar la gravedad del cuadro. La automatización de la lectura de las imágenes de tomografía con la integración de estos datos podría no sólo acelerar el diagnóstico además proporcionar una clasificación precisa del tipo de hemorragia, su volumen estimado y su impacto anatómico incluyendo una desviación de la línea media, la posible compresión de las cisternas basales o la presencia de herniación. La implementación de algoritmos de aprendizaje profundo entrenados con imágenes de tomografía permite detectar características visuales con una precisión superior al diagnóstico realizado por un médico radiólogo en condiciones de alta carga laboral, estos modelos y algoritmos se pueden diseñar para clasificar imágenes de tomografía en función de la presencia o ausencia de hemorragia y una vez confirmada su existencia determinar el tipo, la extensión y localización. La capacidad del modelo para segmentar automáticamente la zona hemorrágica puede ser útil para calcular el volumen y monitorizar la progresión en los estudios seriados o de evolución prolongada (6)

La microcirculación cerebral se ve compuesta por redes capilares que desempeñan el intercambio de oxígeno y nutrientes con el tejido neural, estas redes presentan una alta complejidad arquitectónica con múltiples bifurcaciones vasculares que influyen en la dinámica del flujo sanguíneo. La distribución del flujo y de los eritrocitos en estas redes no

es uniforme y puede variar en función de las múltiples factores hemodinámicos y estructurales, como suele ser el diámetro de los vasos, la viscosidad de la sangre, las interacciones celulares y célula-vaso. Esto llega a afectar directamente la perfusión cerebral y condicionar la extensión de un daño potencialmente existente. Los modelos teóricos tradicionales presentan a la red vascular como segmentos unidimensionales que presentan limitaciones importantes ya que no incorporan adecuadamente la curvatura vascular ni las interacciones dinámicas entre los eritrocitos y la pared vascular, estos modelos dependen de las relaciones empíricas para describir la viscosidad sanguínea o la partición de las células en las bifurcaciones lo cual restringe su capacidad predictiva ante las condiciones fisiopatológicas como las observadas en la hemorragia intracraneal. Los nuevos modelos computacionales con alta fidelidad han demostrado capacidad para simular con precisión la dinámica a nivel microvascular incluyendo deformaciones celulares, bifurcaciones jerárquicas y redes vasculares tridimensionales con morfología realista, sin embargo el alto costo que implica desarrollarlos los hace poco prácticos y asequibles para el análisis extenso o en tiempo real limitando su aplicación. El uso de algoritmos de machine learning surge entonces como una alternativa para superar las barreras computacionales y mejorar la predicción de variables hemodinámicas complejas, los modelos de machine learning se entrenan con datos de simulaciones numéricas de alta resolución que han mostrado ser capaces de predecir tanto la distribución del flujo como los de los eritrocitos en redes vasculares extensas y con estructuras diversas, dentro de los modelos más relevantes se encuentran las redes neuronales artificiales y las redes de memoria a largo plazo las cuales permiten abordar problemas de regresión, clasificación y predicción de series temporales, en el caso de las redes neuronales artificiales la estructura flexible que poseen les proporciona capacidad para manejar relaciones no lineales complejas haciéndolas idóneas para predecir la distribución promedio de flujo y hematocrito en cambio los modelos de memoria a largo plazo son adecuados para capturar las variaciones temporales de estos parámetros incluyendo oscilaciones rápidas inducidas por interacciones eritrocitarias o cambios en la geometría vascular. El proceso de entrenamiento de estos modelos parte de datos simulados obtenidos mediante simulaciones tridimensionales con resolución celular en las cuales se modela el flujo de eritrocitos deformables a través de redes vasculares, las redes neuronales desarrolladas con estos datos han logrado predecir con alta precisión la distribución de flujo

y las células en diversas condiciones hemodinámicas y arquitectónicas demostrando que pueden predecir el caudal sanguíneo en vasos hijos a partir de variables geométricas y del flujo del vaso madre en bifurcaciones con una precisión superior al 90% durante el entrenamiento y alrededor del 80% en las predicciones, de igual forma permitiendo reproducir la dinámica temporal del flujo y del hematocrito con una fidelidad alta capturando las características estadísticas como la media, desviación estándar, asimetría y curtosis lo que es esencial en la hemodinámica para reflejar eventos patológicos en evolución. (7)

A lo largo del tiempo con base en la identificación temprana de alguna patología y el pronóstico funcional se han ido desarrollando escalas pronósticas que posteriormente se han estandarizado, actualmente los métodos convencionales enfrentan limitaciones al manejar grandes volúmenes de variables interrelacionadas tanto clínicas como imagenológicas, frente a esta complejidad surgen los modelos de aprendizaje automático que se han consolidado como una herramienta para identificar patrones no lineales y sinergias entre variables que no son fácilmente perceptibles mediante análisis estadísticos tradicionales, de los más destacables el algoritmo de búsqueda aleatorio random forest ha demostrado un notable rendimiento en la estratificación pronóstica al combinar múltiples árboles de decisión que votan en conjunto para predecir un resultado. El desarrollo de este tipo de modelos requiere una selección rigurosa de las características predictivas que reflejen de manera integral el estado clínico y estructural del paciente en el momento de su ingreso, para este fin se han empleado algoritmos como Boruta que permiten identificar variables con un mayor poder discriminativo, dentro de las variables que suelen emplearse para construir este tipo de modelos existen diez variables clave: la puntuación en la escala de coma de Glasgow, índice de comorbilidades de Charlson, puntuación en la escala de hemorragia intracraneal, volumen del hematoma, estado pupilar, localización troncoencefálica y cerebelosa, edad, uso de anticoagulantes o antiplaquetarios y la presencia de hemorragia intraventricular, siendo seleccionadas frecuentemente estas variables por el peso en la predicción del estado funcional a los seis meses. El algoritmo empleado en el estudio de Trevisi y colaboradores muestra un alto rendimiento de discriminativo con un AUC: 0.96 para mortalidad, AUC: 0.89 para mal pronóstico funcional y AUC: 0.93 para un buen pronóstico, validándose internamente con un conjunto de prueba independiente, poniendo de manifiesto la capacidad del modelo para clasificar adecuadamente a los pacientes en función de su evolución clínica y además su

potencial utilidad en los entornos médicos para poder guiar decisiones clínicas en tiempo real. (8)

En contraste con las puntuaciones como BAT, BRAIN o los nueve puntos, los modelos basados en machine learning aprovechan múltiples variables clínicas e imagenológicas para construir predictores robustos y generalizables, dentro de los algoritmos utilizados se destacan los k-nearest neighbors, regresión logística, máquinas de soporte vectoriales, bosques aleatorios y el método de gradiente potenciado XGBoost, cada uno de estos modelos posee una arquitectura matemática distinta permitiendo detectar patrones no lineales y relaciones complejas entre las variables de entrada lo que se traduce en un mejor desempeño comparativo respecto a los convencionales, en el estudio de Tanioka et al. se utilizaron datos multicéntricos de imágenes de tomografía computarizada obtenidas al ingreso y a las 30 horas provenientes de distintos equipos y protocolos técnicos esto con el objetivo de generar un modelo ampliamente aplicable, la inclusión de variables como la edad, uso de anticoagulantes o antiagregantes plaquetarios, puntuación en la escala de coma de Glasgow, cifras tensionales, biomarcadores séricos y variables imagenológicas como el volumen inicial del hematoma, hipodensidades, morfología irregular, presencia de extensión ventricular o signos tomográficos específicos como el “blend sign” o “spot sign”, permitió crear un modelo predictivo de alta sensibilidad con un diseño de k-nearest neighbors alcanzando un AUC: 0.79 superando el BRAIN score con un AUC: 0.67, este modelo se desempeñó bajo la selección rigurosa de características predictivas basadas en análisis univariados priorizando aquellas con una mayor significancia estadística entre los grupos con y sin expansión del hematoma, las variables como el volumen basal del hematoma, hipodensidades, tiempos desde el inicio del primer estudio, uso de anticoagulantes y la forma irregular del hematoma permitieron demostrar una alta capacidad discriminativa reflejando el comportamiento dinámico del sangrado cerebral, este modelo también incorporó procedimientos técnicos como la normalización de las variables, sobremuestreo aleatorio para corregir el desbalance de clases y la validación cruzada estratificada de 30 pliegues con ajustes manuales de hiperparámetros en cada algoritmo garantizando la robustez, estabilidad y la generalización del modelo evitando así el sobreajuste y optimizando la precisión diagnóstica de nuevos conjuntos de datos, la validación externa se realizó con una cohorte independiente confirmando el buen desempeño del modelo en términos de sensibilidad 84%,

especificidad 73%, y precisión total de 77%. Un aspecto en la construcción de los modelos predictivos es la posibilidad de la interpretación y la aplicación práctica ya que si bien los modelos más complejos como el XGBoost presentan un alto rendimiento, su naturaleza de caja negra puede dificultar su adopción en entornos hospitalarios reales en cambio con un modelo k-nearest neighbors ofrece una mecánica más intuitiva que se basa en la similitud con casos previos facilitando así la comprensión por parte del personal médico. (9)

La utilidad clínica de cada modelo desarrollado radica en la capacidad para identificar de forma temprana a los pacientes con alto riesgo de expansión de hematoma o la identificación adecuada de estos mismos permitiendo implementar estrategias más agresivas para el manejo y el control intensivo de la presión arterial o la reversión de anticoagulación previo a un evento, en casos donde se ha evidenciado ya un proceso a través de imagen la aplicación de los diversos modelos permite no solo adoptar las matemáticas aplicadas a la medicina sino además la optimización en la toma de decisiones que aseguren un mejor pronóstico posterior al evento.

3.2 Antecedentes generales

La hemorragia intracraneal (HIC) es un tipo de accidente cerebrovascular producto de la extravasación de sangre en el parénquima cerebral. La HIC corresponde entre el 10 y el 15% de todos los ictus e incluso es una de las formas más mortales, con una mortalidad temprana en los primeros 30 días que oscila entre el 30 y el 40% (10). La incidencia es diferente entre los diversos grupos de edad, el sexo y el origen étnico, con tasas más altas en poblaciones de origen asiático o afrodescendiente que en quienes tienen ascendencia europea. (11)

Desde los años 80 se ha generalizado el uso de la tomografía computarizada para permitir realizar estudios poblacionales precisos sobre la hemorragia intracraneal superando la sensibilidad de los sistemas clínicos tradicionales para diferenciar entre los distintos tipos de ictus, a pesar de los avances en neuroimagen los datos revelan que la incidencia global de hemorragia intracraneal se ha mantenido relativamente constante con una media de 24.6 casos por 100000 personas al año sin evidenciar una reducción sustancial en el periodo de 1980 a 2008, este estancamiento se contrasta con la notable disminución en la incidencia de infartos cerebrales requiriendo estrategias que permitan ser preventivas y eficaces, en el análisis epidemiológico realizado por Asch y colaboradores mencionan que la incidencia de hemorragia intracraneal se incrementa con la edad alcanzando un ratio de incidencia de 96 veces mayor en mayores de 85 años estos respecto al grupo de referencia de 45 a 54 años, la influencia de los factores como la hipertensión arterial, la angiopatía amiloide cerebral y el uso de anticoagulantes en adultos mayores puede explicar esta tendencia ascendente, este patrón demográfico junto con la letalidad a un mes que se aproxima al 40.4% plantea la urgencia de métodos para una detección temprana y clasificación precisa del tipo de hemorragia, los cuales pueden ser asistidos por algoritmos de machine learning entrenados con imágenes de tomografía computarizada, en cuanto a la mortalidad indicaron que la tasa de fatalidad no ha mejorado a través del tiempo y aunque existen diferencias regionales como una letalidad observada en Japón de 16.7% estas pueden atribuirse más a las variaciones en los métodos de captura de los casos o en las estrategias terapéuticas que a mejoras estructurales de la atención, los factores como el retraso en el diagnóstico por imagen o la falta de acceso oportuno a las unidades especializadas para atender este tipo de eventos podrían perpetuar esta elevada mortalidad. Los datos disponibles suelen ser limitados pero

indican que sólo entre el 12% y el 39% de los sobrevivientes logran una independencia funcional a mediano o largo plazo, esto dependiendo de la edad y la severidad del evento, reforzando la necesidad de mejorar la detección además de una clasificación precisa y la predicción del pronóstico, las cuales son áreas en las que el aprendizaje automático puede tener un impacto al integrarse a la cadena de algoritmos de actuación clínica. La diversidad en la presentación clínica de la hemorragia intracraneal al igual que con su etiología y localización representa un desafío para los sistemas tradicionales de diagnóstico ya que el uso de tomografía computarizada como una primera herramienta de imagen en pacientes con sospecha de posible hemorragia proporciona datos cuantificables y reproducibles adecuados para poder procesarse mediante algoritmos de redes neuronales convolucionales. Estos modelos de aprendizaje profundo permiten extraer patrones de imágenes de tomografía computarizada incluso los más imperceptibles al ojo humano demostrando un potencial importante para las tareas de segmentación clasificación y predicción de los resultados. En su estudio Asch et al. mencionan la importancia de contar con bases de datos robustas, controladas y de alta calidad, que permita ser aplicado en la construcción de un conjunto de datos para el entrenamiento de algoritmos de aprendizaje automático garantizando que los modelos aprendan sobre datos verificados y representativos de la población objetivo, al igual que se debe mantener pendiente la heterogeneidad poblacional en los modelos predictivos, adicionalmente los algoritmos deben ser adaptativos y validados localmente en una primera instancia para poder ser reproducibles y posteriormente ser escalados incluyendo factores epidemiológicos, sociales y culturales que puedan brindar respuesta a la incidencia de este tipo de eventos. (11)

La hemorragia intracraneal es una de las formas más graves de accidente cerebrovascular que se caracteriza por la extravasación espontánea de sangre dentro del parénquima del encéfalo que puede ser por o en ausencia de traumatismo, su elevada letalidad, la escasa posibilidad de recuperación funcional y la limitada eficacia de las terapias que existen hacen que el diagnóstico requiera ser precoz y preciso siendo una prioridad absoluta, las guías de práctica clínica para el manejo de la hemorragia intracraneal establecen que la neuroimagen es fundamental para poder llevar a cabo su diagnóstico y por tanto es insustituible en el servicio de urgencias, la lectura manual de los estudios de tomografía se puede ver afectada por limitaciones operativas, experiencia del personal, disponibilidad y

volumen de trabajo por lo que la integración de los algoritmos de aprendizaje automático para el análisis de imágenes es una innovación técnica con una profunda repercusión a largo plazo. La tomografía computarizada permite confirmar el diagnóstico de hemorragia intracraneal ya que posee una alta sensibilidad, permitiendo localizar el sitio de sangrado, cuantificar su volumen, evaluar la desviación de la línea media y detectar la extensión interventricular, todos estos factores se vinculan a un pronóstico funcional y vital. La localización de las hemorragias suele ser más frecuente en los ganglios basales, tálamo, cerebelo y el tronco encefálico. Las hemorragias intracraneales lobares aunque son menos frecuentes se suelen asociar con una angiopatía amiloide mientras que las profundas se relacionan con hipertensión arterial, estas características de imagen junto con los parámetros clínicos como el nivel de conciencia, la edad del paciente, el uso de anticoagulantes y la presión arterial sistólica al ingreso forman parte de las escalas pronósticas que permiten orientar las decisiones terapéuticas y quirúrgicas. La posibilidad de automatizar la identificación y cuantificar de estos factores un resultado que permita optimizar la evaluación inicial del paciente al igual que reducir la variabilidad interobservador y acelerar la toma de decisiones se puede lograr a través de varios modelos de aprendizaje automático. De igual forma no sólo en la detección hemorragia intracraneal sino además en la detección de los eventos secundarios que pueden agravar el daño neurológico, incluyendo la formación de edema perilesional, disrupción de la barrera hematoencefálica, activación de la respuesta inflamatoria y el compromiso del flujo sanguíneo cerebral. El volumen inicial del hematoma y su expansión en las primeras horas son los factores pronósticos de mayor relevancia ya que la detección temprana de los signos de expansión activa como el “signo de blend” o la heterogeneidad del hematoma pueden ser detectadas por un radiólogo experimentado pero de igual forma pueden ser identificadas por modelos de visión computacional entrenados con redes neuronales convolucionales capaces de detectar patrones de densidad, forma, bordes y simetría con una sensibilidad superior a la humana en condiciones con una alta carga laboral. Las guías además destacan la importancia de clasificar a los pacientes desde el ingreso de acuerdo al riesgo de deterioro neurológico, utilizando escalas como la de Glasgow, el score del hematoma y el volumen calculado del sangrado, siendo este último frecuentemente utilizado y calculado mediante la fórmula del ABC/2, el cual puede ser calculado de forma automatizada mediante algoritmos de segmentación que se basen en aprendizaje profundo,

permitiendo mejorar la precisión y reducir los tiempos de análisis. De igual forma se menciona la necesidad de monitorizar de forma intensiva a los pacientes con hemorragia intracraneal dado el riesgo elevado de un deterioro súbito, las variables como el empeoramiento del nivel de conciencia, la aparición de herniación, el aumento de la presión intracraneal o la extensión del sangrado pueden anticiparse mediante un análisis comparativo de estudios secuenciales, permitiendo adaptar el algoritmo para realizar un seguimiento a longitudinal. (12)

En la actualidad el manejo de la hemorragia intracraneal junto con el diagnóstico temprano es solo un medio para iniciar las intervenciones médicas correspondientes sino además un determinante directo del pronóstico del paciente existen protocolos de intervención temprana como el INTERACT3, que integran una reducción intensiva de la presión arterial, reversión de la anticoagulación, control térmico y metabólico y una referencia a neurocirugía organizada en forma de "care bundle", este enfoque se ha probado en hospitales de baja, media y alta complejidad reduciendo la mortalidad y mejorando la funcionalidad de los pacientes a los 30 días siempre y cuando la implementación ocurra dentro de las primeras 6 horas del evento, el reconocimiento automatizado de hemorragias intracraneales puede actuar como un disparador de este bundle terapéutico permitiendo reducir el tiempo de latencia entre la adquisición de la imagen y la intervención clínica, adicionalmente el tipo de hemorragia tiene implicaciones terapéuticas directas. El creciente uso de anticoagulantes en particular los inhibidores del factor Xa, alteran el perfil de los pacientes con hemorragia intracraneal y la reversión con andexanet alfa ha demostrado una superioridad sobre el tratamiento convencional pero para ello se requiere una identificación rápida del tipo de anticoagulante utilizado y la cuantificación precisa del hematoma, tareas que pueden ser facilitadas por sistemas de algoritmos de aprendizaje automático capaces de integrar imagen y datos clínicos, los algoritmos propuestos actualmente buscan incorporar módulos de clasificación para discriminar entre hemorragia intracraneal asociada o no asociada, a anticoagulantes directos permitiendo flujos diferenciados de tratamiento en servicios de urgencia y cuidados intensivos. La evaluación precoz del hematoma mediante las técnicas mínimamente invasivas ha cobrado un renovado interés tras los resultados del ensayo ENRICH, el cual demostró que una craniectomía transulcal parafascicular asistida por navegación endoscópica mejora la recuperación funcional comparada con el tratamiento

conservador en pacientes con hemorragia intracraneal lobar. La selección de candidatos para este tipo de intervención se puede ver beneficiada por algoritmos que analicen automáticamente la localización, volumen, accesibilidad quirúrgica y además signos de herniación en la tomografía inicial, permitiendo una detección automatizada de estas variables se lograrían estandarizar los criterios de derivación a neurocirugía y evitar demoras en la toma de decisiones. La incorporación de datos longitudinales provenientes de estudios seriados de imagen es otra opción que permitiría desarrollar modelos predictivos para detectar la expansión del hematoma, progresión de edema o una respuesta intervenciones como la craniectomía descompresiva lo cual ha sido probado en el ensayo SWITCH. (12)

Los hallazgos realizados por Penckofer et al. mencionan que el diagnóstico y la intervención dependen del reconocimiento temprano del hematoma, la caracterización precisa del origen y la evaluación continua de su evolución, es por esto que la tomografía computarizada y la resonancia magnética juegan un papel en la etapa temprana como en el seguimiento longitudinal de cada caso, no obstante se requiere una interpretación rápida, confiable y estandarizada de estos estudios ya que suelen presentar un reto operativo en los hospitales de alta especialidad motivando a la búsqueda de soluciones automatizadas como los modelos de machine learning, estos algoritmos se pueden crear con la finalidad de analizar secuencias completas de imagen y aprender patrones específicos asociados a diferentes etiologías hemorrágicas, es el caso de una red neuronal convolucional que se puede entrenar para distinguir automáticamente entre un patrón sugestivo de hemorragia intracraneal primaria por hipertensión con una localización profunda o sin lesiones estructurales asociadas frente a una hemorragia intracraneal secundaria a un tumor o una malformación vascular de lesión nodular subyacente con refuerzo de contraste y morfología heterogénea, además al integrar las variables clínicas como la edad, antecedentes de hipertensión, uso de anticoagulantes el modelo puede generar clasificaciones útiles para la toma de decisiones incluyendo alertas de riesgo o sugerencias de diagnóstico diferencial. (13)

Dentro de los factores de riesgo más importantes se halla la hipertensión arterial, la cual es responsable para la mayoría de los casos de HIC primaria. Otros factores que pueden causar HIC son la angiopatía amiloide cerebral, las coagulopatías, el uso de anticoagulantes y las malformaciones vasculares. (13) También existe la necesidad de hacer un claro

diagnóstico entre HIC primaria e HIC secundaria, ya que esta última puede estar asociada a aneurismas, tumores cerebrales, traumatismos y trastornos de la coagulación.

Diagnóstico por Imagen en la Hemorragia Intracraneal

Los estudios recientes han demostrado que la incidencia de HIC no ha tenido un decremento significativo en las últimas décadas, a diferencia del ictus isquémico, lo que refuerza la necesidad de mejorar la prevención, el diagnóstico y el manejo de la HIC. El diagnóstico rápido y preciso de la hemorragia intracerebral (HIC) es extremadamente importante para el tratamiento y para el pronóstico del paciente. En este contexto la tomografía computarizada (TC) sin contraste sigue siendo el método de referencia por su buena sensibilidad para la detección de sangre en el parénquima cerebral y por la rapidez en la obtención de imágenes. Sin embargo, la TC tiene limitaciones en la diferenciación de la hemorragia intracerebral primaria con la secundaria. En este sentido, la resonancia magnética (RM) con secuencias de susceptibilidad magnética y eco-planar gradient echo ha mostrado ser útil para la detección de lesiones subyacentes como las malformaciones arteriovenosas o la angiopatía amiloide cerebral. El uso de biomarcadores radiológicos y la introducción de inteligencia artificial en la interpretación de las imágenes están revolucionando el diagnóstico de la HIC. El uso de ciertos patrones de neuroimagen ha demostrado que pueden predecir la expansión del hematoma lo que permitiría una mejor estratificación del riesgo en estos pacientes. (14)

Aplicación del Machine Learning en el Diagnóstico y Clasificación de la HIC

El Machine Learning (ML) ha surgido como una herramienta extremadamente efectiva en el campo de la radiología, proporcionando importantes beneficios en la detección y clasificación de la hemorragia intracerebral (HIC). Los modelos de aprendizaje profundo han demostrado una notable capacidad para identificar hemorragias, alcanzando altos niveles de sensibilidad y especificidad, lo que reduce la variabilidad entre diferentes observadores. (15) Diversos estudios han explorado la implementación de algoritmos de radiómica basados en ML para distinguir entre HIC primaria y secundaria, logrando una precisión que supera la de los radiólogos tradicionales. Estos modelos tienen la capacidad de analizar características de la imagen que no son evidentes a simple vista, lo que permite una clasificación más

objetiva de los diferentes subtipos de HIC. Por otra parte, el pronóstico de pacientes que presentan hemorragias subaracnoideas ha visto mejoras significativas gracias al uso de máquinas de soporte vectorial (SVM) y redes neuronales artificiales. Esto podría influir considerablemente en las decisiones terapéuticas que se tomen posteriormente.

Retos y Oportunidades en la Implementación de Machine Learning en Neuroimágenes

A pesar de los progresos en la aplicación de inteligencia artificial en imágenes médicas, se presentan varios obstáculos para su implementación en la práctica clínica. Uno de los principales desafíos es la escasez de bases de datos de alta calidad y correctamente anotadas que sean necesarias para el entrenamiento de los modelos. Además, la variabilidad en los protocolos de adquisición de imágenes y las diferencias entre las poblaciones de estudio pueden influir en la capacidad de generalización de los modelos de ML. Por otra parte, la interpretabilidad de los resultados producidos por estos algoritmos continúa siendo un desafío. De hecho, son muchos los modelos de Deep learning que funcionan como "cajas negras" y en los que no podemos determinar fácilmente qué factores determinan la predicción que ofrecen estos modelos. Esto también implica que resulta relevante trabajar la explicación y desarrollar modelos explicables, así como utilizar modelos híbridos que integren la inteligencia artificial y la experiencia clínica. Finalmente, la validación multicéntrica de los modelos y su implementación en la toma de decisiones clínicas requiere desarrollar trabajos interdisciplinarios que incluyan radiólogos, neurólogos, científicos de datos y expertos en inteligencia artificial. No obstante, los avances actuales en radiómica y Deep learning permiten afirmar que existe un potencial amplio para que se lleve a cabo una mejora del diagnóstico y tratamiento de la HIC.

Aprendizaje automático

La segmentación y clasificación de imágenes de apoyo diagnóstico juegan actualmente un rol importante en el diagnóstico médico, una identificación temprana de la condición ayuda a brindar una respuesta más rápida. (15) Las técnicas de inteligencia artificial para analizar datos son diversas dentro de las dos principales se encuentran el machine learning y el Deep learning dentro de cada uno existen diferentes métodos y aquellos

métodos que representan mejor a cada uno suele ser la máquina de soporte vectorial para el machine learning y para el Deep learning el uso de redes neuronales convolucionales como lo refiere Wang et al. (16) en donde compararon la precisión que poseen ambas técnicas para clasificar caracteres encontrando con un ligero porcentaje pero mayor dependiendo del tamaño de la imagen a analizar reportando hasta una precisión con redes neuronales convolucionales de 0.95 frente a 0.61 en un set de datos con escala de imagen de 256 x 256 píxeles, y el conjunto más pequeño contaba con escalas de imagen de 64 x 64 logrando una precisión para redes de 0.71 y para máquinas de soporte vectorial de 0.62 una diferencia en cuanto al análisis. Las técnicas de aprendizaje automático utilizan la segmentación y el perfil de intensidad de la imagen para procesar los datos. (15)

En el estudio realizado por Diaz y colaboradores donde utilizan una red neuronal convolucional para la clasificación de imágenes de tumores cerebrales utilizaron 708 cortes de imágenes de meningiomas, 1426 cortes de gliomas, y 930 de tumores de la pituitaria, utilizando este set de datos el 80% siendo 2452 lo dedicaron para entrenar el modelo y 20% siendo 612 imágenes para validación, estas imágenes cabe aclarar tenían una resolución de 512 x 521 píxeles, al igual que utilizaron aumentación de datos para prevenir el sobreajuste de la red neuronal. Para ello el modelo fue entrenado por un total de 80 épocas usando un optimizador basado en descenso del gradiente, al enfrentarse a la validación externa con datos reales la sensibilidad para detección de imágenes de meningiomas fue de 0.93, para glioma fue de 0.99 y para tumores de la pituitaria fue de 0.98 obteniendo un total de precisión del modelo de 0.973. (15)

Una red neuronal convolucional es un tipo de red neuronal “feedforward” que permite extraer características de datos con estructuras de convolución. Una red neuronal convolucional no requiere extraer datos manualmente. (16,17) La similitud en la que se basa una red neuronal es similar a lo biológico ya que una neurona artificial pretende establecerse como análoga a lo que es una neurona biológica los núcleos o kernels de diferentes receptores pueden responder a distintos estímulos, funciones de activación que pueden simularse solo como señales eléctricas que exceden el límite o umbral que puede ser transmitido a la siguiente neurona. (17)

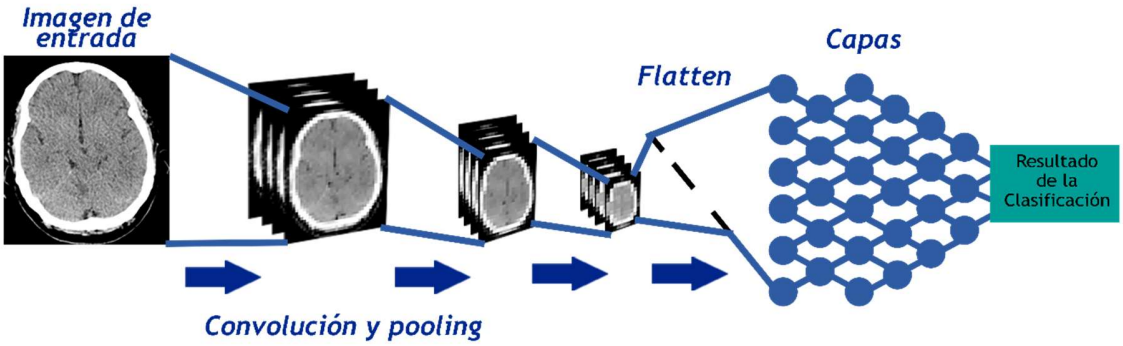


Figura 1. Representación esquemática del funcionamiento de una red neuronal convolucional para el análisis de imágenes de tomografía computarizada. Se muestra cómo la imagen de entrada atraviesa el proceso de convolución y pooling hasta reducirse a un vector unidimensional (flatten) que posteriormente es procesado por una red neuronal densa que realiza la clasificación final.

Al igual que para implementar y optimizar el flujo de un modelo justamente como se llaman existen los optimizadores que se basan en la precisión y pérdida de los modelos entrenados para poder ajustar los hiperparámetros que mejor correspondan al modelo y brindar una mejor precisión y menor pérdida. Las funciones de pérdida y optimizadores son mecanismos que el ser humano invento para enseñar a un sistema de redes neuronales convolucionales a aprender lo que se esperaba que aprendiese. (17) Estos dos conceptos son muy importantes al momento de la validación del modelo ya que permiten estimar a través de sus valores como se ha desempeñado el modelo en la tarea que decidimos otorgarle.

Las redes neuronales convolucionales se utilizan con múltiples propósitos desde reconocimiento de patrones, reconocimiento facial, identificación de objetos (18). En el estudio reportado por Tuli et al. compararon dos modelos un modelo basado en atención y transformación y una red neuronal convolucional resultando en que los transformadores mantenían la precisión en comparación con las redes neuronales convolucionales, y esto pudo deberse a que los modelos basados en atención permiten el enfoque en la parte de la imagen que es importante para la categorización. (18)

Se requieren 4 componentes para construir un modelo de red neuronal convolucional, la convolución que es el medio por el que se extraerá información, las salidas de la

convolución o “outputs” que pueden ser llamados mapas de características, “padding” que es cuando se introduce el valor de cero ajustando los valores de salida y por último el “stride” que es la forma de controlar la densidad de la convolución siendo que entre más largo el stride menor es la densidad. Para poder reducir un área de valores densa se suele utilizar una convolución de 3x3 permitiendo reducir la cantidad de valores a unos mínimos deformando una convolución para enfocarse en lo más representativo del mapa de características. Existen múltiples modelos de redes neuronales convolucionales a medida que se fue desarrollando la primera neurona artificial se comenzaron a volver más complejos, LeCun et al. desarrollaron una red neuronal convolucional capaz de ser entrenada con backpropagation para reconocimiento de caracteres manuscritos, este modelo creado se llamó LeNet-5 compuesto por siete capas de entrenamiento con dos capas convolucionales, dos de pooling, este modelo fue utilizado para lectura y reconocimiento de caracteres en cheques de bancos. Posteriormente se creó la AlexNet que fue de las primeras que utilizó la función ReLU de activación, dropout, y respuesta local de normalización en una red neuronal convolucional. (19)

Una similitud muy importante es que la respuesta local de normalización o local response normalization tiene como propósito simular la inhibición del sistema nervioso lo que permite que las neuronas que reciben cierta actividad de estimulación inhiban la estimulación de las neuronas periféricas. Los kernels de convolución de distinto tamaño pueden extraer diferente información del mapa de características y dependiendo de este tipo de extracción y la escala del kernel es posible obtener el representativo.

Dentro de las múltiples técnicas de machine learning y sus aplicaciones existen dos categorías muy importantes siendo el aprendizaje supervisado y el aprendizaje no supervisado, el aprendizaje no supervisado suele utilizarse para extraer conclusiones a partir de set de datos sin respuestas previas o sin que les hayamos proporcionado una posible solución al problema que se les destina a resolver, permitiendo así que los algoritmos se encarguen de encontrar una relación entre los datos que se les proporciona. En cambio las técnicas con aprendizaje supervisado suelen variar en cuanto a la problemática de clasificación las 5 principales son las técnicas basadas en lógica, la técnica basada en perceptrón, técnicas de aprendizaje a partir de estadística, máquinas de soporte vectorial y

aprendizaje basado en instancias dentro de todas estas existen más técnicas como es el árbol de decisión que permite el modelado y seguimiento de cómo se va realizando la clasificación a través del cálculo del valor requerido a través de la introducción de diversas variables, este tipo de algoritmo se construyen a través del crecimiento del árbol o "tree growth" que consiste en que el conjunto de datos de entrenamiento que se brinda al algoritmo se basa en los datos óptimos o etiquetados apropiadamente para la clasificación permitiendo entrenar el modelo hasta obtener la misma partición de datos que el conjunto original; posteriormente se continua con la poda de árboles o el "tree pruning" en donde se reduce el tamaño del árbol de decisión buscando una partición adecuada de los datos. Algunas de las ventajas de los árboles de decisión son entender el proceso de clasificación y como se fue iterando para realizar la toma de decisiones el único contratiempo llega a ser el manejo de datos faltantes y su optimización.

Las redes Bayesianas permiten asociaciones de probabilidad entre variables y estas pueden desarrollar el aprendizaje a través de la estructura DAG de aprendizaje y la determinación de parámetros y al igual que los árboles de decisión suele ser difícil tratar con los datos faltantes. En cuanto al "k-nearest neighbors" que se mide con respecto al valor de k que define que tan cercanos deben ser los valores vecinos para examinar o describir una parte de los datos, este se basa principalmente en que con el conjunto de datos para entrenamiento se calcula la distancia en datos y basado en el entrenamiento se establece el punto de menor distancia entre ambos datos permitiendo otorgar un valor; una de las ventajas de las técnicas del k-nearest neighbors es su aplicación para conjuntos de datos amplios y dentro de los principales obstáculos pudiesen ser el requerimiento de una mayor cantidad de tiempo para el entrenamiento o clasificación que se requiere. Además de todas estas técnicas de clasificación una en particular que es la máquina de soporte vectorial que se basa en resolver problemas a través de aprendizaje basado en la teoría estadística y realiza esto mediante la clasificación de datos utilizando vectores en un hiperplano con múltiples dimensiones permitiendo la mejor clasificación a través de la creación de un margen que permita clasificar los datos proporcionados; la principal ventaja es la capacidad para lidiar con una amplia variedad de problemas de clasificación, incluyendo problemas de alta dimensión y no separables linealmente.

La hemorragia causada por ruptura del aneurisma subaracnoideo posee un alto riesgo de mortalidad y morbilidad, por lo que el brindar una respuesta terapéutica rápida a este tipo de pacientes permite reducir las complicaciones para los sobrevivientes. (20) La inteligencia artificial que apoya el diagnóstico por imágenes y el uso de las técnicas derivadas para su aplicación nos permite ver un ejemplo concreto en este tipo de pacientes en la optimización del tiempo desde la ruptura del aneurisma o signos de evento isquémico hasta el traslado, posteriormente el escaneo realizado por la tomografía computarizada en donde una gran diferencia es que desde que se está generando la imagen en el computador se permitiría al operador intervenir con software de elección y modelos cargados previamente para la detección del área a intervenir y reduciendo aún más el tiempo a la intervención trasladándose al paciente de forma inmediata a quirófano, permitiendo al mismo tiempo generar un reporte automático de la lesión con las principales características para conocimiento del cirujano.

En cuanto a modelos de machine learning entrenados previamente para el propósito de detección y análisis existe el antecedente de Colasurdo y colaboradores que desarrollaron un modelo con el uso de 340 casos con historial de trauma craneoencefálico permitiendo entrenar al modelo para detección de imágenes de hematoma subdural únicamente y su desarrollo, reportaron una sensibilidad del modelo de 91.4% mostrando un adecuado desempeño para la identificación de este padecimiento. (21)

Un obstáculo para la detección de imágenes y su análisis por parte de modelos de machine learning y en particular las redes neuronales convolucionales es el ruido en las imágenes es por ello que Amin et al se vieron limitados en este aspecto y recomiendan la adquisición de imágenes con alta calidad que permitan su análisis y una mejor clasificación, ya que eliminar el ruido de las imágenes de resonancia magnética resulta una tarea compleja debido a todas las estructuras afectadas por este problema, de igual forma mencionan que la precisión de una clasificación y análisis de la imagen para detectar tumores resulta aún demandante debido a la morfología del tumor en cuanto a tamaño forma y estructura. (22)

La inteligencia artificial ha progresado rápidamente en los últimos años al igual que el auge y atención que se las ha brindado han permitido aprovechar sus funciones para apoyo diagnóstico que pudiesen apoyar en los roles de función diagnóstica donde son más participes

los humanos, aunque esto aún sigue pareciendo controversial debido a que aún no se han empleado para tareas concretas y permitido al mismo tiempo su refinación para estas tareas.

4. PLANTEAMIENTO DEL PROBLEMA

En el año 2019 la American Heart Association (AHA), no solo actualizó la base de grupo de trabajo, sino que emite recomendaciones relacionadas con herramientas para la detección de eventos cerebrovasculares, así como escalas de triaje para determinar la transferencia hacia un centro donde se pueda realizar tratamiento endovascular. De este modo presentaron el cuarto nivel de certificación en el cual se puede realizar tratamiento endovascular pero no son considerados como centros destinados a este tratamiento. (23) El tratamiento es individualizado tal como para cada evento aunque se pretende protocolizar el actuar frente a estos sucesos para mejorar los tiempos de actuación y así poder brindar el mejor tratamiento por eso, es que en el momento en el que se redactaron las guías para el tratamiento de un evento cerebrovascular agudo de tipo hemorrágico indica que se han instaurado diferentes algoritmos según la región debido a la proximidad de los centros de tratamiento así como a la capacidad de estos centros para brindar el tratamiento endovascular, además indican que se debe realizar el triaje según las escalas definidas para dar el tratamiento de mayor nivel posible.

La capacidad de realizar diagnósticos de un episodio hemorrágico se basa en la práctica de la detección clínica, aunque también en el soporte que ofrecen los auxiliares de diagnóstico como algo que puede ofrecer una tomografía computarizada, un segundo aspecto que repercute considerablemente es la variabilidad de la interpretación de la imagen que puede dar lugar a diagnósticos inconsistentes y retrasos en el tratamiento, lo que conlleva unos resultados clínicos pobres. En escenarios de urgencia, y cuando el tiempo es un factor esencial, la necesidad de disponer de una herramienta automatizada y eficaz para la identificación y la clasificación de hemorragias pasa a ser una exigencia para favorecer la determinación de la hemorragia. Los algoritmos de machine learning, con muy especial atención a las redes neuronales convolucionales, han demostrado un potencial relevante en el análisis de imágenes médicas, y ofrecen una solución interesante para la resolución de este problema. Las redes neuronales convolucionales son capaces de aprender características complejas y patrones en las imágenes, favoreciendo la clasificación de forma precisa y reproductible.

Aún con el avance de la aplicación del machine learning, la aplicabilidad de estas tecnologías concretas a la determinación y a la clasificación de hemorragias requiere de una validez de forma asociada y una demostración práctica de la misma tal como sus resultados. Un requerimiento para tener en cuenta para llevar estos proyectos a cabo y poder construir este tipo de modelo es la recogida de datos grandes con el fin de permitir un correcto entrenamiento del modelo, un ejemplo de los usos de estas técnicas es la capacidad predictiva y de clasificación en los modelos actuales es tal que existen estudios mediante los cuales se puede seguir la interacción entre células con los modelos, y prever la hemodinámica en vasos. Este trabajo se propone tratar el reto de cómo poner en práctica estas técnicas que, aunque nueva la aplicación en la actualidad, el conocimiento y diseño se ha originado ya en los años 1950's. El uso de estas técnicas en el ámbito de la medicina permitirá mejorar la celeridad y precisión de los diagnósticos, al igual que disminuir la carga de trabajo de los radiólogos, permitiéndoles concentrarse en aquellos casos más complejos. Todo ello con la finalidad de acelerar el tiempo de respuesta mejorando el apoyo diagnóstico para aquellos eventos que produzcan una crisis cerebrovascular.

5. OBJETIVOS

5.1 Objetivo General

Desarrollar y validar un algoritmo de machine learning basado en redes neuronales convolucionales para identificar y clasificar hemorragias intracraneales con un nivel del 95% de confianza y un margen de error máximo estimado del 5%, en imágenes de tomografía computarizada de pacientes atendidos en el Hospital de Especialidades “Dr. Antonio Fraga Mouret” del Centro Médico Nacional “La Raza”.

5.2 Objetivos específicos

Evaluar el rendimiento del algoritmo con mejor precisión y menor pérdida desarrollado en términos de sensibilidad, especificidad y precisión diagnóstica.

Comparar diferentes arquitecturas de algoritmos de machine learning desarrolladas para seleccionar la más eficiente en el análisis de las imágenes de tomografía computarizada sin contraste.

Comparar las métricas de valores predictivos positivo y negativo para validar el algoritmo externamente con mejores métricas de precisión y pérdida.

Validar el algoritmo final en un conjunto independiente de imágenes para determinar su aplicabilidad clínica.

6. MATERIAL Y MÉTODOS

Para el presente estudio se adoptó el diseño de prueba diagnóstica siendo observacional, retrospectivo y transversal, empleando técnicas de Deep learning, machine learning y redes neuronales convolucionales. Este estudio se ha llevado a cabo en la UMAE Hospital de Especialidades Dr. Antonio Fraga Mouret del Centro Médico Nacional La Raza.

Datos

Se han recolectado las imágenes de tomografía computarizada sin uso de contraste provenientes de pacientes con diagnóstico confirmado mediante reporte del servicio de imagenología, dentro de los cuales se encuentran diagnósticos de hemorragia intracraneal epidural, subdural, subaracnoidea e intraparenquimatosa y sin diagnóstico patológico alguno para la primera fase del desarrollo del algoritmo de machine learning con redes neuronales convolucionales, las imágenes para el modelo se han obtenido del periodo del primero de enero de 2024 al 31 de diciembre de 2024, en la segunda fase de validación externa con el modelo desarrollado se han recolectado imágenes del periodo del primero de enero de 2025 al 31 de enero del 2025 de los pacientes ingresados para evaluación quirúrgica durante este mismo periodo.

Participantes

Para este estudio las imágenes de los pacientes se obtuvieron de los pacientes que se les realizó estudio de tomografía computarizada en la Unidad Médica de Alta Especialidad Hospital de Especialidades “Dr. Antonio Fraga Mouret” del Centro Médico Nacional “La Raza” en Ciudad de México; se realizó en el centro de imagen al momento de ingreso de cada paciente para evaluación u hospitalización, quedando almacenada en el servidor de estudios de imagen del Hospital de Especialidades permitiendo así la consulta de los archivos de imagen junto con su respectivo reporte por parte del médico radiólogo a través de la plataforma de imagen.

Al momento de realizar la selección de las imágenes requeridas para la primera fase del desarrollo del algoritmo, se consideraron los siguientes criterios de inclusión imágenes de pacientes con diagnóstico confirmado de hematoma intracraneal que se les haya realizado estudio y apoyo diagnóstico mediante tomografía computarizada en la unidad, imágenes de

tomografía computarizada con tamaño de 512 x 512 píxeles, imágenes de pacientes adultos mayores de 18 años, imágenes de pacientes que estén disponibles en corte transversal de la zona del encéfalo, imágenes de tomografía sin contraste con diagnóstico confirmado de alguno de estos hallazgos en alguna de las áreas de la bóveda craneal. Se han excluido las imágenes de acuerdo a los siguientes criterios de exclusión imágenes de pacientes de tomografía computarizada incompletas o de baja calidad, imágenes de tomografía computarizada que no tengan un diagnóstico confirmado por un médico radiólogo o un reporte previo especificando los hallazgos de una hemorragia intracraneal, imágenes de pacientes con condiciones coexistentes que puedan complicar la interpretación de las imágenes (tumores cerebrales benignos o malignos), imágenes de pacientes con más de un tipo de hematoma intracraneal, imágenes que no se ajusten a la resolución de píxeles requeridas para el preprocesamiento de datos, imágenes de pacientes que hayan presentado hidrocefalia durante la toma del estudio, imágenes de pacientes donde se visualicen los senos esfenoidales, maxilares o alguna otra cavidad que no corresponda al sistema ventricular, imágenes de pacientes de tomografía computarizada que correspondan a un estudio contrastado, en cuanto a criterios de eliminación fueron los siguientes: imágenes que no cumplieron con los criterios de calidad durante el preprocesamiento siendo imágenes más grandes o con una resolución limitada, datos con información incompleta o inconsistencias en las etiquetas.

La segunda fase de recolección se ha realizado de acuerdo con los pacientes ingresados para tomografía computarizada ya sea para apoyo diagnóstico al ingreso y programación quirúrgica de los pacientes durante el periodo del primero de enero de 2025 al 31 de enero de 2025.

Preparación de datos

Para la fase inicial del desarrollo del algoritmo se comenzó por organizar las imágenes de acuerdo a la presencia o ausencia de hemorragia intracraneal correspondiendo los modelos desarrollados a un algoritmo de aprendizaje supervisado de tipo clasificación binaria, clasificando manualmente 12,392 imágenes de tomografía sin contraste de corte transversal en dos carpetas digitales en las cuales 6066 correspondieron a imágenes con hemorragia intracraneal de algún subtipo, y 6326 imágenes correspondientes a imágenes sin presencia de

hemorragia intracraneal, esto de acuerdo a la interpretación y reporte de estas imágenes revisadas por un médico radiólogo, para garantizar la selección de las imágenes adecuadas se tomaron imágenes en plano transversal únicamente excluyendo aquellas en otro corte, posteriormente se llevó a cabo el preprocesamiento de los datos construyendo y ejecutando un código para redimensionar las imágenes a un tamaño de 100 x 100 píxeles, al igual que se aplicaron técnicas de aumentación de datos, como rotaciones y desplazamientos para evitar el sobreajuste.

El conjunto de 12,392 imágenes se dividió en dos subconjuntos: correspondiendo 85% para datos de entrenamiento y 15% para datos de validación interna y medir métricas del modelo de precisión y pérdida, para la segunda fase de validación externa del modelo se obtuvo una muestra adicional de 355 imágenes de los cuales solo se procesaron 343, se implementó una red neuronal convolucional utilizando el framework TensorFlow en Python. La arquitectura del modelo final incluyó capas convolucionales, capas de pooling y capas densas con funciones de activación ReLU para introducir no linealidad, se utilizó el optimizador Adam y la función de pérdida de entropía cruzada binaria.

Desenlace

El resultado predicho por el modelo fue la presencia o ausencia de hemorragia intracraneal en imágenes de tomografía computarizada, se utilizó como estándar de referencia el diagnóstico confirmado por el reporte escrito por un médico radiólogo del departamento de imagenología, basándose en el criterio de la validez clínica del reporte y la disponibilidad del informe detallado en el sistema. La evaluación del modelo se realizó mediante la comparación directa entre la predicción del modelo como una clasificación binaria “Con hemorragia” o “Sin hemorragia”.

Predictores

Para la selección de predictores se fundamentó en la consulta con un médico radiólogo y un médico neurólogo, quienes realizaron la evaluación de la imagen de tomografía de acuerdo a las características de la imagen como la densidad media en unidades Hounsfield de un rango de 40-90 UH (unidades Hounsfield) para sangre aguda, localización anatómica categorizada de acuerdo a los lóbulos cerebrales, el efecto de masa que se cuenta como

presente si existe desplazamiento de estructuras medias ≥ 5 mm, la textura de la imagen calculada mediante filtros de Gabor y el tamaño del hematoma calculado de acuerdo al diámetro máximo en eje transversal; se debe mencionar que estos no fueron utilizados como características del modelo final y que en cambio fueron los criterios clínicos y radiológicos de referencia.

Muestra

El cálculo del tamaño de la muestra para la fase de validación externa se ha efectuado utilizando la fórmula de estudio de prueba diagnóstica recuperada del libro titulado "MUESTREO Y TAMAÑO DE MUESTRA Una guía práctica para personal de salud que realiza investigación" de Velasco R. y colaboradores (24), buscando obtener una sensibilidad mayor al 70% y especificidad esperada menor al 30% con un nivel de confianza del 95%, para ello resultó en el requerimiento de 323 imágenes y ajustando por posibles pérdidas del 10% resultó en un total de 355 imágenes requeridas para el estudio. A continuación se muestra la fórmula empleada para el cálculo de la muestra para la validación externa buscando estimar un 70% de sensibilidad a detectar con el uso del modelo para detección y clasificación de hemorragias.

Fórmula general de prueba diagnóstica:

$$n = \frac{4(Z_a)^2(pq)}{IC^2}$$

En donde:

n = total de sujetos a estudiar.

Zα = es la desviación normal estandarizada para el nivel de significación establecido (en este caso 1.96).

p = es la proporción esperada, son los valores de sensibilidad que se espera encontrar que es 70%.

q = 1 – p.

IC2 = es la amplitud máxima permitida del intervalo de confianza alrededor del cual consideramos que está el verdadero valor de la sensibilidad o especificidad o intervalo de confianza de 95%.

Sustitución y desarrollo con los valores:

$$n = \frac{4(1.96)^2(0.70)(0.30)}{0.1^2}$$

$$n = \frac{15.36(0.21)}{0.01}$$

$$n = \frac{3.2269}{0.01} = 322.69$$

Datos

La base de datos generada se encuentra en la hoja de cálculo de Microsoft Excel que se encuentra disponible en el repositorio de GitHub en el siguiente link: https://github.com/EDWINZGS136/base-de-datos-proyecto-tesis.

Datos perdidos

Se han omitido del modelo y de la fase de validación externa del modelo aquellos casos que no contaran con reporte de médico radiólogo o la imagen no fuese clara para el diagnóstico mencionándose a través del mismo reporte diagnóstico.

Método de análisis

Para el desarrollo de cada modelo se ajustaron de forma manual los hiperparámetros de cada uno permitiendo entrenarse hasta 13 modelos dentro de los cuales se incluyen en el modelo 3 un modelo con 5 folds diferentes, en el modelo 4 con 20 folds diferentes y el modelo 5 con 4 diferentes folds, en los modelos CNN-1, CNN-2, CNN-6, CNN-7, CNN-8, CNN-9, CNN-10, CNN-11, CNN-12, CNN-13 no se aplicó un entrenamiento con diferentes folds. El desarrollo y entrenamiento de cada modelo convolucional se realizó de acuerdo con

el diagrama de flujo que se muestra en la figura 13. La clasificación se realizó a través de una interfaz que se desarrolló para cargar los modelos de clasificación entrenados y observar su rendimiento de acuerdo con el diagnóstico de cada imagen, para el acceso a esta interfaz a través de una ventana del navegador web se realizó a través de la consola de anaconda prompt (véase figura 8), para posteriormente ingresar al servidor de forma local con un enlace que redirigía a la interfaz en la ventana del navegador (véase figura 9), la interfaz se desarrolló en un bloc de notas a través de un archivo html (véase figura 11) que permite la visualización gráfica de la interfaz, además de un archivo .py que permitía la carga y ejecución del modelo en la interfaz para poder cargar las imágenes y posteriormente probar la clasificación de acuerdo al modelo cargado (véase figura 12), esto permitió evaluar el desempeño de cada modelo respecto al resultado de clasificación.

Para el análisis y desarrollo de los modelos se ha utilizado una computadora laptop Huawei D16 con software Python 3.12.4, Visual Studio Code, con un sistema de 16 GB de RAM y un disco duro de estado sólido de 500 GB, junto con un procesador AMD Ryzen 5 4600H para realizar el proceso de desarrollo del algoritmo, desde lectura de los archivos clasificación de las imágenes y fases de entrenamiento en el entorno de desarrollo de Visual Studio Code, en cuanto a requerimientos del algoritmo por ser un algoritmo de machine learning creado con los requisitos previamente mencionados tanto de software y de hardware fue posible entrenarlo y cargarlo debido a un reescalado de imágenes a una menor dimensión de 100 x 100 píxeles para no sobrepasar el rendimiento del procesador ni de la tarjeta gráfica integrada al momento de entrenar el modelo.

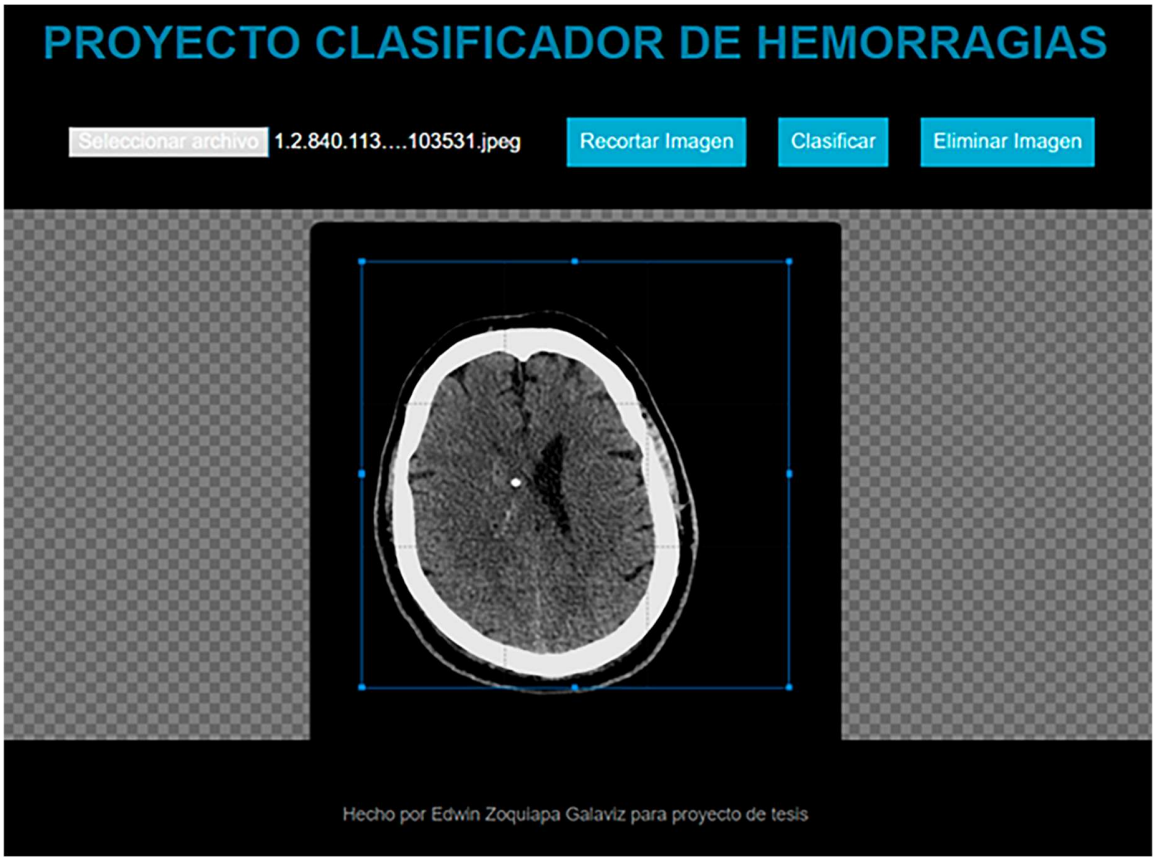


Figura 2. Captura de pantalla de la interfaz gráfica sobre la cual se cargó cada uno de los modelos previamente entrenados, el área seleccionada por el recuadro azul era el área objetivo a procesar e interpretar.

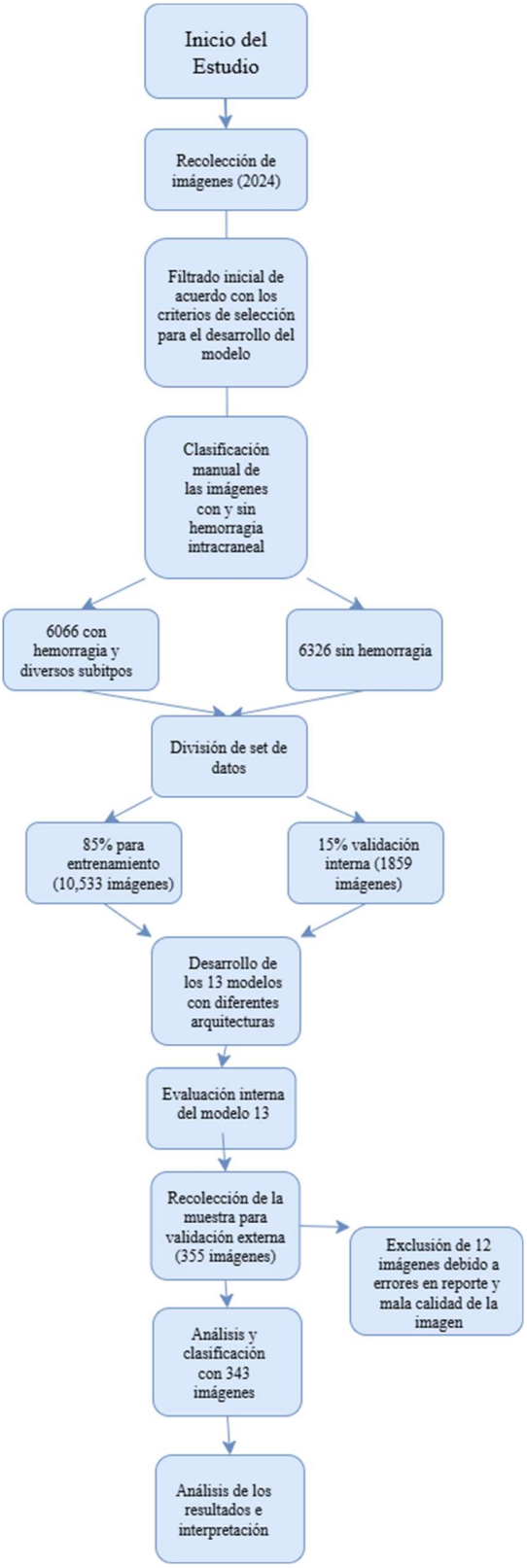


Figura 3. Diagrama de flujo del estudio.

Aprobación ética

El desarrollo de este estudio tanto en la fase de planeación, ejecución, análisis y reporte se ha apegado a los principios éticos en la Declaración de Helsinki (25) y asimismo, se han seguido los lineamientos de la Ley General de Salud en su última reforma publicada en el Diario Oficial de la Federación el 7 de junio de 2024 (26), así como la normativa presente en el Reglamento de la Ley General de Salud en Materia de investigación para la Salud (27), junto con lo establecido en la norma oficial NOM-012-SSA3-2012. (28) Contando con dictamen aprobatorio por parte del comité de ética en investigación del Hospital de Especialidades Dr. Antonio Fraga Mouret del Centro Médico Nacional La Raza con número de registro CLIS R-2024-3501-148.

Financiación

Este estudio se ha llevado a cabo sin la financiación de patrocinadores externos, el presupuesto utilizado para el desarrollo de este estudio ha sido aportado por los miembros del equipo de investigación.

Conflicto de interés

El autor declara no tener conflicto de interés alguno al momento de la planeación y desarrollo de este estudio.

7. RESULTADOS

El desarrollo del modelo de red neuronal convolucional se denominó CNN-13, se realizó con 12,392 imágenes de tomografía computarizada de cráneo sin contraste de las cuales 6,066 (48.9%) tenían reporte de hemorragia intracraneal y 6,326 (51.1%) sin hemorragia intracraneal, para los distintos modelos evaluados se utilizaron 10,533 imágenes para el entrenamiento 5156 con hemorragia y 5377 sin hemorragia correspondiendo a un 85% para entrenamiento y 15% para validación interna que fueron 1,859 imágenes, 910 con hemorragia y 949 sin hemorragia, se aplicaron técnicas de aumentación de datos con un rango de rotación de 30º de las imágenes de forma aleatoria, desplazamiento horizontal, desplazamiento vertical, distorsión y zoom aleatorio.

Para la validación y obtención de métricas de validez diagnóstica se recolectaron 355 archivos de imágenes adicionales de los cuales solo se han seleccionado 343 casos válidos para el análisis se han descartado 12 archivos debido a la mala calidad de la imagen, la media de edad de los pacientes cuyos archivos fueron recolectados fue de 52.66 años [Desviación estándar 16.76], de los cuales 162 (47.2%) fueron masculinos y 181 (52.8%) femeninos. Se realizó una clasificación de los casos de los cuales confirmado con el diagnóstico en reporte del médico radiólogo se identificaron 249 (72.6%) casos sin hemorragia intracraneal y 94 (27.4%) se identificaron con hemorragia intracraneal.

Se construyeron 13 modelos de red neuronal convolucional con TensorFlow/Keras, cada uno se denominó con el prefijo CNN refiriéndose como modelo de red neuronal convolucional seguido del número del modelo que se ha ido mejorando variando de acuerdo a los hiperparámetros, para posteriormente evaluar el que tuviese mejor métrica de precisión y valor de pérdida para seleccionarse el más apropiado, en los modelos 3, 4 y 5 se hizo una validación cruzada con 5, 20 y 4 folds respectivamente, en todos los modelos se puso como límite 100 épocas de entrenamiento y se configuró un earlystopping con paciencia de 30 épocas, el modelo CNN-1 obtuvo una precisión posterior a su entrenamiento de 83%, con una pérdida del 39%, el cual se entrenó por 46 épocas, al final con la evaluación externa obtuvo una sensibilidad del 50% y una especificidad del 30% con un valor predictivo positivo (VPP) del 21%, con un valor predictivo negativo del 62% y un F1-Score de 0.3. El modelo CNN-2 se observó con una precisión del 75%, pérdida del 52%, el cual se entrenó por 17

épocas, y en la evaluación con el conjunto de datos externo obtuvo una sensibilidad del 58% y una especificidad del 27% con un VPP del 23% y un VPN del 63% reflejando un F1-Score de 0.33. El modelo CNN-3 fold 1 mostró una precisión de 86%, pérdida del 31%, el cual se entrenó por 32 épocas, observándose una sensibilidad del 37%, especificidad del 33%, VPP: 17%, VPN: 58%, F1-Score de 0.23. El modelo CNN-3 fold 2 mostró una precisión de 84%, pérdida del 33%, el cual se entrenó por 40 épocas, observándose una sensibilidad del 33%, especificidad del 45%, VPP: 18%, VPN: 64%, F1-Score de 0.23; el modelo CNN-3 fold 3, mostró una precisión de 85%, pérdida del 33%, el cual se entrenó por 34 épocas, observándose una sensibilidad del 46%, especificidad del 28%, VPP: 19%, VPN: 58%, F1-Score de 0.27. El modelo CNN-3 fold 4 mostró una precisión de 87%, pérdida del 31%, el cual se entrenó por 54 épocas, observándose una sensibilidad del 54%, especificidad del 20%, VPP: 20%, VPN: 54%, F1-Score de 0.29. El modelo CNN-3 fold 5 mostró una precisión de 82%, pérdida del 38%, el cual se entrenó por 34 épocas, observándose una sensibilidad del 63%, especificidad del 12%, VPP: 21%, VPN: 48%, F1-Score de 0.32. El modelo CNN-4 fold 1 mostró una precisión de 86%, pérdida del 31%, el cual se entrenó por 37 épocas, observándose una sensibilidad del 52%, especificidad del 26%, VPP: 21%, VPN: 59%, F1-Score de 0.3. El modelo CNN-4 fold 2 mostró una precisión de 86%, pérdida del 32%, el cual se entrenó por 54 épocas, observándose una sensibilidad del 45%, especificidad del 27%, VPP: 19%, VPN: 57%, F1-Score de 0.27. El modelo CNN-4 fold 3 mostró una precisión de 84%, pérdida del 34%, el cual se entrenó por 31 épocas, observándose una sensibilidad del 60%, especificidad del 16%, VPP: 21%, VPN: 53%, F1-Score de 0.31. El modelo CNN-4 fold 4 mostró una precisión de 85%, pérdida del 33%, el cual se entrenó por 30 épocas, observándose una sensibilidad del 38%, especificidad del 29%, VPP: 17%, VPN: 55%, F1-Score de 0.23. El modelo CNN-4 fold 5 mostró una precisión de 86%, pérdida del 32%, el cual se entrenó por 43 épocas, observándose una sensibilidad del 61%, especificidad del 20%, VPP: 22%, VPN: 59%, F1-Score de 0.33. El modelo CNN-4 fold 6 mostró una precisión de 82%, pérdida del 37%, el cual se entrenó por 34 épocas, observándose una sensibilidad del 29%, especificidad del 49%, VPP: 18%, VPN: 65%, F1-Score de 0.22. El modelo CNN-4 fold 7 mostró una precisión de 81%, pérdida del 40%, el cual se entrenó por 35 épocas, observándose una sensibilidad del 46%, especificidad del 21%, VPP: 18%, VPN: 51%, F1-Score de 0.26. El modelo CNN-4 fold 8 mostró una precisión de 87%, pérdida del 32%, el

cual se entrenó por 45 épocas, observándose una sensibilidad del 69%, especificidad del 20%, VPP: 24%, VPN: 63%, F1-Score de 0.36. El modelo CNN-4 fold 9 mostró una precisión de 76%, pérdida del 49%, el cual se entrenó por 16 épocas, observándose una sensibilidad del 59%, especificidad del 13%, VPP: 20%, VPN: 47%, F1-Score de 0.30. El modelo CNN-4 fold 10, mostró una precisión de 85%, pérdida del 32%, el cual se entrenó por 30 épocas, observándose una sensibilidad del 37%, especificidad del 36%, VPP: 18%, VPN: 60%, F1-Score de 0.24. El modelo CNN-4 fold 11 mostró una precisión de 83%, pérdida del 38%, el cual se entrenó por 32 épocas, observándose una sensibilidad del 29%, especificidad del 40%, VPP: 15%, VPN: 60%, F1-Score de 0.20. El modelo CNN-4 fold 12 mostró una precisión de 84%, pérdida del 36%, el cual se entrenó por 31 épocas, observándose una sensibilidad del 44%, especificidad del 28%, VPP: 19%, VPN: 58%, F1-Score de 0.26. El modelo CNN-4 fold 13 mostró una precisión de 78%, pérdida del 45%, el cual se entrenó por 22 épocas, observándose una sensibilidad del 37%, especificidad del 30%, VPP:16%, VPN: 56%, F1-Score de 0.23. El modelo CNN-4 fold 14 mostró una precisión de 83%, pérdida del 36%, el cual se entrenó por 27 épocas, observándose una sensibilidad del 53%, especificidad del 24%, VPP: 21%, VPN: 58%, F1-Score de 0.30. El modelo CNN-4 fold 15 mostró una precisión de 90%, pérdida del 26%, el cual se entrenó por 69 épocas, observándose una sensibilidad del 54%, especificidad del 22%, VPP: 21%, VPN: 57%, F1-Score de 0.30. El modelo CNN-4 fold 16 mostró una precisión de 83%, pérdida del 41%, el cual se entrenó por 28 épocas, observándose una sensibilidad del 54%, especificidad del 25%, VPP: 21%, VPN: 59%, F1-Score de 0.30. El modelo CNN-4 fold 17 mostró una precisión de 86%, pérdida del 33%, el cual se entrenó por 29 épocas, observándose una sensibilidad del 33%, especificidad del 37%, VPP: 16%, VPN: 59%, F1-Score de 0.22. El modelo CNN-4 fold 18 mostró una precisión de 81%, pérdida del 43%, el cual se entrenó por 31 épocas, observándose una sensibilidad del 50%, especificidad del 27%, VPP: 20%, VPN: 59%, F1-Score de 0.29. El modelo CNN-4 fold 19 mostró una precisión de 84%, pérdida del 33%, el cual se entrenó por 30 épocas, observándose una sensibilidad del 43%, especificidad del 33%, VPP: 19%, VPN: 61%, F1-Score de 0.27. El modelo CNN-4 fold 20 mostró una precisión de 86%, pérdida del 31%, el cual se entrenó por 52 épocas, observándose una sensibilidad del 64%, especificidad del 20%, VPP: 23%, VPN: 60%, F1-Score de 0.34. El modelo CNN-5 fold 1 mostró una precisión de 86%, pérdida del 30%, el cual se entrenó por 48 épocas,

observándose una sensibilidad del 56%, especificidad del 16%, VPP: 20%, VPN: 50%, F1-Score de 0.29. El modelo CNN-5 fold 2 mostró una precisión de 83%, pérdida del 36%, el cual se entrenó por 29 épocas, observándose una sensibilidad del 54%, especificidad del 20%, VPP: 20%, VPN: 54%, F1-Score de 0.29. El modelo CNN-5 fold 3 mostró una precisión de 85%, pérdida del 33%, el cual se entrenó por 44 épocas, observándose una sensibilidad del 37%, especificidad del 36%, VPP: 18%, VPN: 60%, F1-Score de 0.24. El modelo CNN-5 fold 4 mostró una precisión de 81%, pérdida del 40%, el cual se entrenó por 26 épocas, observándose una sensibilidad del 46%, especificidad del 26%, VPP: 19%, VPN: 56%, F1-Score de 0.27. El modelo CNN-6, mostró una precisión de 88%, pérdida del 26%, el cual se entrenó por 40 épocas, observándose una sensibilidad del 36%, especificidad del 35%, VPP: 17%, VPN: 59%, F1-Score de 0.23. El modelo CNN-7 mostró una precisión de 75%, pérdida del 77%, el cual se entrenó por 25 épocas, observándose una sensibilidad del 34%, especificidad del 57%, VPP: 23%, VPN: 69%, F1-Score de 0.27. El modelo CNN-8 mostró una precisión de 64%, pérdida del 81%, el cual se entrenó por 6 épocas, observándose una sensibilidad del 13%, especificidad del 81%, VPP: 22%, VPN: 71%, F1-Score de 0.17. El modelo CNN-9 mostró una precisión de 78%, pérdida del 59%, el cual se entrenó por 16 épocas, observándose una sensibilidad del 47%, especificidad del 39%, VPP: 23%, VPN: 66%, F1-Score de 0.31. El modelo CNN-10 mostró una precisión de 70%, pérdida del 71%, el cual se entrenó por 12 épocas, observándose una sensibilidad del 29%, especificidad del 45%, VPP: 17%, VPN: 62%, F1-Score de 0.21. El modelo CNN-11 mostró una precisión de 78%, pérdida del 72%, el cual se entrenó por 16 épocas, observándose una sensibilidad del 36%, especificidad del 54%, VPP: 23%, VPN: 69%, F1-Score de 0.28. El modelo CNN-12 mostró una precisión de 65%, pérdida del 84%, el cual se entrenó por 6 épocas, observándose una sensibilidad del 8%, especificidad del 92%, VPP: 30%, VPN: 72%, F1-Score de 0.13. Los resultados de la curva AUC-ROC, con las métricas de precisión y pérdida de cada modelo se encuentran en el anexo H. La arquitectura junto con el código desarrollado para cada modelo se encuentra disponible en el anexo I.

Las métricas de cada modelo se pueden observar en la tabla 1, el modelo que mostró un mejor rendimiento en sensibilidad fue el modelo denominado CNN-13 que se entrenó durante 40 épocas obteniéndose una precisión de 83% y una pérdida de 42% en la evaluación interna, posteriormente fue seleccionado para la evaluación externa observándose una

sensibilidad de 74%, especificidad del 44%, valor predictivo positivo del 33%, valor predictivo negativo del 82% y un F1-Score de 0.46, mostrando un desempeño moderado con un AUC: 0.59, IC-95%: 0.54-0.65.

Con la aplicación del modelo denominado CNN-13 como clasificador se encontraron 134 observaciones (39.1%) sin hemorragia intracraneal y 209 observaciones (60.9%) con hemorragia intracraneal, los casos se introdujeron a una matriz de confusión a partir de la cual se obtuvieron 70 (20.4%) verdaderos positivos (VP), 110 (32.1%) verdaderos negativos (VN), 139 (40.5%) falsos positivos (FP), 24 (7%) falsos negativos (FN) como se puede apreciar en la figura 4. El total de verdaderos positivos (VP) y verdaderos negativos ha sido de 180.

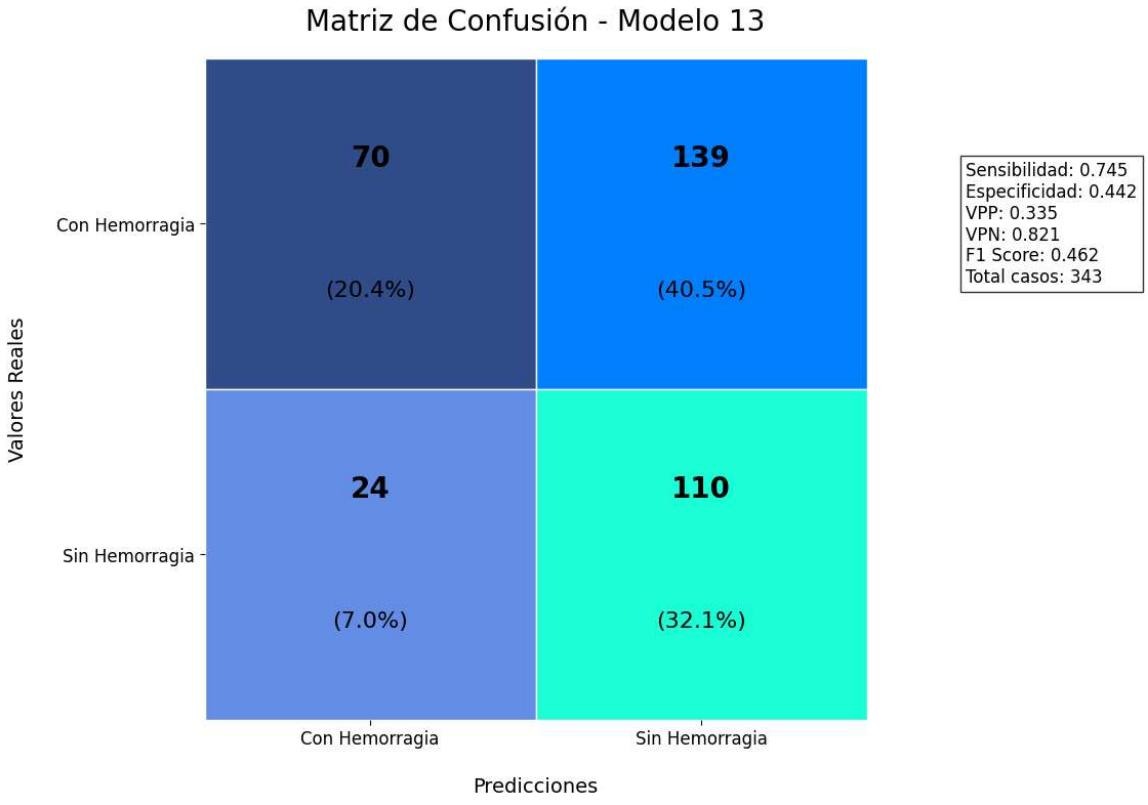


Figura 4. Matriz de confusión de los casos analizados con el modelo 13.

El análisis de la curva ROC mostró un AUC: 0.59, IC-95%: 0.54-0.65, como se puede observar en la figura 5.

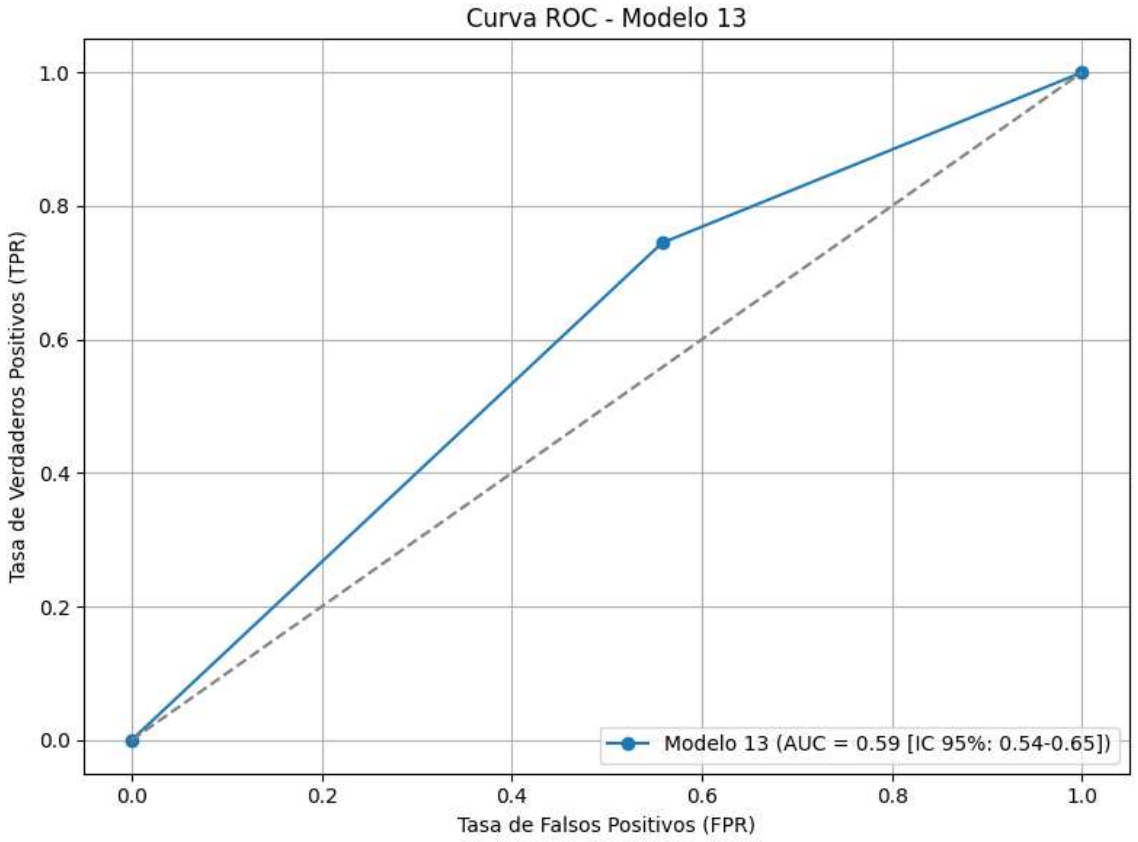


Figura 5. Curva AUC-ROC evaluando el desempeño del modelo 13 de clasificación.

En cuanto a las métricas de precisión obtenidas por el modelo durante el entrenamiento y la validación interna se observó un ligero sobreajuste del modelo CNN-13 posterior a la época 25 y posteriormente continúa por debajo de la curva de entrenamiento como se observa en la figura 6.

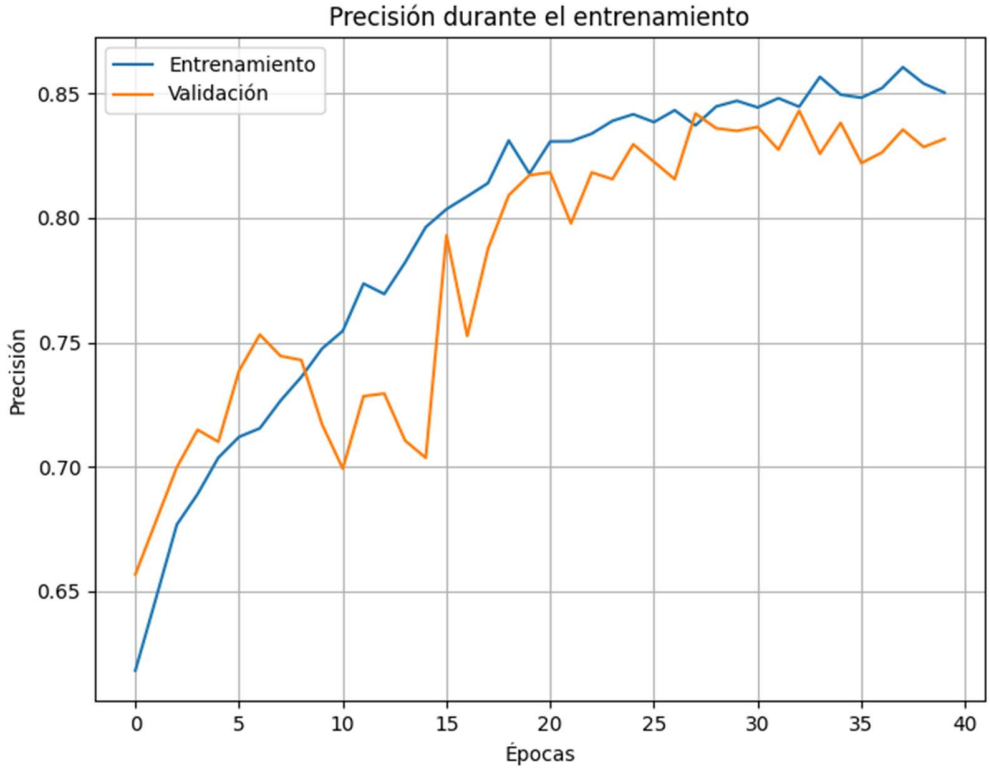


Figura 6. Evolución de la precisión del Modelo 13 durante el Entrenamiento y Validación.

Respecto a la evolución de la pérdida del modelo CNN-13 se puede observar un sobreajuste más marcado por una elevación posterior a la época 5 que se mantiene elevado con picos en épocas como la 14 y la 21 pero manteniéndose por encima de la línea azul correspondiente al entrenamiento como se observa en la figura 7.

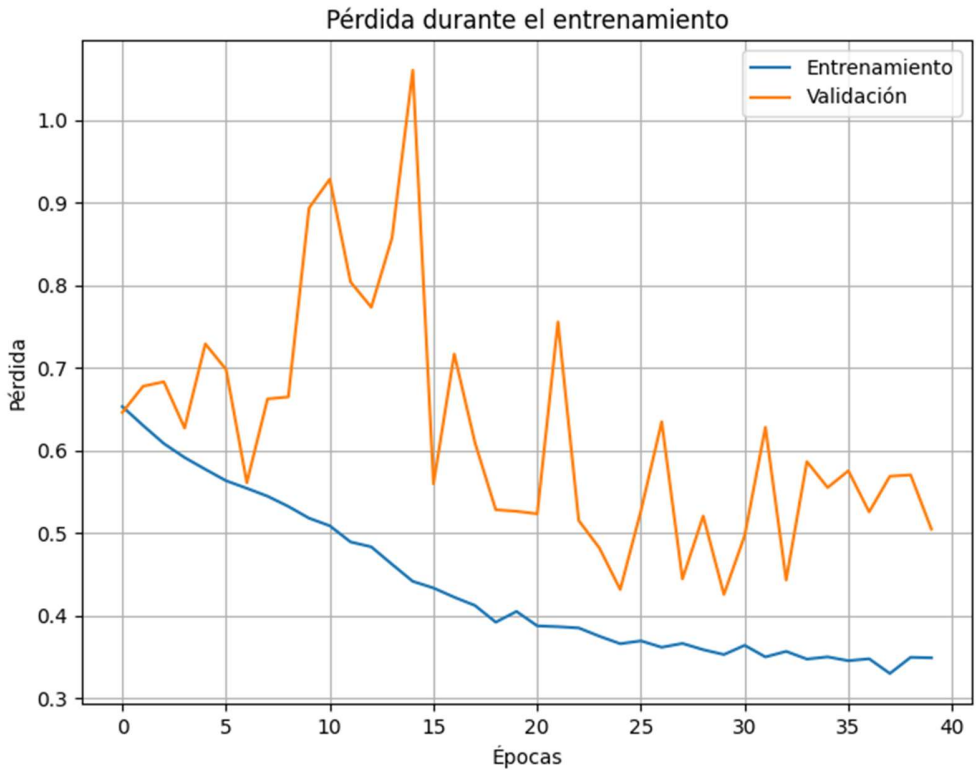


Figura 7. Evolución de la pérdida del Modelo 13 durante el Entrenamiento y Validación.

Modelo	Precisión	Pérdida	Épocas	Sensibilidad	Especificidad	VPP	VPN	F1-Score
CNN-1	0.83	0.39	46	0.5	0.309	0.215	0.621	0.3
CNN-2	0.75	0.52	17	0.585	0.277	0.234	0.639	0.334
CNN-3 fold 1	0.86	0.31	32	0.372	0.337	0.175	0.587	0.238
CNN-3 fold 2	0.84	0.33	40	0.33	0.45	0.185	0.64	0.237
CNN-3 fold 3	0.85	0.33	34	0.468	0.281	0.197	0.583	0.278
CNN-3 fold 4	0.87	0.31	54	0.543	0.209	0.206	0.547	0.298
CNN-3 fold 5	0.82	0.38	34	0.638	0.129	0.217	0.485	0.323
CNN-4 fold 1	0.86	0.31	37	0.521	0.261	0.21	0.591	0.3
CNN-4 fold 2	0.86	0.32	54	0.457	0.273	0.192	0.571	0.27
CNN-4 fold 3	0.84	0.34	31	0.606	0.169	0.216	0.532	0.318
CNN-4 fold 4	0.85	0.33	30	0.383	0.293	0.17	0.557	0.235
CNN-4 fold 5	0.86	0.32	43	0.617	0.209	0.227	0.591	0.332
CNN-4 fold 6	0.82	0.37	34	0.298	0.498	0.183	0.653	0.227
CNN-4 fold 7	0.81	0.40	35	0.468	0.213	0.183	0.515	0.263
CNN-4 fold 8	0.87	0.32	45	0.691	0.201	0.246	0.633	0.363
CNN-4 fold 9	0.76	0.49	16	0.596	0.137	0.207	0.472	0.307
CNN-4 fold 10	0.85	0.32	30	0.372	0.369	0.182	0.609	0.245
CNN-4 fold 11	0.83	0.38	32	0.298	0.402	0.158	0.602	0.207
CNN-4 fold 12	0.84	0.36	31	0.447	0.289	0.192	0.581	0.268
CNN-4 fold 13	0.78	0.45	22	0.372	0.301	0.167	0.56	0.231
CNN-4 fold 14	0.83	0.36	27	0.532	0.245	0.21	0.581	0.301
CNN-4 fold 15	0.90	0.26	69	0.543	0.229	0.21	0.57	0.303
CNN-4 fold 16	0.83	0.41	28	0.543	0.257	0.216	0.598	0.309
CNN-4 fold 17	0.86	0.33	29	0.33	0.373	0.166	0.596	0.221
CNN-4 fold 18	0.81	0.43	31	0.5	0.277	0.207	0.595	0.293
CNN-4 fold 19	0.84	0.33	30	0.436	0.333	0.198	0.61	0.272
CNN-4 fold 20	0.86	0.31	52	0.649	0.205	0.236	0.607	0.346
CNN-5 fold 1	0.86	0.30	48	0.564	0.165	0.203	0.5	0.299
CNN-5 fold 2	0.83	0.36	29	0.543	0.205	0.205	0.543	0.297
CNN-5 fold 3	0.85	0.33	44	0.372	0.361	0.18	0.604	0.243
CNN-5 fold 4	0.81	0.40	26	0.468	0.265	0.194	0.569	0.274
CNN-6	0.88	0.26	40	0.362	0.357	0.175	0.597	0.236
CNN-7	0.75	0.77	25	0.34	0.574	0.232	0.698	0.276
CNN-8	0.64	0.81	6	0.138	0.815	0.22	0.715	0.17
CNN-9	0.78	0.59	16	0.479	0.398	0.231	0.669	0.311
CNN-10	0.70	0.71	12	0.298	0.45	0.17	0.629	0.216
CNN-11	0.78	0.72	16	0.362	0.546	0.231	0.694	0.282
CNN-12	0.65	0.84	6	0.085	0.928	0.308	0.729	0.133
CNN-13	0.83	0.42	40	**0.745***	0.442	0.335	**0.821***	0.462

Tabla 1. Rendimiento de modelos: En la primera columna se observa el modelo desarrollado con su respectivo fold en caso de tenerlo para validación cruzada, la segunda columna

corresponde a la precisión obtenida del modelo posterior a su entrenamiento, en la tercera se encuentra la pérdida de cada modelo, en la cuarta columna el número de épocas por el cual se entrenó, en la quinta columna se encuentra la sensibilidad del modelo posterior a su aplicación con un conjunto de datos distintos del entrenamiento, en la sexta columna se encuentra la especificidad de cada modelo, en la séptima y octava se ubican los valores predictivo positivo y predictivo negativo de cada modelo, por último en la novena columna se ubica el F1-Score de cada modelo.

8. DISCUSIÓN

Interpretación

Dentro de los modelos evaluados aquel que mostró un mejor rendimiento fue el modelo CNN-13 esto en cuanto al objetivo, en el cual se observó una precisión de 0.83 con una pérdida de 0.42 con un valor de sensibilidad de 74% y especificidad del 44% aunque los valores predictivos no se consideraron igual de favorecedores encontrándose un VPP de 0.33 y un VPN de 0.82 con un F1-Score de 0.46, en modelos previos como el caso del CNN-3, 4 y 5 se realizó la validación cruzada mediante múltiples folds considerando que esto mejoraría los resultados de los tres modelos en los que se aplicó, en cambio aquel con mayor cantidad de folds fue el modelo CNN-3 fold 4 mostrando una precisión de 0.87 con una pérdida del modelo de 0.31 y un F1-Score de 0.29 considerando que quizás el aumento de folds puede llegar a incrementar la pérdida, dentro de los modelos evaluados aquellos que se desarrollaron aplicando validación cruzada requirieron una menor cantidad de épocas para el entrenamiento en comparación con las épocas que requirieron los otros modelos, aun así en comparación con el modelo CNN-13 los resultados mostraron que la identificación de hemorragia intracraneal se logró con una precisión diagnóstica aceptable, en el estudio de Moldonavu; en donde se evaluó un enfoque híbrido entre un modelo de machine learning con una red neuronal convolucional en imágenes por resonancia magnética para clasificar meningiomas se obtuvo una precisión de hasta el 99.5% con la combinación de EfficienNet-B0/B4. Aunque las imágenes utilizadas difieren en la naturaleza con las que se utilizaron en este estudio debido a que en ese estudio se utilizaron imágenes de resonancia magnética y en el actual estudio se utilizaron imágenes de tomografía computarizada, en este estudio se destaca como la integración de modelos pre-entrenados con clasificadores adicionales puede llegar a mejorar la precisión, el cual puede apreciarse como un enfoque para optimizar el rendimiento y entrenamiento de modelos. (29) Dentro de la aplicación de algoritmos de aprendizaje automático con imágenes de tomografía cerebral para la detección de hemorragias intracraneales Mäenpää y Korja realizaron una revisión sistemática en la que aunque los modelos de red neuronal convolucional pueden alcanzar sensibilidades superiores al 90%, las especificidades han mostrado una variabilidad considerable llegando desde el 58% hasta el 97.7%, coincidiendo con las limitaciones observadas en este estudio. (30) En

otro estudio realizado por Etli et al. desarrollaron un modelo basado en EfficienNet-B0/B4 para distinguir la hemorragia subaracnoidea, donde se observaron valores de F1-Score, sensibilidad y precisión cercanos al 100%. (31) La diferencia en el rendimiento podría atribuírsele a los factores como el tamaño del bloque de datos ya que en este estudio se incluyeron más de 23.000 imágenes, la calidad de las etiquetas o la inclusión de capas específicas se encontraban adaptadas a la identificación de la hemorragia subaracnoidea únicamente, en cambio nuestro modelo fue entrenado con imágenes en escala de grises de menor resolución de 100 x 100 píxeles lo que probablemente haya limitado la capacidad de extracción de características discriminantes profundas.

Limitaciones

Dentro de las principales limitaciones se encontró que la dependencia de los datos de un único centro hospitalario introduce el sesgo geográfico y demográfico, una problemática que se comparte con el 60% de los estudios en el rubro por lo que la homogeneidad que se presenta puede no ser adecuada al generalizar los resultados a otras poblaciones, el redimensionamiento de las imágenes también parece ser un reto ya que lo ideal sería contar con un clasificador de imágenes que leyera la imagen sin procesos de convolución eso requiere equipo de alto nivel por ejemplo de 32 GB únicamente de RAM y una GPU dedicada, por lo que también deben considerarse los equipamientos y material con el que se desarrollan, esto permite suponer que con mejor hardware como procesador, memoria RAM, inclusive una tarjeta de video que permita manejar procesos y distribuirlos entre la cantidad total de memoria podría ayudar a mejorar este tipo de algoritmos al grado de ser excelentes, aunque no se descarta que en un futuro el hardware requerido pueda ser más accesible y que permita mejores resultados con la redimensión de imágenes.

Adicionalmente un proceso esencial son los datos debido a que las métricas de precisión se basan en la calidad de los datos y la cantidad en este estudio aunque se utilizaron una gran cantidad de imágenes de más de 12,000 y adicionalmente a la transformación de los mismos estos no permitieron llegar a métricas por encima del 71% evidenciando que aunque detecta los cambios en la morfología o los hallazgos extraños en la morfología habitual de las estructuras encefálicas, no ha permitido diferenciar apropiadamente la diversa

cantidad de escenarios probables que puedan existir además de no detectar aquellos casos en los que verdaderamente se padecía con la hemorragia y esto pudo deberse a las diferentes densidades observadas y su distribución en el encéfalo, junto con las estructuras presentes en cada plano analizado que van desde un plano superior donde solo se encuentra la porción encefálica hasta un nivel inferior donde se pueden observar los senos frontales, los ventrículos o incluso la diferencia en el radio o diámetro de la hemorragia a diversas alturas, por lo que una cantidad adecuada de datos con las condiciones y características apropiadas como imágenes de tomografía para cada plano con controles sin evidencia de calcificaciones adicionales y morfologías alteradas, empleando conjuntos de datos robustos permitiría armar una herramienta no solo para la clasificación, también para la detección de cualquier anormalidad a nivel encefálico permitiendo brindarle al algoritmo de entrenamiento todas las morfologías posibles para un control adecuado y distribución de una hemorragia en los diferentes planos, permitiendo así no solo clasificar las hemorragias con base en los hallazgos del algoritmo o a patrones preestablecidos en el entrenamiento, sino que además permitiría evidenciar sospecha de alteración en la morfología de las estructuras asociándolas no solo a hemorragia, sino a otras condiciones ameritando una intervención. Adicionalmente si se añadiera un mapa de gradientes de acuerdo con las densidades de cada estructura en el cráneo se podría evidenciar la concentración de cada tipo de sustancia o morfología de acuerdo con la densidad mostrada en la imagen, permitiendo brindar al médico un área sobre la cual centrar la interpretación de la imagen.

Uso del modelo en el contexto del cuidado actual

El modelo actual puede implementarse para el aprendizaje o el test de otros modelos que se lleguen a desarrollar en el futuro y poder evaluar como las métricas evolucionan y se modulan respecto a los hiperparámetros, considerándose como un referente previo para la elaboración de un algoritmo o estructura más compleja que permita una mejor clasificación. La integración de este modelo requeriría estandarizar el preprocesamiento de imágenes para disminuir artefactos, al igual que una capacitación completa y constante al personal operador y de interpretación en el departamento de imagenología, la aplicación de este algoritmo se recomienda restringirlo a solo población adulta debido a que no se evaluó población pediátrica, por lo que la prioridad actual sería validar el modelo en otras poblaciones para

permitir la generalización e implementación de esta tecnología, al igual que desarrollar una interfaz accesible para facilitar el uso a personal no técnico para obtener mejores resultados. Permitiendo identificar las posibles debilidades y limitaciones del actual modelo en otras poblaciones para actualizarlo a nuevas versiones.

El presente modelo aunque no es óptimo para brindar un diagnóstico preciso sobre la presencia de hemorragia intracraneal permite identificar anormalidades en la imagen mediante la detección de gradientes. La aplicación próxima se puede realizar en centros médicos con alta demanda que permitan integrar a los algoritmos con este tipo de tecnología y técnica para que pueda brindarse prioridad a los casos donde la imagen aparezca con mayor cantidad de criterios sugestivos de hemorragia intracraneal y confirmarlo posteriormente mediante el diagnóstico por parte del médico radiólogo. Esto permitirá apoyar a los equipos de médicos que cuentan con un entrenamiento deficiente en la identificación e interpretación de imágenes, permitiendo delegar tareas en donde los servicios se encuentren saturados y optimizar el flujo de trabajo para procedimientos que requieran intervención inmediata.

Además se propone que el desarrollo de nuevos modelos reforzará la capacidad de los médicos y de la implementación de esta tecnología en el área de la salud para la detección de imágenes, progresión y evolución de condiciones que pudiesen complicarse hasta situaciones irreversibles.

El trabajo presentado es una contribución a las herramientas de apoyo diagnóstico para los médicos con entrenamiento y sin entrenamiento en imágenes de tomografía computarizada, con una posible aplicación como herramienta de triaje para asistir a la identificación de los casos relevantes sin sustituir el criterio clínico como elemento central en la toma de decisiones. La presente tesis establece bases para integrar modelos de inteligencia artificial y algoritmos de aprendizaje automático en flujos para toma de decisiones en el área médica y de las ciencias de la salud.

9. CONCLUSIONES

El modelo de red neuronal convolucional desarrollado denominado CNN-13 mostró una capacidad moderada para identificar hemorragias intracraneales en imágenes de tomografía computarizada con una sensibilidad del 74%, IC-95%: 64.4-82.9, y una especificidad del 44%, IC-95%: 37.9-50.6, aunque la sensibilidad sugirió potencial para la aplicación en el triaje inicial, el bajo valor predictivo del 33% refleja una tasa de falsos positivos limitando su aplicabilidad clínica. Aunque la especificidad del 44.1% y el valor predictivo positivo del 33% fueron métricas limitadas, se podría validar su aplicación como un sistema de apoyo diagnóstico, no sustitutivo que pudiese integrarse a los flujos de trabajo para priorizar los casos urgentes y optimizar la toma de decisiones en tiempo real, contribuyendo a una atención médica ágil y eficiente, fungiendo como una herramienta adicional para el diagnóstico y que médicos con falta de experiencia o con entrenamiento insuficiente pudiesen emplear. Se debe destacar que el modelo CNN-13 se realizó en una laptop de marca Huawei modelo D16 2022 considerada como una laptop de gama media con un procesador AMD Ryzen 5 4600H, 16GB de RAM y una memoria tipo SSD de 500 GB sin emplear GPU dedicada, demostrando que con recursos limitados de hardware es posible crear e implementar algoritmos de aprendizaje profundo, permitiendo además la aplicación de esta tecnología en entornos hospitalarios con una infraestructura informática básica.

10. BIBLIOGRAFÍA

1. Smorchkova AK, Khoruzhaya AN, Kremneva EI, Petryaikin AV. Machine learning technologies in CT-based diagnostics and classification of intracranial hemorrhages. Zhurnal Voprosy Neirokhirurgii Imeni NN Burdenko. 2023 Jan 1;87(2):85-91. DOI: 10.17116/neiro20238702185.
2. Shi Y, Zhang G, Ma C, Xu J, Xu K, Zhang W, Wu J, Xu L. Machine learning algorithms to predict intraoperative hemorrhage in surgical patients: a modeling study of real-world data in Shanghai, China. BMC Medical Informatics and Decision Making. 2023 Aug 10;23(1):156. DOI: 10.1186/s12911-023-02253-w.
3. Soofi AA, Awan A. Classification techniques in machine learning: applications and issues. Journal of Basic & Applied Sciences. 2017 Jan 5;13:459-65.
4. Lyu J, Xu Z, Sun H, Zhai F, Qu X. Machine learning-based CT radiomics model to discriminate the primary and secondary intracranial hemorrhage. Scientific Reports. 2023 Mar 6;13(1):3709. DOI: 10.1038/s41598-023-30678-w.
5. Dengler NF, Madai VI, Unteroberdörster M, Zihni E, Brune SC, Hilbert A, Livne M, Wolf S, Vajkoczy P, Frey D. Outcome prediction in aneurysmal subarachnoid hemorrhage: a comparison of machine learning methods and established clinico-radiological scores. Neurosurgical Review. 2021 Oct;44(5):2837-46. DOI: 10.1007/s10143-020-01453-6.
6. Centro Nacional de Excelencia Tecnológica (CENETEC). Guía de Práctica Clínica: Diagnóstico y tratamiento de la hemorragia intracerebral en adultos [Internet] Ciudad de México: Secretaría de Salud; 2022 [citado el 5 de agosto de 2025] Disponible en: https://cenetec-difusion.com/CMGPC/GPC-S-102-22/RR.pdf.
7. Ebrahimi S, Bagchi P. Application of machine learning in predicting blood flow and red cell distribution in capillary vessel networks. Journal of the Royal Society Interface. 2022 Aug 10;19(193):20220306. DOI: 10.1098/rsif.2022.0306.
8. Trevisi G, Caccavella VM, Scerrati A, Signorelli F, Salamone GG, Orsini K, Fasciani C, D'Arrigo S, Auricchio AM, D'Onofrio G, Salomi F. Machine learning model prediction of 6-month functional outcome in elderly patients with intracerebral hemorrhage. Neurosurgical Review. 2022 Aug;45(4):2857-67. DOI: 10.1007/s10143-022-01802-7.

9. Tanioka S, Yago T, Tanaka K, Ishida F, Kishimoto T, Tsuda K, Ikezawa M, Araki T, Miura Y, Suzuki H. Machine learning prediction of hematoma expansion in acute intracerebral hemorrhage. Scientific Reports. 2022 Jul 21;12(1):12452. DOI: 10.1038/s41598-022-15400-6.
10. Rodríguez-Yáñez M, Castellanos M, Freijo MM, Fernández JL, Martí-Fàbregas J, Nombela F, Simal P, Castillo J, Díez-Tejedor E, Fuentes B, de Leciñana MA. Guías de actuación clínica en la hemorragia intracerebral. Neurología. 2013 May 1;28(4):236-49.
11. Van Asch CJ, Luitse MJ, Rinkel GJ, van der Tweel I, Algra A, Klijn CJ. Incidence, case fatality, and functional outcome of intracerebral haemorrhage over time, according to age, sex, and ethnic origin: a systematic review and meta-analysis. The Lancet Neurology. 2010 Feb 1;9(2):167-76. DOI: 10.1016/S1474-4422(09)70340-0.
12. Seiffge DJ, Anderson CS. Treatment for intracerebral hemorrhage: dawn of a new era. International Journal of Stroke. 2024 Jun;19(5):482-9. DOI: 10.1177/17474930241250259.
13. Penckofer M, Kazmi KS, Thon J, Tonetti DA, Ries C, Rajagopalan S. Neuro-imaging in intracerebral hemorrhage: updates and knowledge gaps. Frontiers in neuroscience. 2024 May 9;18:1408288. DOI: 10.3389/fnins.2024.1408288.
14. Greenberg SM, Ziai WC, Cordonnier C, Dowlatshahi D, Francis B, Goldstein JN, Hemphill III JC, Johnson R, Keigher KM, Mack WJ, Mocco J. 2022 guideline for the management of patients with spontaneous intracerebral hemorrhage: a guideline from the American Heart Association/American Stroke Association. Stroke. 2022 Jul;53(7):e282-361.
15. Díaz-Pernas FJ, Martínez-Zarzuela M, Antón-Rodríguez M, González-Ortega D. A deep learning approach for brain tumor classification and segmentation using a multiscale convolutional neural network. InHealthcare 2021 Feb 2 (Vol. 9, No. 2, p. 153). MDPI. DOI: 10.3390/healthcare9020153.
16. Wang P, Fan E, Wang P. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. Pattern recognition letters. 2021 Jan 1;141:61-7. DOI: 10.1016/j.patrec.2020.07.042.

17. Li Z, Liu F, Yang W, Peng S, Zhou J. A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems. 2021 Jun 10;33(12):6999-7019. DOI: 10.1109/TNNLS.2021.3084827.
18. Tuli S, Dasgupta I, Grant E, Griffiths TL. Are convolutional neural networks or transformers more like human vision?. arXiv preprint arXiv:2105.07197. 2021 May 15. DOI: 10.48550/arXiv.2105.07197.
19. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 2002 Aug 6;86(11):2278-324. DOI: 10.1109/5.726791.
20. Salman S, Gu Q, Sharma R, Wei Y, Dherin B, Reddy S, Tawk R, Freeman WD. Artificial intelligence and machine learning in aneurysmal subarachnoid hemorrhage: future promises, perils, and practicalities. Journal of the Neurological Sciences. 2023 Nov 15;454:120832. DOI: 10.1016/j.jns.2023.120832.
21. Colasurdo M, Leibushor N, Robledo A, Vasandani V, Luna ZA, Rao AS, Garcia R, Srinivasan VM, Sheth SA, Avni N, Madziva M. Automated detection and analysis of subdural hematomas using a machine learning algorithm. Journal of neurosurgery. 2022 Sep 30;138(4):1077-84. DOI: 10.3171/2022.8.JNS22888.
22. Amin J, Sharif M, Haldorai A, Yasmin M, Nayak RS. Brain tumor detection and classification using machine learning: a comprehensive survey. Complex & intelligent systems. 2022 Aug;8(4):3161-83. DOI: 10.1007/s40747-021-00563-y.
23. Jauch EC, Schwamm LH, Panagos PD, Barbazzeni J, Dickson R, Dunne R, Foley J, Fraser JF, Lassers G, Martin-Gill C, O'Brien S. Recommendations for regional stroke destination plans in rural, suburban, and urban communities from the prehospital stroke system of care consensus conference: a consensus statement from the American Academy of Neurology, American Heart Association/American Stroke Association, American Society of Neuroradiology, National Association of EMS Physicians, National Association of State EMS Officials, Society of NeuroInterventional Surgery, and Society of Vascular and Interventional Neurology: endorsed by .... Stroke. 2021 May;52(5):e133-52. DOI: 10.1161/STROKEAHA.120.033228.

24. Rodríguez VM, Verónica Araceli Martínez O., Hernández JR, Huazano F, Armando Nieves R. Muestreo y tamaño de la muestra: una guía práctica para personal de salud que realiza investigación. El Cid Editor; 2003.
25. World Medical Association. Declaración de Helsinki – Principios éticos para las investigaciones médicas en seres humanos [Internet]. México: Comisión Nacional de Bioética; [citado 2025 Jul 7]. Disponible en: https://www.conbioetica-mexico.salud.gob.mx/descargas/pdf/helsinki.pdf
26. Secretaría de Salud. Ley General de Salud [Internet]. México: Diario Oficial de la Federación; [citado 2025 Jul 7]. Disponible en: https://salud.gob.mx/unidades/cdi/legis/lgs/LEY_GENERAL_DE_SALUD.pdf
27. Secretaría de Salud. Reglamento de la Ley General de Salud en Materia de Investigación para la Salud [Internet]. México: Diario Oficial de la Federación; [citado 2025 Jul 7]. Disponible en: https://salud.gob.mx/unidades/cdi/nom/compi/rlgsmis.html
28. Secretaría de Gobernación. Norma Oficial Mexicana NOM-012-SSA3-2012, Que establece los criterios para la ejecución de proyectos de investigación para la salud en seres humanos [Internet]. Diario Oficial de la Federación; 2013 ene 4 [citado 2025 Jul 7]. Disponible en: https://dof.gob.mx/nota_detalle.php?codigo=5284148&fecha=04/01/2013
29. Moldovanu S, Tăbăcaru G, Barbu M. Convolutional neural Network–Machine learning model: hybrid model for meningioma tumour and healthy brain classification. Journal of Imaging. 2024 Sep 20;10(9):235. DOI: 10.3390/jimaging10090235.
30. Mäenpää SM, Korja M. Diagnostic test accuracy of externally validated convolutional neural network (CNN) artificial intelligence (AI) models for emergency head CT scans–A systematic review. International Journal of Medical Informatics. 2024 Sep 1;189:105523. DOI: 10.1016/j.ijmedinf.2024.105523.
31. Etli MU, Başarslan MS, Varol E, Sarıkaya H, Çakıcı YE, Öndüç GG, Bal F, Kayalar AE, Aykılıç Ö. Evaluating Deep Learning Techniques for Detecting Aneurysmal Subarachnoid Hemorrhage: A Comparative Analysis of Convolutional Neural

Network and Transfer Learning Models. World Neurosurgery. 2024 Jul 1;187:e807-13. DOI: 10.1016/j.wneu.2024.04.168.

11. ANEXOS

Anexo A. Instrumento de recolección de datos.

Desarrollo de un algoritmo de Machine Learning para identificar y clasificar hemorragias intracraneales con tomografías en el Hospital de Especialidades del CMN La Raza

1. **Ingresar el nombre del archivo de imagen asignado**
2. **De acuerdo con las siguientes categorías evalué la imagen y proporcioné una categoría para la imagen**
    a. **1)Epidural**
    b. **2)Subdural**
    c. **3)Intraparenquimatoso**
    d. **4)subaracnoidea**

ID paciente	Nombre de imagen asignado	Categoría asignada por neurólogo	Categoría asignada por especialista en radiología e imagen	Diagnóstico de acuerdo con el reporte de radiología	Categoría asignada en fase de entrenamiento
01					
02					
03					
04					
05					
06					
07					
08					
09					
10					
11					
12					
13					
14					
15					
16					

Formato ilustrativo. El formato se encontrará en una tabla de Excel para poder ser completado con los datos de forma digital hasta obtener la muestra requerida.

Anexo B. Matriz de confusión del modelo de red neuronal convolucional denominado CNN-13.

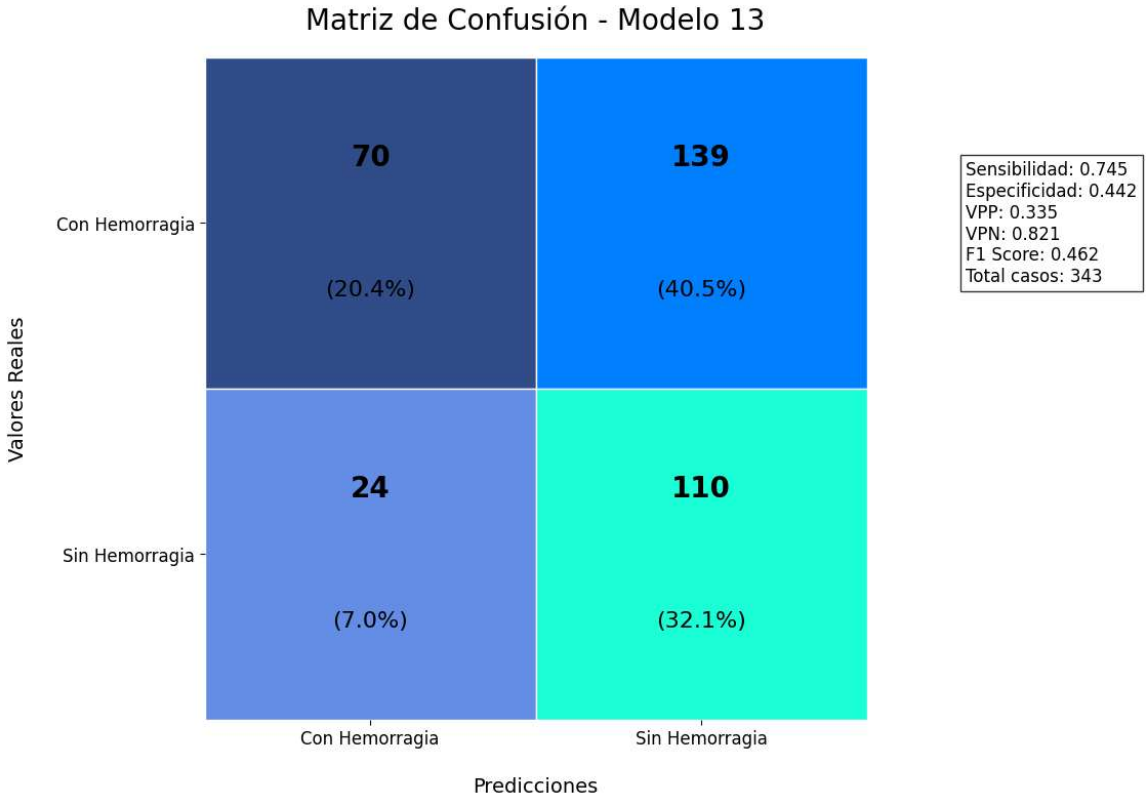


Figura 4. Matriz de confusión de los casos analizados con el modelo CNN-13.

Anexo C. Conjunto de gráficos del modelo de red neuronal convolucional denominado CNN-13 con sus respectivos gráficos de métricas de precisión.

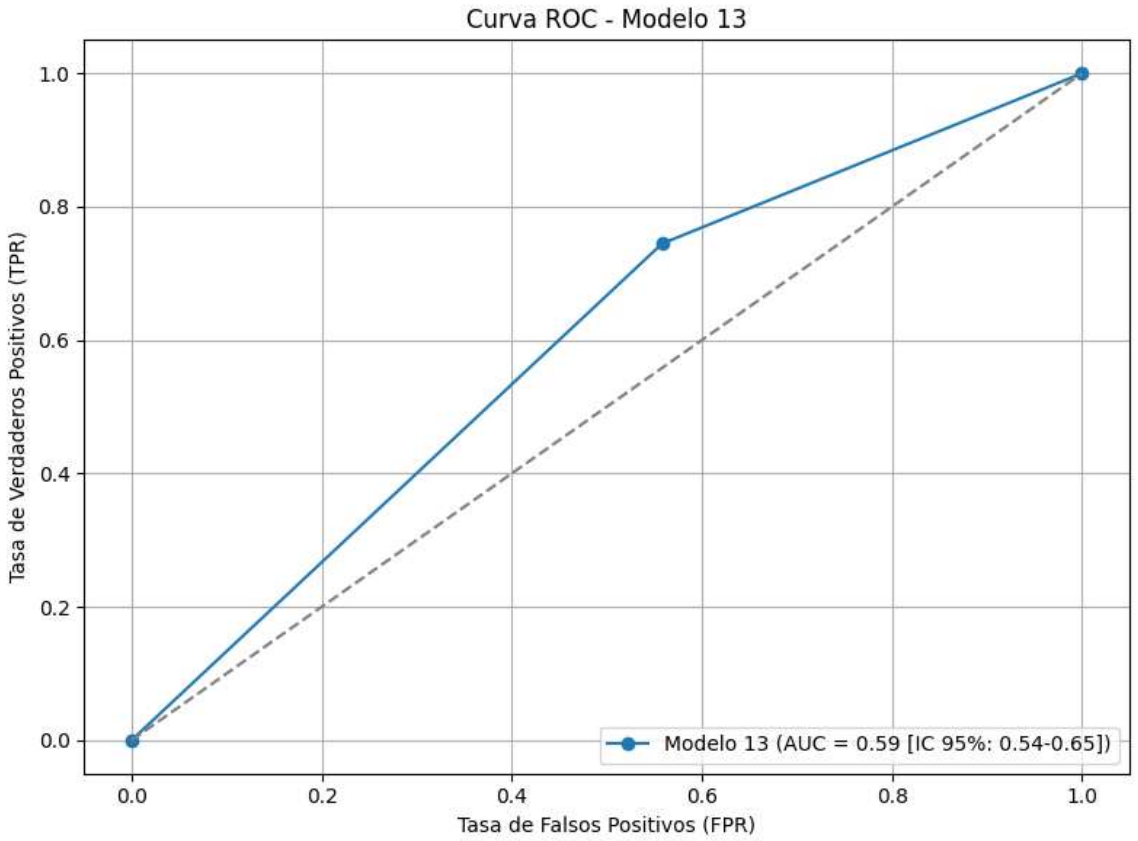


Figura 5. Curva AUC-ROC evaluando el desempeño del modelo CNN-13 de clasificación.

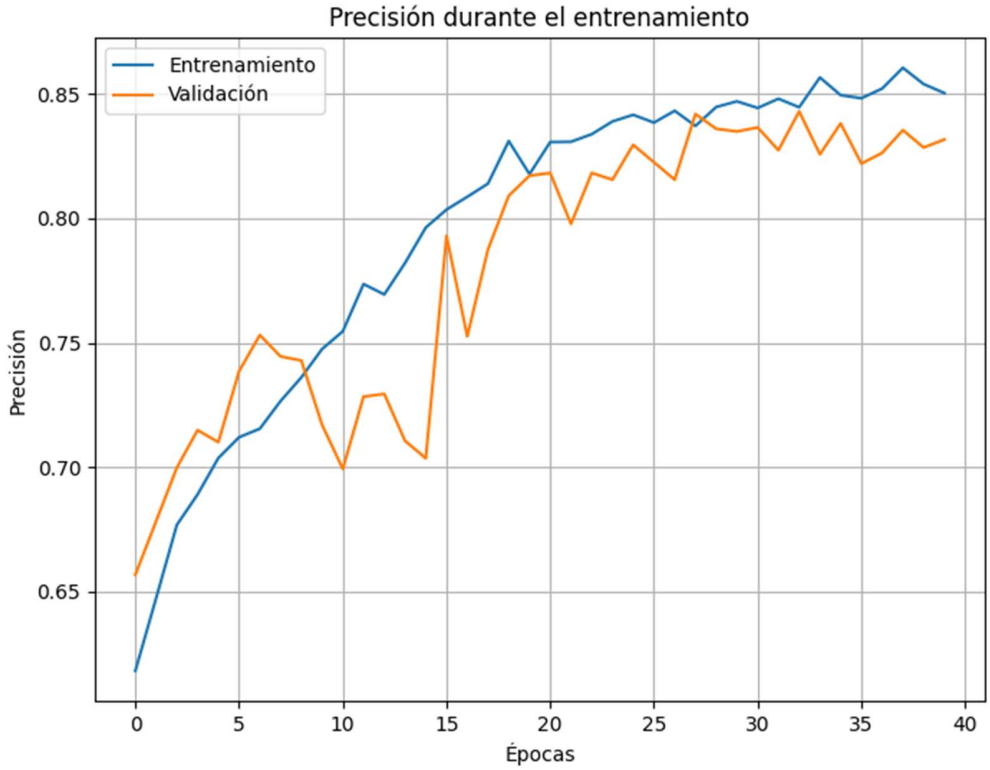


Figura 6. Evolución de la precisión del modelo CNN-13 durante el Entrenamiento y Validación.

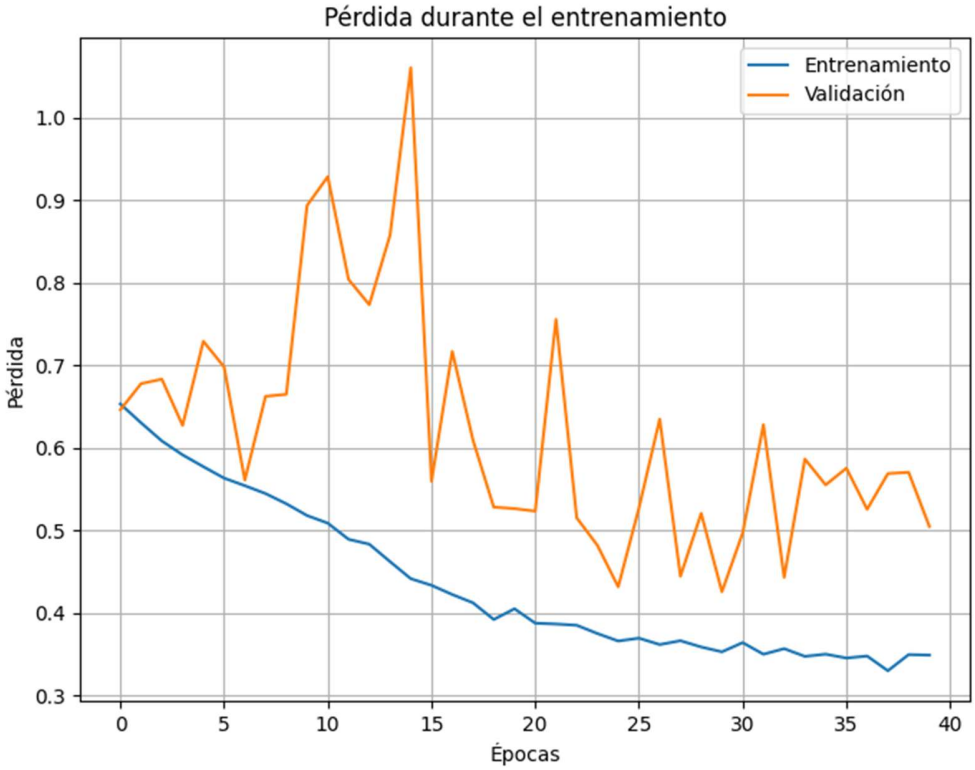


Figura 7. Evolución de la pérdida del modelo CNN-13 durante el Entrenamiento y Validación.

Anexo D. Tabla del rendimiento con la cantidad de épocas de entrenamiento de cada modelo, los respectivos folds, métricas de sensibilidad, especificidad, valor predictivo positivo y negativo con su respectivo F1-Score.

Modelo	Precisión	Pérdida	Épocas	Sensibilidad	Especificidad	VPP	VPN	F1-Score
CNN-1	0.83	0.39	46	0.5	0.309	0.215	0.621	0.3
CNN-2	0.75	0.52	17	0.585	0.277	0.234	0.639	0.334
CNN-3 fold 1	0.86	0.31	32	0.372	0.337	0.175	0.587	0.238
CNN-3 fold 2	0.84	0.33	40	0.33	0.45	0.185	0.64	0.237
CNN-3 fold 3	0.85	0.33	34	0.468	0.281	0.197	0.583	0.278
CNN-3 fold 4	0.87	0.31	54	0.543	0.209	0.206	0.547	0.298
CNN-3 fold 5	0.82	0.38	34	0.638	0.129	0.217	0.485	0.323
CNN-4 fold 1	0.86	0.31	37	0.521	0.261	0.21	0.591	0.3
CNN-4 fold 2	0.86	0.32	54	0.457	0.273	0.192	0.571	0.27
CNN-4 fold 3	0.84	0.34	31	0.606	0.169	0.216	0.532	0.318
CNN-4 fold 4	0.85	0.33	30	0.383	0.293	0.17	0.557	0.235
CNN-4 fold 5	0.86	0.32	43	0.617	0.209	0.227	0.591	0.332
CNN-4 fold 6	0.82	0.37	34	0.298	0.498	0.183	0.653	0.227
CNN-4 fold 7	0.81	0.40	35	0.468	0.213	0.183	0.515	0.263
CNN-4 fold 8	0.87	0.32	45	0.691	0.201	0.246	0.633	0.363
CNN-4 fold 9	0.76	0.49	16	0.596	0.137	0.207	0.472	0.307
CNN-4 fold 10	0.85	0.32	30	0.372	0.369	0.182	0.609	0.245
CNN-4 fold 11	0.83	0.38	32	0.298	0.402	0.158	0.602	0.207
CNN-4 fold 12	0.84	0.36	31	0.447	0.289	0.192	0.581	0.268
CNN-4 fold 13	0.78	0.45	22	0.372	0.301	0.167	0.56	0.231
CNN-4 fold 14	0.83	0.36	27	0.532	0.245	0.21	0.581	0.301
CNN-4 fold 15	0.90	0.26	69	0.543	0.229	0.21	0.57	0.303
CNN-4 fold 16	0.83	0.41	28	0.543	0.257	0.216	0.598	0.309
CNN-4 fold 17	0.86	0.33	29	0.33	0.373	0.166	0.596	0.221
CNN-4 fold 18	0.81	0.43	31	0.5	0.277	0.207	0.595	0.293
CNN-4 fold 19	0.84	0.33	30	0.436	0.333	0.198	0.61	0.272
CNN-4 fold 20	0.86	0.31	52	0.649	0.205	0.236	0.607	0.346
CNN-5 fold 1	0.86	0.30	48	0.564	0.165	0.203	0.5	0.299
CNN-5 fold 2	0.83	0.36	29	0.543	0.205	0.205	0.543	0.297
CNN-5 fold 3	0.85	0.33	44	0.372	0.361	0.18	0.604	0.243
CNN-5 fold 4	0.81	0.40	26	0.468	0.265	0.194	0.569	0.274
CNN-6	0.88	0.26	40	0.362	0.357	0.175	0.597	0.236
CNN-7	0.75	0.77	25	0.34	0.574	0.232	0.698	0.276
CNN-8	0.64	0.81	6	0.138	0.815	0.22	0.715	0.17
CNN-9	0.78	0.59	16	0.479	0.398	0.231	0.669	0.311
CNN-10	0.70	0.71	12	0.298	0.45	0.17	0.629	0.216
CNN-11	0.78	0.72	16	0.362	0.546	0.231	0.694	0.282
CNN-12	0.65	0.84	6	0.085	0.928	0.308	0.729	0.133

CNN-13	0.83	0.42	40	**0.745***	0.442	0.335	**0.821***	0.462

Tabla 1. Rendimiento de modelos: En la primera columna se observa el modelo desarrollado con su respectivo fold en caso de tenerlo para validación cruzada, la segunda columna corresponde a la precisión obtenida del modelo posterior a su entrenamiento, en la tercera se encuentra la pérdida de cada modelo, en la cuarta columna el número de épocas por el cual se entrenó, en la quinta columna se encuentra la sensibilidad del modelo posterior a su aplicación con un conjunto de datos distintos del entrenamiento, en la sexta columna se encuentra la especificidad de cada modelo, en la séptima y octava se ubican los valores predictivo positivo y predictivo negativo de cada modelo, por último en la novena columna se ubica el F1-Score de cada modelo.

Anexo E. Interfaz desarrollada para cargar los modelos de clasificación.

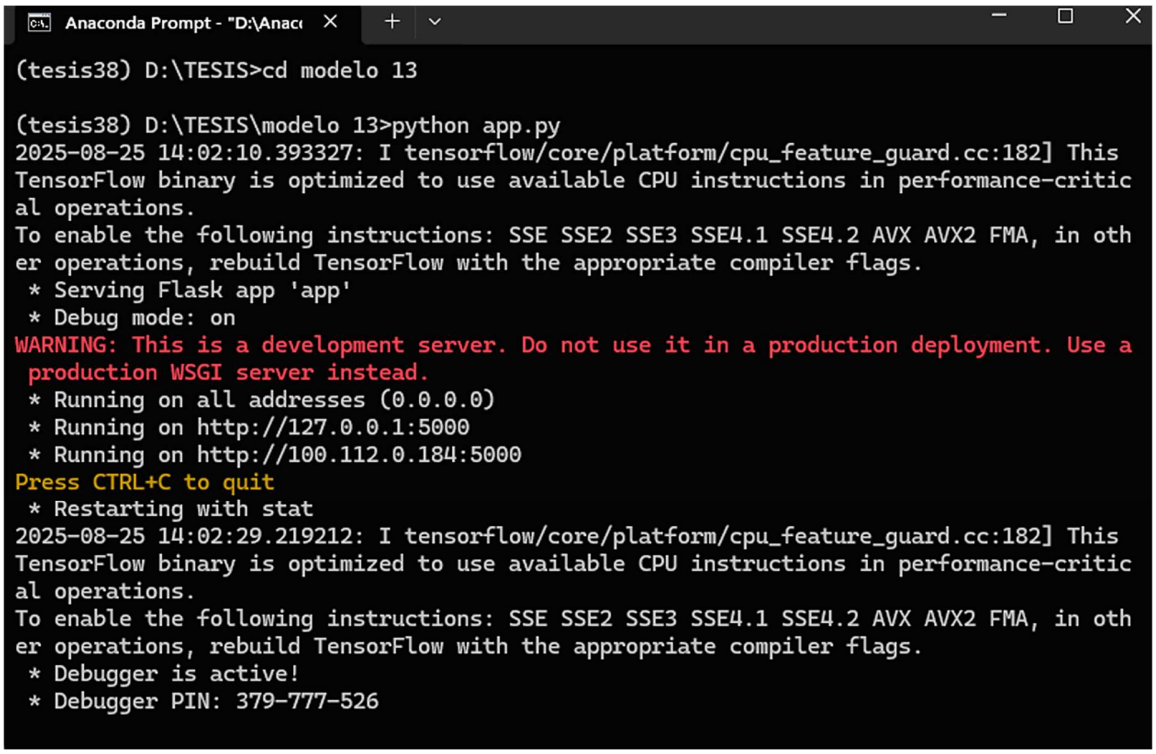


Figura 8. Captura de pantalla de la ventana de anaconda prompt para ejecutar la interfaz ya cargada con el modelo a ejecutar.

Figura 9. Captura de pantalla de la interfaz para cargar las imágenes y proceder a realizar la clasificación.

Figura 10. Captura de pantalla del archivo index que se ejecuta a través de la aplicación app.py para iniciar la interfaz gráfica en ventana de navegador.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Clasificación de Hemorragias</title>
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/cropperjs/1.5.12/cropper.min.css">
  <style>
    :root{ --accent:#00A9CE; }
    body{background:#000;color:#fff;text-align:center;font-family:Arial,
sans-serif}
    h1{color:var(--accent);font-size:36px;margin:20px 0 10px}
    form{margin-top:10px}
    input,button{font-size:16px;padding:10px;margin:10px}

button{background:var(--accent);color:#fff;border:none;cursor:pointer;border-rad
ius:6px}
    button:hover{background:#008bb0}
    #result{font-size:20px;margin:18px 0 30px}

.image-container{margin-top:12px;display:flex;justify-content:center;align-items
:center;flex-direction:column;gap:14px}
    .panel{background:#111;border:2px solid
#222;border-radius:12px;padding:10px;display:inline-block}
    img.preview{max-width:100%;max-height:420px;border:2px solid
#fff;border-radius:10px}
    img.thumb{max-width:250px;max-height:250px;border:2px solid
#fff;border-radius:14px}
    img.gradcam{
      max-width:min(95vw,1000px);
      max-height:700px;
      border:3px solid var(--accent);
      border-radius:16px;
      cursor:pointer;
      transition: transform 0.2s;
    }
    img.gradcam:hover{ transform:scale(1.02); }
    footer{margin-top:40px;font-size:14px;color:gray}
    .row{display:flex;gap:12px;flex-wrap:wrap;justify-content:center}
    #overlay{
      position:fixed;top:0;left:0;width:100%;height:100%;
      background:rgba(0,0,0,0.9);
      display:none;justify-content:center;align-items:center;
      z-index:1000;
    }
    #overlay img{
      max-width:95%;max-height:95%;
      border:4px solid var(--accent);
      border-radius:16px;
    }
  </style>
</head>
<body>
```

```
  <h1>PROYECTO CLASIFICADOR DE HEMORRAGIAS</h1>

  <form id="uploadForm" enctype="multipart/form-data">
    <div class="row">
      <input type="file" id="file" accept="image/*" required
onchange="previewImage(event)">
      <button type="button" onclick="cropImage()">Recortar Imagen</button>
      <button type="submit">Clasificar</button>
      <button type="button" onclick="resetImage()">Eliminar Imagen</button>
    </div>
  </form>

  <div class="image-container">
    <div class="panel">
      <img id="selectedImage" class="preview" style="display:none;">
    </div>
    <img id="uploadedImage" class="thumb" src="" alt="Imagen recortada"
style="display:none;">
    <img id="gradcamImage" class="gradcam" src="" alt="Grad-CAM"
style="display:none;" onclick="showOverlay(this.src)">
  </div>

  <h3 id="result"></h3>

  <footer>Hecho por Edwin Zoquiapa Galaviz para proyecto de tesis</footer>

  <div id="overlay" onclick="hideOverlay()">
    <img id="overlayImg" src="">
  </div>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/cropperjs/1.5.12/cropper.min.js"></s
cript>

  <canvas id="workCanvas" width="100" height="100"
style="display:none;"></canvas>

  <script>
    let cropper;

    function previewImage(event){
      const file = event.target.files[0];
      const reader = new FileReader();
      reader.onload = function(e){
        const image = document.getElementById("selectedImage");
        image.src = e.target.result;
        image.style.display = "block";
        if (cropper) cropper.destroy();
        cropper = new Cropper(image, { aspectRatio: 1, viewMode: 2 });
      };
      reader.readAsDataURL(file);
    }
```

```
    function cropImage(){
      if (!cropper) return;
      const croppedCanvas = cropper.getCroppedCanvas({ width: 256, height: 256
});
      const croppedImage = croppedCanvas.toDataURL("image/png");
      const imgElement = document.getElementById("uploadedImage");
      imgElement.src = croppedImage;
      imgElement.style.display = "block";
      document.getElementById("selectedImage").dataset.croppedImage =
croppedImage;
      const preDataUrl = preprocessClient(croppedCanvas);
      document.getElementById("selectedImage").dataset.preImage = preDataUrl;
    }

    function resetImage(){
      document.getElementById("file").value = "";
      for (const id of ["selectedImage","uploadedImage","gradcamImage"]) {
        const el = document.getElementById(id);
        el.src = "";
        el.style.display = "none";
      }
      document.getElementById("result").innerText = "";
      if (cropper) { cropper.destroy(); cropper = null; }
    }

    function preprocessClient(sourceCanvas){
      const targetW = 100, targetH = 100;
      const work = document.getElementById('workCanvas');
      work.width = targetW; work.height = targetH;
      const ctx = work.getContext('2d');
      ctx.drawImage(sourceCanvas, 0, 0, targetW, targetH);
      let imgData = ctx.getImageData(0, 0, targetW, targetH);
      let data = imgData.data;
      for (let i = 0; i < data.length; i += 4) {
        const r = data[i], g = data[i+1], b = data[i+2];
        const gray = Math.round(0.299*r + 0.587*g + 0.114*b);
        data[i] = data[i+1] = data[i+2] = gray;
      }
      const capHigh = 200, low = 20, high = 200, scale = 255 / (high - low);
      for (let i = 0; i < data.length; i += 4) {
        let g = data[i];
        if (g > capHigh) g = capHigh;
        let stretched = (g - low) * scale;
        if (stretched < 0) stretched = 0;
        if (stretched > 255) stretched = 255;
        data[i] = data[i+1] = data[i+2] = stretched;
      }
      const cx = targetW/2, cy = targetH/2, radius = Math.min(targetW, targetH)
* 0.47;
      for (let y = 0; y < targetH; y++){
        for (let x = 0; x < targetW; x++){
          const dx = x - cx, dy = y - cy;
          if (dx*dx + dy*dy > radius*radius) {
            const idx = (y*targetW + x)*4;
```

```
            data[idx] = data[idx+1] = data[idx+2] = 0;
          }
        }
      }
      ctx.putImageData(imgData, 0, 0);
      return work.toDataURL("image/png");
    }

    document.getElementById("uploadForm").addEventListener("submit",
function(event){
      event.preventDefault();
      const preData = document.getElementById("selectedImage").dataset.preImage;
      const cropped =
document.getElementById("selectedImage").dataset.croppedImage;
      if (!cropped) {
        alert("Por favor recorta la imagen antes de enviarla.");
        return;
      }
      const imgDataUrl = preData || cropped;
      fetch(imgDataUrl)
        .then(res => res.blob())
        .then(blob => {
          const formData = new FormData();
          formData.append("file", blob, "preprocessed.png");
          fetch("/predict", { method: "POST", body: formData })
            .then(response => response.json())
            .then(data => {
              if (data.error) {
                document.getElementById("result").innerText = "Error: " +
data.error;
              } else {
                document.getElementById("result").innerText = "Clasificado como:
" + data.prediction;
                const gradcamImg = document.getElementById("gradcamImage");
                gradcamImg.src = data.gradcam_url + "?t=" + new
Date().getTime();
                gradcamImg.style.display = "block";
              }
            })
            .catch(err => {
              console.error(err);
              document.getElementById("result").innerText = "Error al procesar
la imagen.";
            });
        });
    });

    function showOverlay(src){
      document.getElementById("overlayImg").src = src;
      document.getElementById("overlay").style.display = "flex";
    }

    function hideOverlay(){
      document.getElementById("overlay").style.display = "none";


    }
  </script>
</body>
</html>
```

Figura 11. Captura de pantalla del código en bloc de notas del archivo index.html a través del cual se ejecuta la interfaz en una ventana del navegador web predeterminado.

D: > TESIS > modelo 13 > app.py > ...

```python
# importación de librerías
from flask import Flask, request, render_template, jsonify
from tensorflow.keras.models import load_model
import tensorflow as tf
import numpy as np
import cv2
import os, uuid, time
from PIL import Image

# ==================== CONFIG ====================
MODEL_PATH   = r"D:\TESIS\modelo 13\modelo 13.h5"  # Ruta al modelo ya entrenado
UPLOAD_DIR   = os.path.join("static", "uploads")
IMG_W, IMG_H = 100, 100                            # tamaño usado al entrenar
USE_SKULL_STRIP = True                             # quitar cráneo en inferencia

os.makedirs(UPLOAD_DIR, exist_ok=True)

# Cargar modelo
model = load_model(MODEL_PATH)

app = Flask(__name__)

# ================ PREPROCESAMIENTO ================
def skull_strip_simple(gray_8u):
    """
    Quita fondo y atenúa cráneo usando la máscara del contorno mayor.
    Entrada: imagen en escala de grises uint8 (0..255)
    Salida: solo región intracraneal (8 bits)
    """
    # umbral simple para separar zonas brillantes
    _, th = cv2.threshold(gray_8u, 30, 255, cv2.THRESH_BINARY)
    th = cv2.morphologyEx(th, cv2.MORPH_CLOSE, np.ones((5,5), np.uint8))
    cnts, _ = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if not cnts:
        return gray_8u
    c = max(cnts, key=cv2.contourArea)
    mask = np.zeros_like(gray_8u)
    cv2.drawContours(mask, [c], -1, 255, thickness=-1)
    cerebro = cv2.bitwise_and(gray_8u, mask)
    return cerebro

def window_and_normalize(gray_8u, low=20, high=200, cap_high=200):
    """
    Ventana cerebral simple:
    - recorta intensidades muy altas (hueso) a 'cap_high'
    - estira el rango [low, high] a [0,1]
    """
    g = gray_8u.astype(np.float32)
    g = np.minimum(g, float(cap_high))
    g = (g - float(low)) / float(high - low)
    g = np.clip(g, 0.0, 1.0)
    return g

def preprocess_image(path_or_bytes, do_skull_strip=USE_SKULL_STRIP):
    """
    Lee PNG/JPG (desde ruta o bytes), aplica: gris -> (opcional) skull-strip ->
    resize 100x100 -> ventana + normaliza [0,1].
    Devuelve:
    x: tensor (1,100,100,1) float32
    gray_8u: imagen 8 bits (100x100) para overlay del Grad-CAM
    """
    # Cargar imagen mediante ruta o bytes
    if isinstance(path_or_bytes, (bytes, bytearray)):
        img = Image.open(io.BytesIO(path_or_bytes)).convert("RGB")
        arr = np.array(img)
    else:
        arr = cv2.imread(path_or_bytes, cv2.IMREAD_COLOR)
        arr = cv2.cvtColor(arr, cv2.COLOR_BGR2RGB)

    gray = cv2.cvtColor(arr, cv2.COLOR_RGB2GRAY)

    if do_skull_strip:
        gray = skull_strip_simple(gray)

    gray = cv2.resize(gray, (IMG_W, IMG_H), interpolation=cv2.INTER_AREA)

    g01 = window_and_normalize(gray, low=20, high=200, cap_high=200)
    x = g01[np.newaxis, ..., np.newaxis].astype(np.float32)
    return x, gray

# ================== GRAD-CAM ====================
def get_last_conv_layer_name(m):
    for layer in reversed(m.layers):
        if isinstance(layer, tf.keras.layers.Conv2D):
            return layer.name
    return None
```

```python
def grad_cam(model, img_tensor, last_conv_name):
    """
    img_tensor: (1,H,W,1) float32 [0,1]
    Devuelve heatmap 8-bits (H,W) en [0..255]
    """
    # Modelo que expone la última convolución y la salida final
    grad_model = tf.keras.models.Model(
        [model.inputs],
        [model.get_layer(last_conv_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_out, preds = grad_model(img_tensor)
        class_channel = preds[:, 0]

    grads = tape.gradient(class_channel, conv_out)

    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    conv_out = conv_out[0]
    heatmap = tf.reduce_sum(conv_out * pooled_grads, axis=-1)

    heatmap = tf.maximum(heatmap, 0)
    maxval  = tf.reduce_max(heatmap)
    heatmap = heatmap / (maxval + 1e-8)

    heatmap = heatmap.numpy()
    heatmap = cv2.resize(heatmap, (IMG_W, IMG_H))
    heatmap = (heatmap * 255).astype(np.uint8)
    return heatmap

def overlay_heatmap(gray_base_8u, heatmap_8u):
    color = cv2.applyColorMap(heatmap_8u, cv2.COLORMAP_JET)
    base  = cv2.cvtColor(gray_base_8u, cv2.COLOR_GRAY2BGR)
    return cv2.addWeighted(base, 0.6, color, 0.4, 0)

# ==================== RUTAS ====================
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No se envió imagen'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No se seleccionó imagen'}), 400

    filename = f"{uuid.uuid4().hex}.png"
    filepath = os.path.join(UPLOAD_DIR, filename)
    os.makedirs(UPLOAD_DIR, exist_ok=True)
    file.save(filepath)

    try:
        x, gray_for_cam = preprocess_image(filepath, do_skull_strip=USE_SKULL_STRIP)
```

```
145
146         # Predicción
147         prob = float(model.predict(x, verbose=0)[0][0])   # prob ∈ [0,1]
148         resultado = "Con Hemorragia" if prob >= 0.5 else "Sin Hemorragia"
149
150         # Grad-CAM
151         gradcam_url = None
152         last_conv = get_last_conv_layer_name(model)
153         if last_conv:
154             heat = grad_cam(model, x, last_conv)
155             overlay = overlay_heatmap(gray_for_cam, heat)
156             grad_name = f"gradcam_{int(time.time())}_{filename}"
157             grad_path = os.path.join(UPLOAD_DIR, grad_name)
158             cv2.imwrite(grad_path, overlay)
159             gradcam_url = f"/static/uploads/{grad_name}"
160
161         return jsonify({
162             'prediction': resultado,
163             'probability': round(prob, 4),
164             'image_url': f"/static/uploads/{filename}",
165             'gradcam_url': gradcam_url
166         })
167
168     except Exception as e:
169         return jsonify({'error': str(e)}), 500
170
171 # ==================== función MAIN ======================
172 if __name__ == '__main__':
173     # host 0.0.0.0 para acceder desde otros dispositivos a través del servidor local
174     app.run(host="0.0.0.0", port=5000, debug=True)
```

Figura 12. Captura del código en visual studio code que permite el despliegue de la interfaz en una ventana del navegador web predeterminado.

Anexo F. Diagrama de flujo del modelo CNN-13.

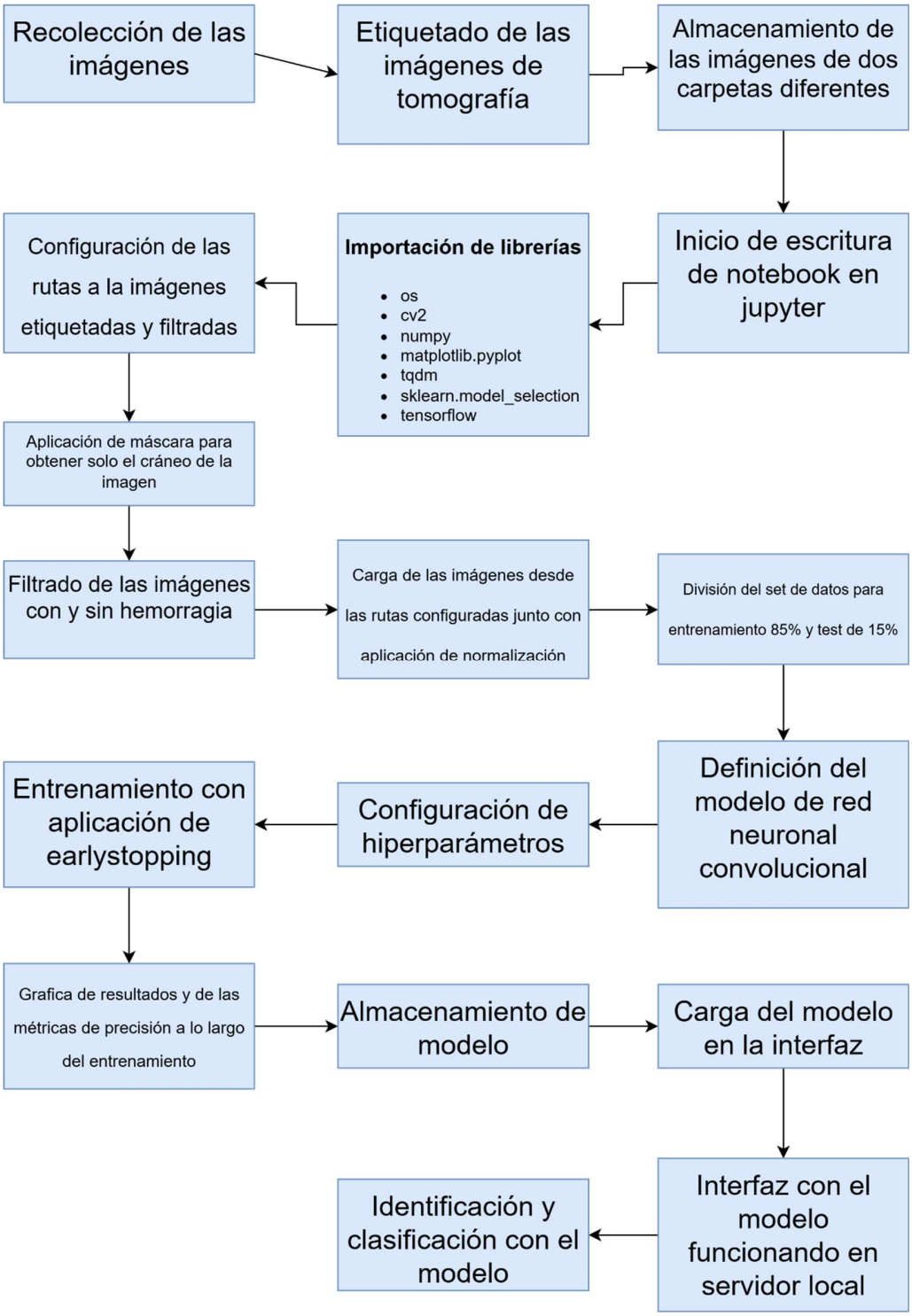


Figura 13. Diagrama de flujo de la ejecución del desarrollo del modelo.

Anexo H: Entrenamiento, métricas de precisión y curva ROC de cada uno de los modelos desarrollados y entrenados.

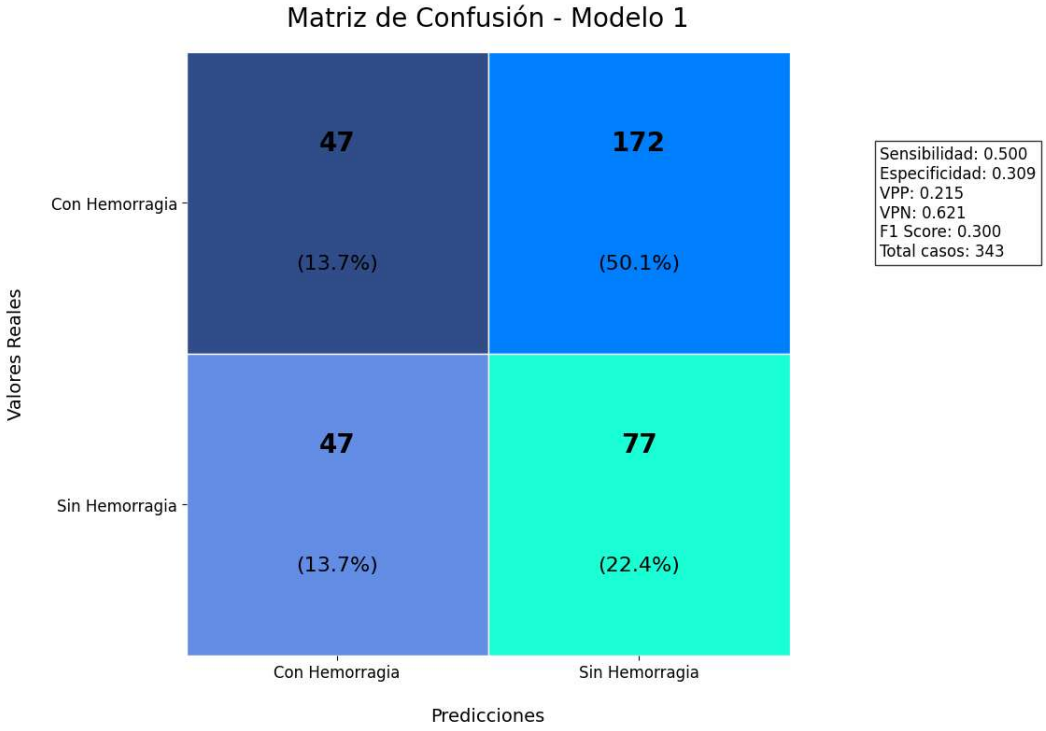


Figura 14. Matriz de confusión de los casos analizados con el modelo 1.

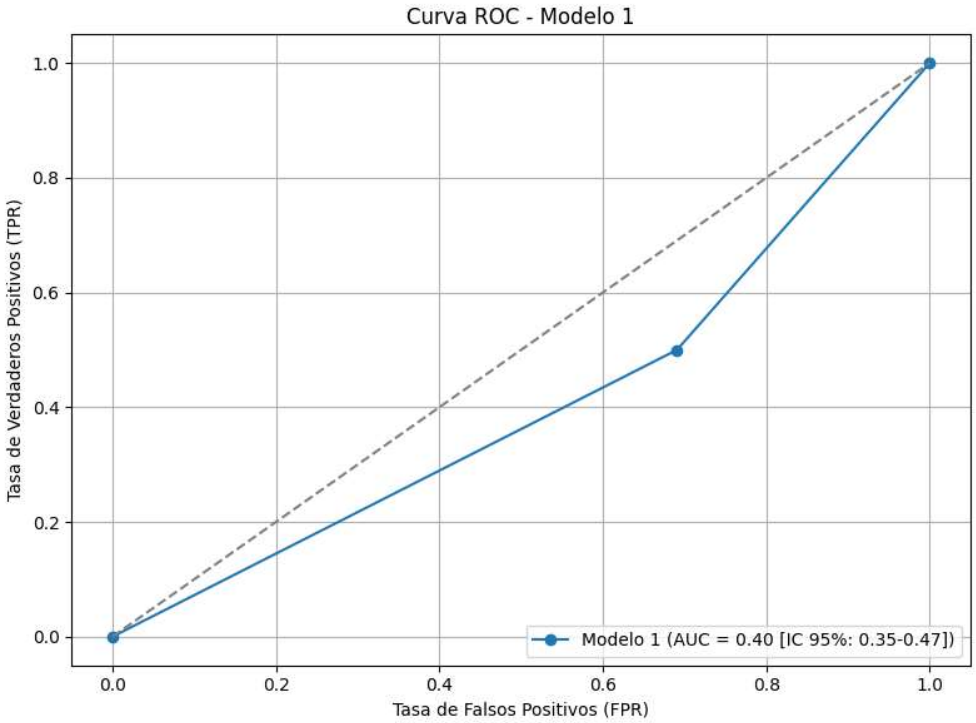


Figura 15. Curva AUC-ROC evaluando el desempeño del modelo 1 de clasificación.

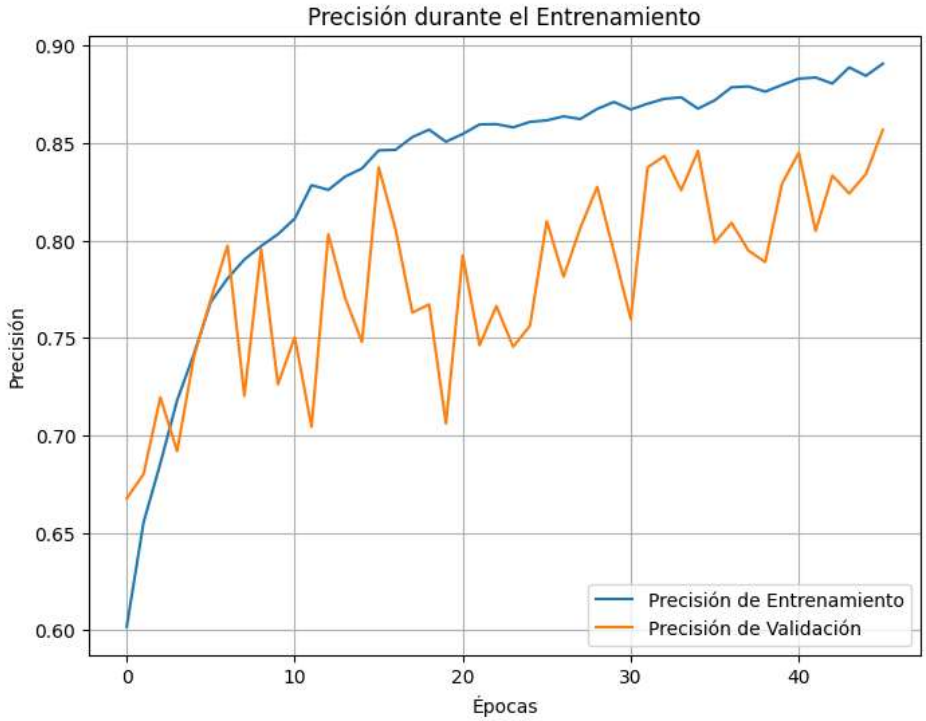


Figura 16. Evolución de la precisión del Modelo 1 durante el Entrenamiento y Validación.

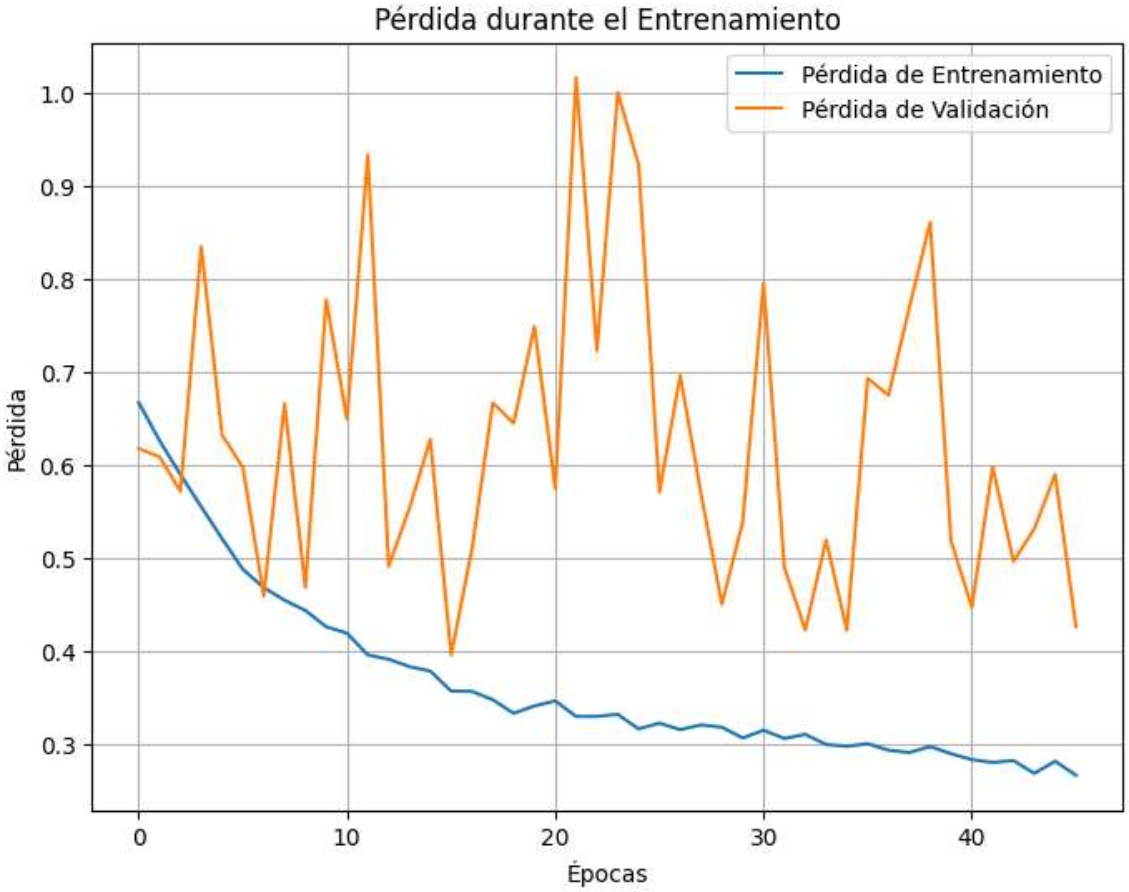


Figura 17. Evolución de la pérdida del Modelo 1 durante el Entrenamiento y Validación.

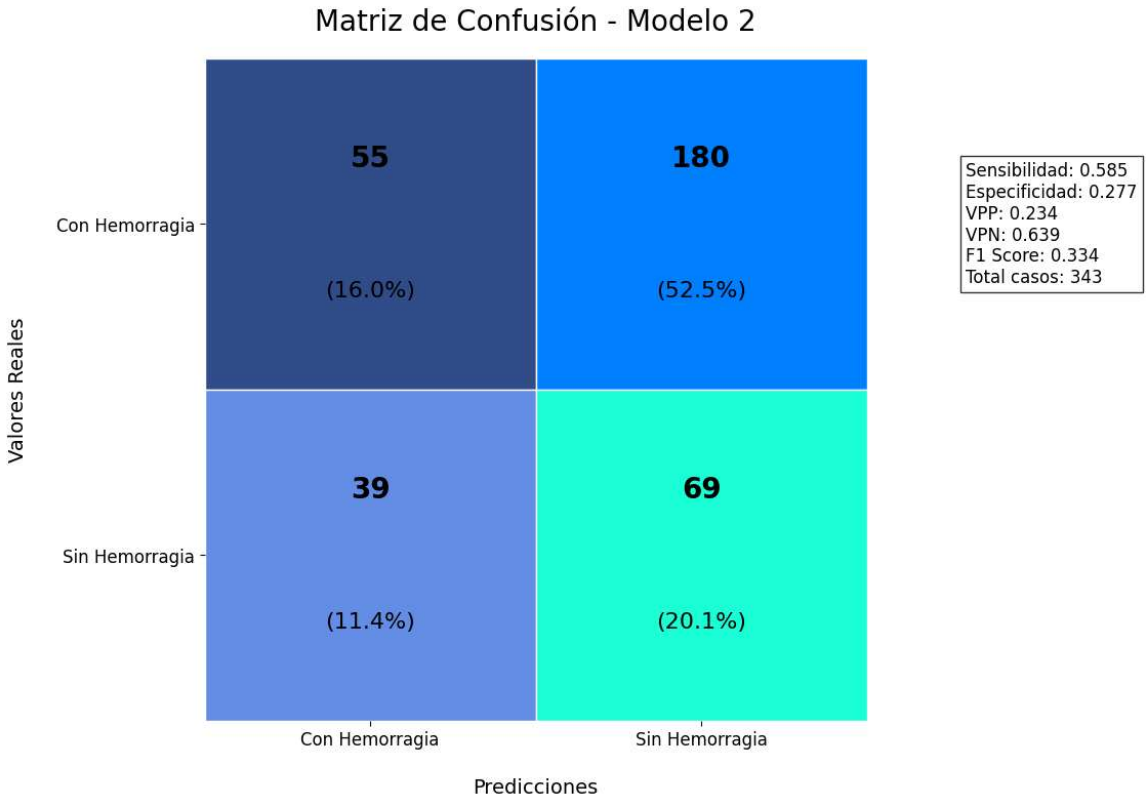


Figura 18. Matriz de confusión de los casos analizados con el modelo 2.

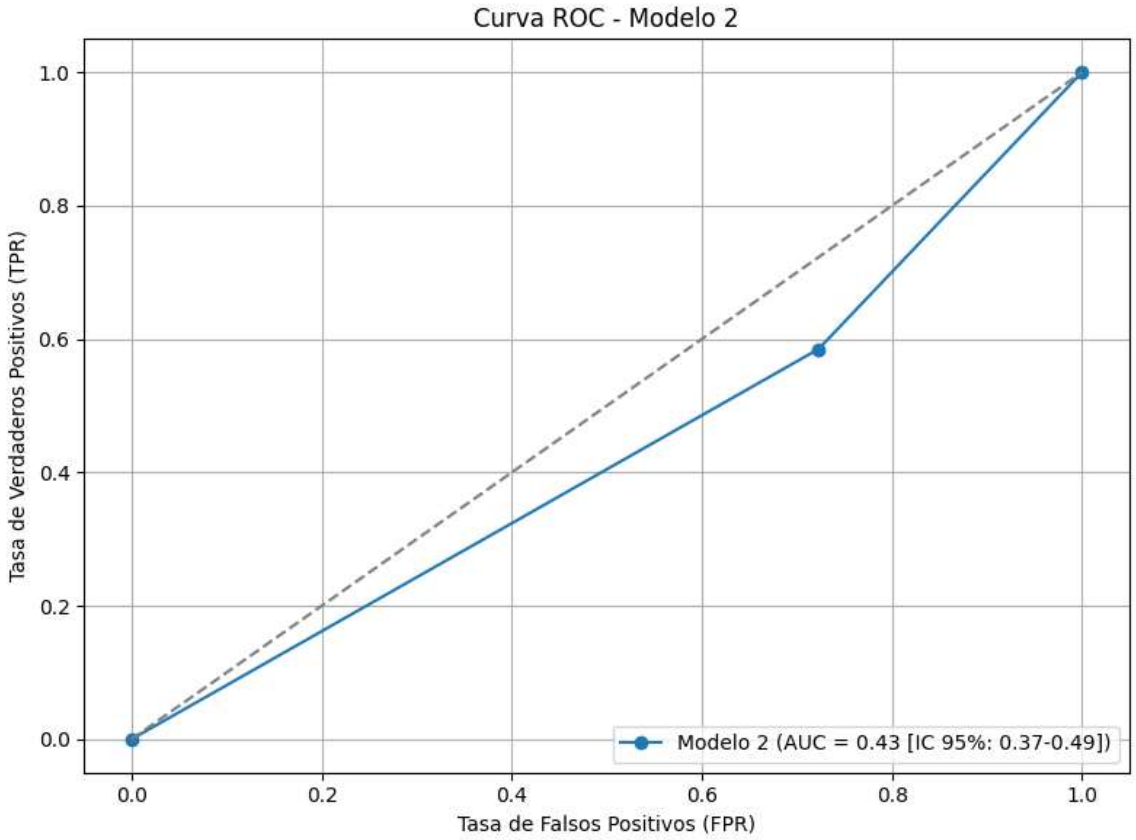


Figura 19. Curva AUC-ROC evaluando el desempeño del modelo 2 de clasificación.

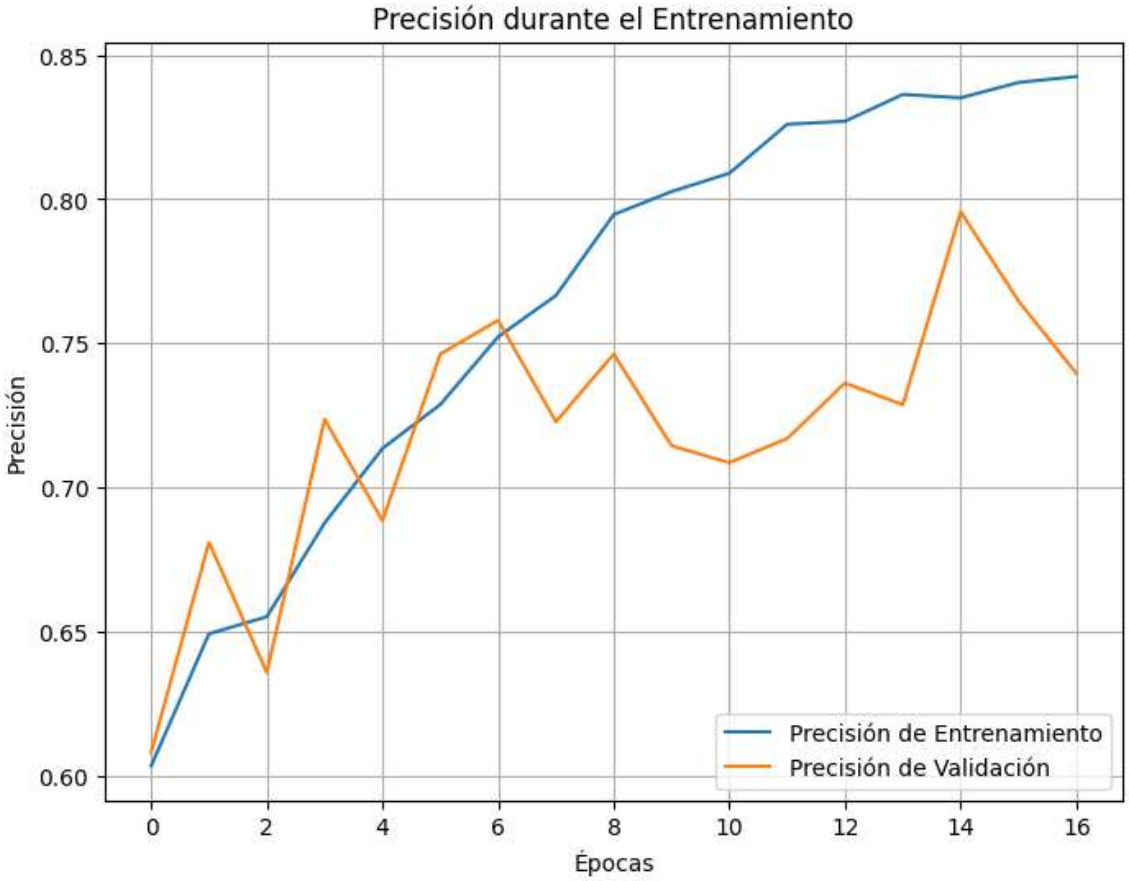


Figura 20. Evolución de la precisión del Modelo 2 durante el Entrenamiento y Validación.

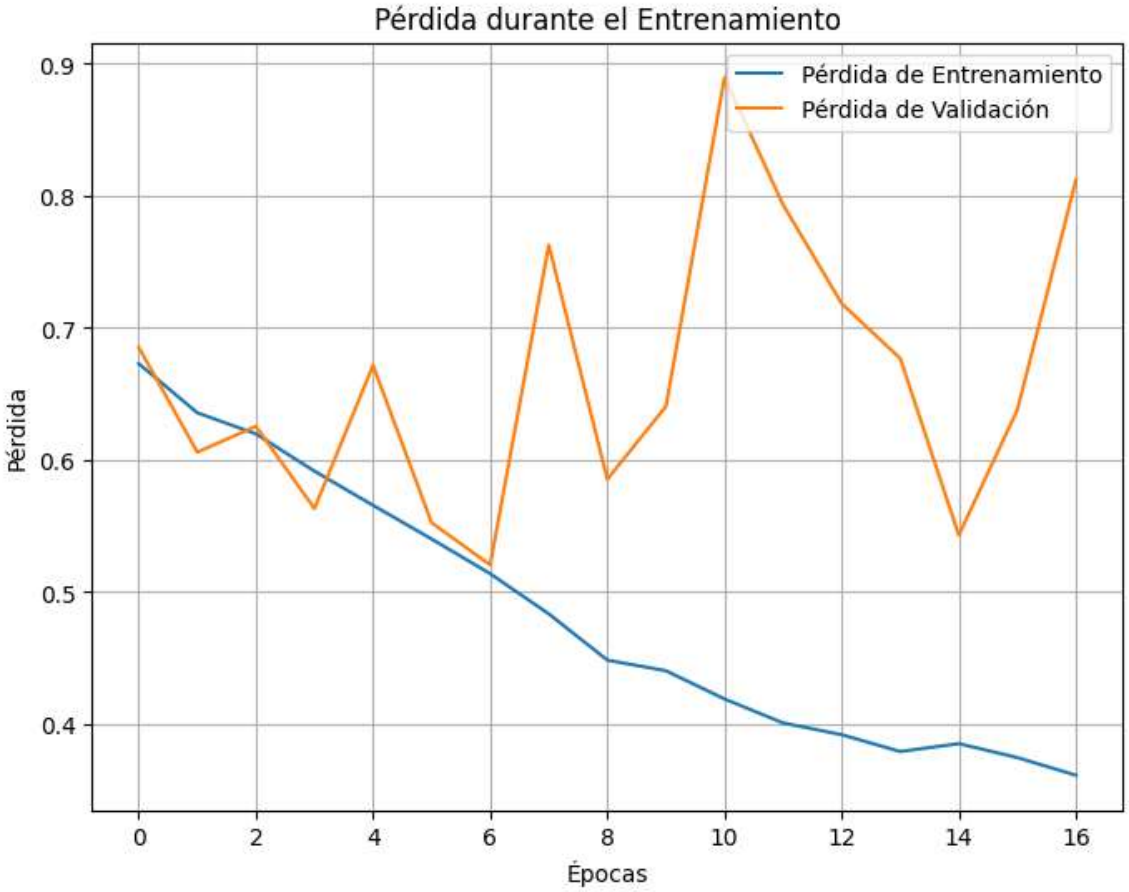


Figura 21. Evolución de la pérdida del Modelo 2 durante el Entrenamiento y Validación.

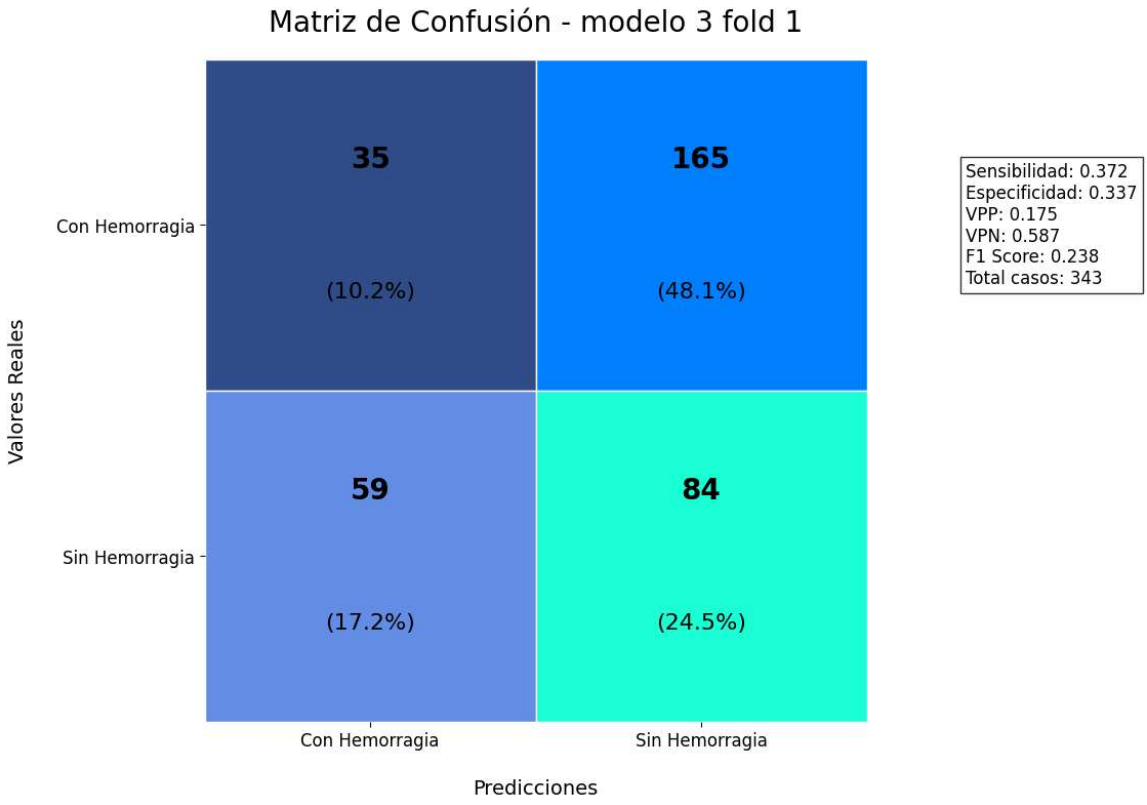


Figura 22. Matriz de confusión de los casos analizados con el modelo 3 (fold 1).

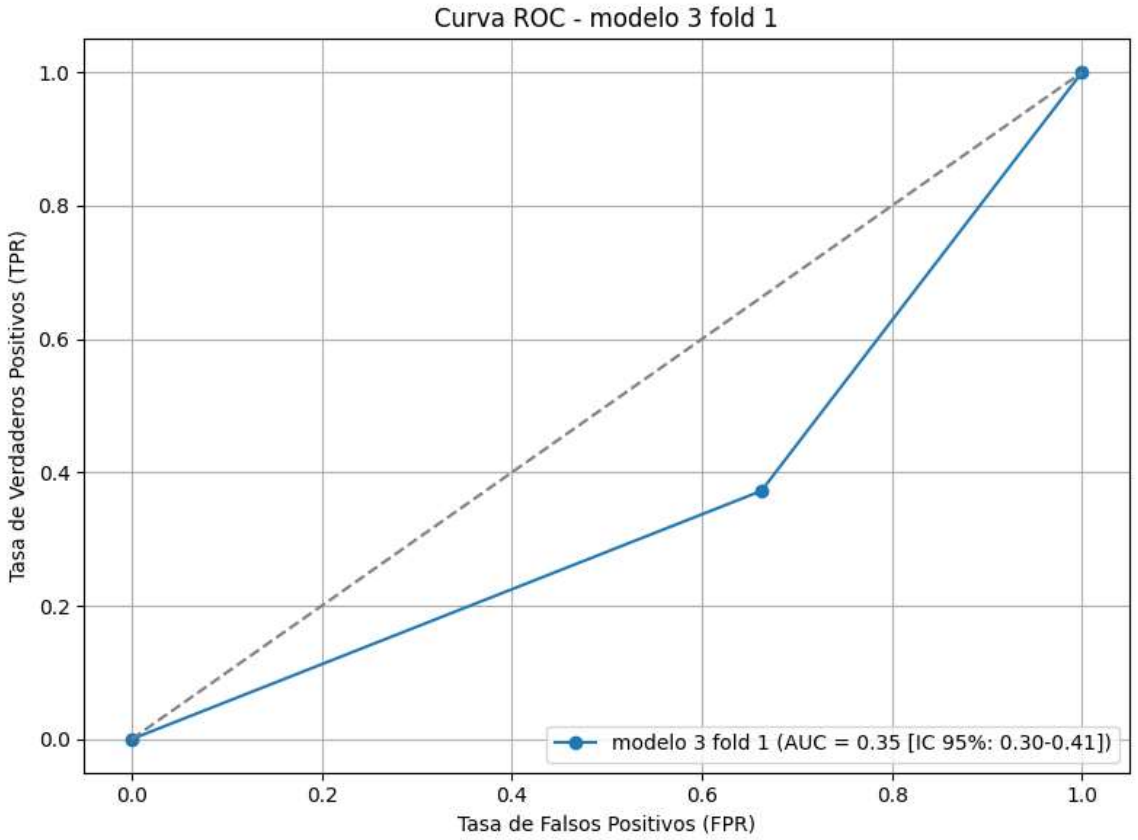


Figura 23. Curva AUC-ROC evaluando el desempeño del modelo 3 de clasificación (fold 1).

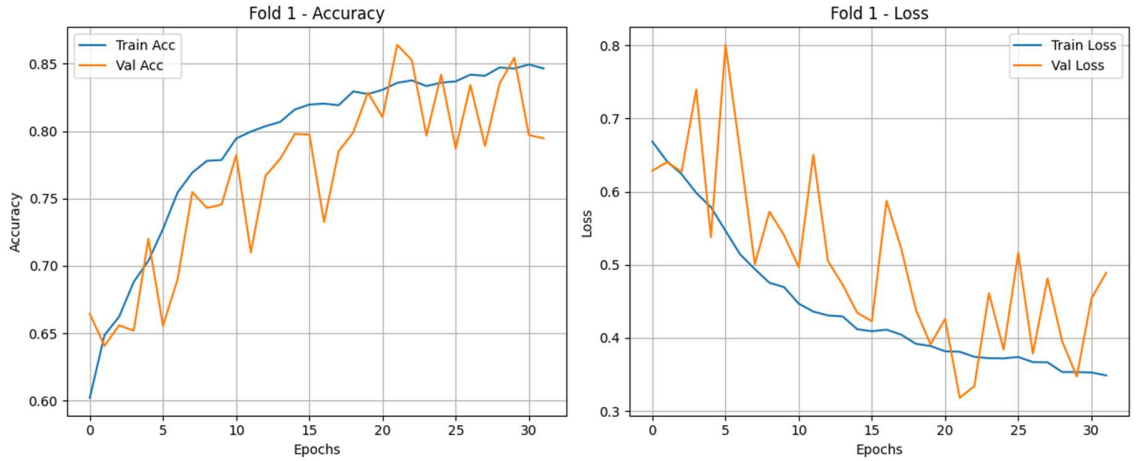


Figura 24. Evolución de la precisión del Modelo 3 durante el Entrenamiento y Validación (fold 1).

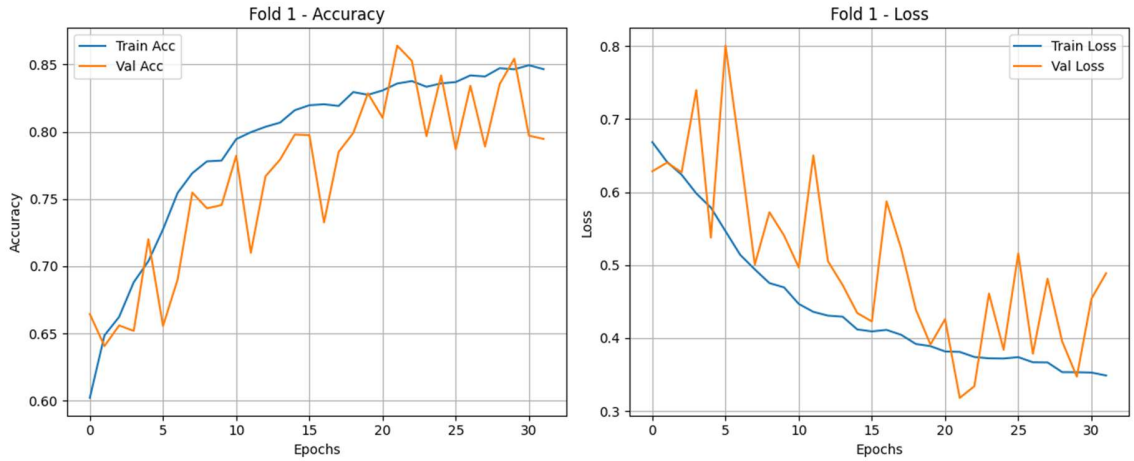


Figura 25. Evolución de la pérdida del Modelo 3 durante el Entrenamiento y Validación (fold 1).

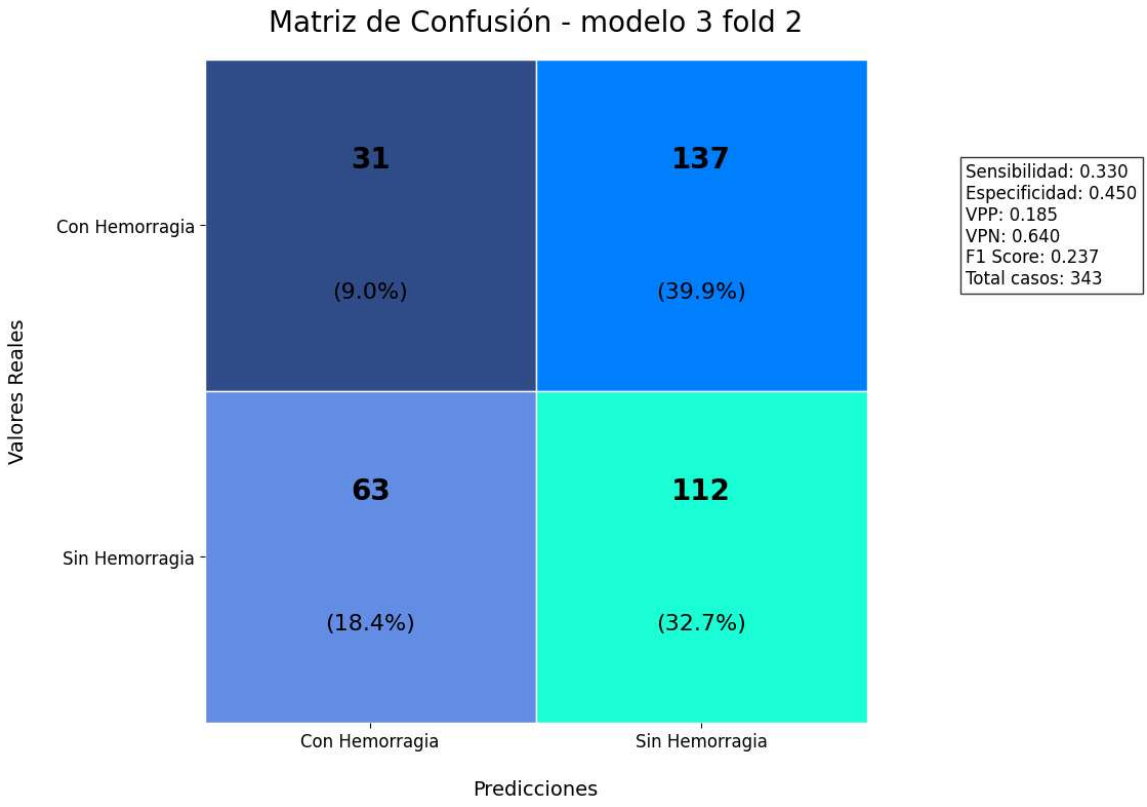


Figura 26. Matriz de confusión de los casos analizados con el modelo 3 (fold 2).

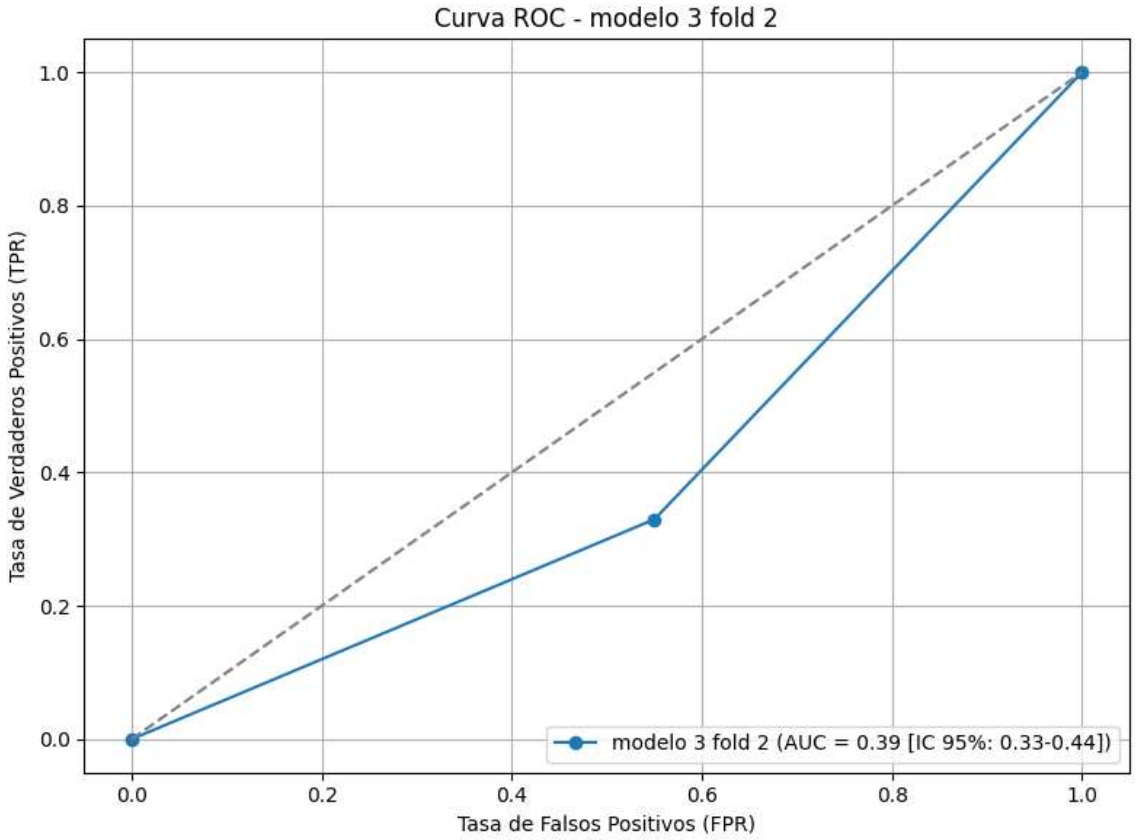


Figura 27. Curva AUC-ROC evaluando el desempeño del modelo 3 de clasificación (fold 2).

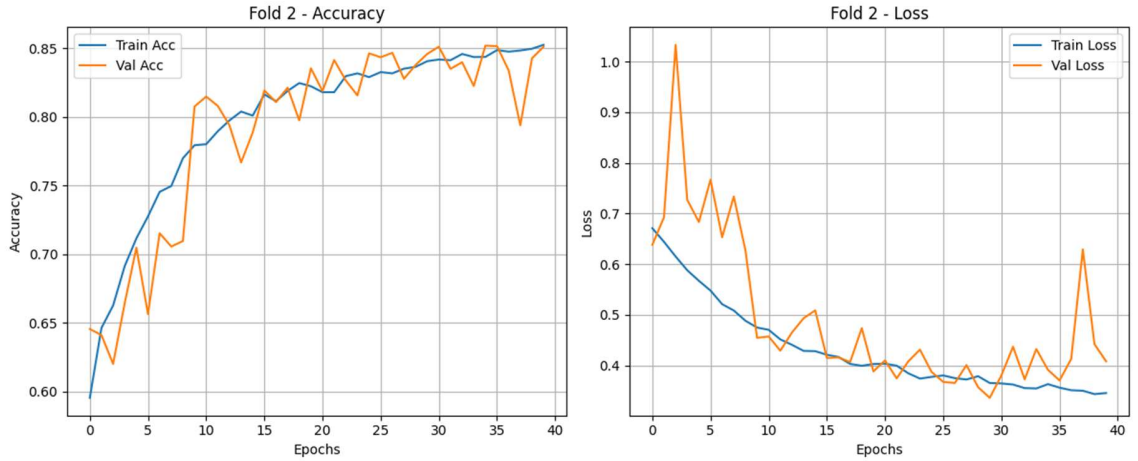


Figura 28. Evolución de la precisión del Modelo 3 durante el Entrenamiento y Validación (fold 2).

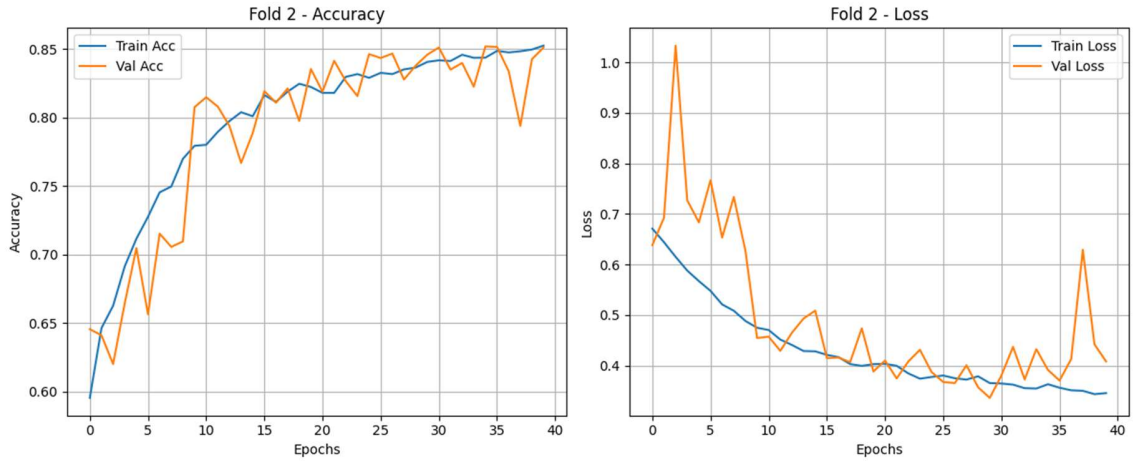


Figura 29. Evolución de la pérdida del Modelo 3 durante el Entrenamiento y Validación (fold 2).

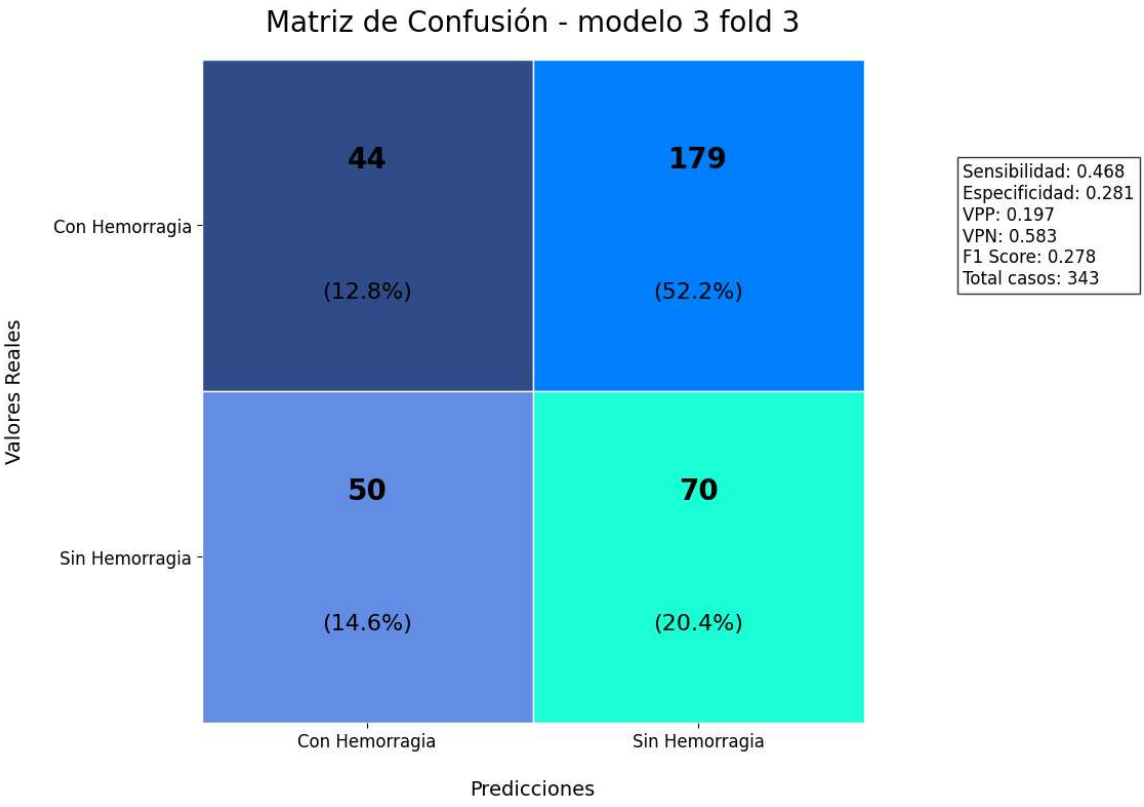


Figura 30. Matriz de confusión de los casos analizados con el modelo 3 (fold 3).

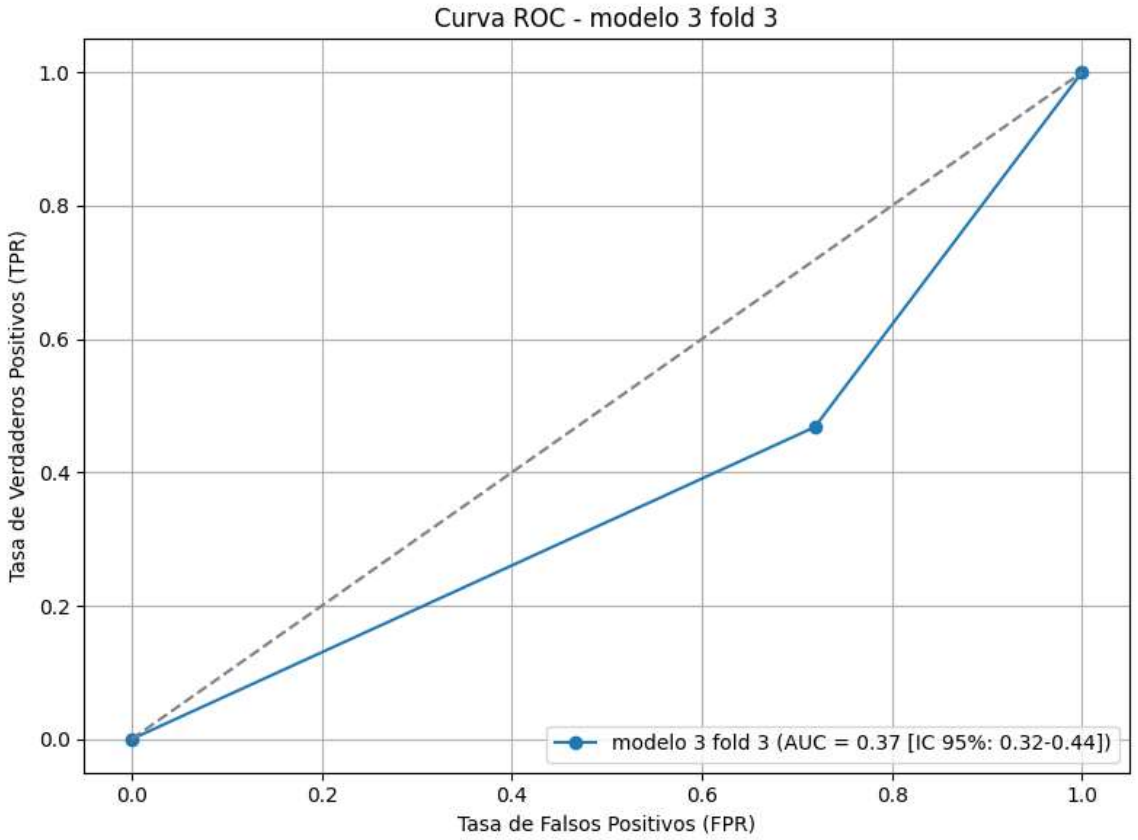


Figura 31. Curva AUC-ROC evaluando el desempeño del modelo 3 de clasificación (fold 3).

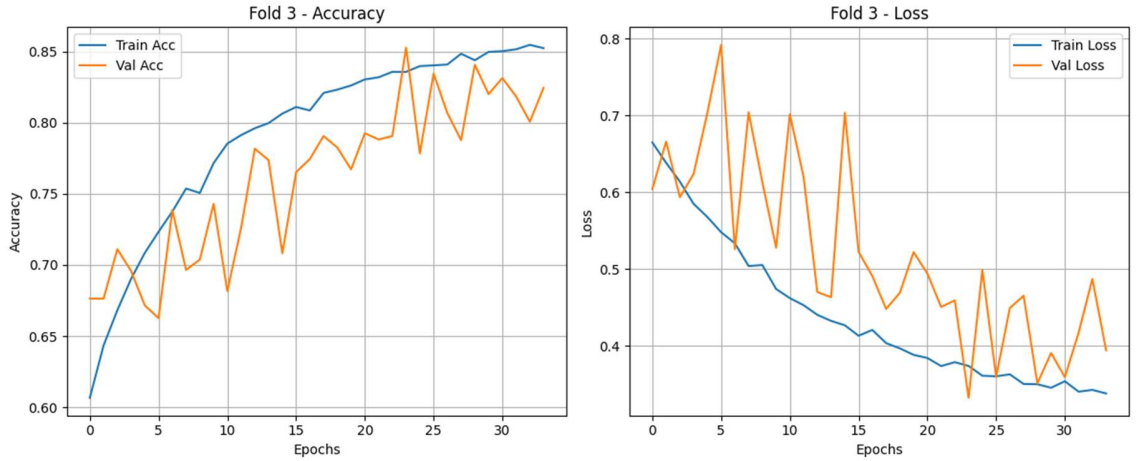


Figura 32. Evolución de la precisión del Modelo 3 durante el Entrenamiento y Validación (fold 3).

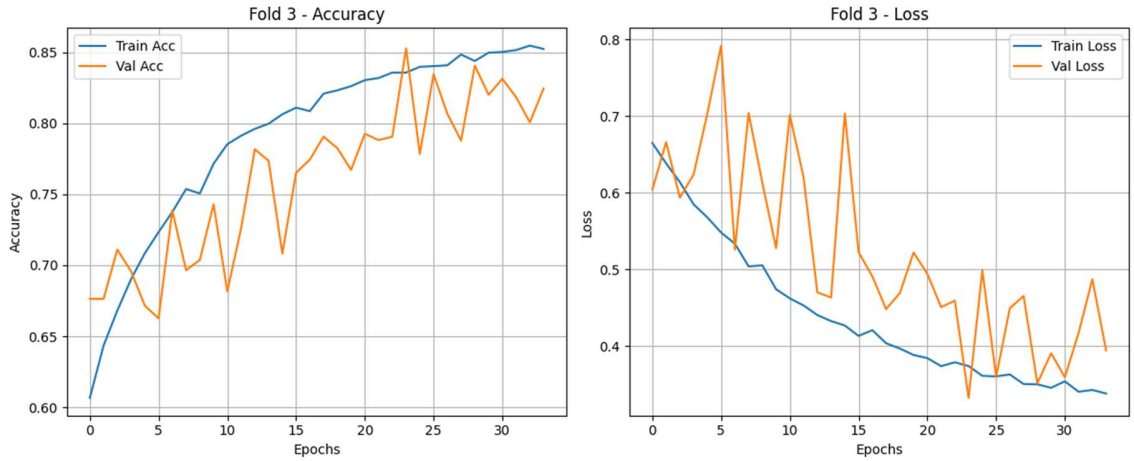


Figura 33. Evolución de la pérdida del Modelo 3 durante el Entrenamiento y Validación (fold 3).

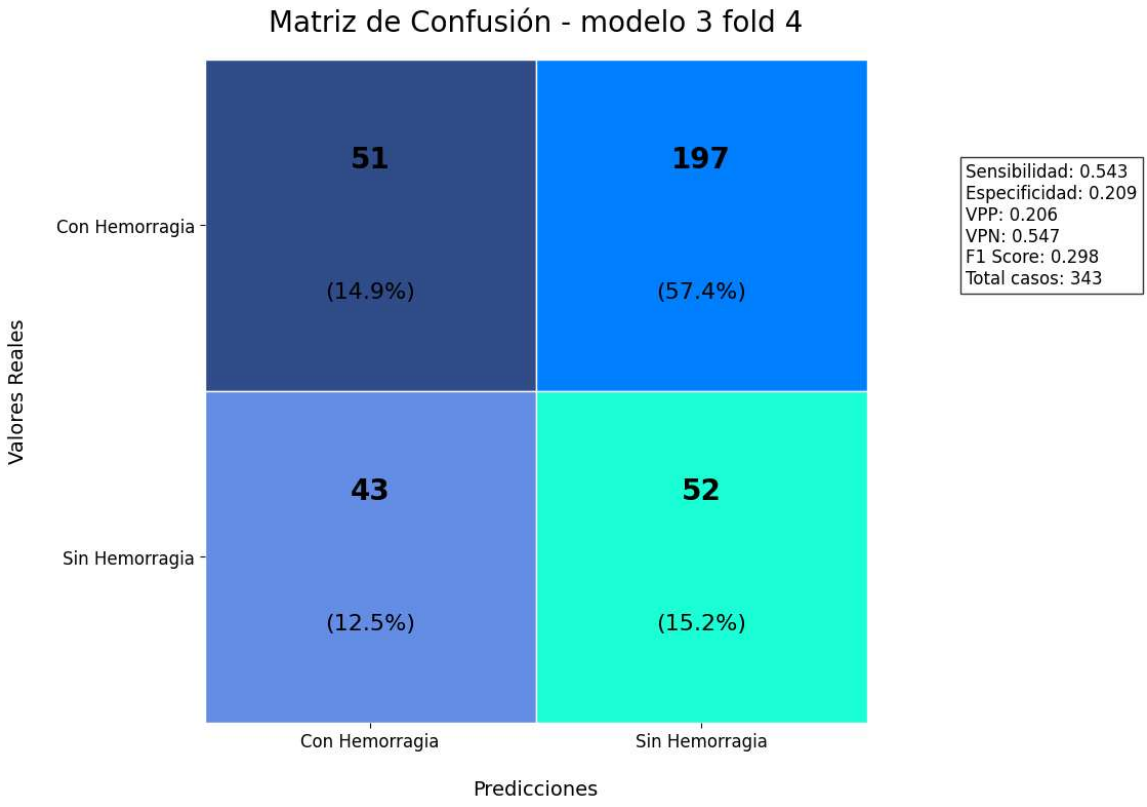


Figura 34. Matriz de confusión de los casos analizados con el modelo 3 (fold 4).

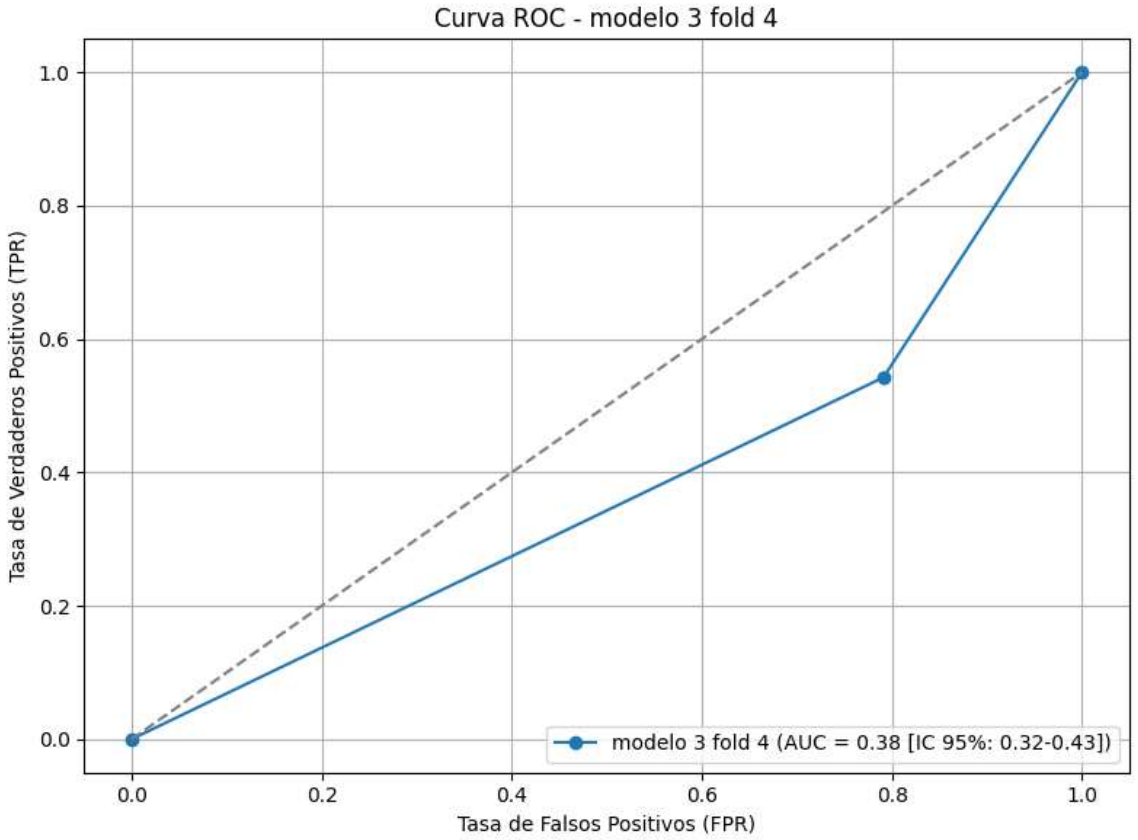


Figura 35. Curva AUC-ROC evaluando el desempeño del modelo 3 de clasificación (fold 4).

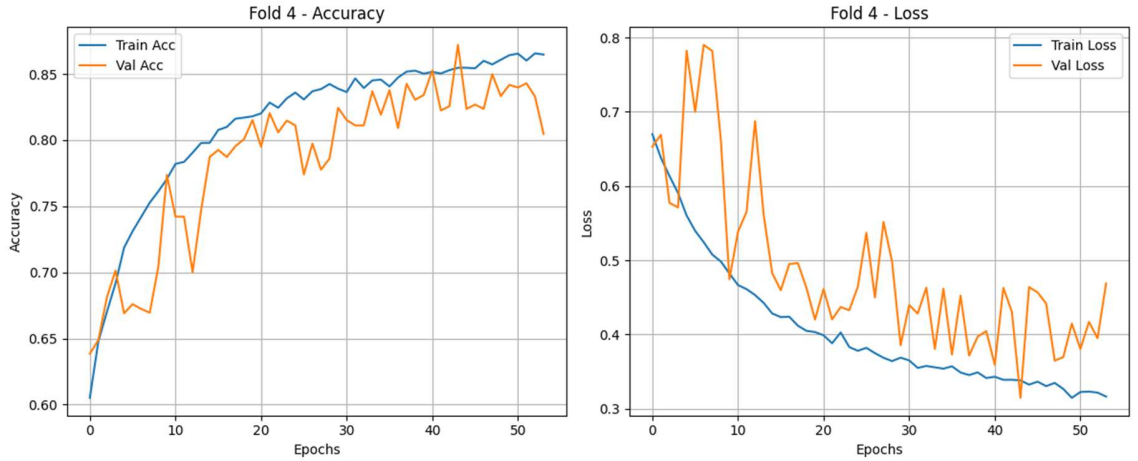


Figura 36. Evolución de la precisión del Modelo 3 durante el Entrenamiento y Validación (fold 4).

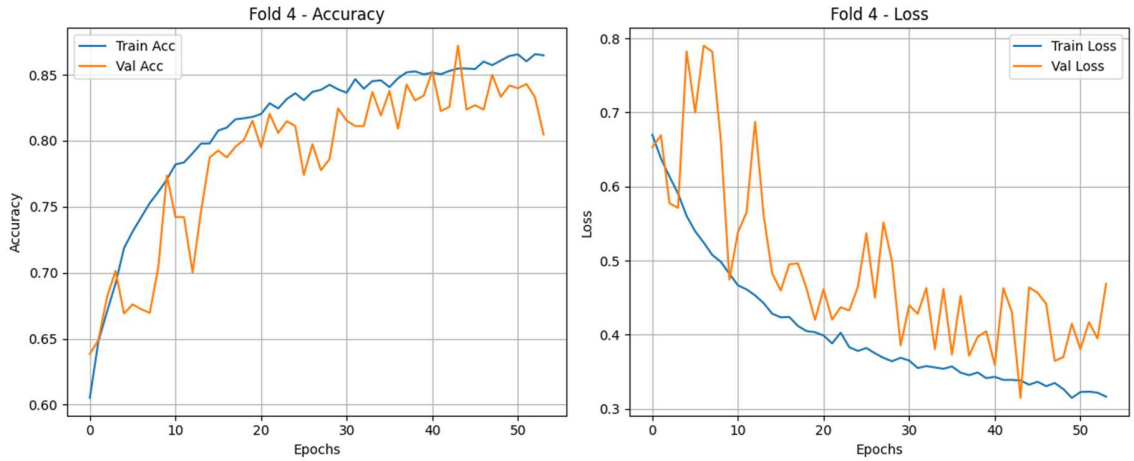


Figura 37. Evolución de la pérdida del Modelo 3 durante el Entrenamiento y Validación (fold 4).

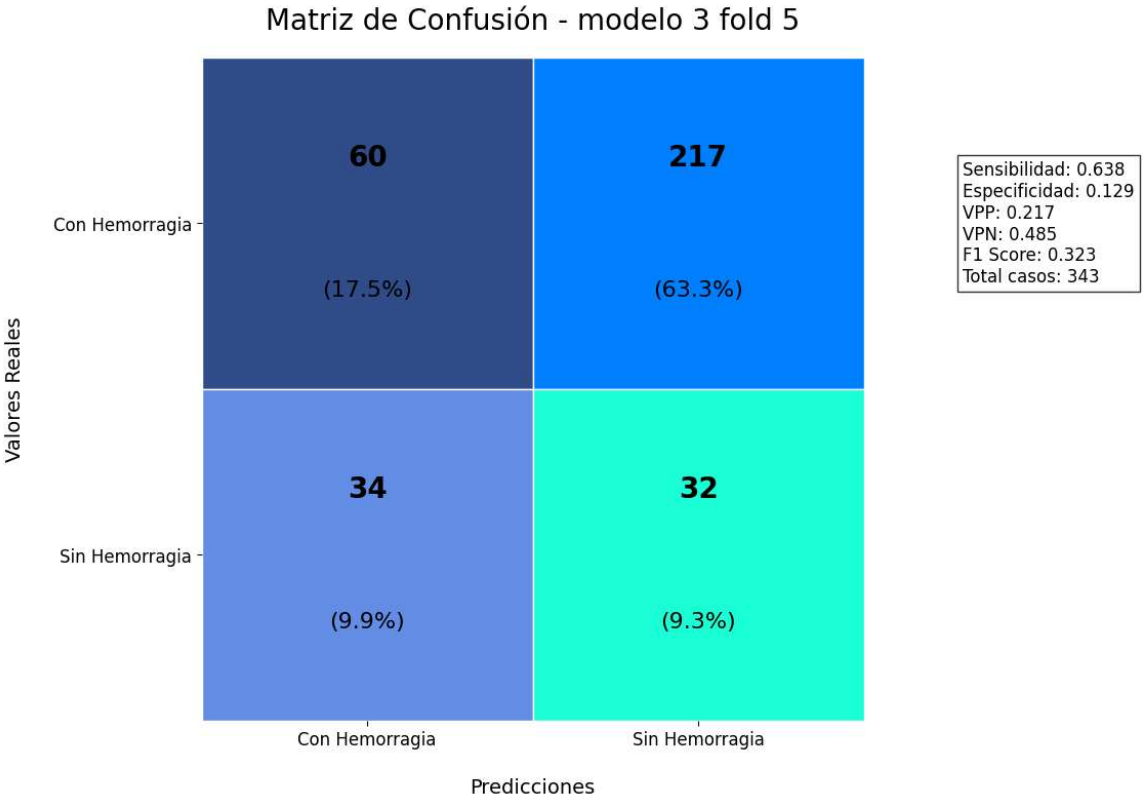


Figura 38. Matriz de confusión de los casos analizados con el modelo 3 (fold 5).

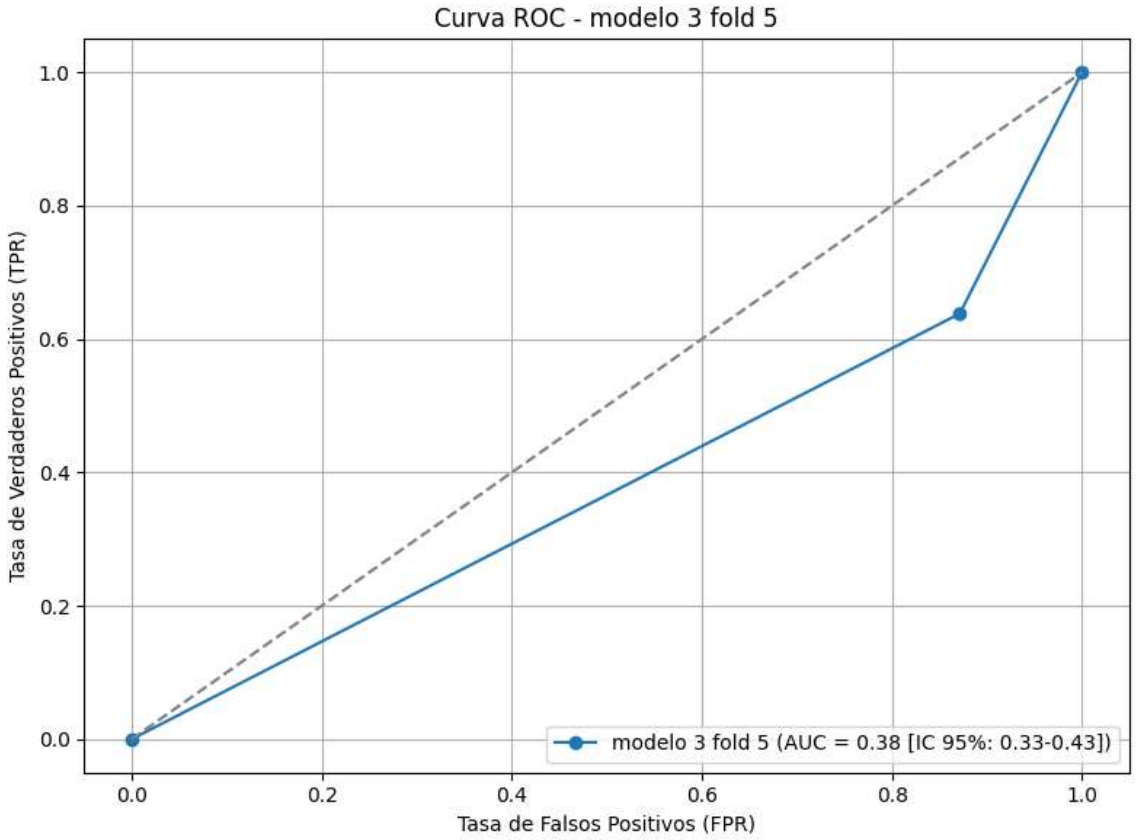


Figura 39. Curva AUC-ROC evaluando el desempeño del modelo 3 de clasificación (fold 5).

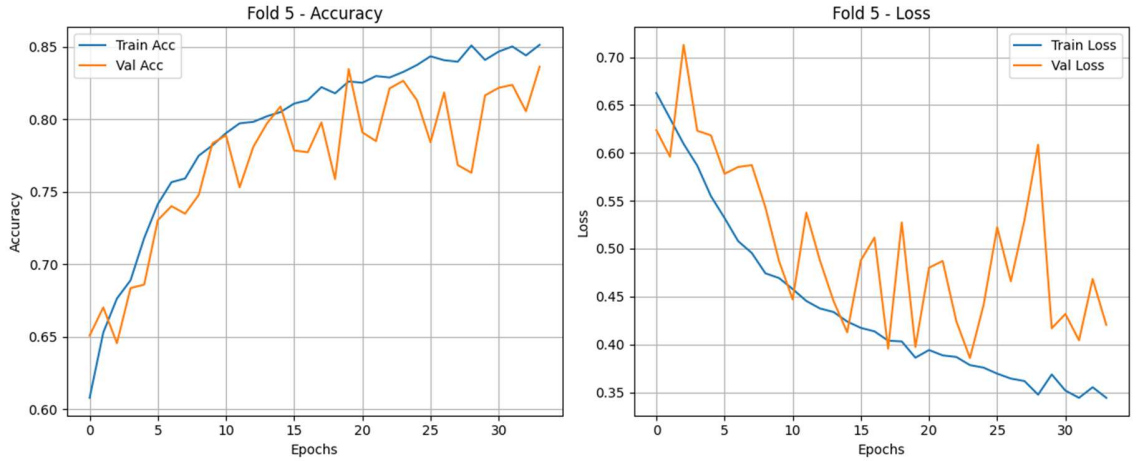


Figura 40. Evolución de la precisión del Modelo 3 durante el Entrenamiento y Validación (fold 5).

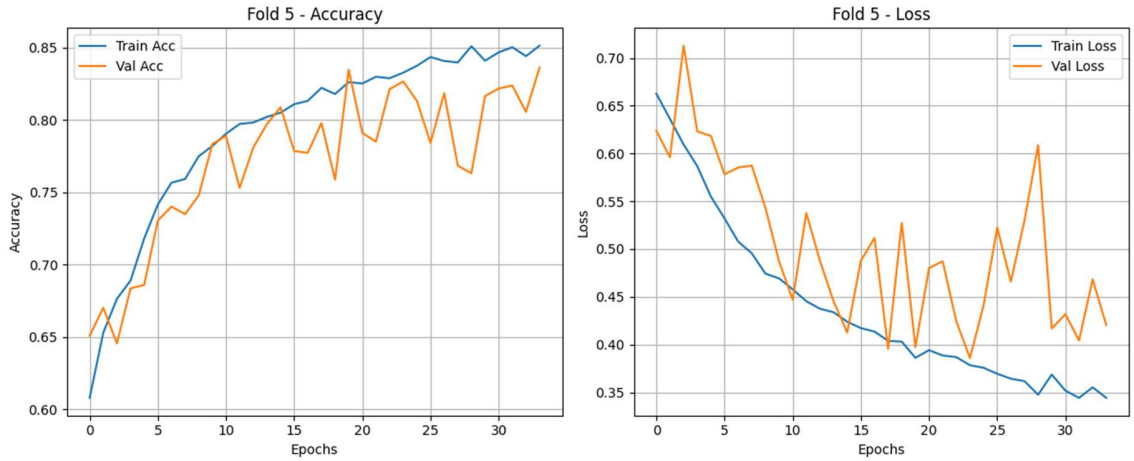


Figura 41. Evolución de la pérdida del Modelo 3 durante el Entrenamiento y Validación (fold 5).

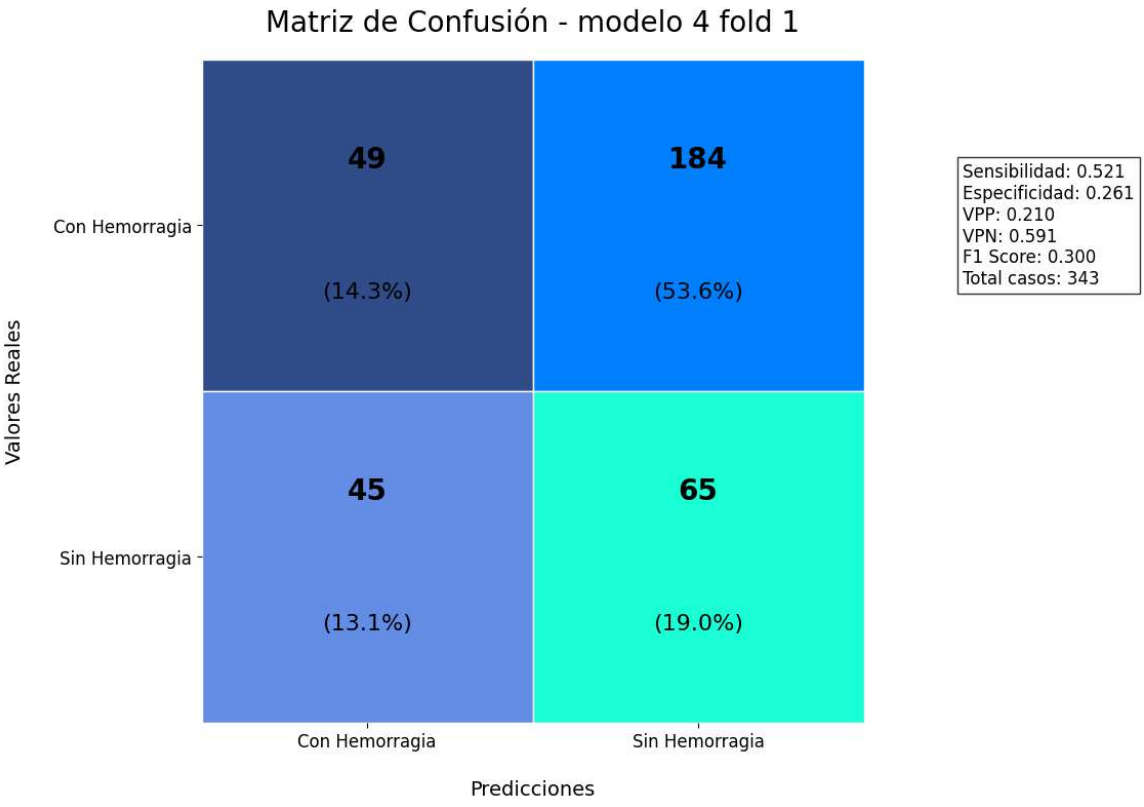


Figura 42. Matriz de confusión de los casos analizados con el modelo 4 (fold 1).

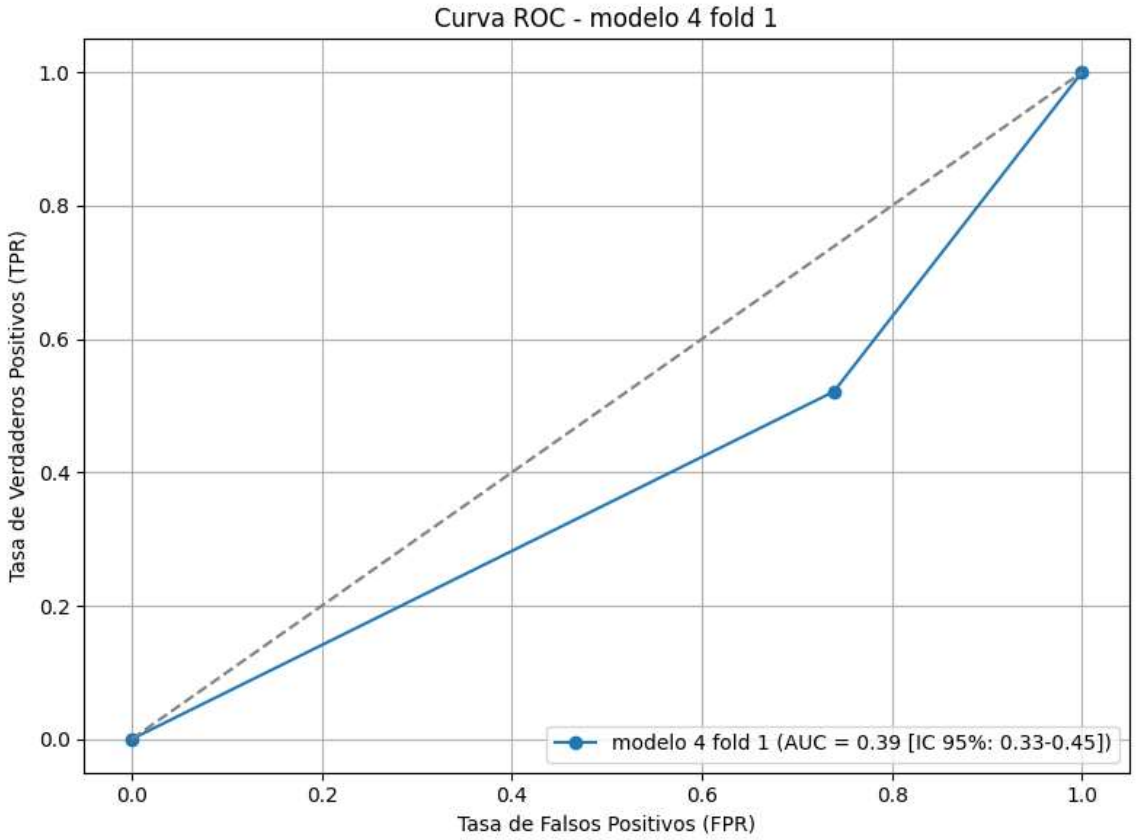


Figura 43. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 1).

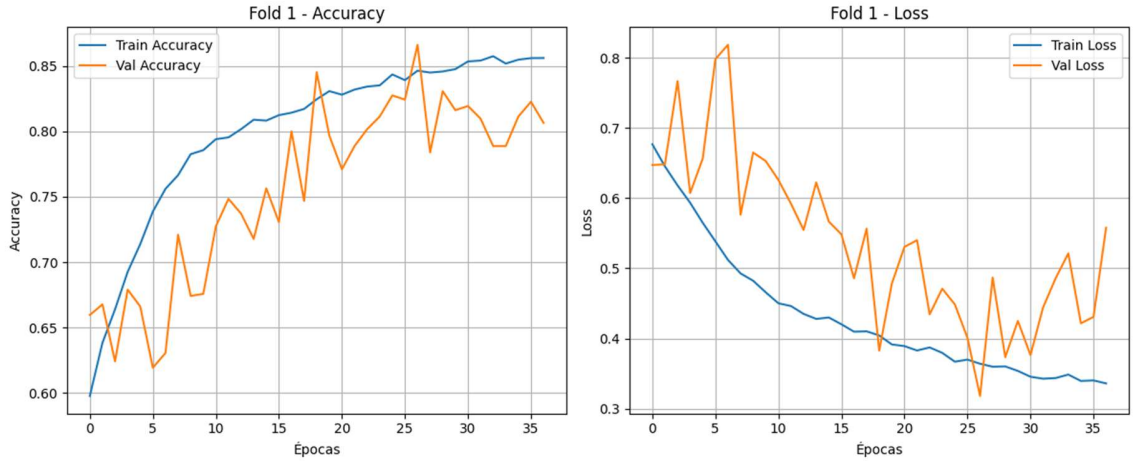


Figura 44. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 1).

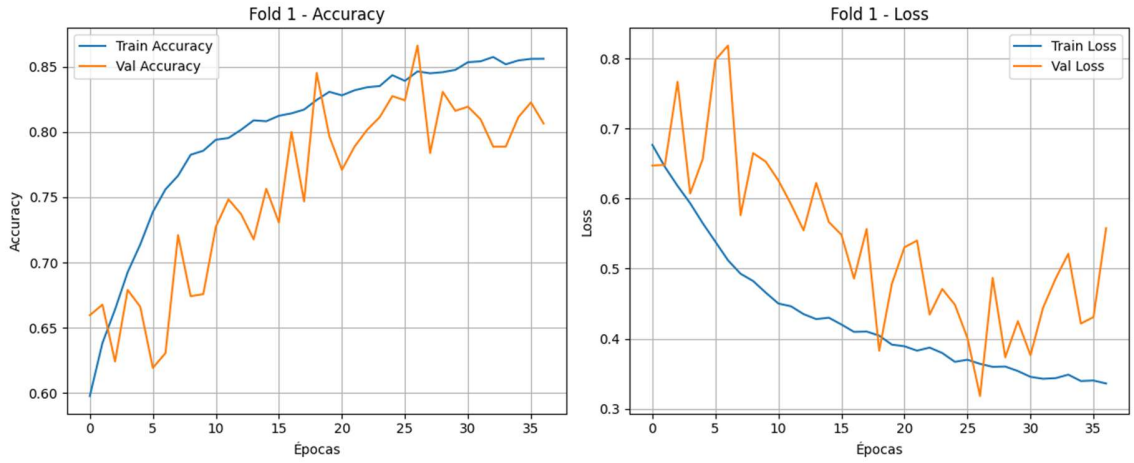


Figura 45. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 1).

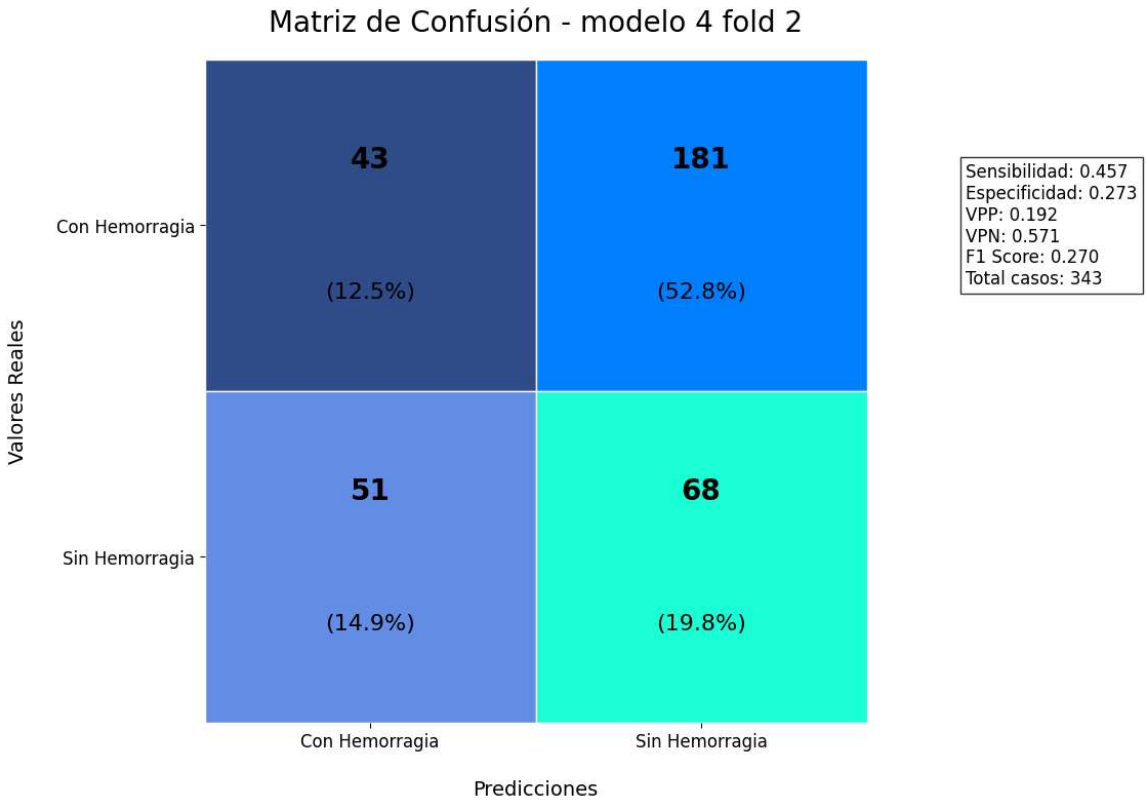


Figura 46. Matriz de confusión de los casos analizados con el modelo 4 (fold 2).

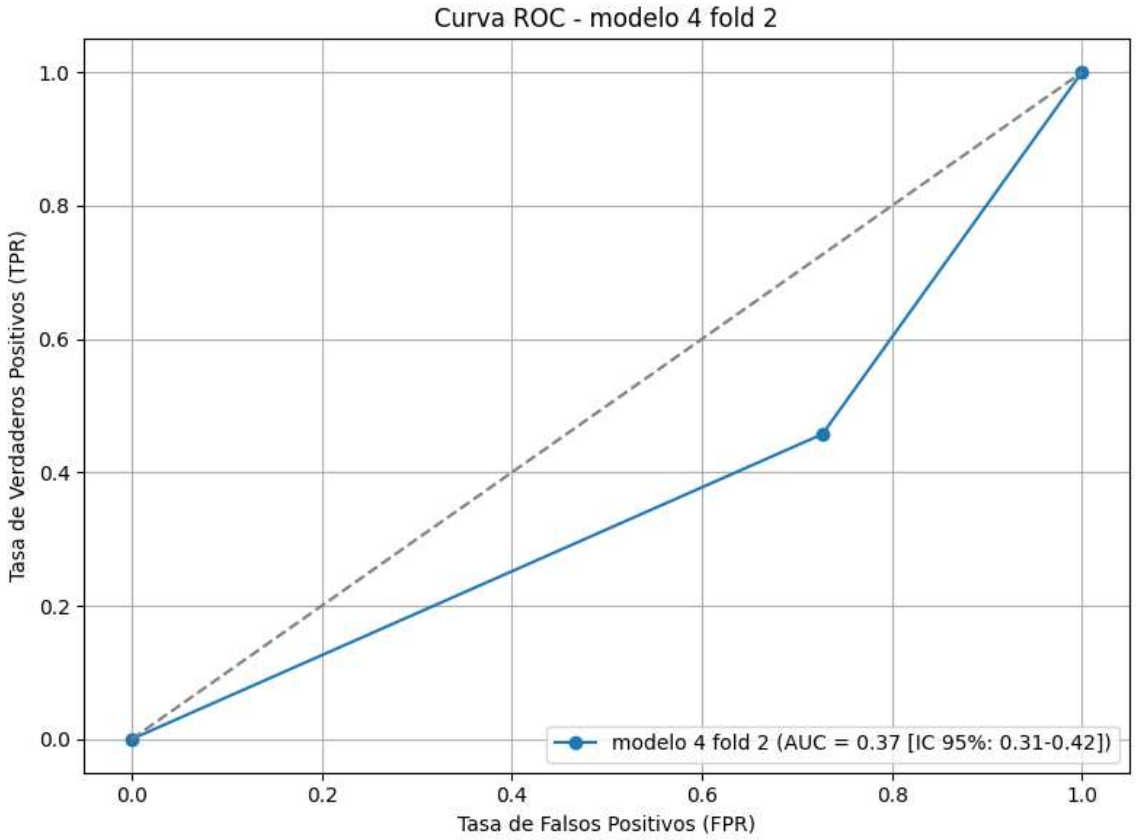


Figura 47. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 2).

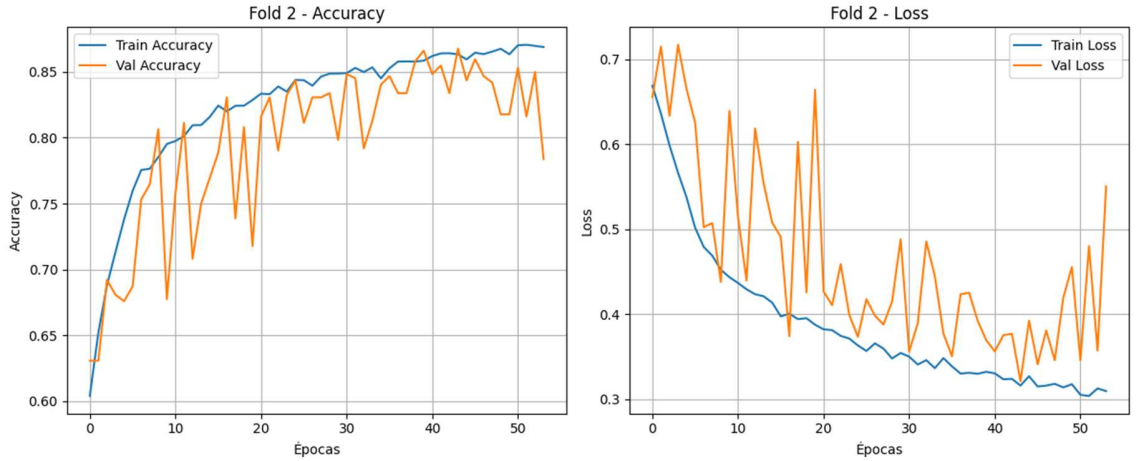


Figura 48. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 2).

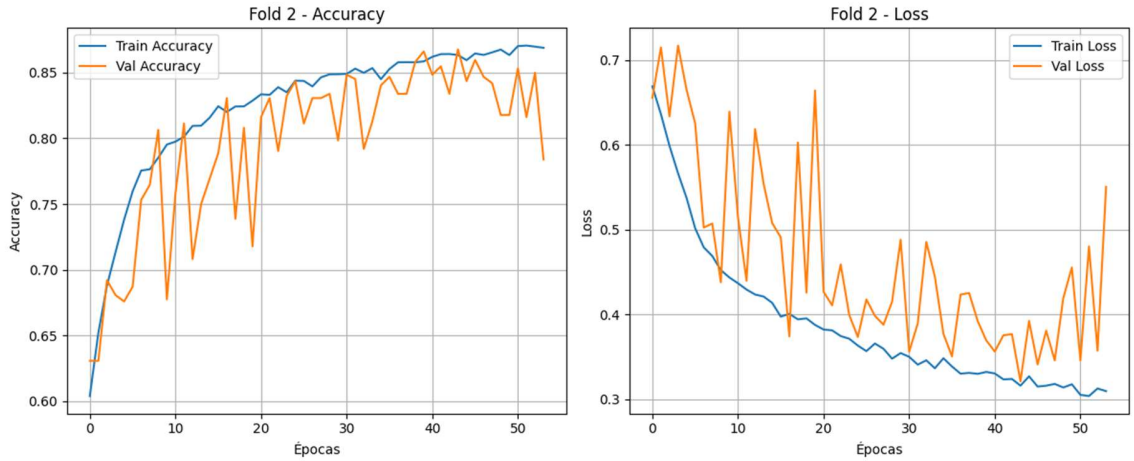


Figura 49. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 2).

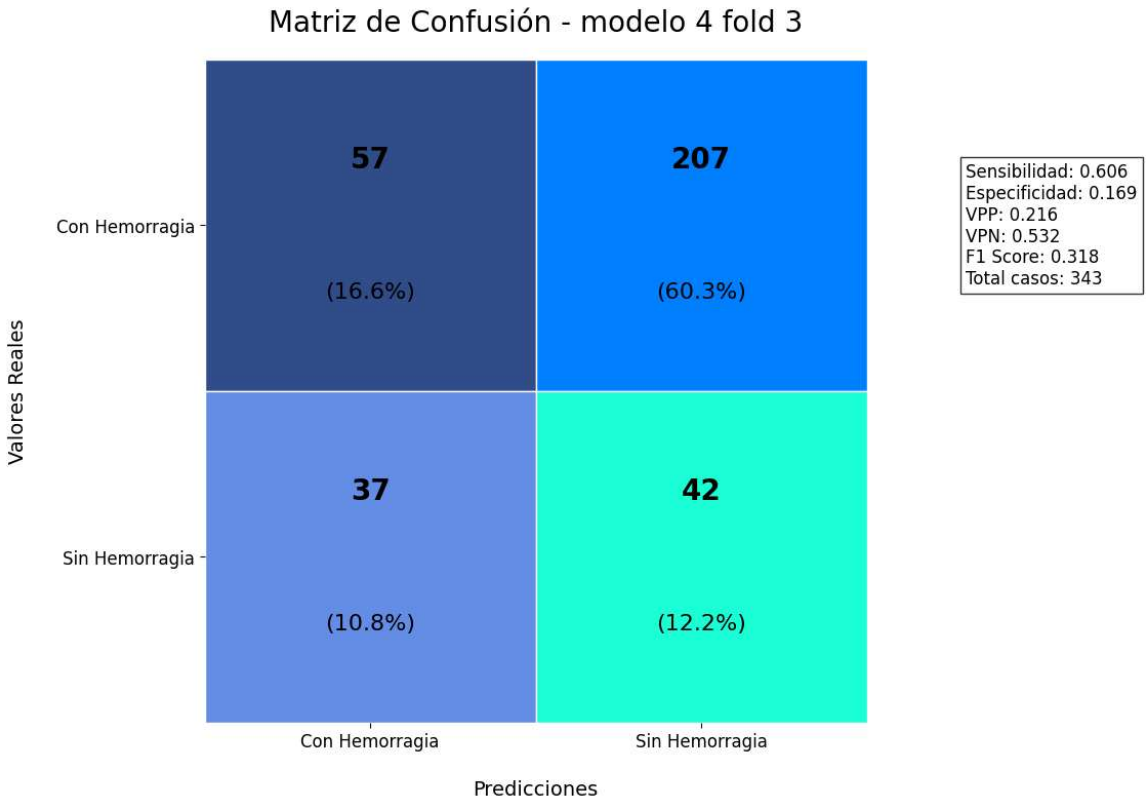


Figura 50. Matriz de confusión de los casos analizados con el modelo 4 (fold 3).

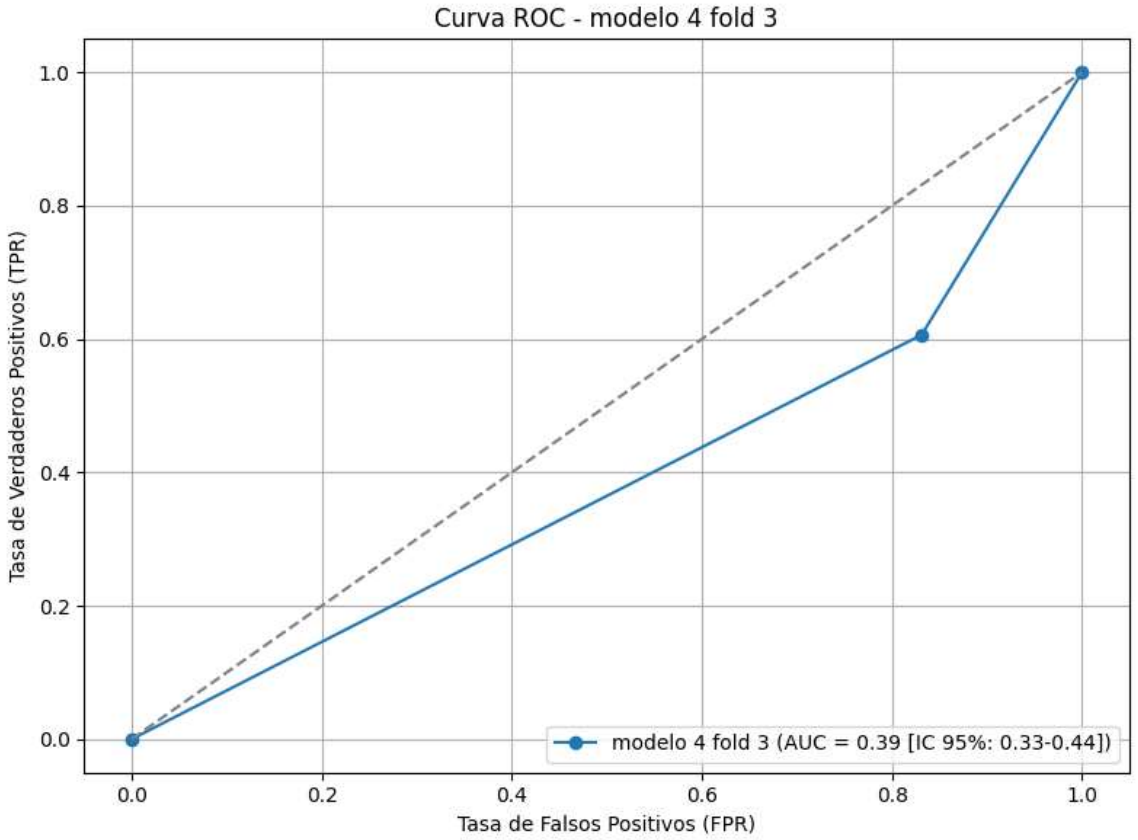


Figura 51. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 3).

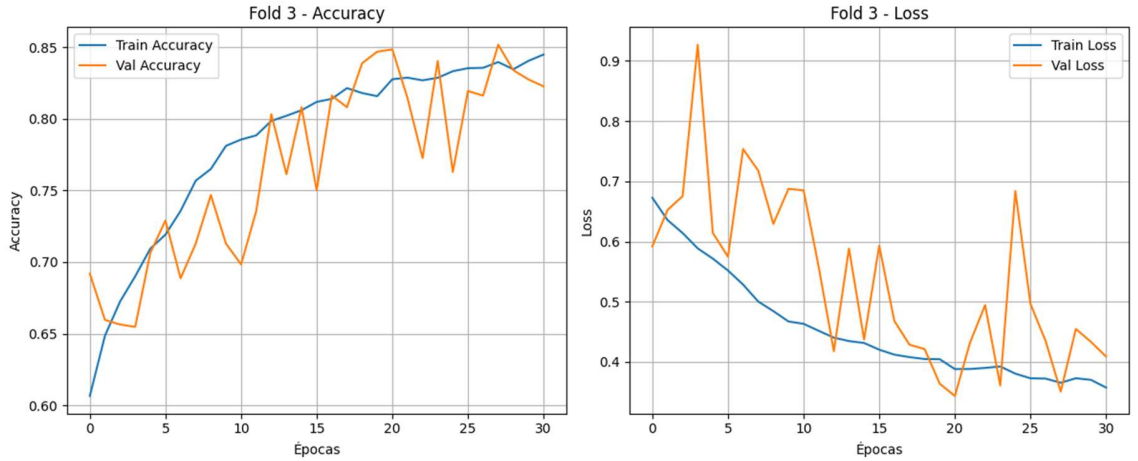


Figura 52. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 3).

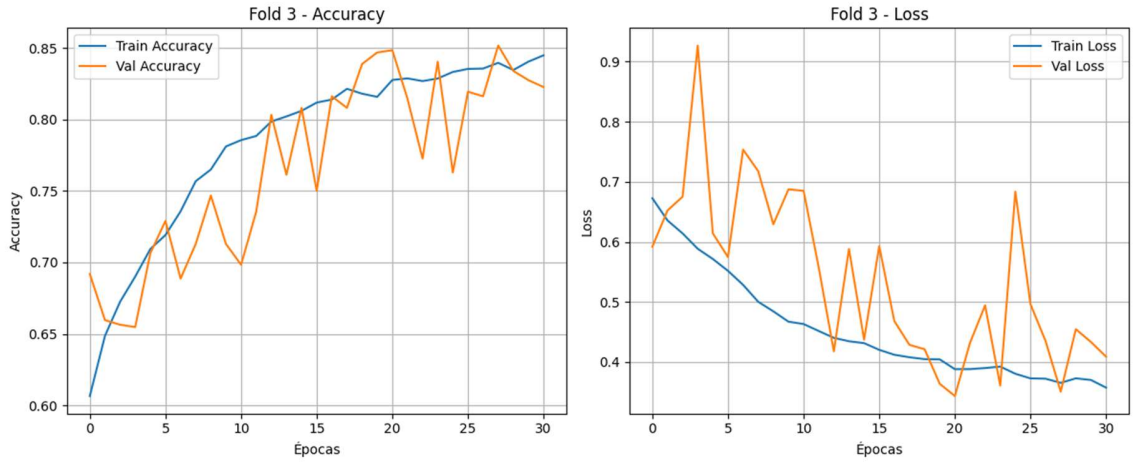


Figura 53. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 3).

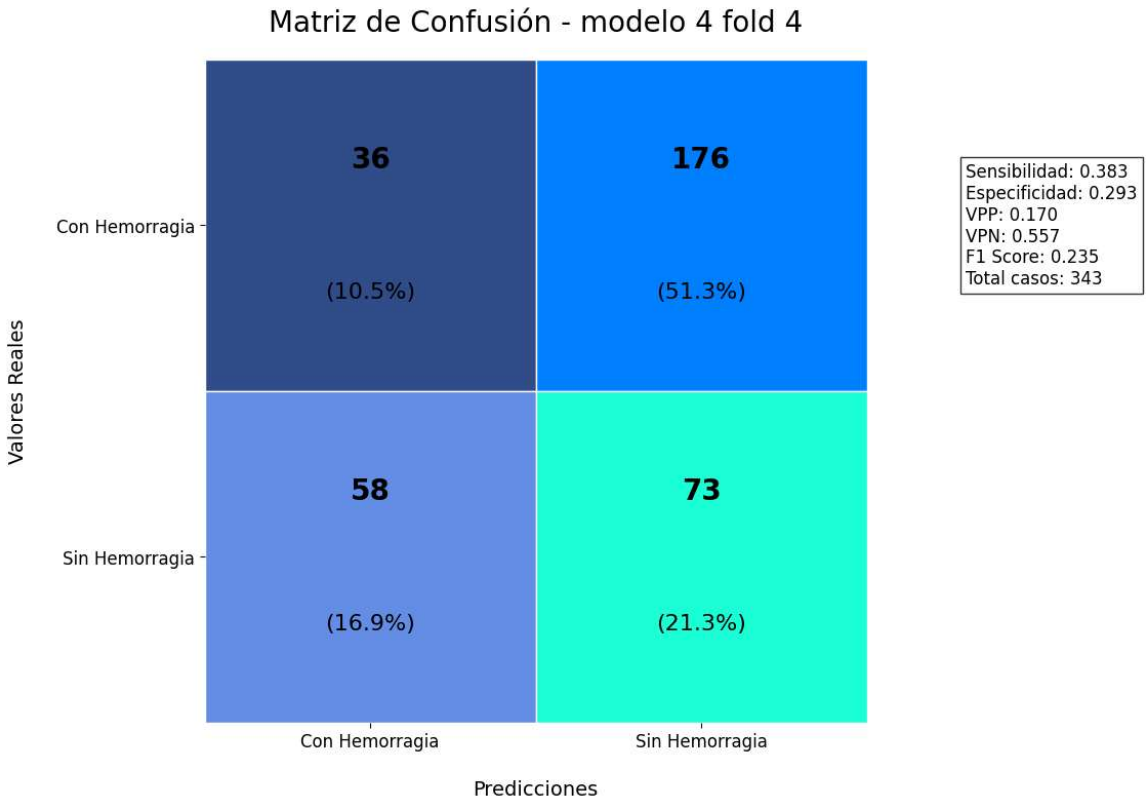


Figura 54. Matriz de confusión de los casos analizados con el modelo 4 (fold 4).

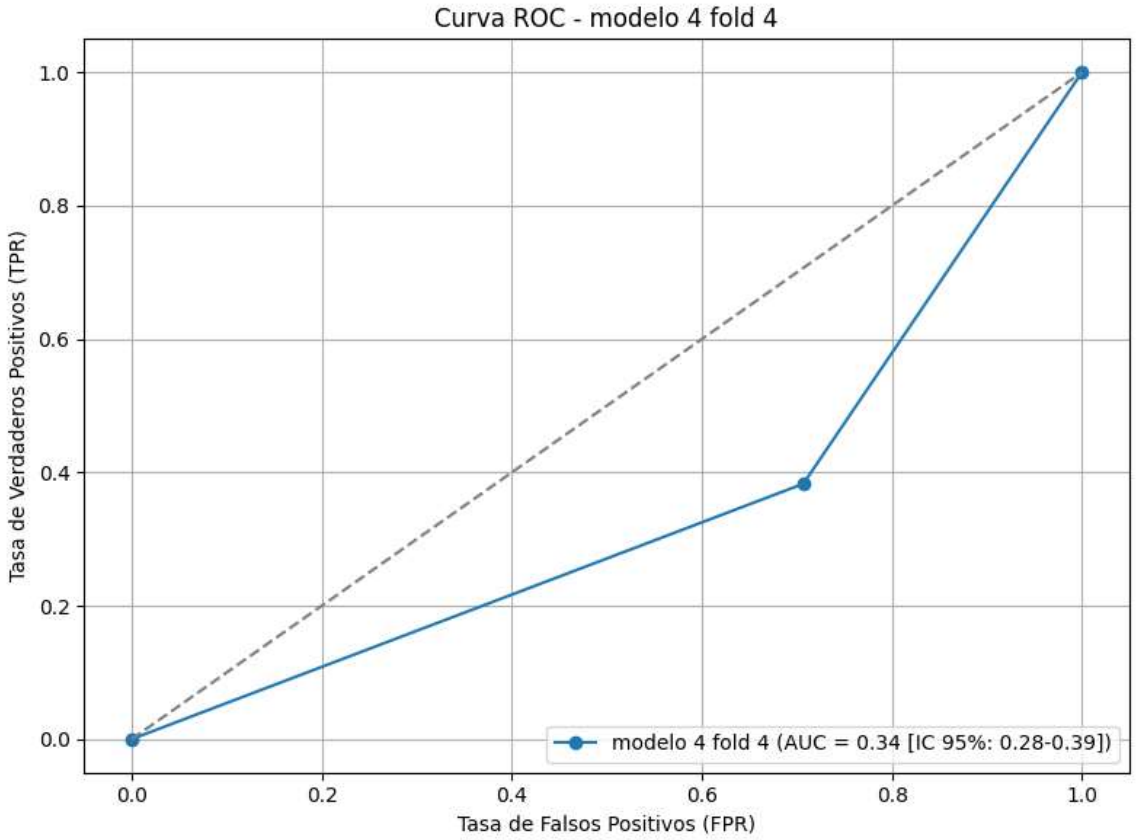


Figura 55. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 4).

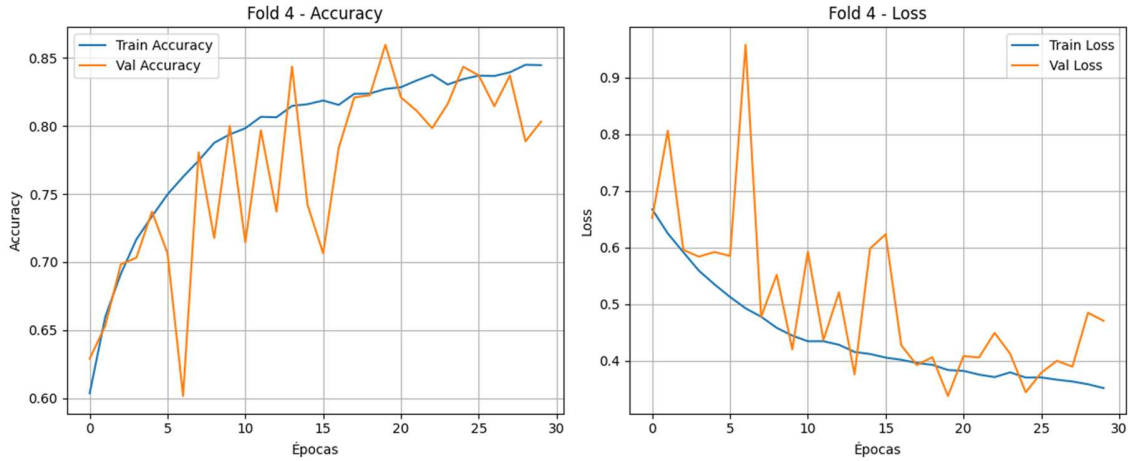


Figura 56. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 4).

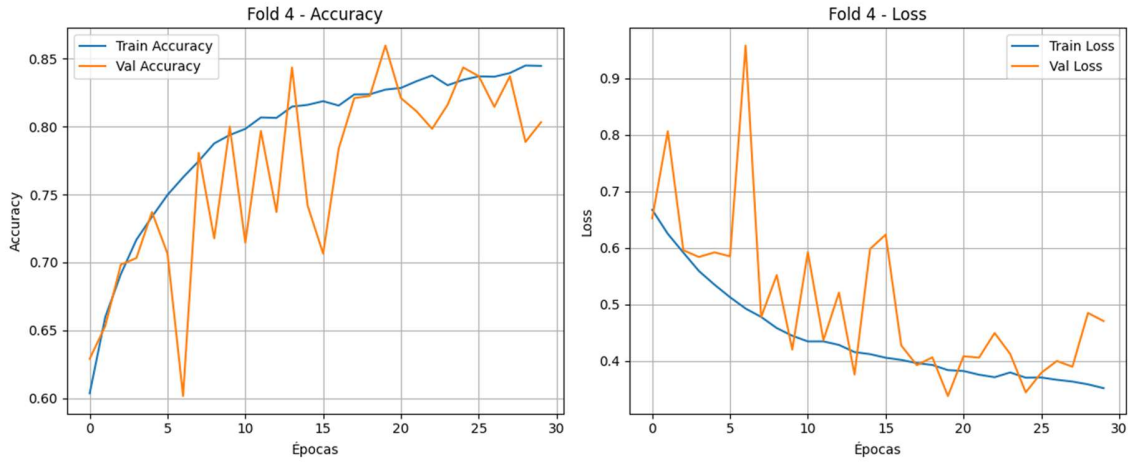


Figura 57. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 4).

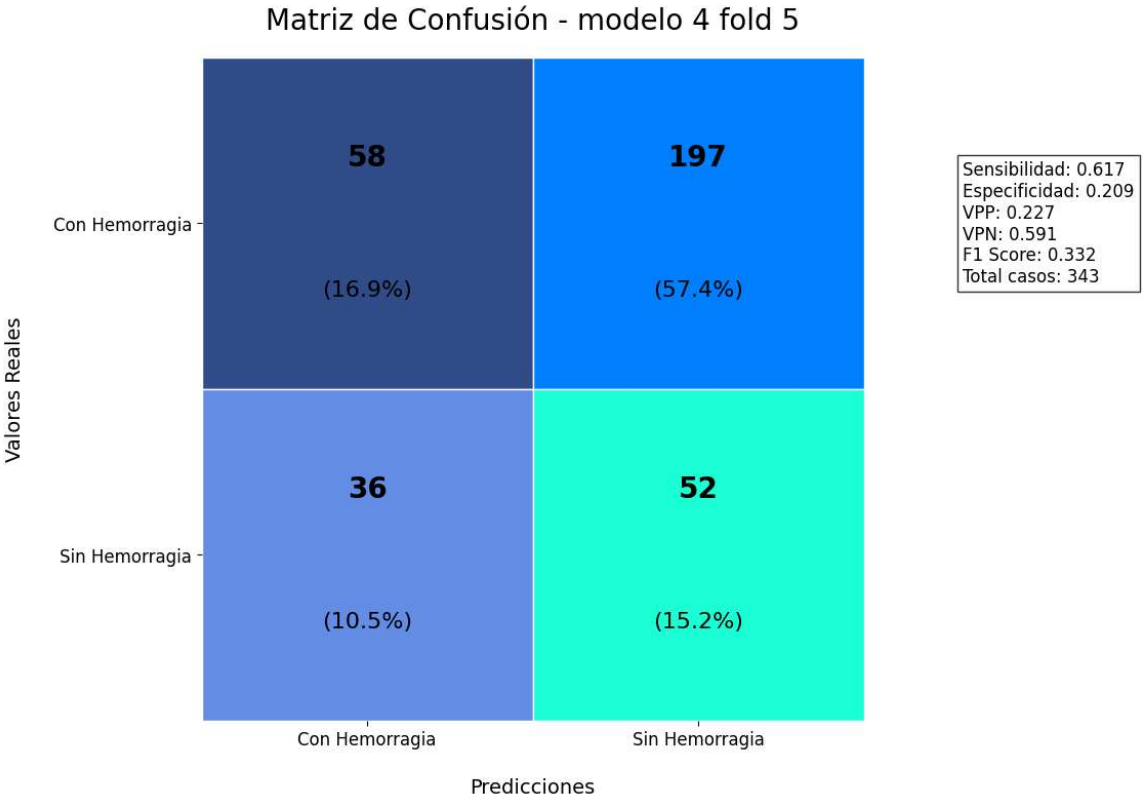


Figura 58. Matriz de confusión de los casos analizados con el modelo 4 (fold 5).

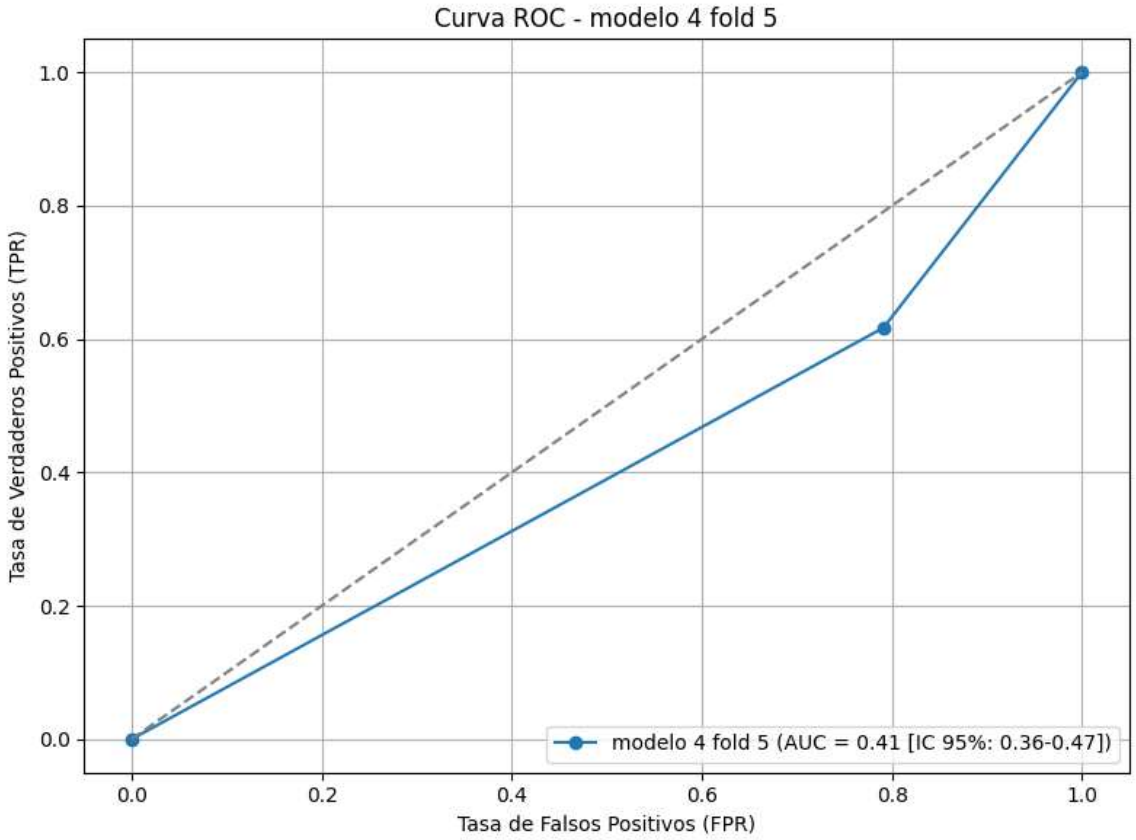


Figura 59. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 5).

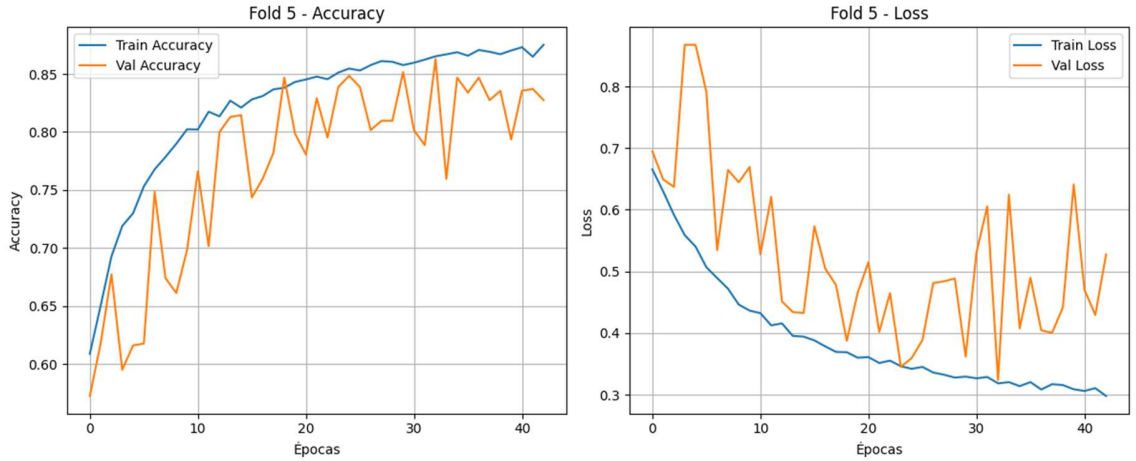


Figura 60. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 5).

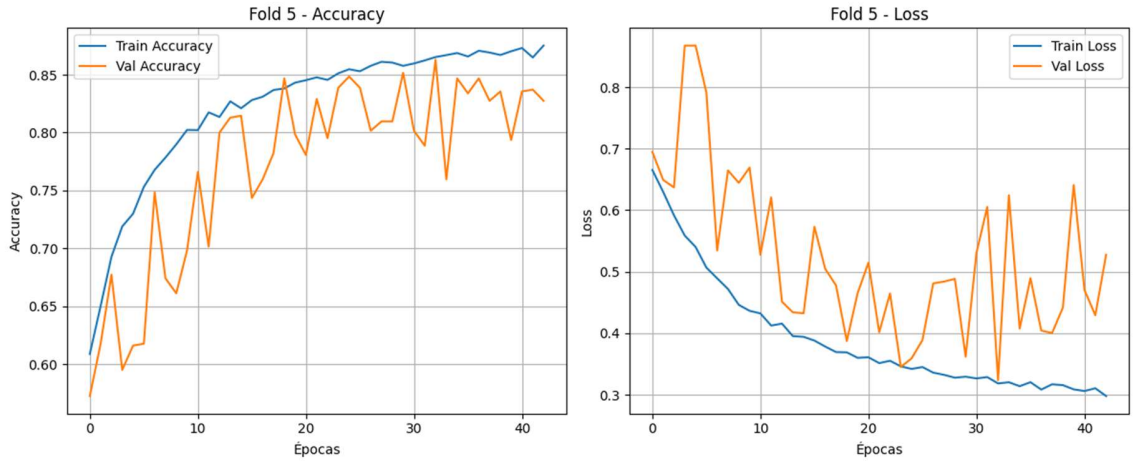


Figura 61. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 5).

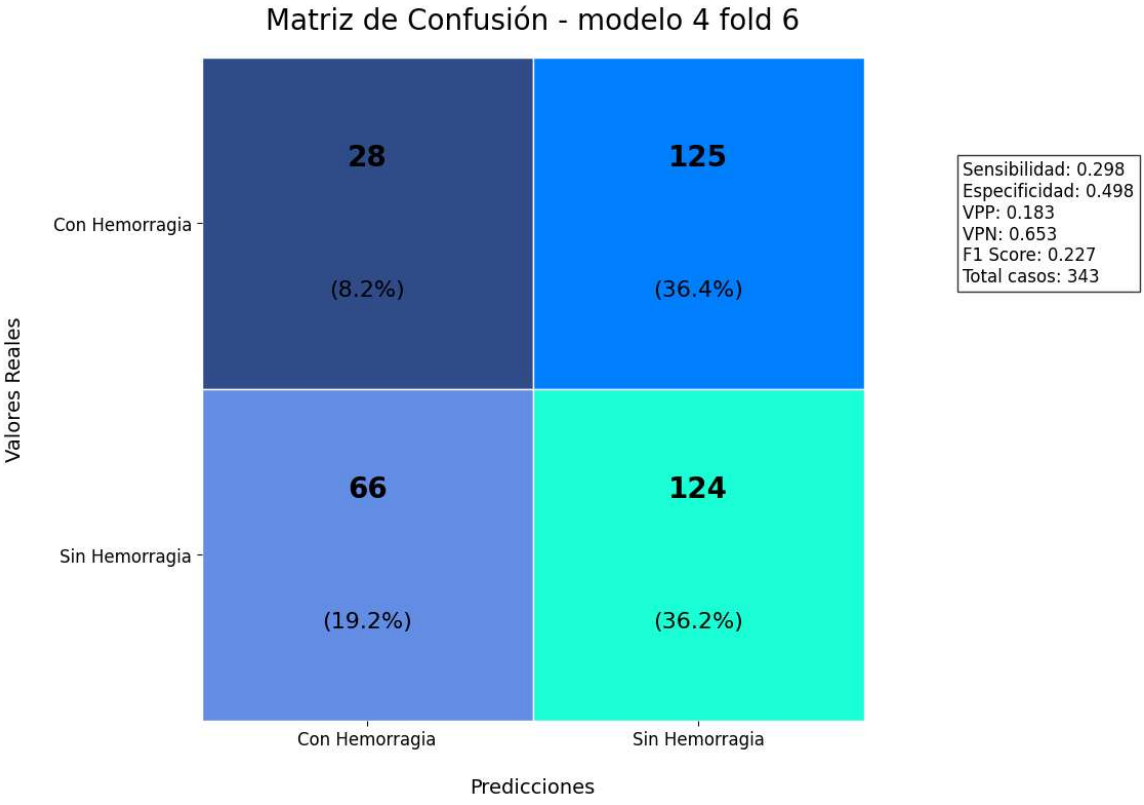


Figura 62. Matriz de confusión de los casos analizados con el modelo 4 (fold 6).

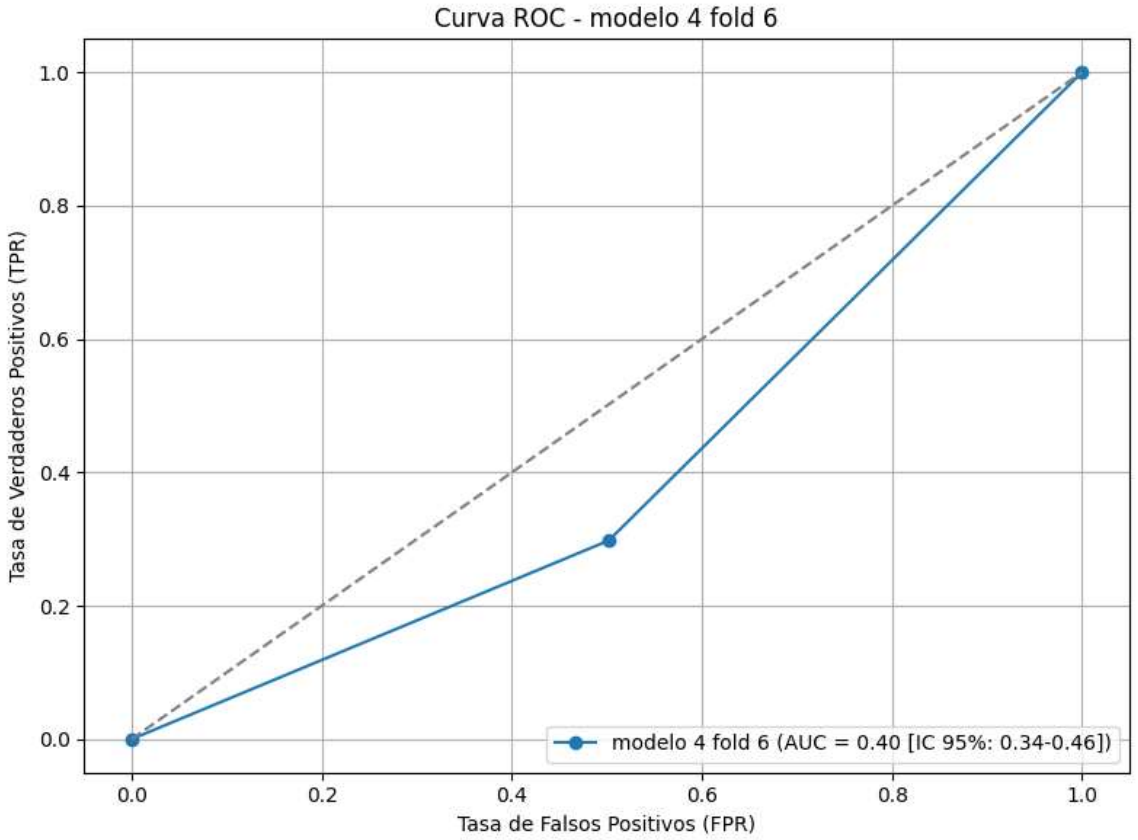


Figura 63. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 6).

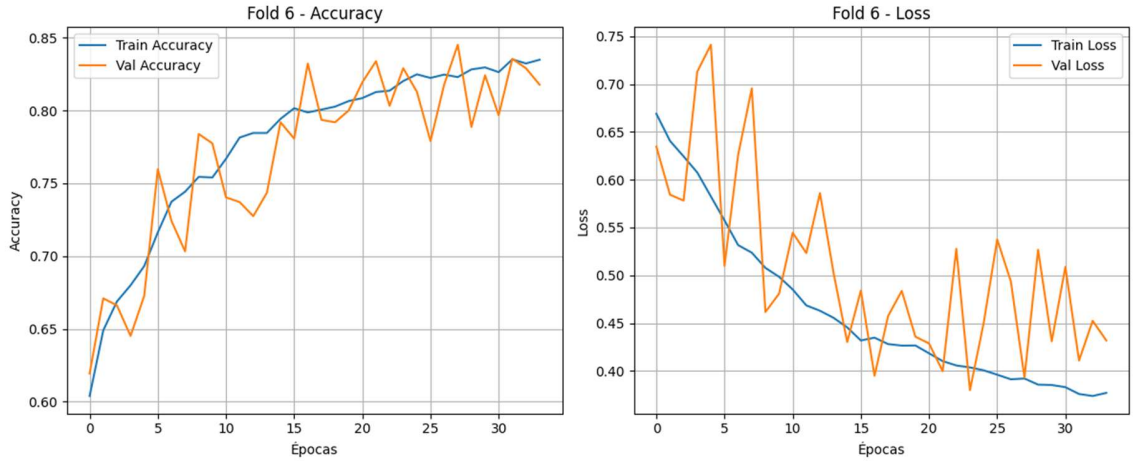


Figura 64. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 6).

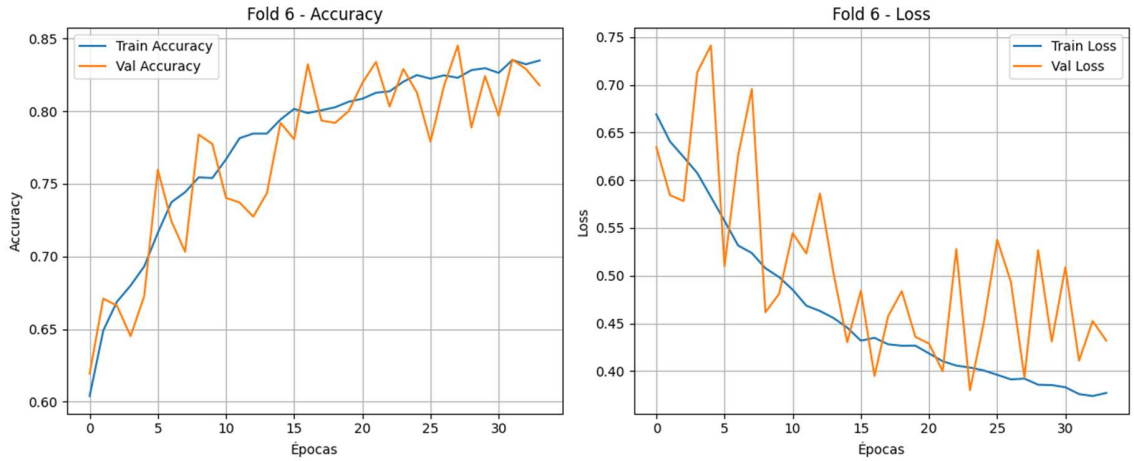


Figura 65. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 6).

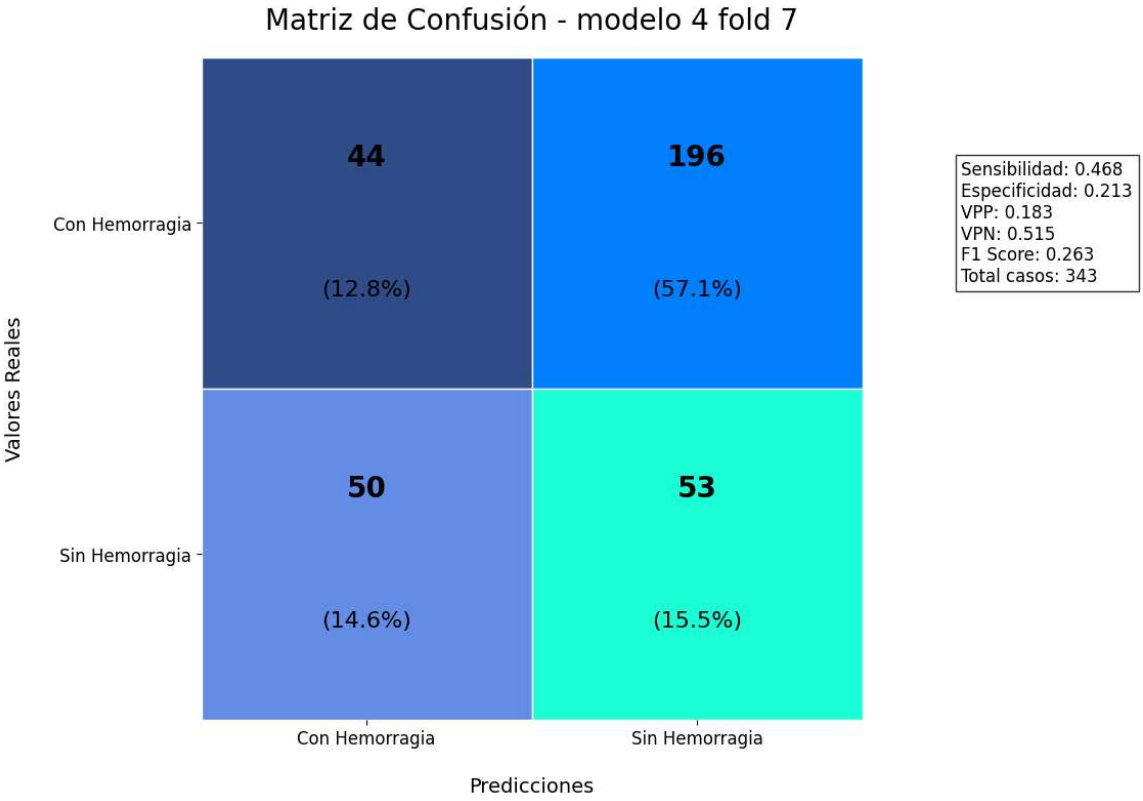


Figura 66. Matriz de confusión de los casos analizados con el modelo 4 (fold 7).

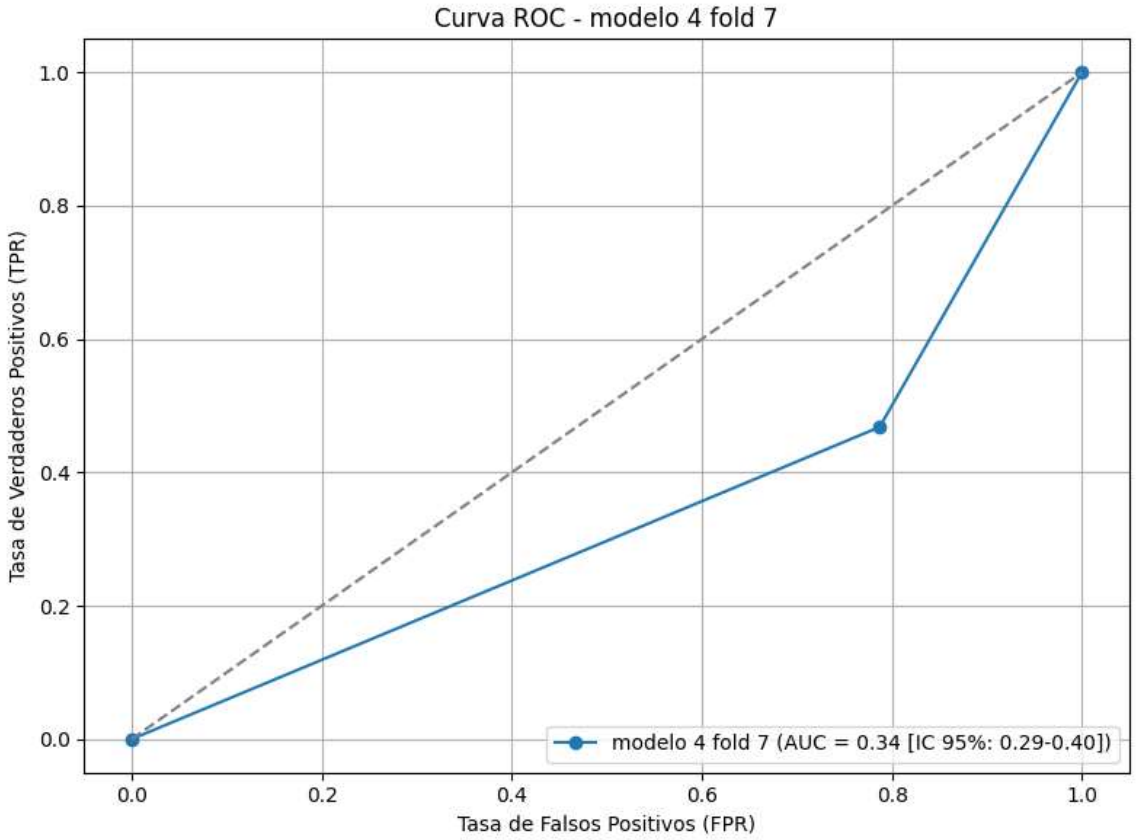


Figura 67. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 7).

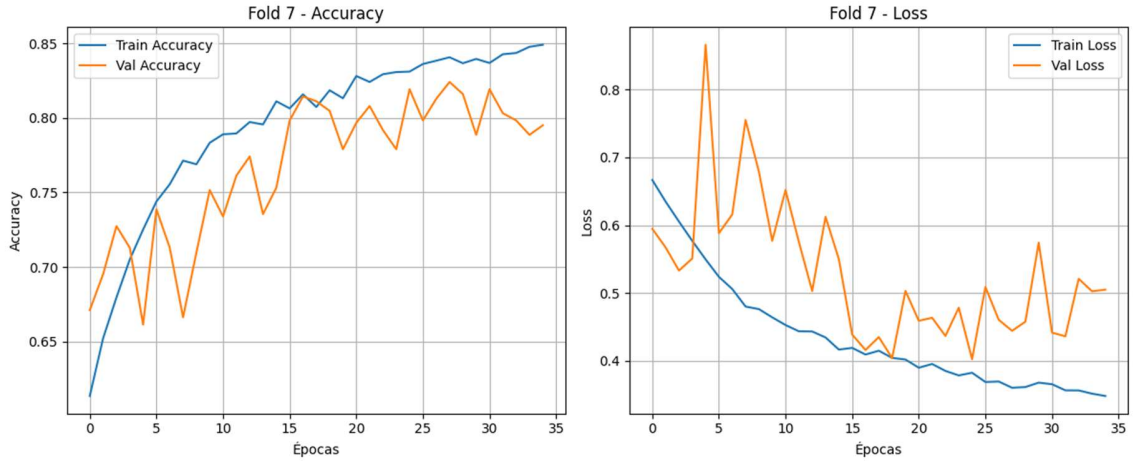


Figura 68. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 7).

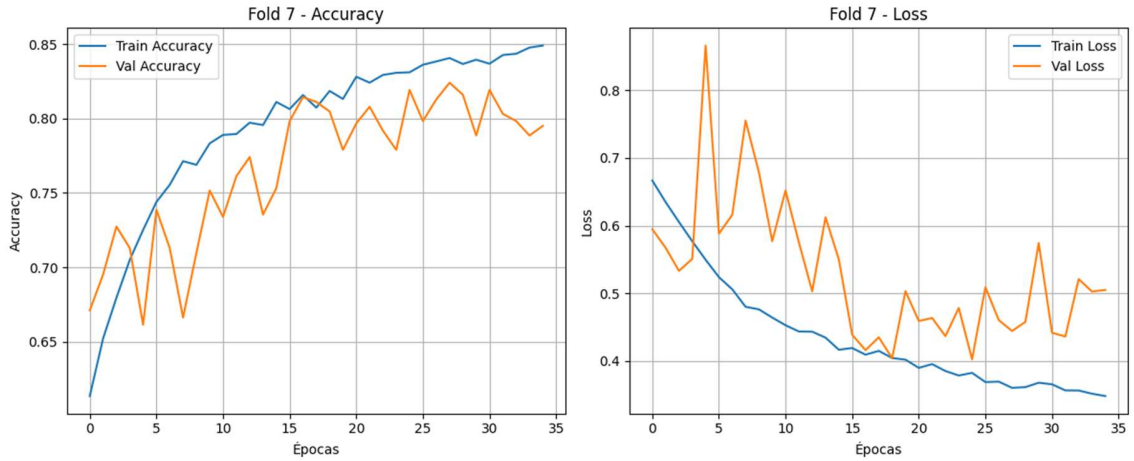


Figura 69. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 7).

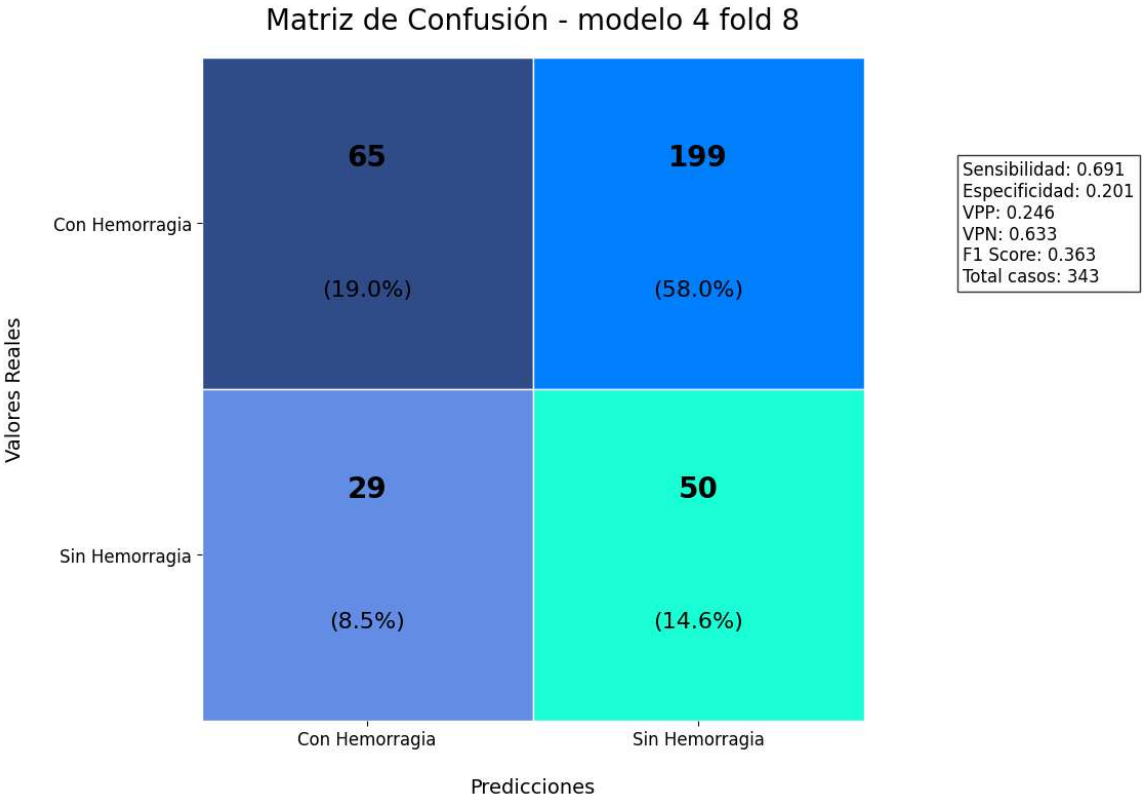


Figura 70. Matriz de confusión de los casos analizados con el modelo 4 (fold 8).

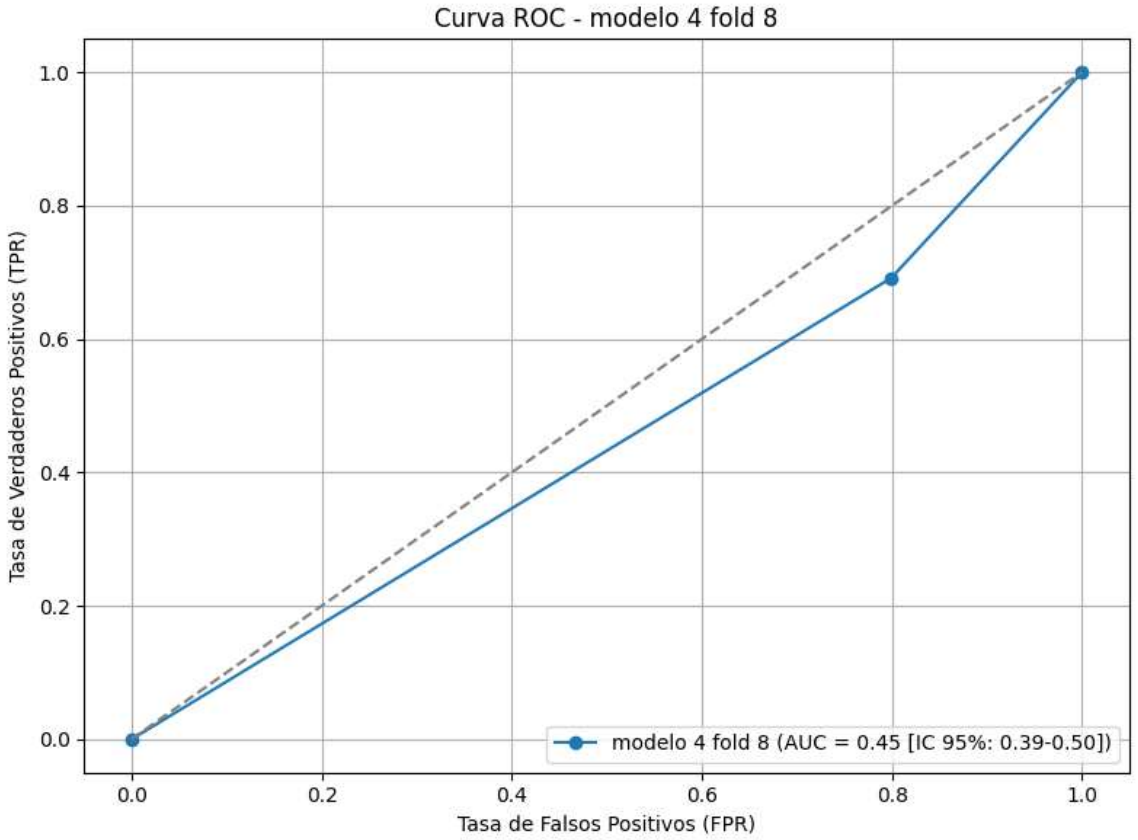


Figura 71. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 8).

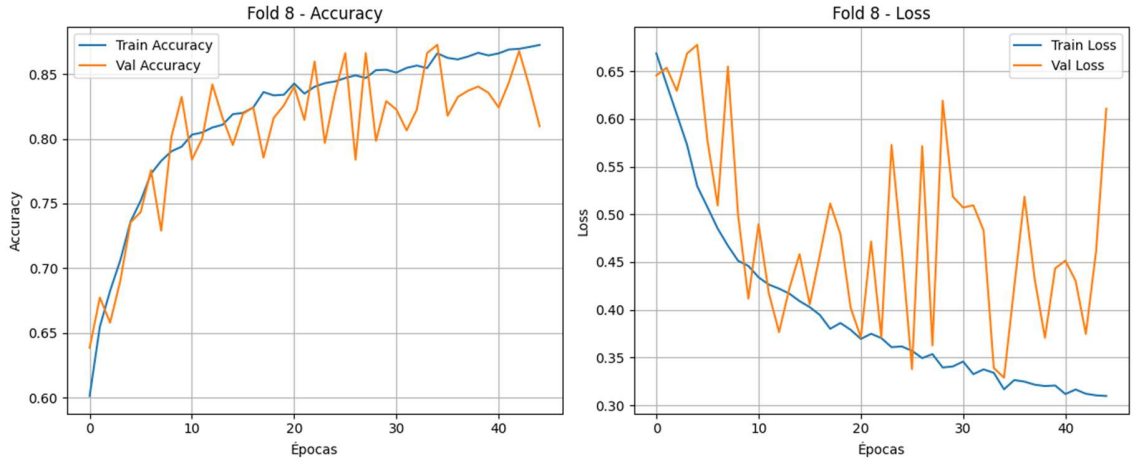


Figura 72. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 8).

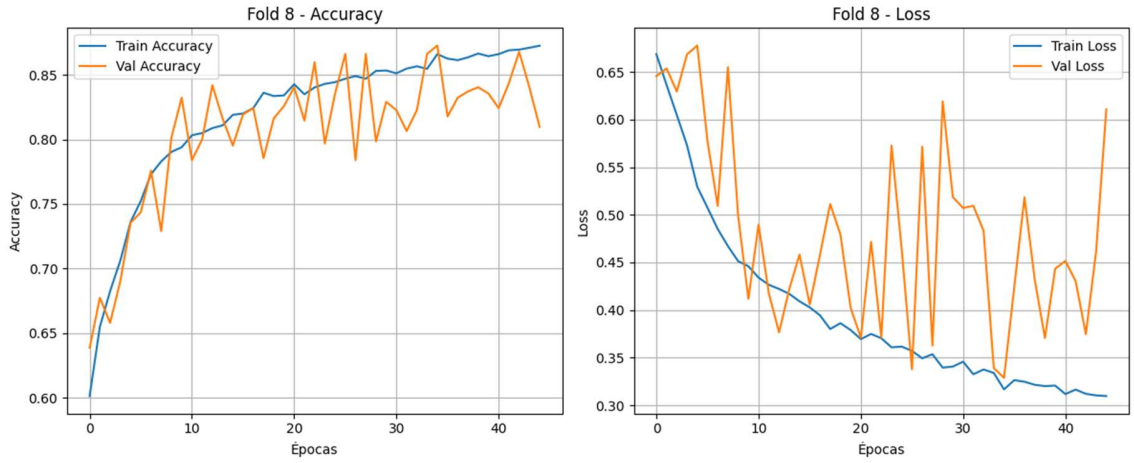


Figura 73. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 8).

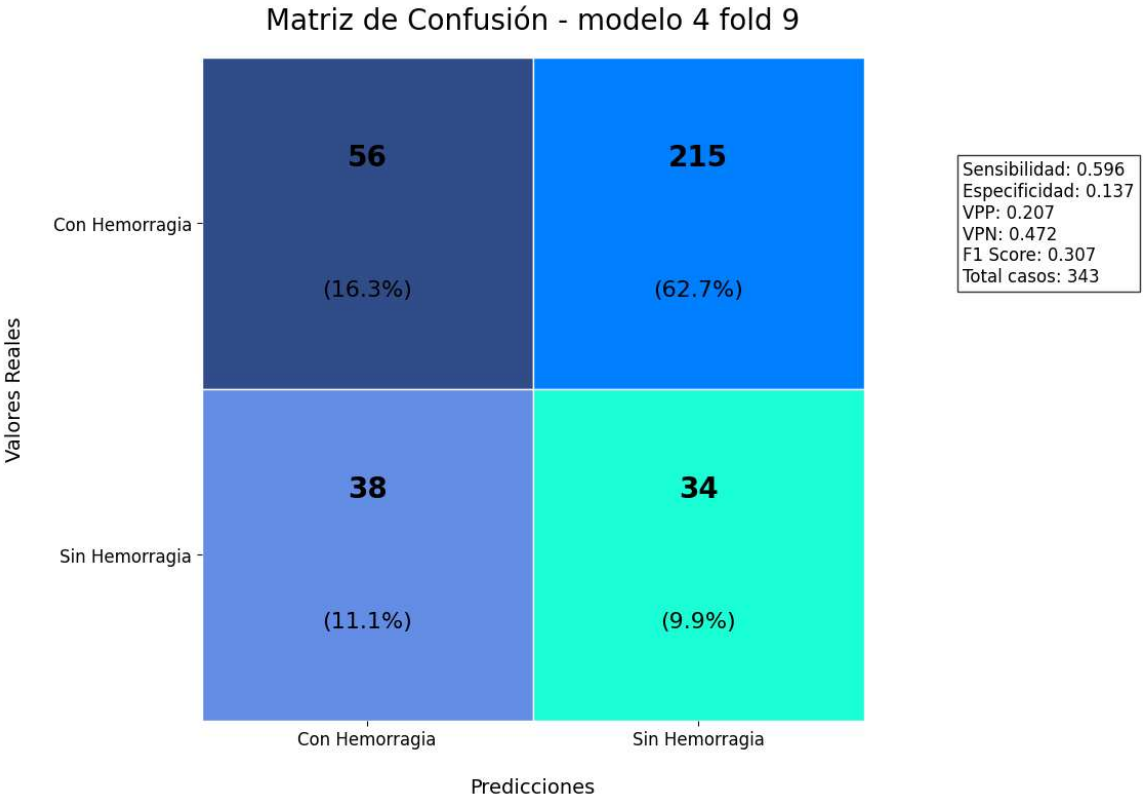


Figura 74. Matriz de confusión de los casos analizados con el modelo 4 (fold 9).

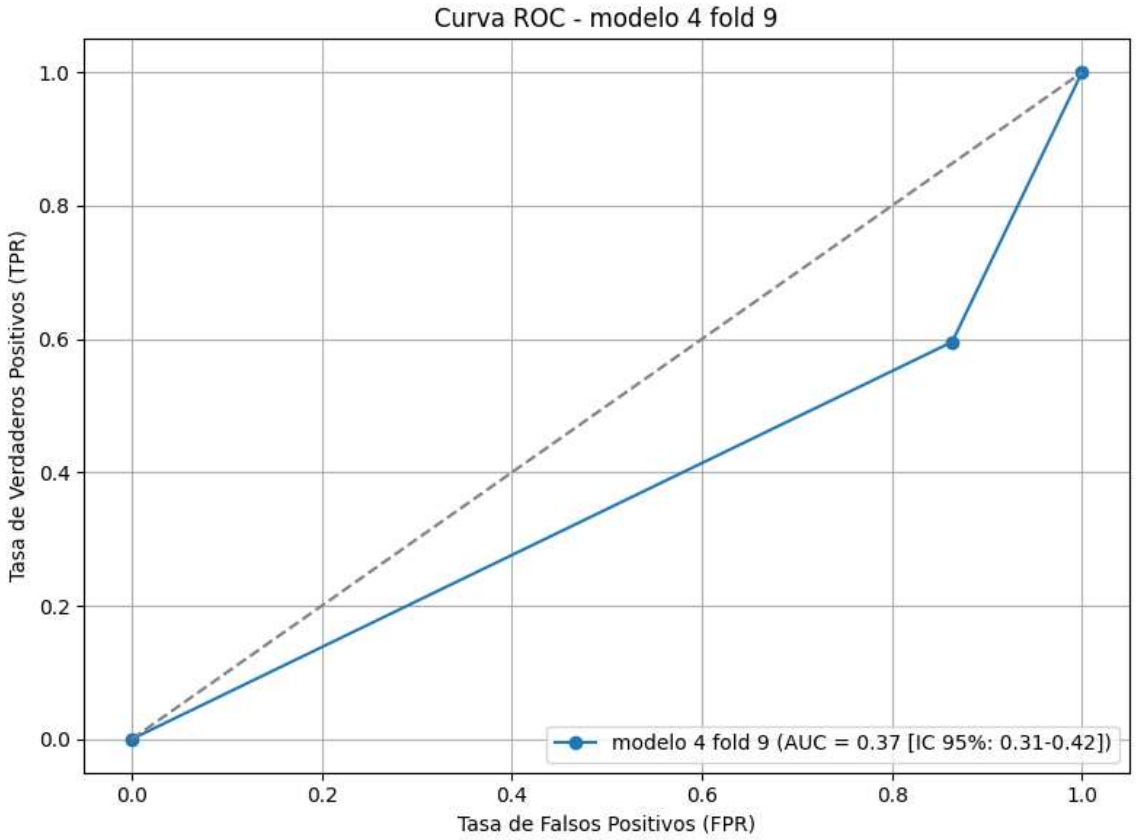


Figura 75. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 9).

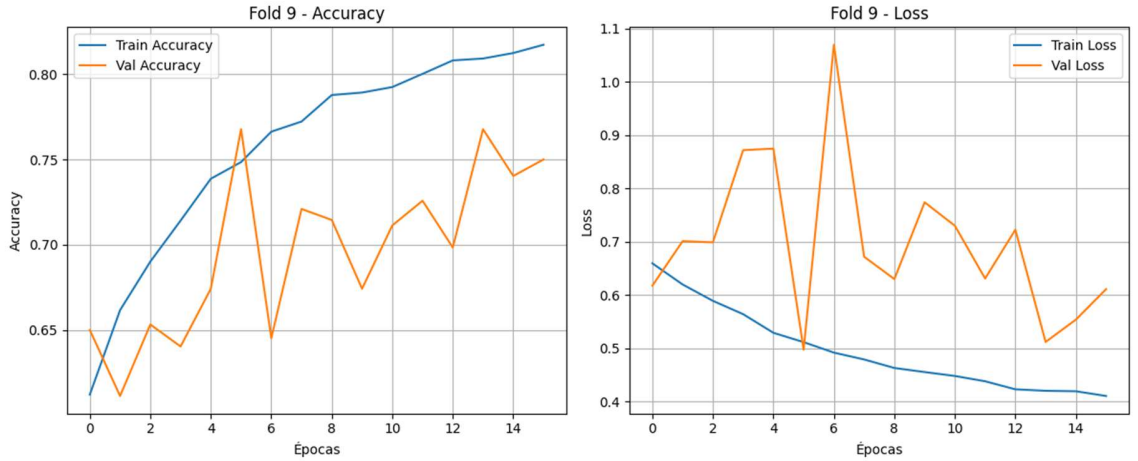


Figura 76. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 9).

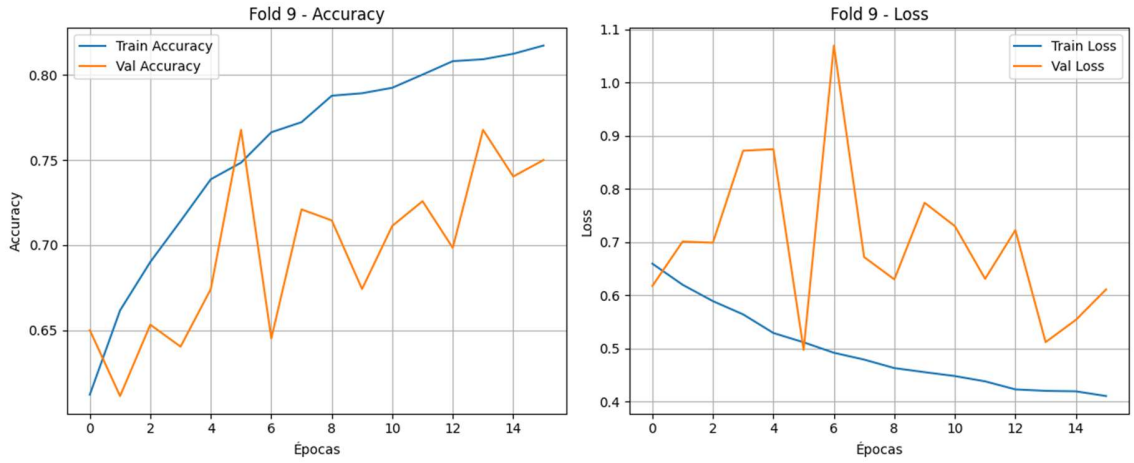


Figura 77. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 9).

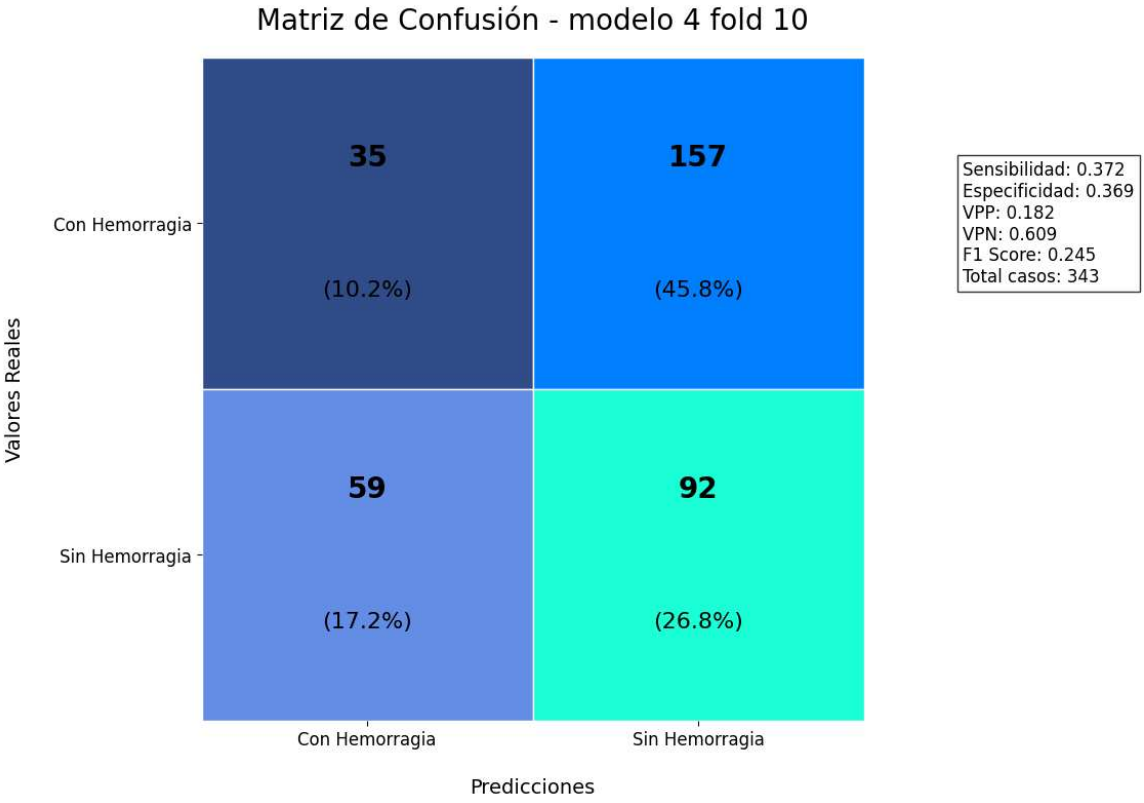


Figura 78. Matriz de confusión de los casos analizados con el modelo 4 (fold 10).

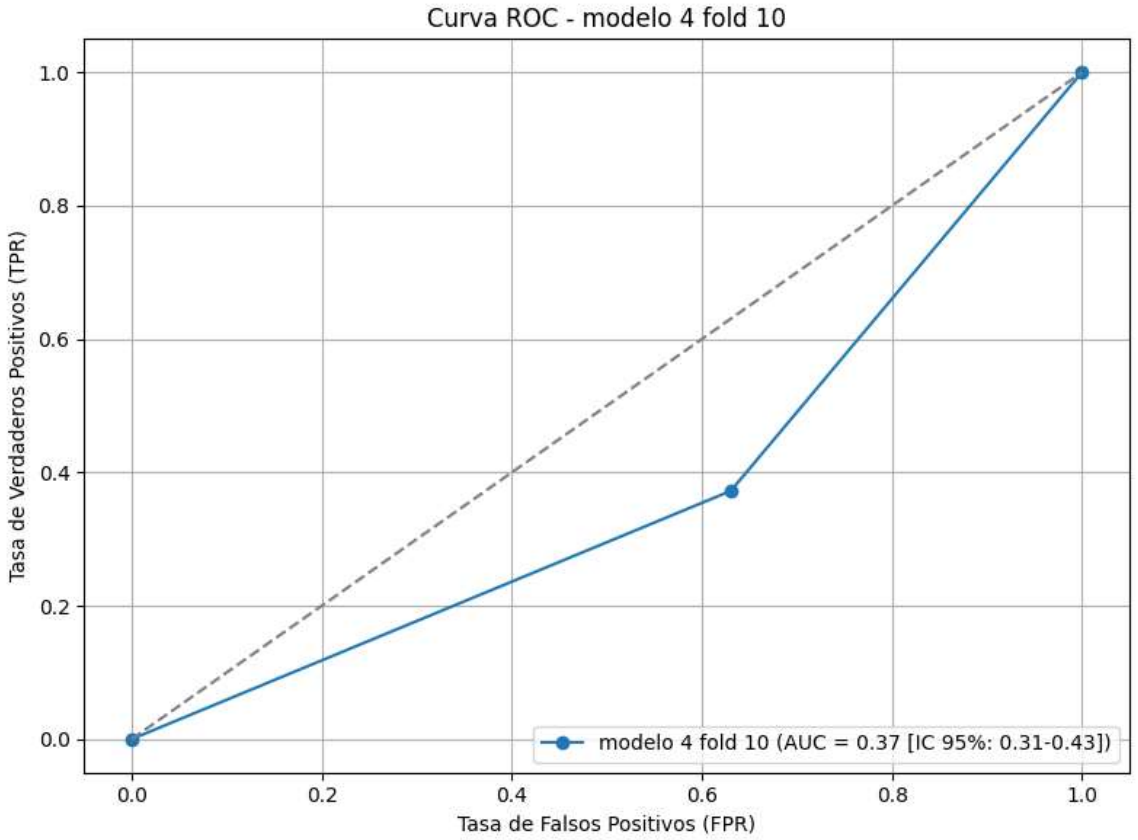


Figura 79. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 10).

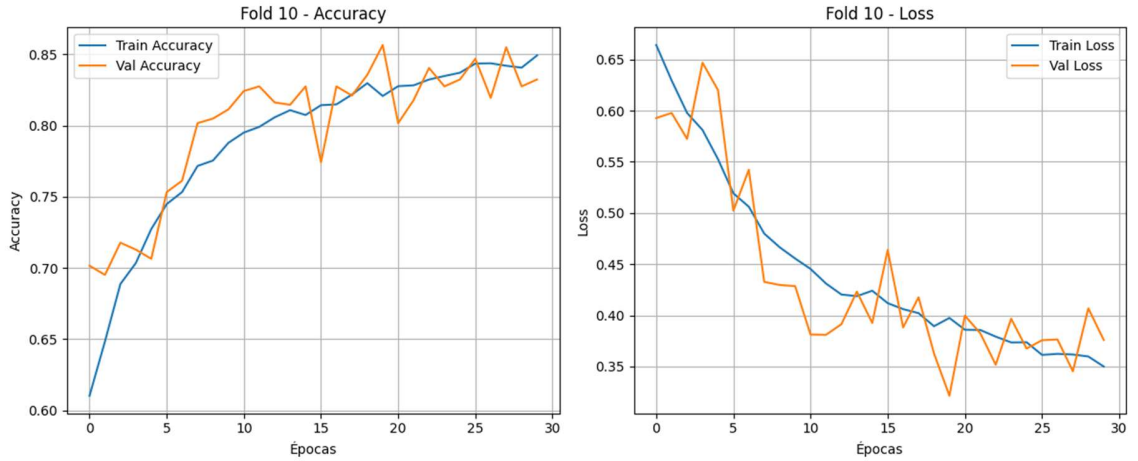


Figura 80. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 10).

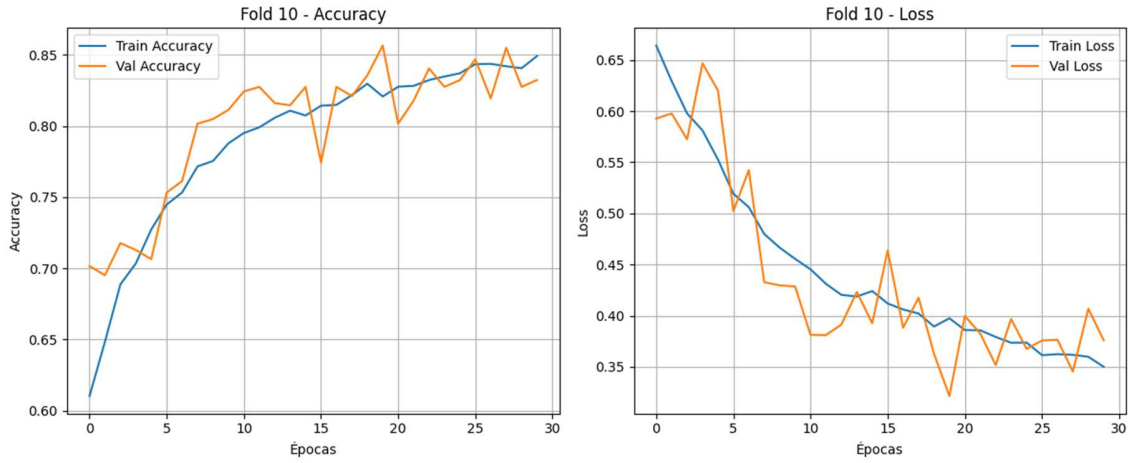


Figura 81. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 10).

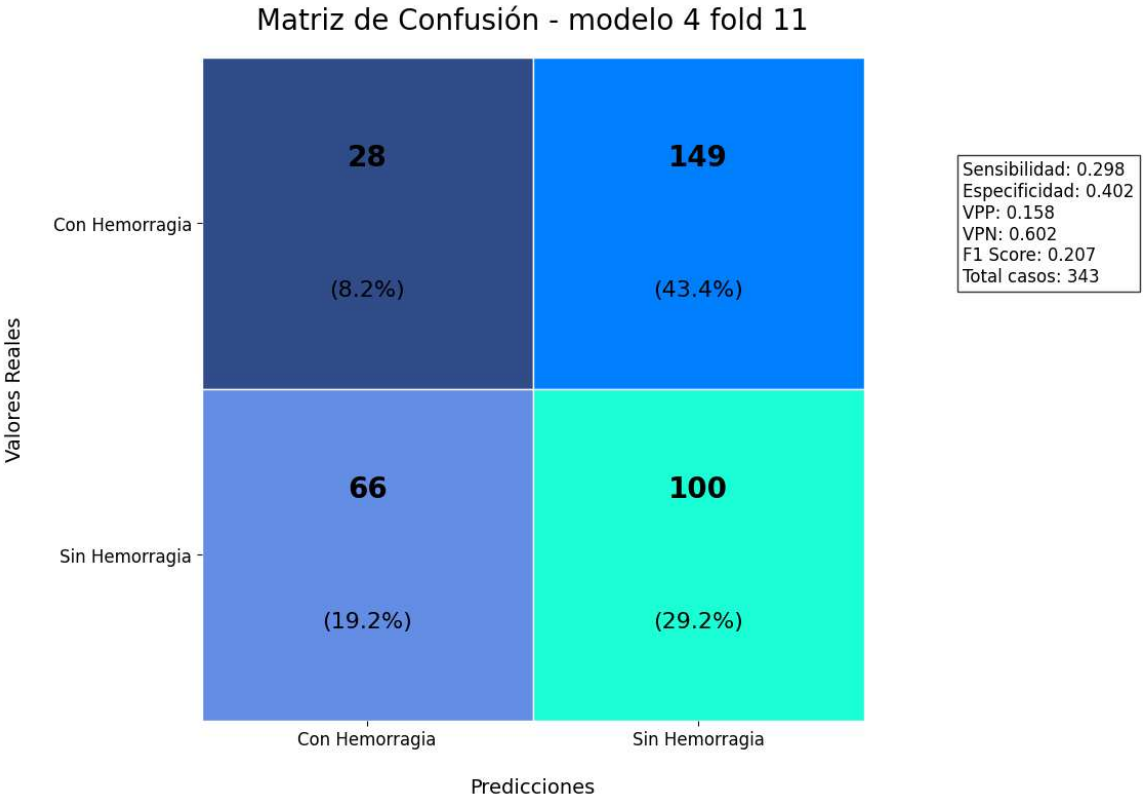


Figura 82. Matriz de confusión de los casos analizados con el modelo 4 (fold 11).

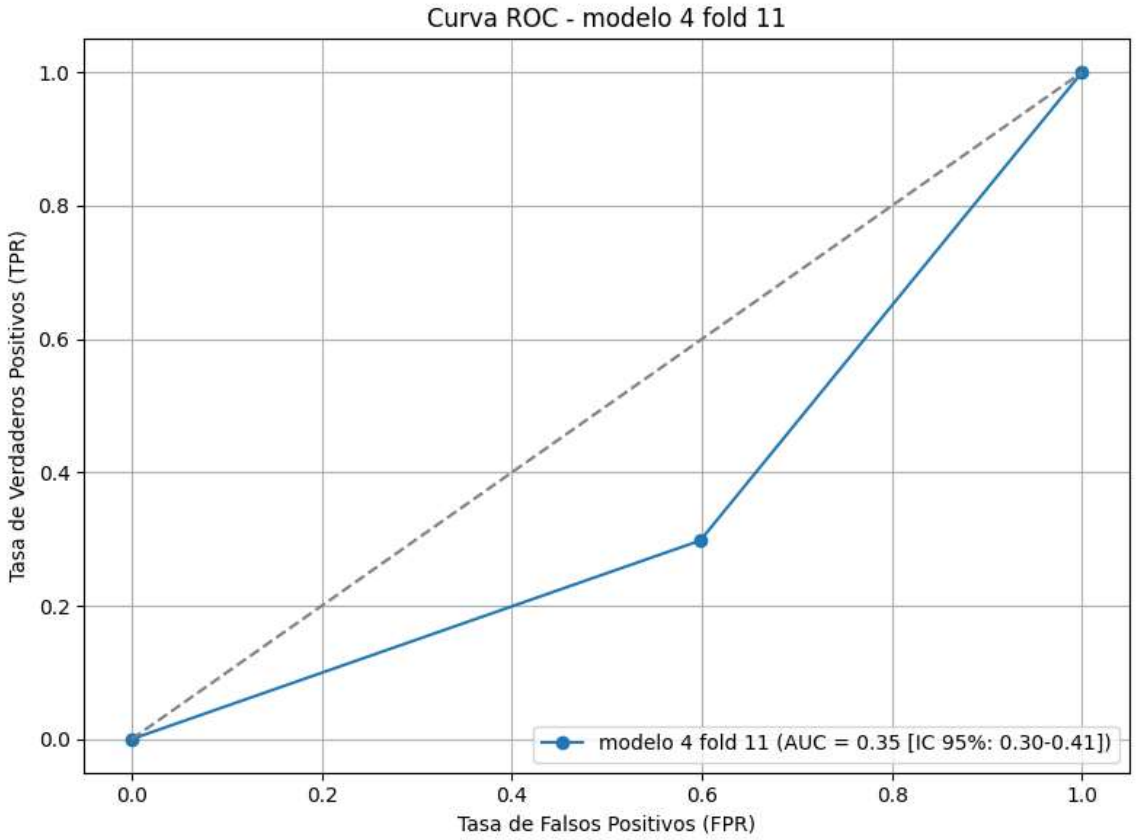


Figura 83. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 11).

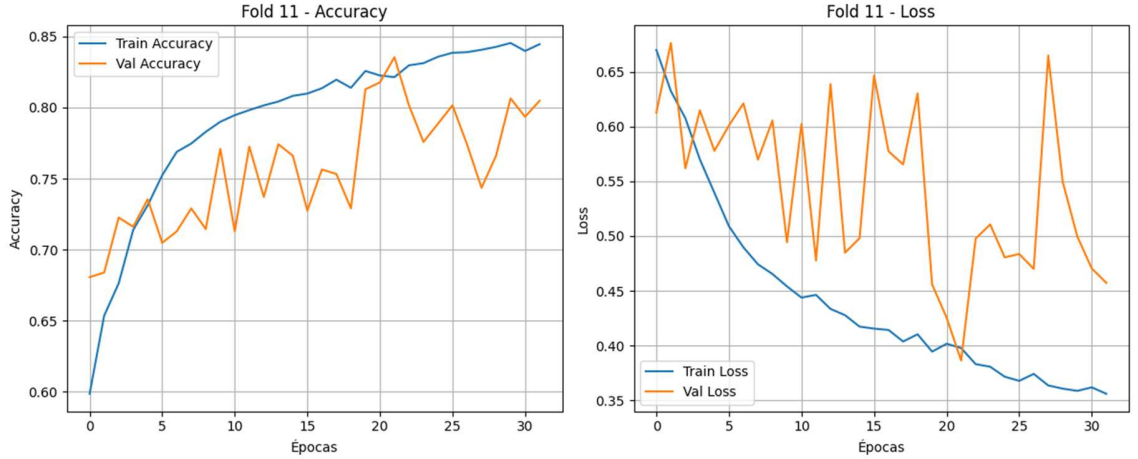


Figura 84. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 11).

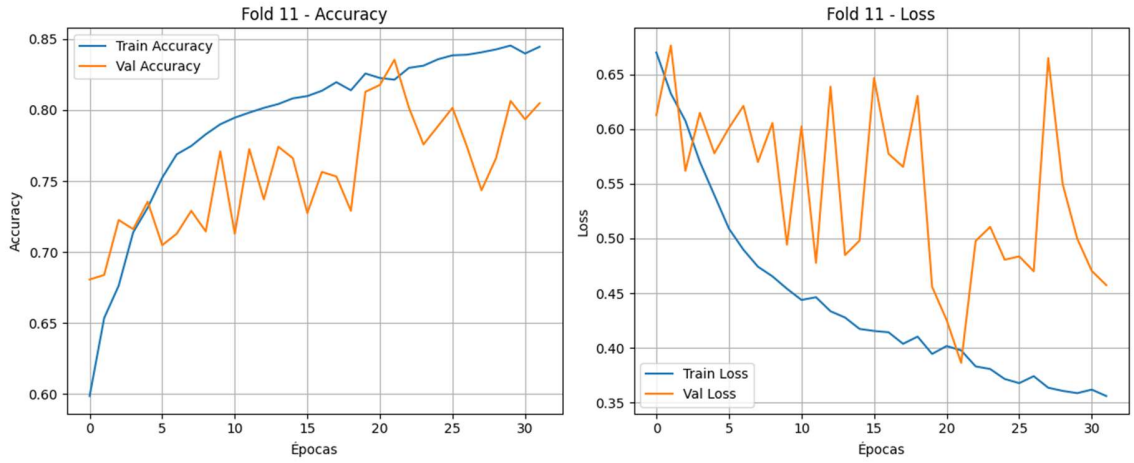


Figura 85. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 11).

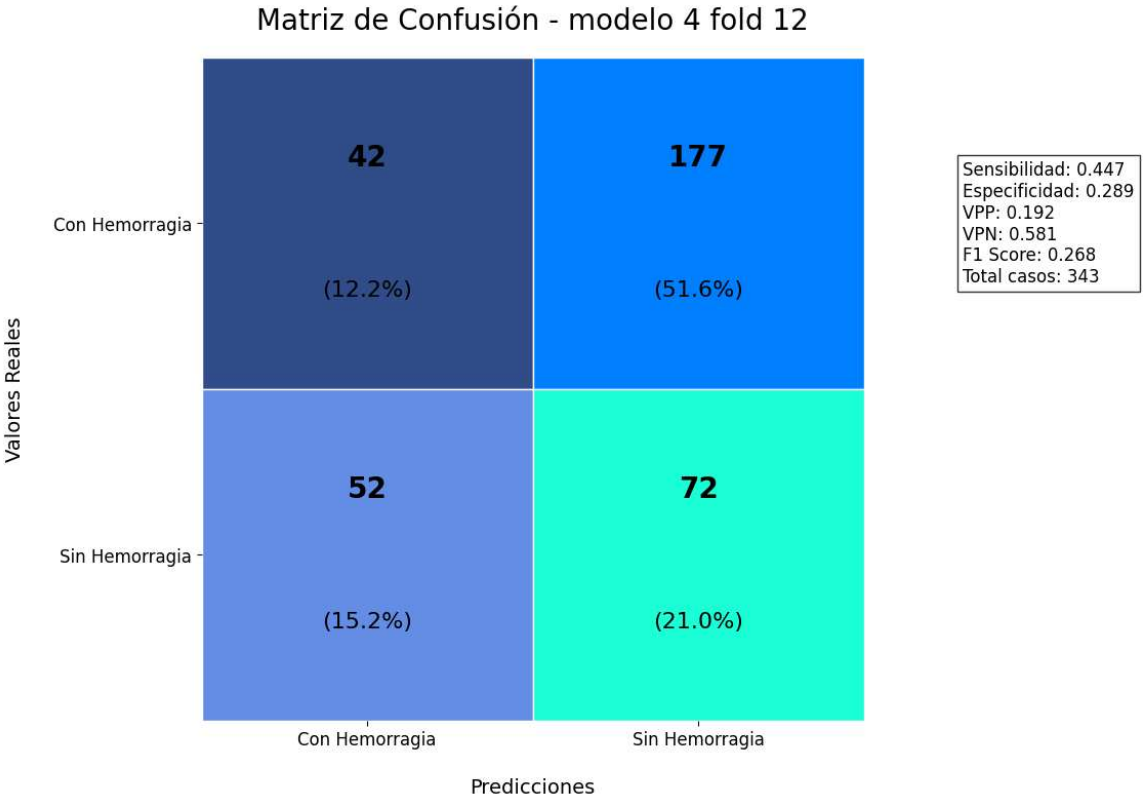


Figura 86. Matriz de confusión de los casos analizados con el modelo 4 (fold 12).

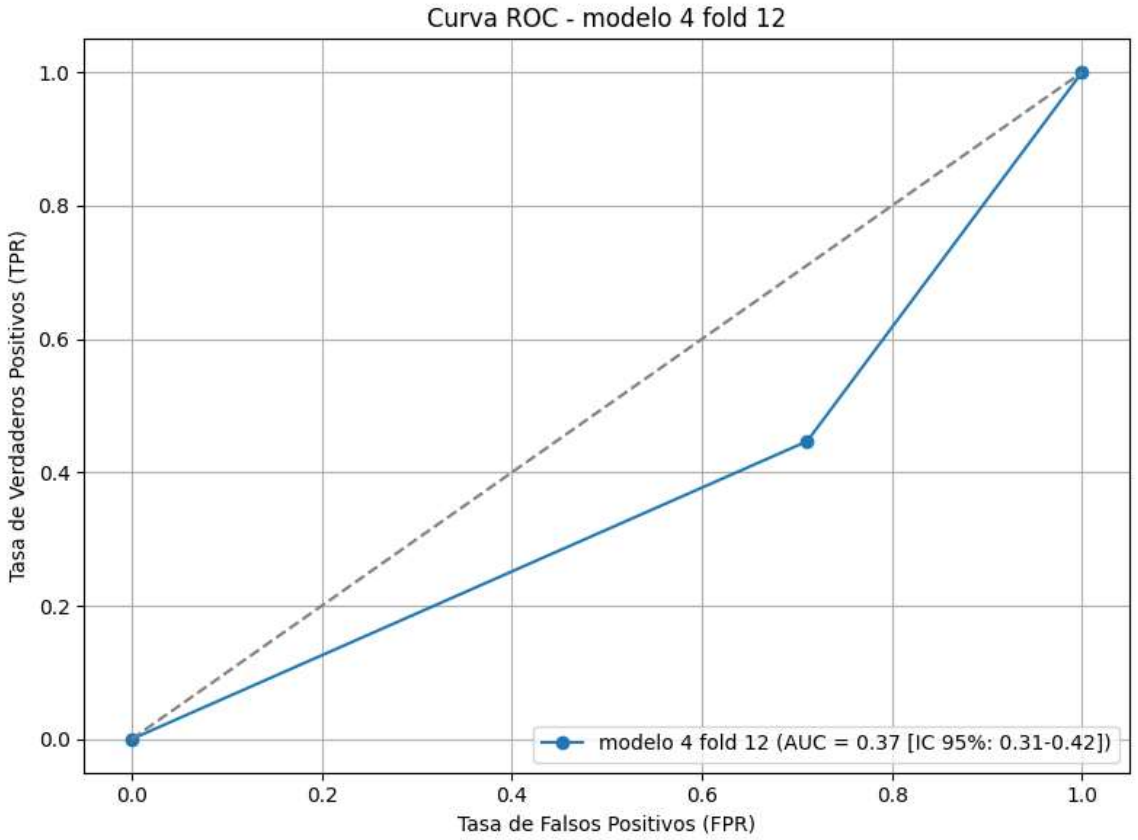


Figura 87. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 12).

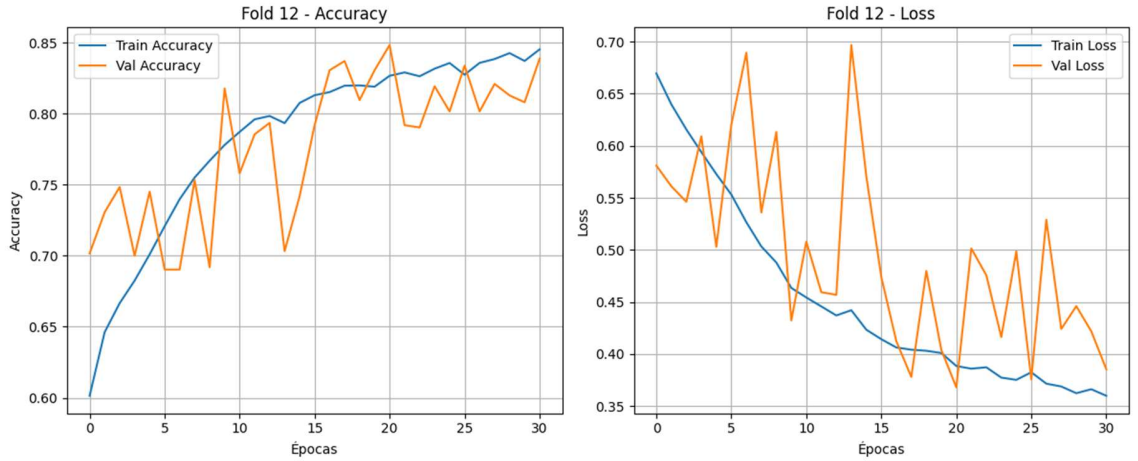


Figura 88. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 12).

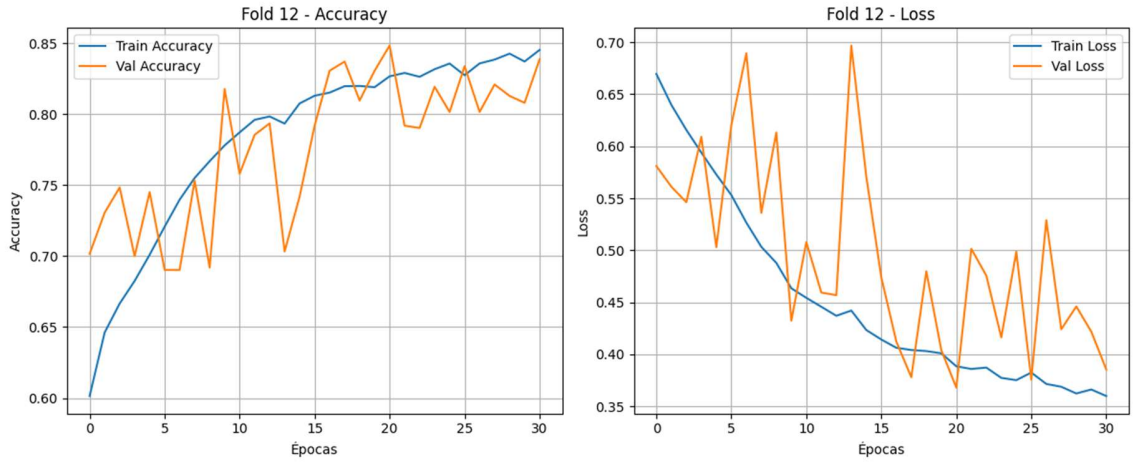


Figura 89. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 12).

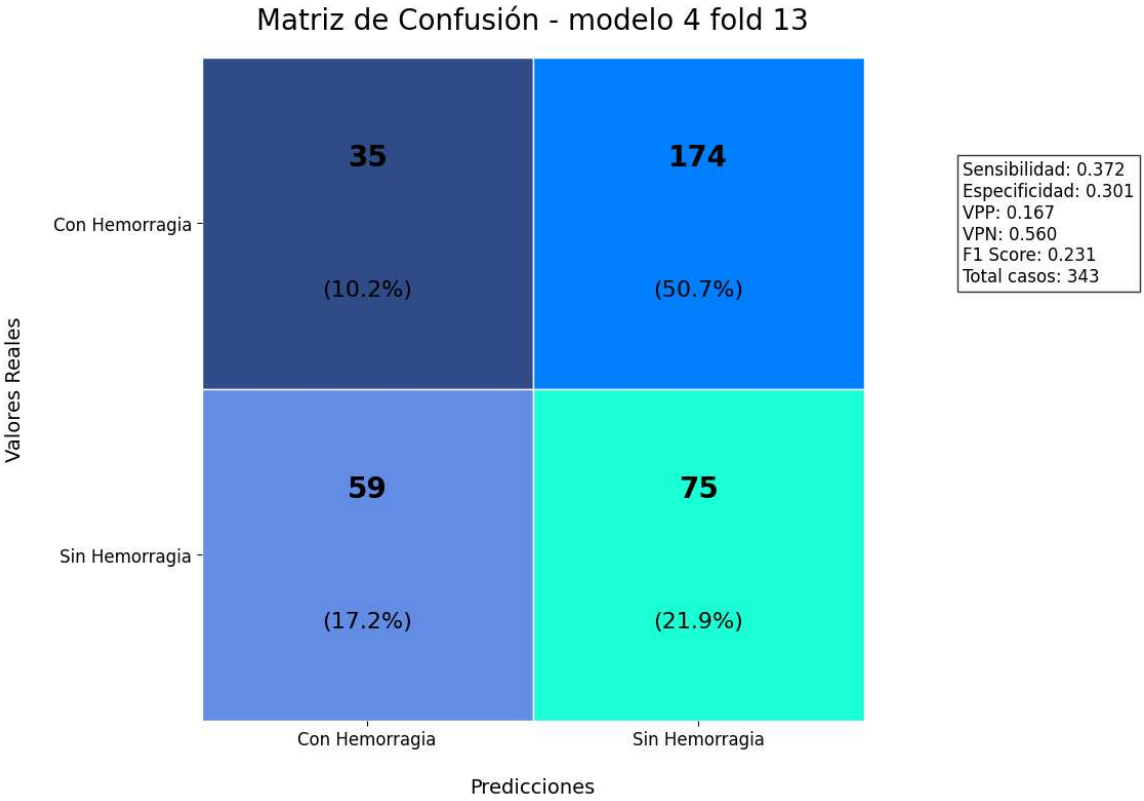


Figura 90. Matriz de confusión de los casos analizados con el modelo 4 (fold 13).

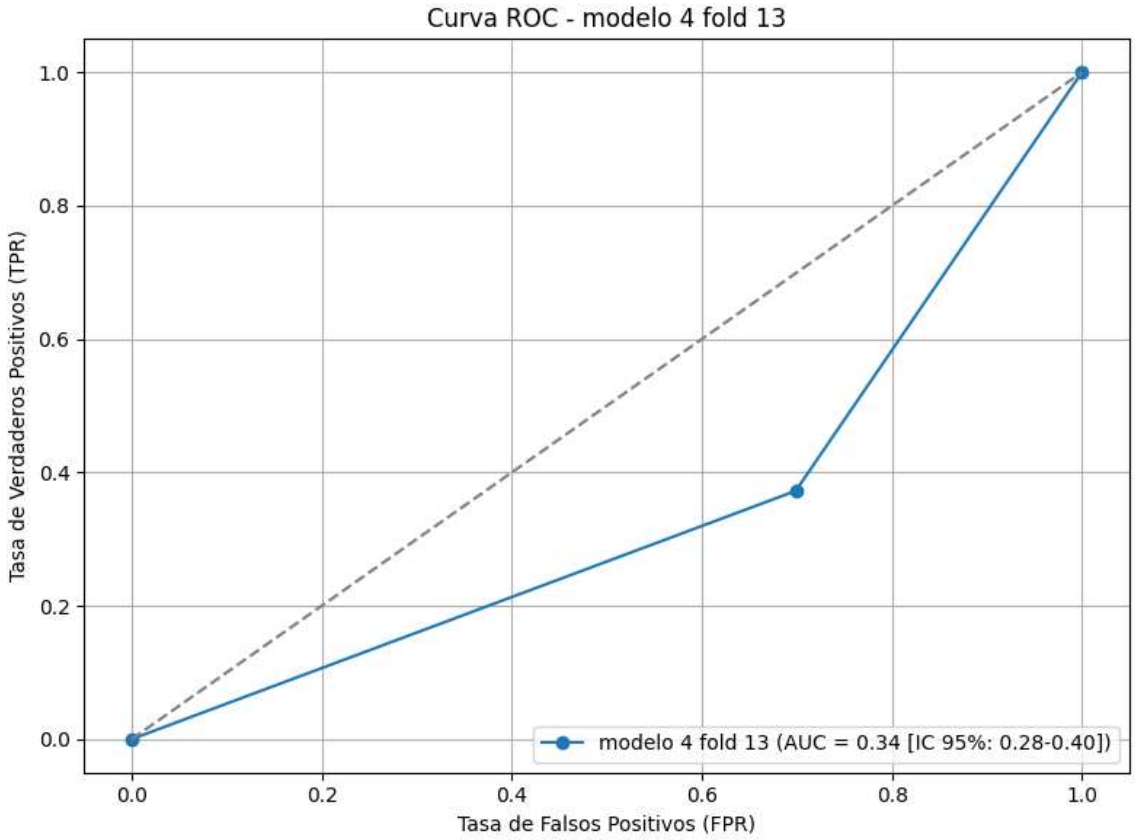


Figura 91. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 13).

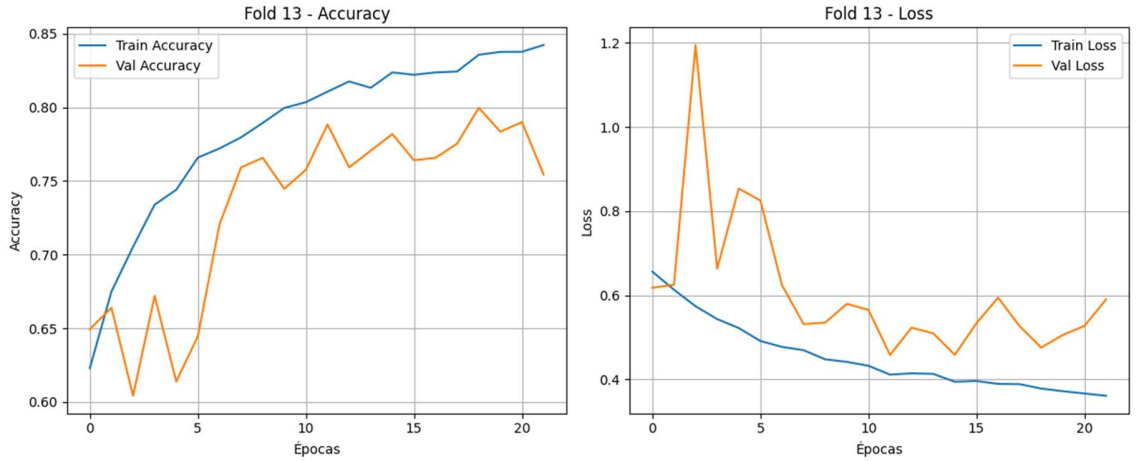


Figura 92. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 13).

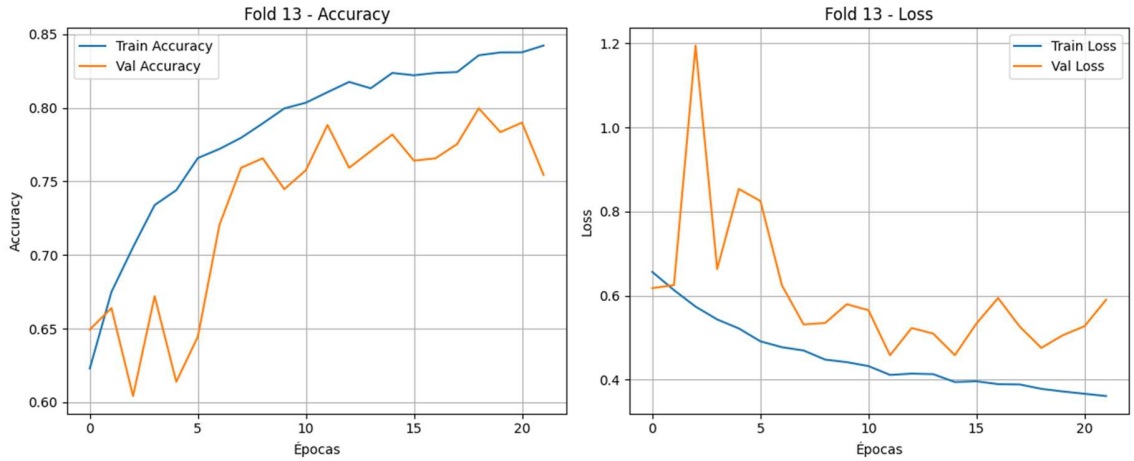


Figura 93. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 13).

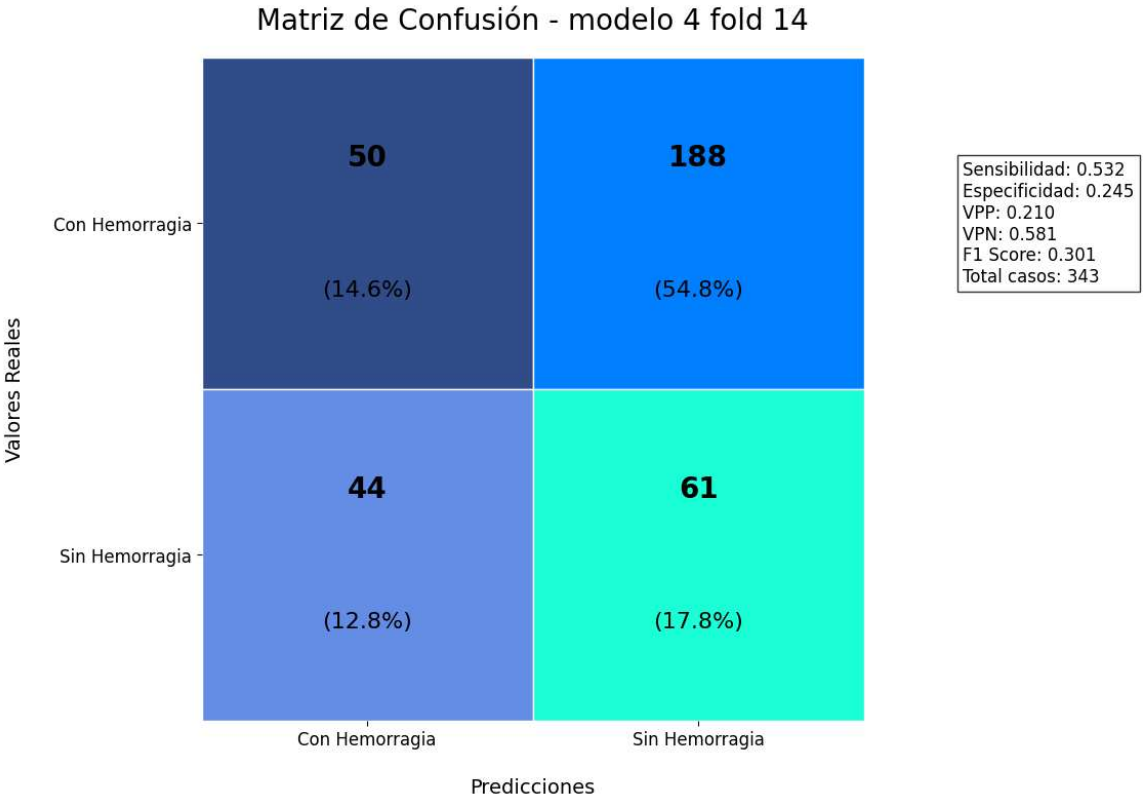


Figura 94. Matriz de confusión de los casos analizados con el modelo 4 (fold 14).

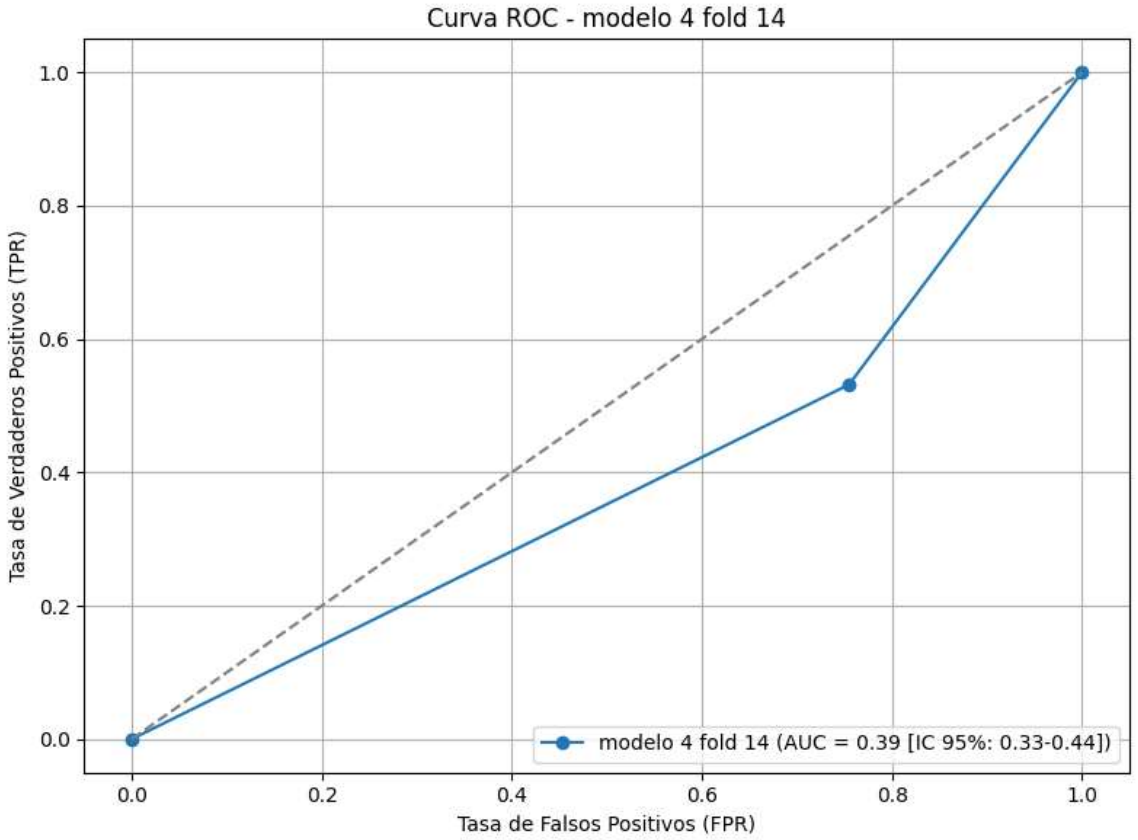


Figura 95. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 14).

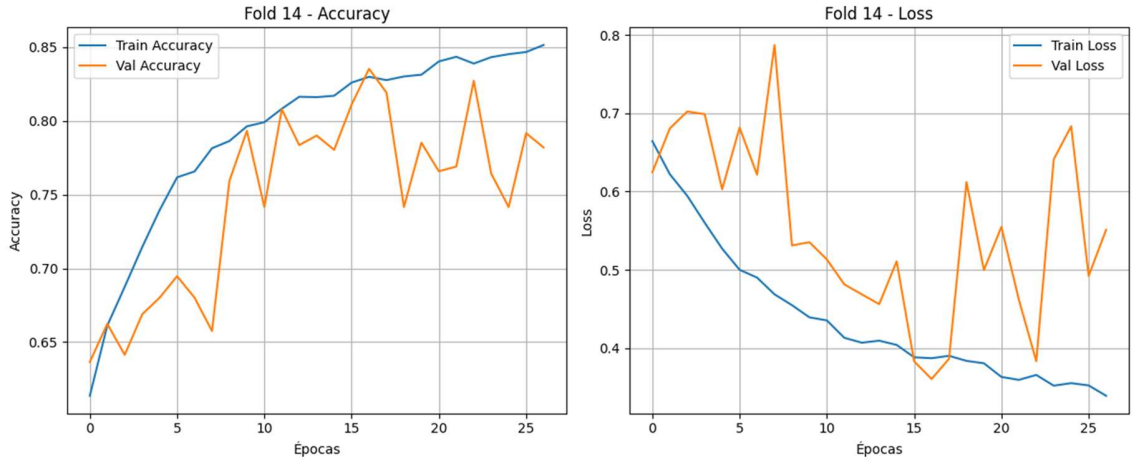


Figura 96. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 14).

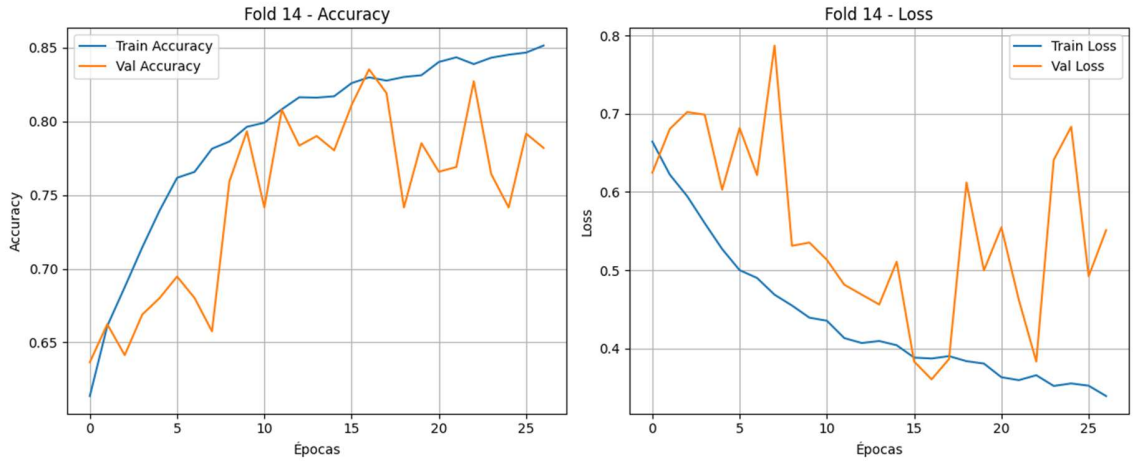


Figura 97. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 14).

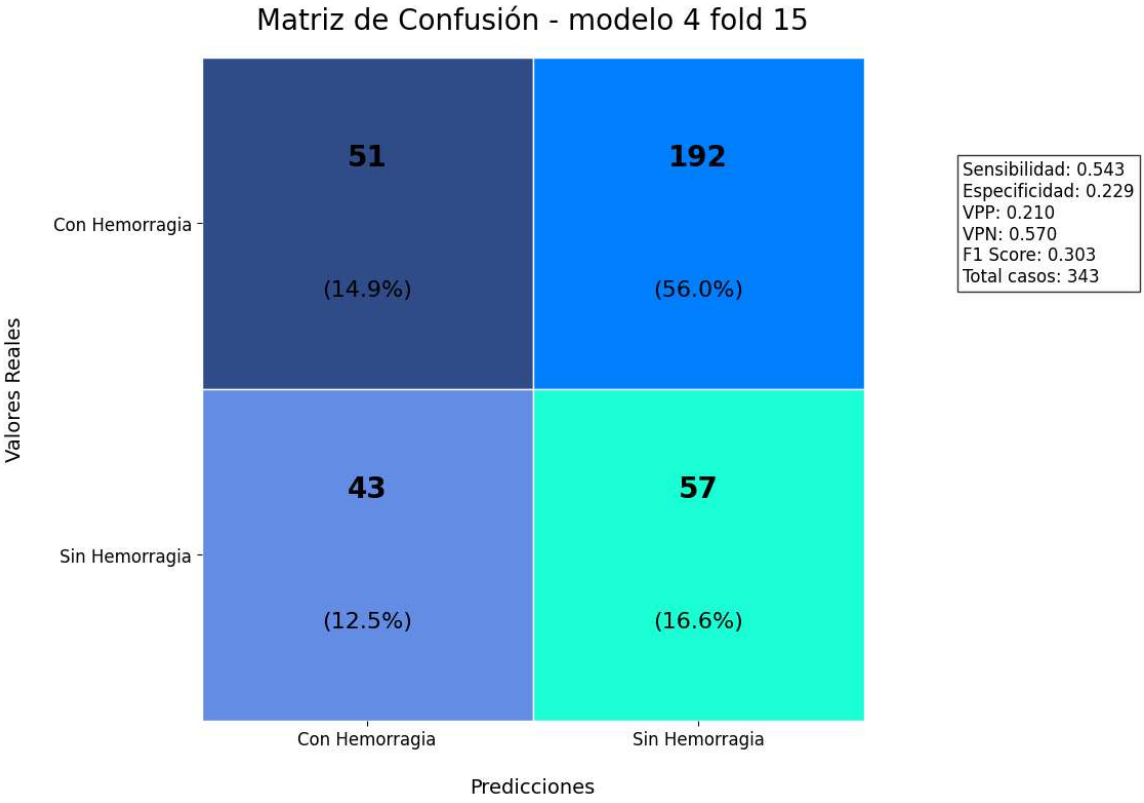


Figura 98. Matriz de confusión de los casos analizados con el modelo 4 (fold 15).

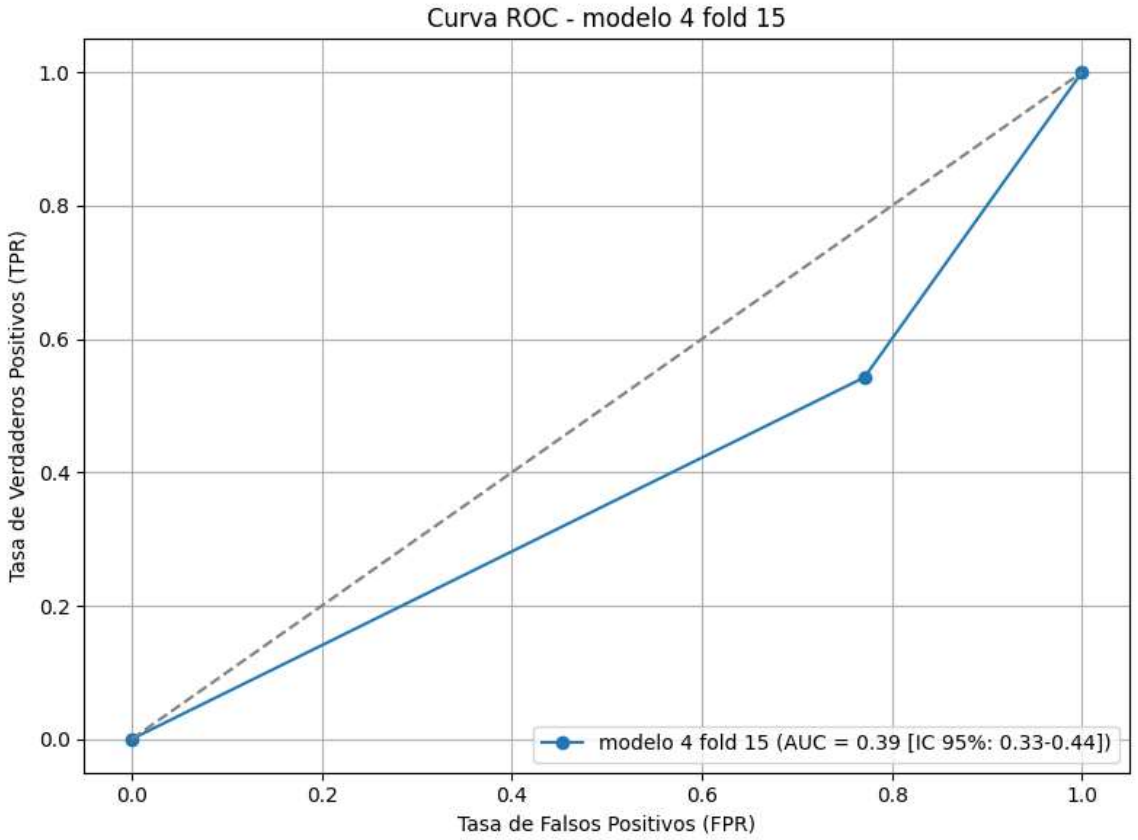


Figura 99. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 15).

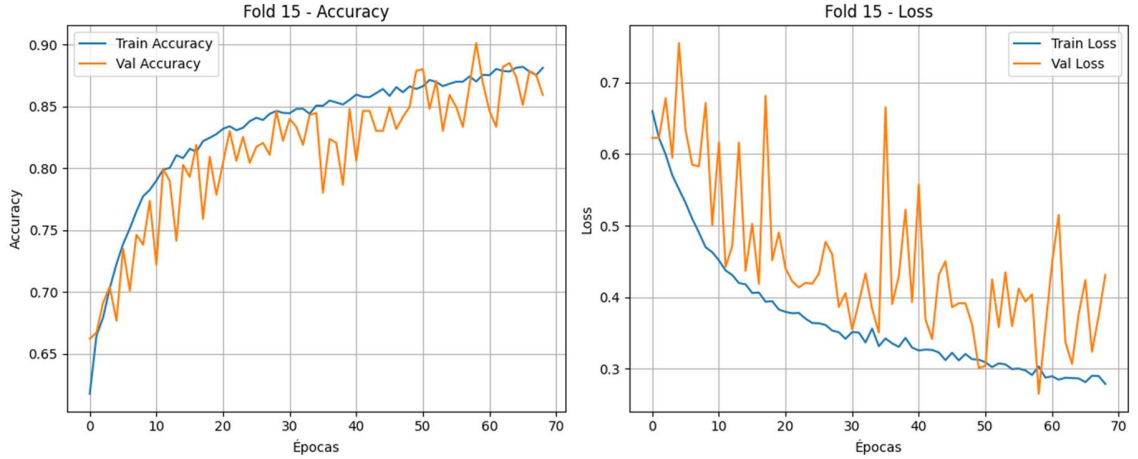


Figura 100. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 15).

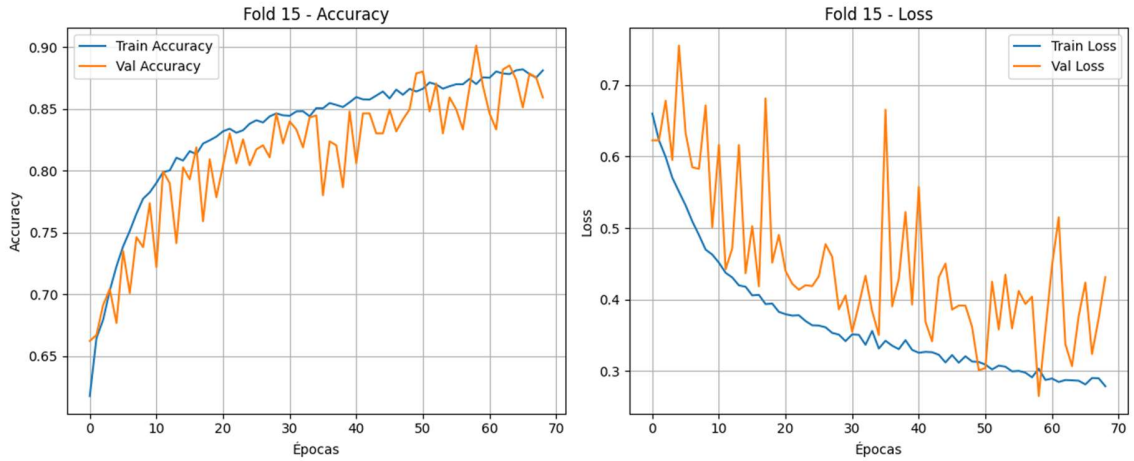


Figura 101. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 15).

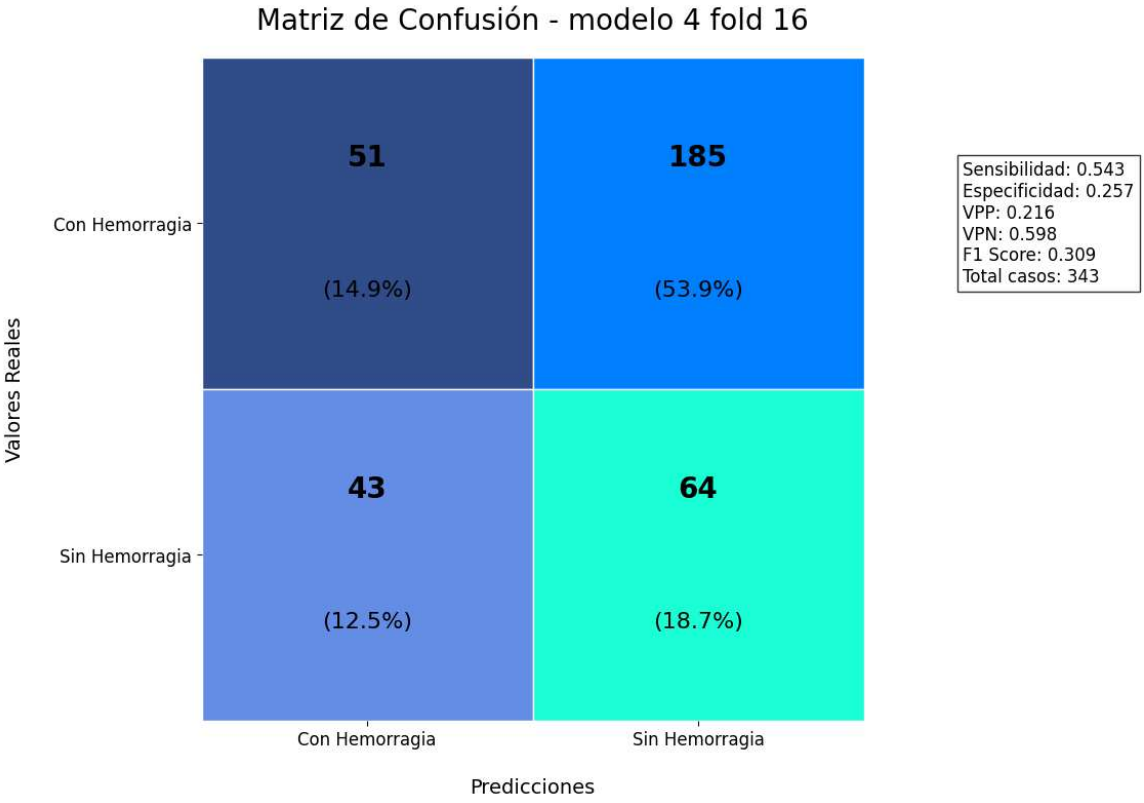


Figura 102. Matriz de confusión de los casos analizados con el modelo 4 (fold 16).

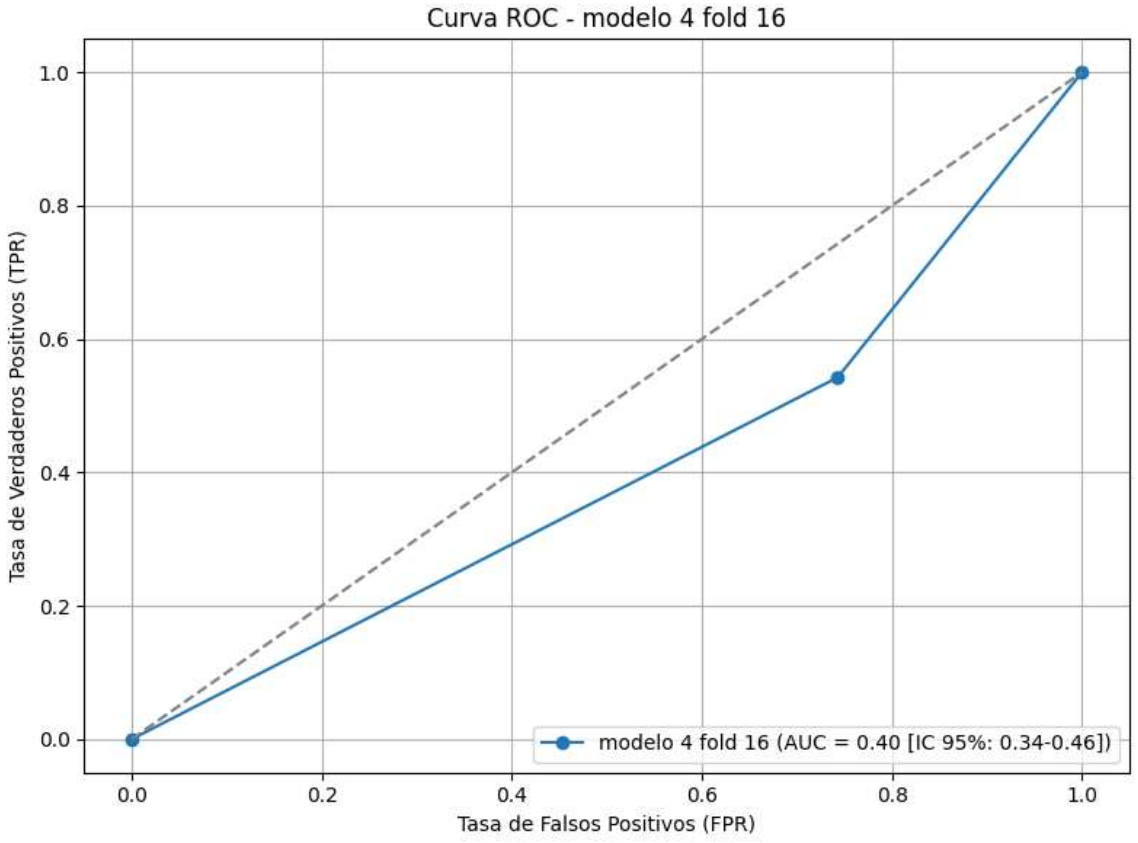


Figura 103. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 16).

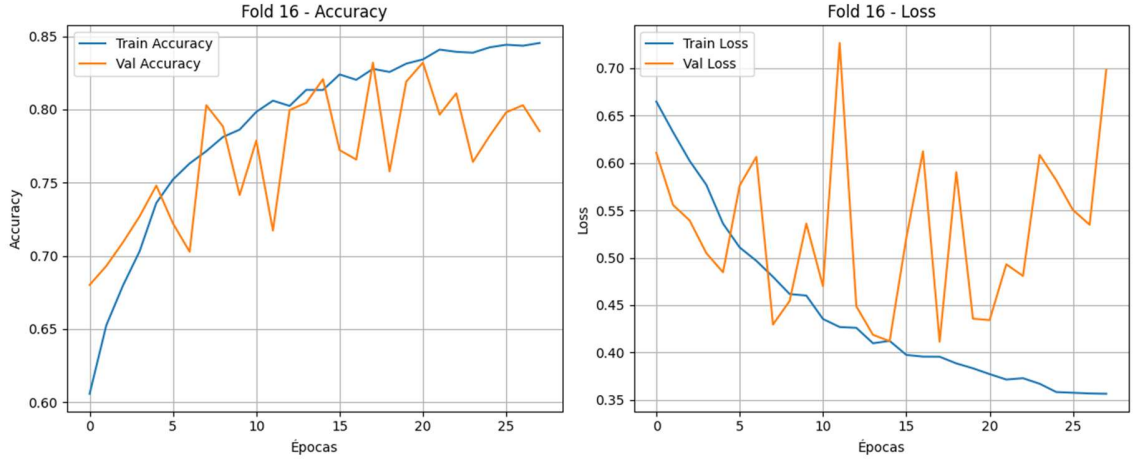


Figura 104. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 16).

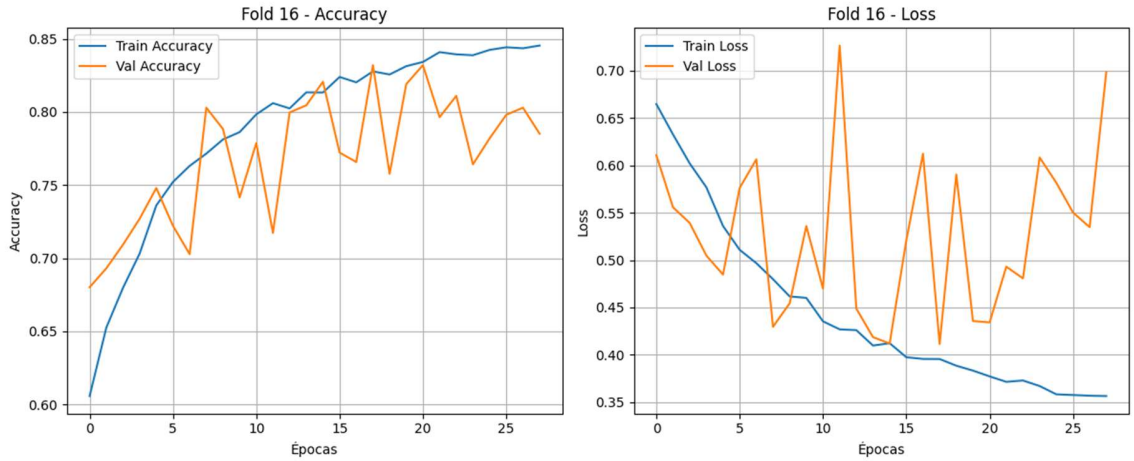


Figura 105. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 16).

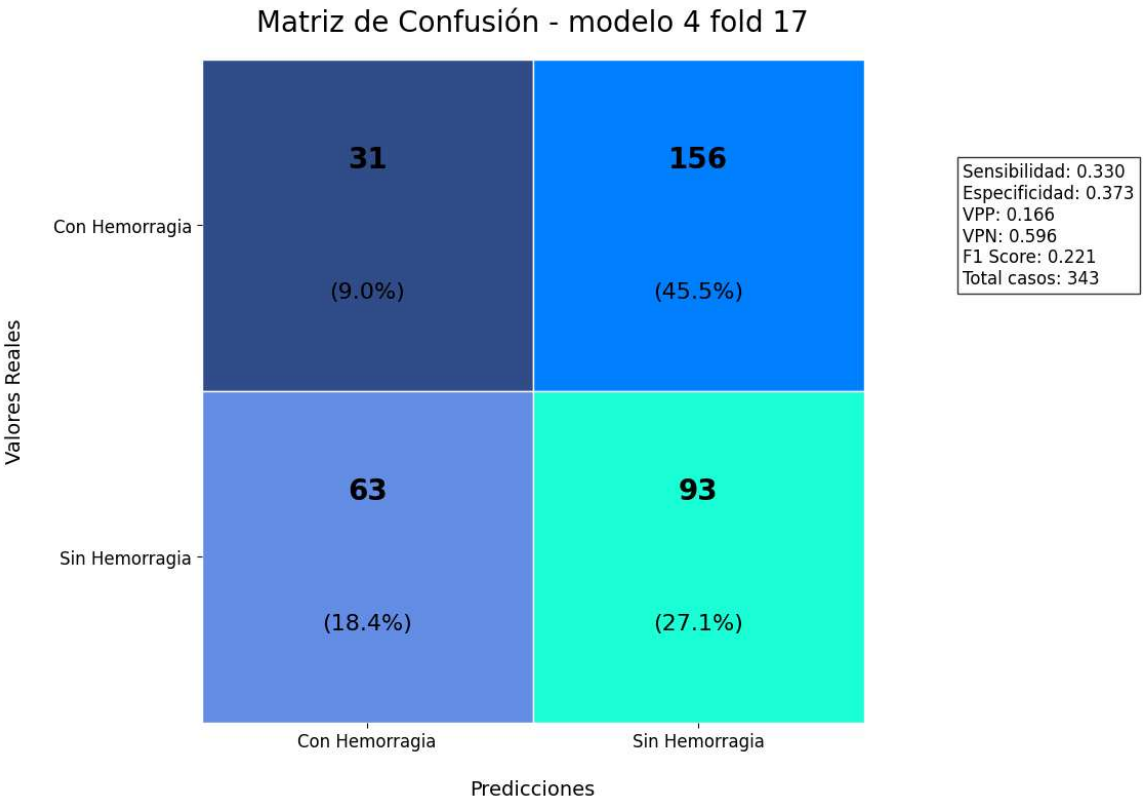


Figura 106. Matriz de confusión de los casos analizados con el modelo 4 (fold 17).

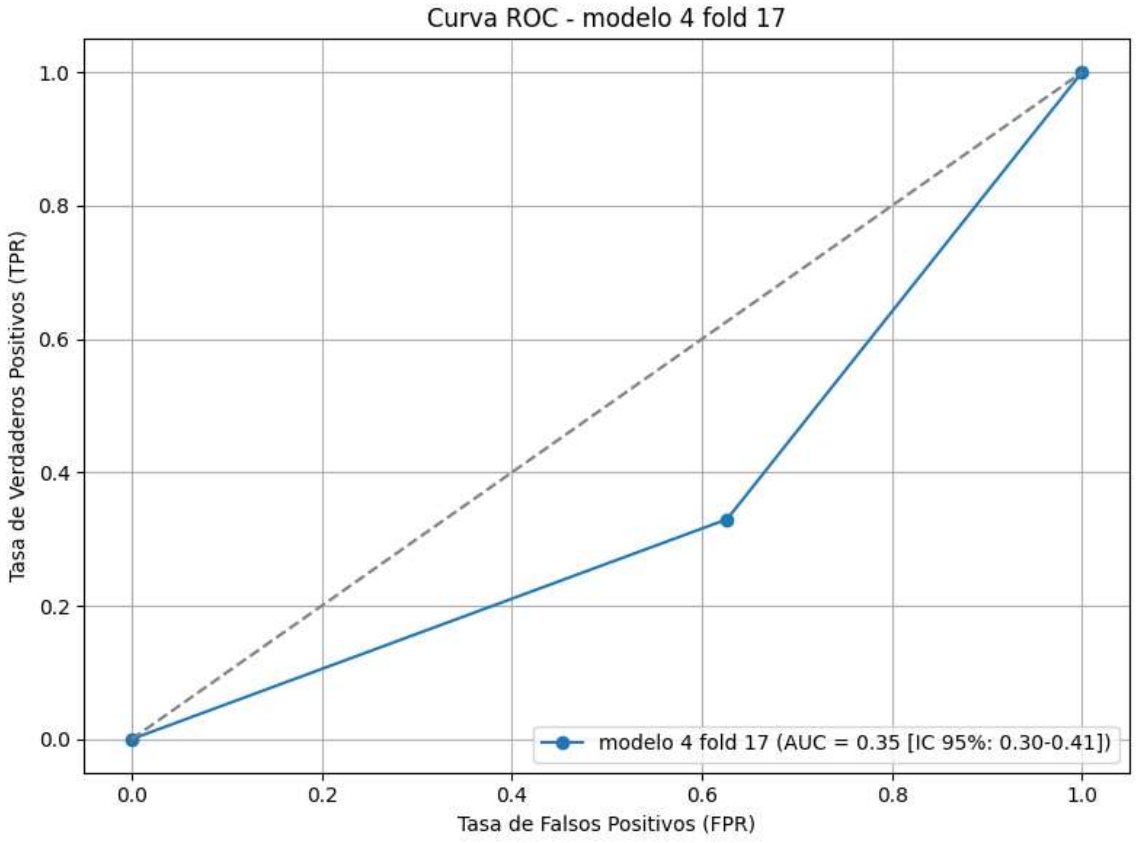


Figura 107. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 17).

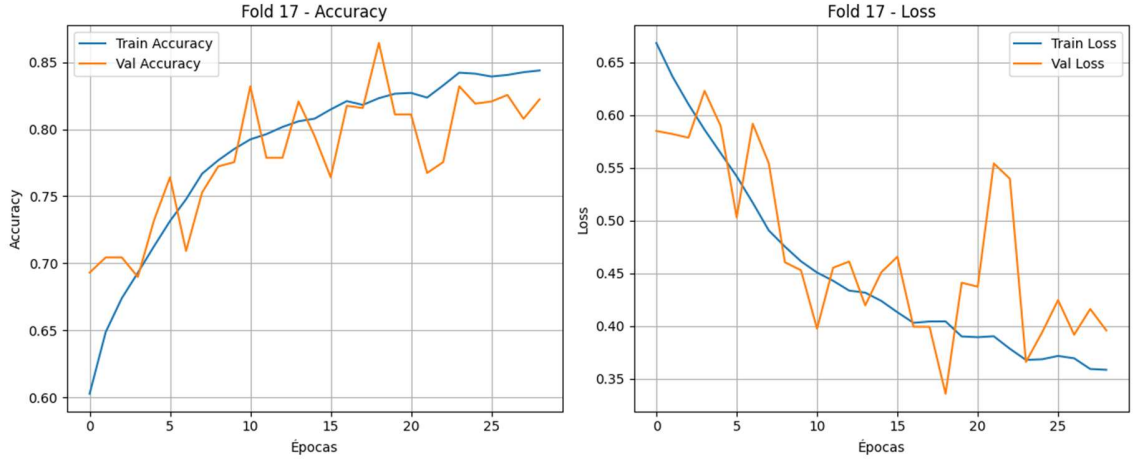


Figura 108. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 17).

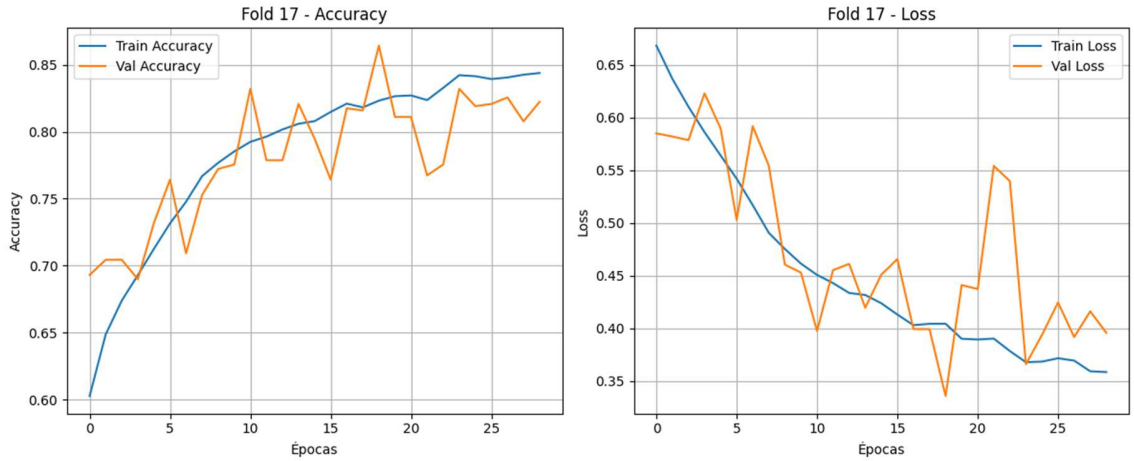


Figura 109. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 17).

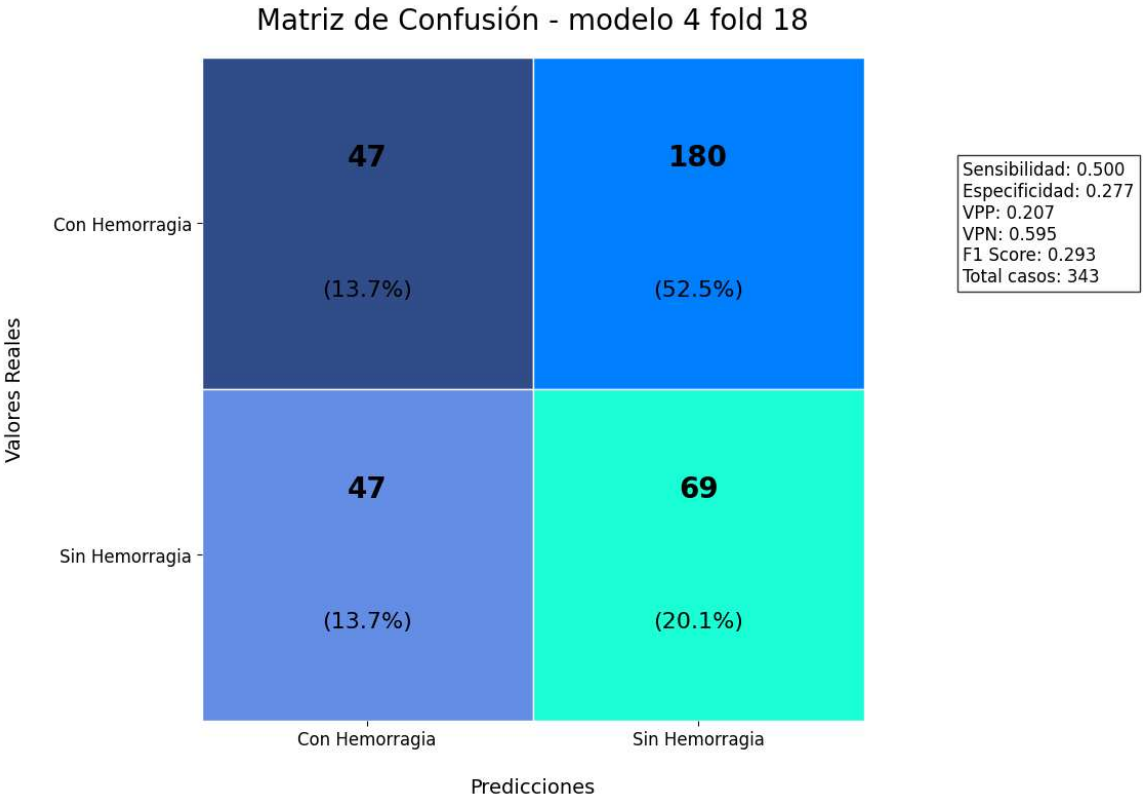


Figura 110. Matriz de confusión de los casos analizados con el modelo 4 (fold 18).

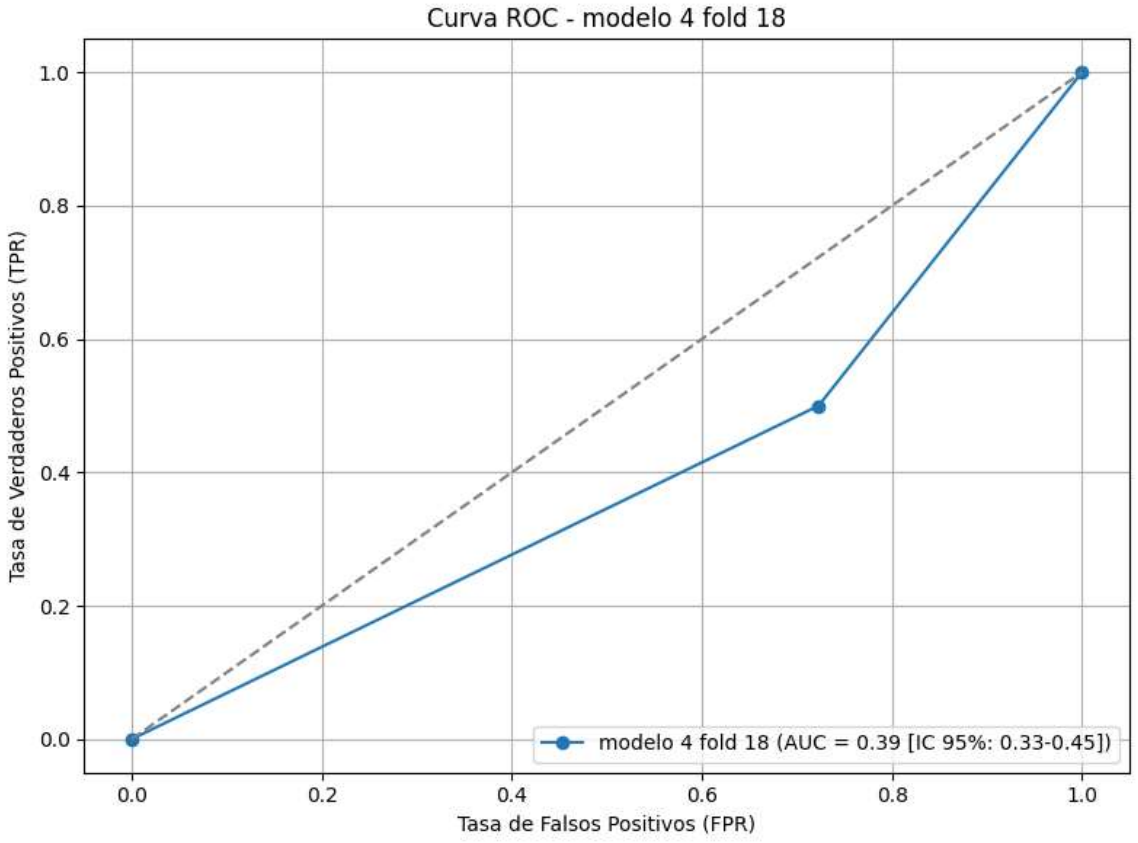


Figura 111. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 18).

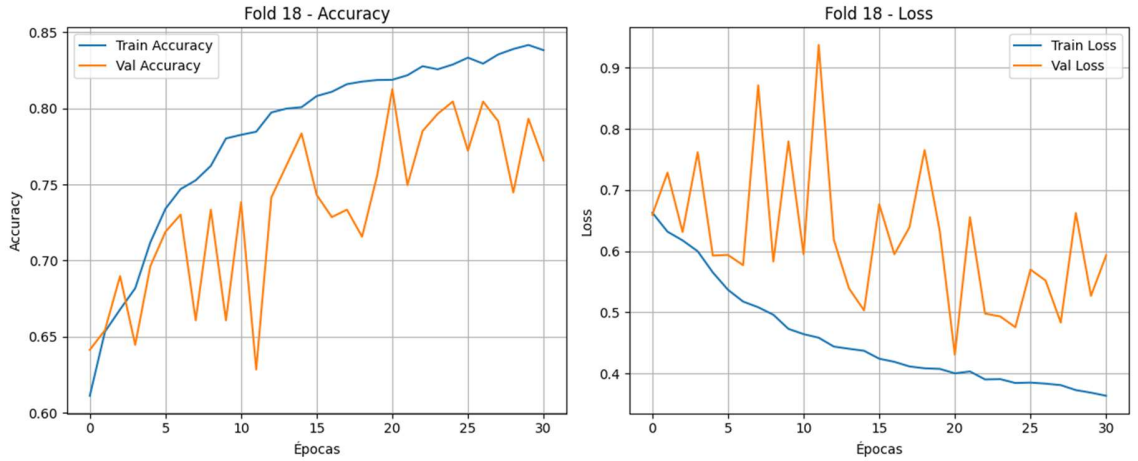


Figura 112. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 18).

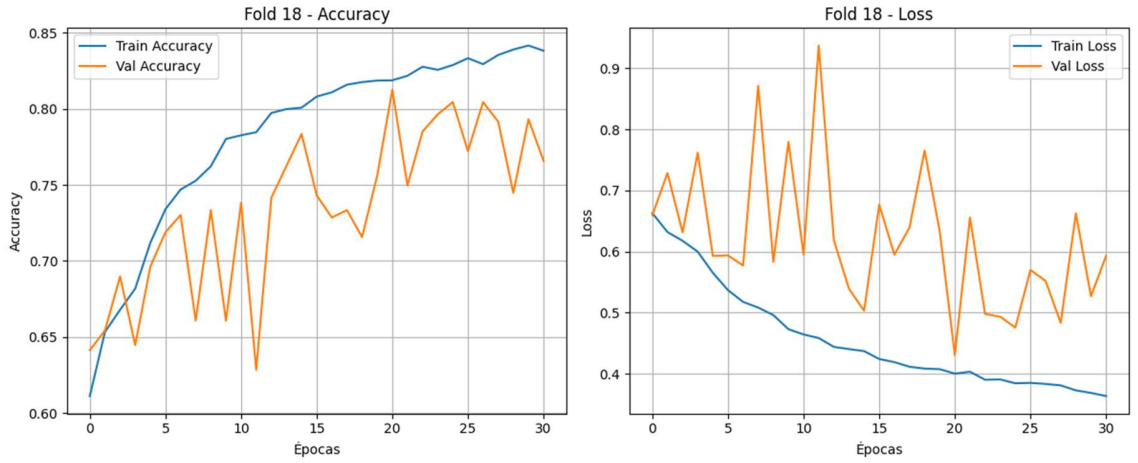


Figura 113. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 18).

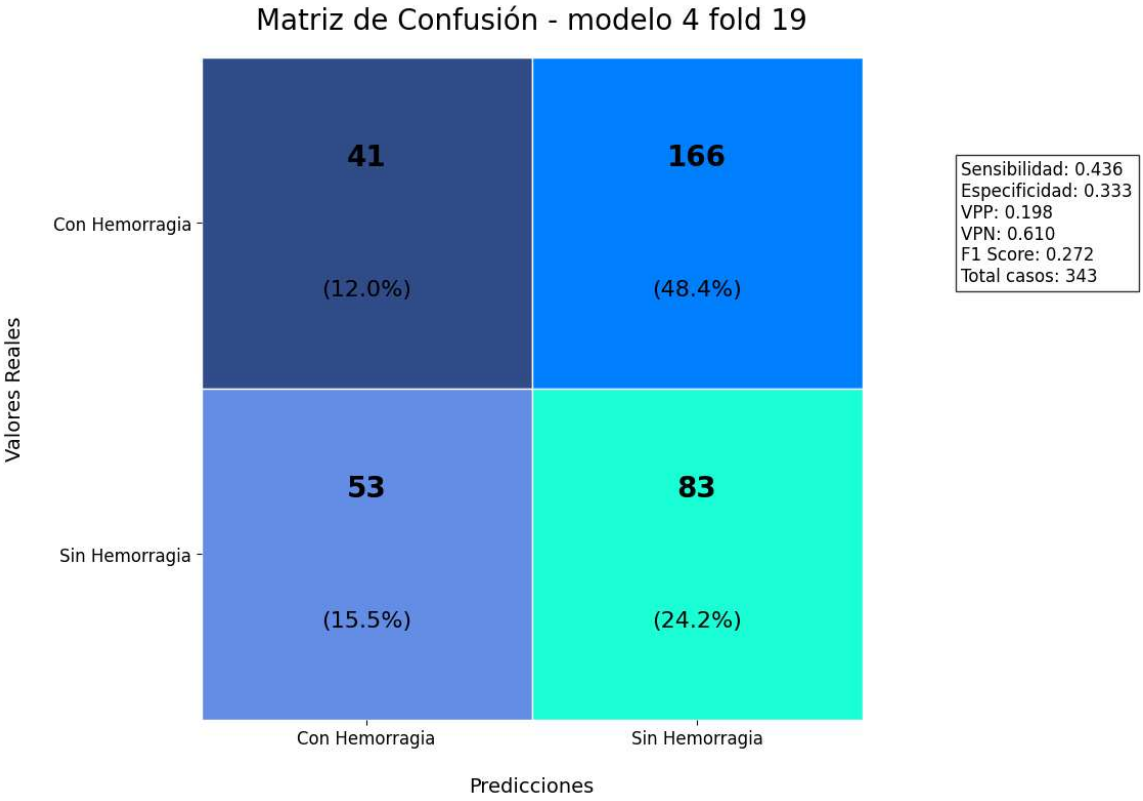


Figura 114. Matriz de confusión de los casos analizados con el modelo 4 (fold 19).

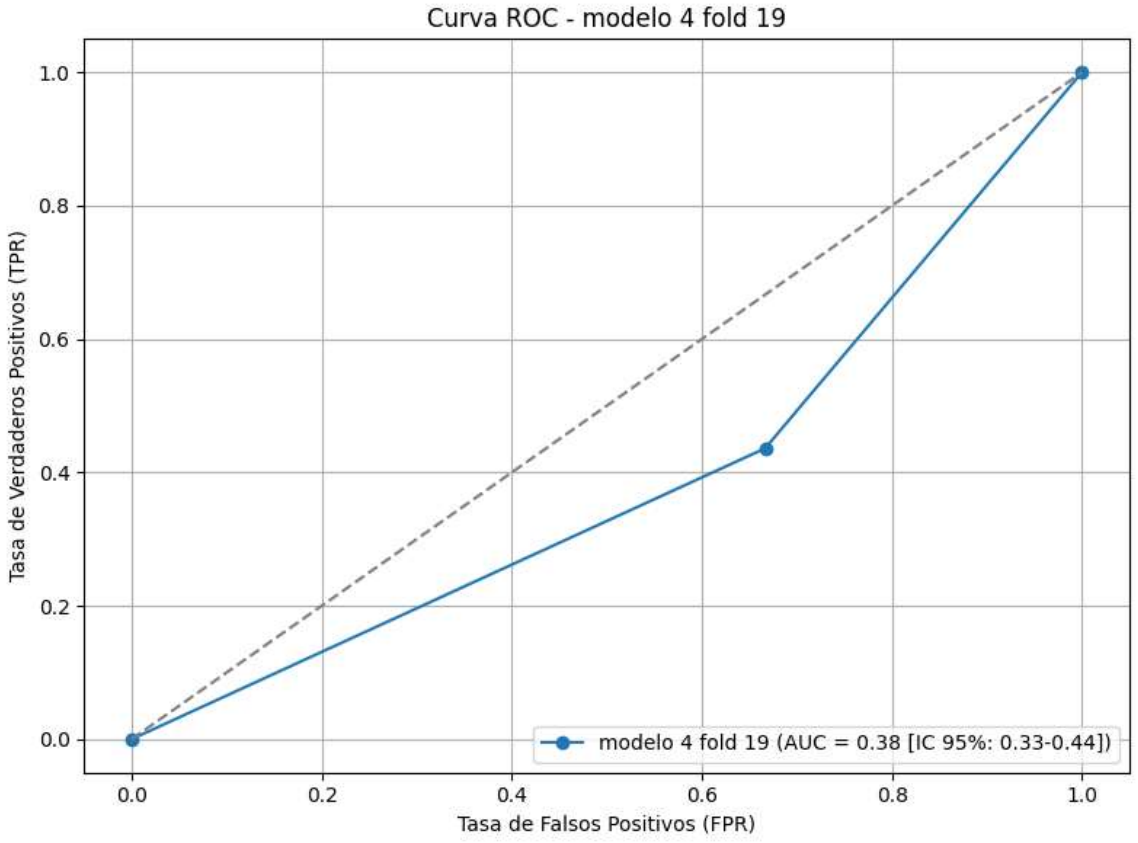


Figura 115. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 19).

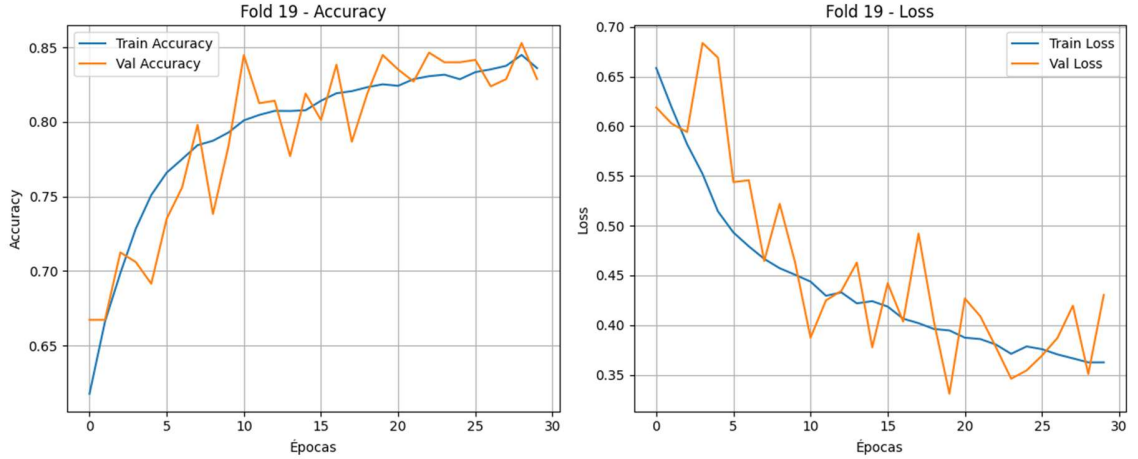


Figura 116. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 19).

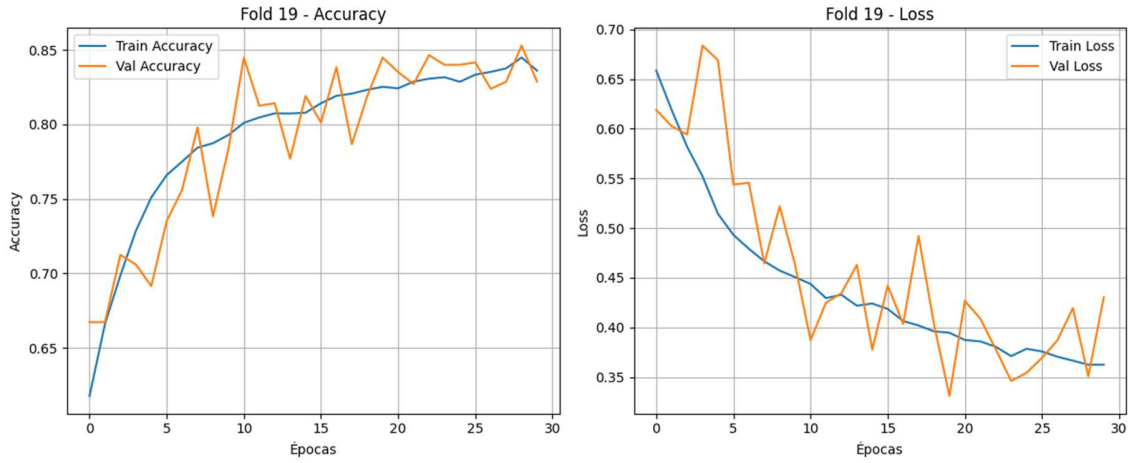


Figura 117. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 19).

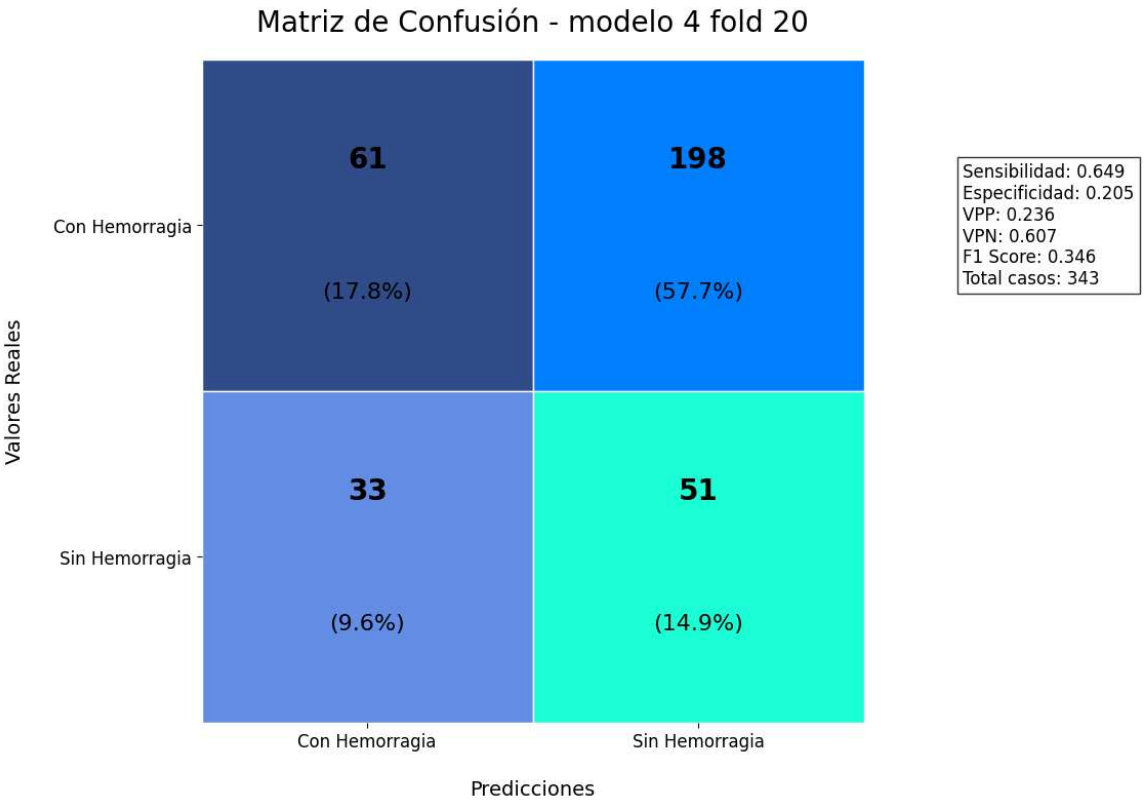


Figura 118. Matriz de confusión de los casos analizados con el modelo 4 (fold 20).

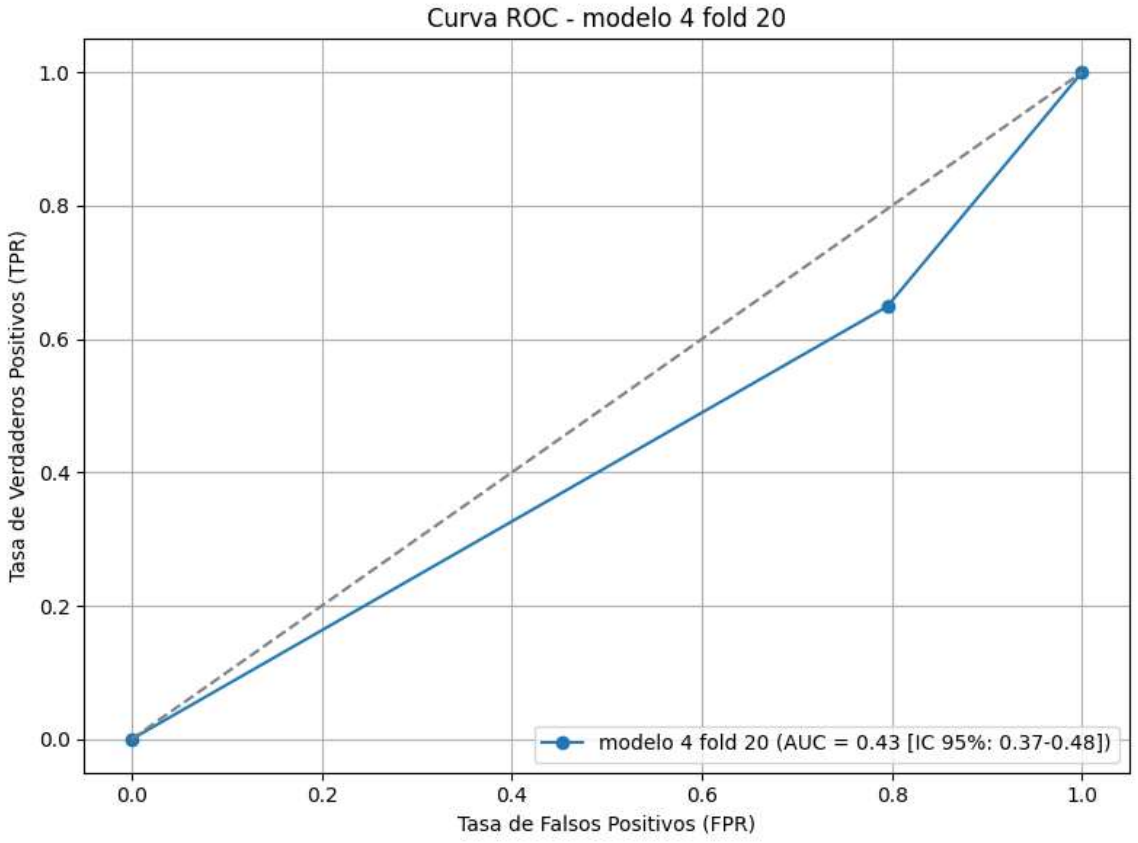


Figura 119. Curva AUC-ROC evaluando el desempeño del modelo 4 de clasificación (fold 20).

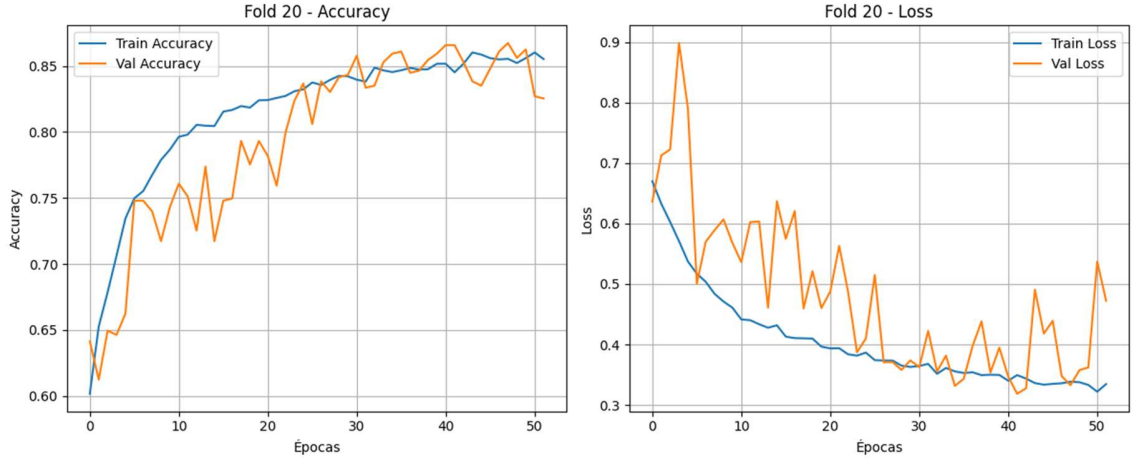


Figura 120. Evolución de la precisión del Modelo 4 durante el Entrenamiento y Validación (fold 20).

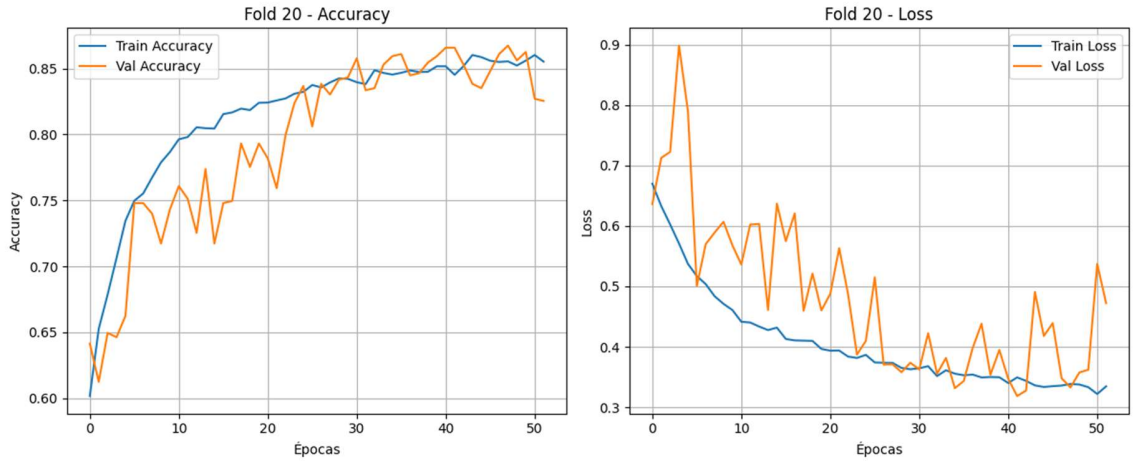


Figura 121. Evolución de la pérdida del Modelo 4 durante el Entrenamiento y Validación (fold 20).

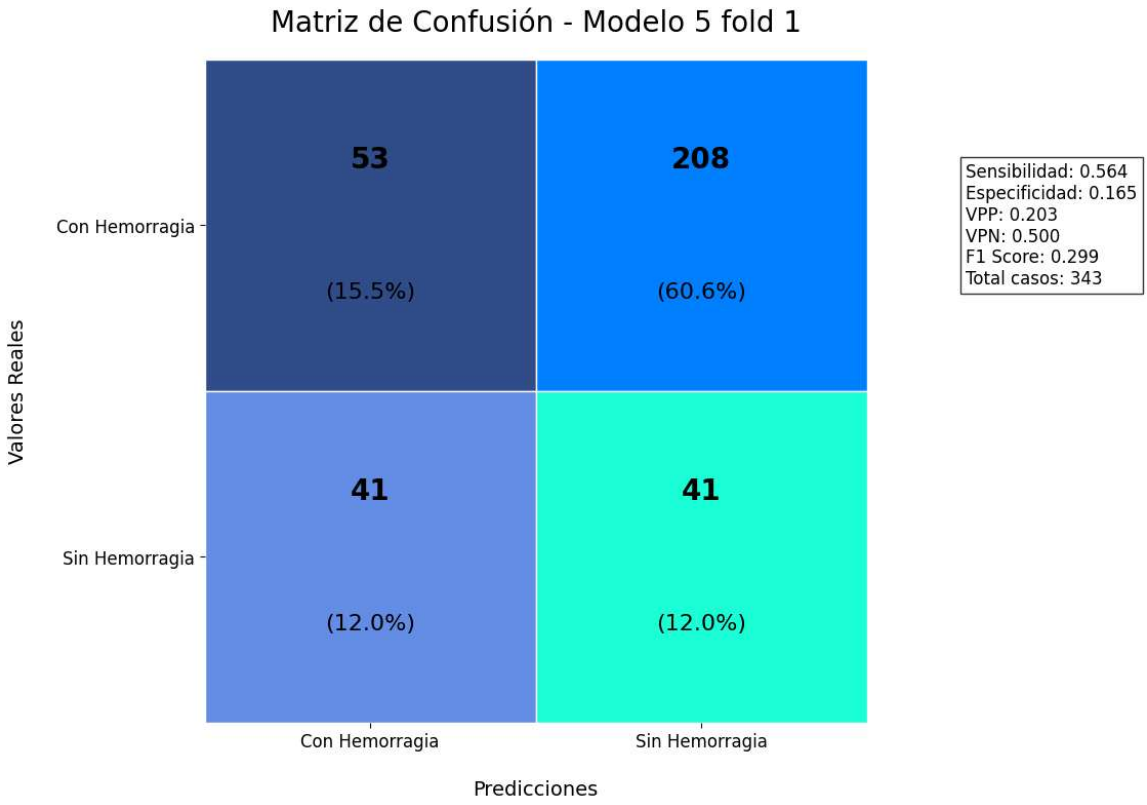


Figura 122. Matriz de confusión de los casos analizados con el modelo 5 (fold 1).

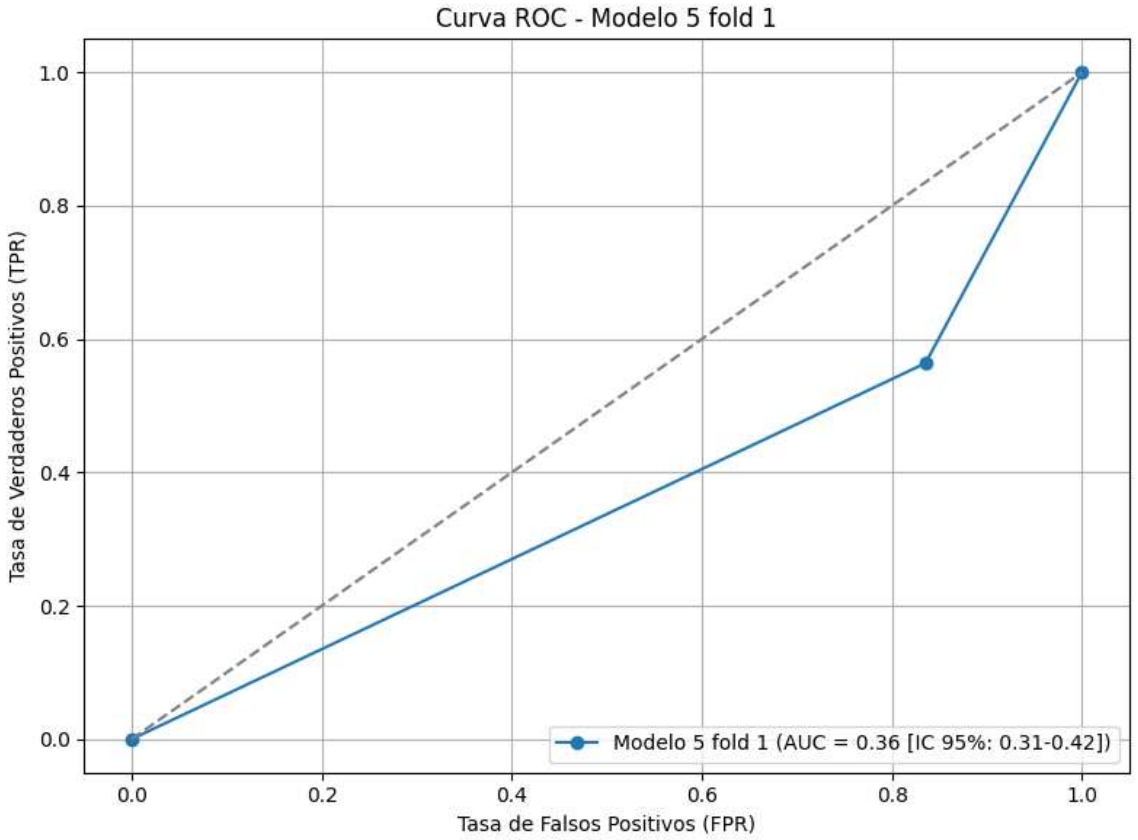


Figura 123. Curva AUC-ROC evaluando el desempeño del modelo 5 de clasificación (fold 1).

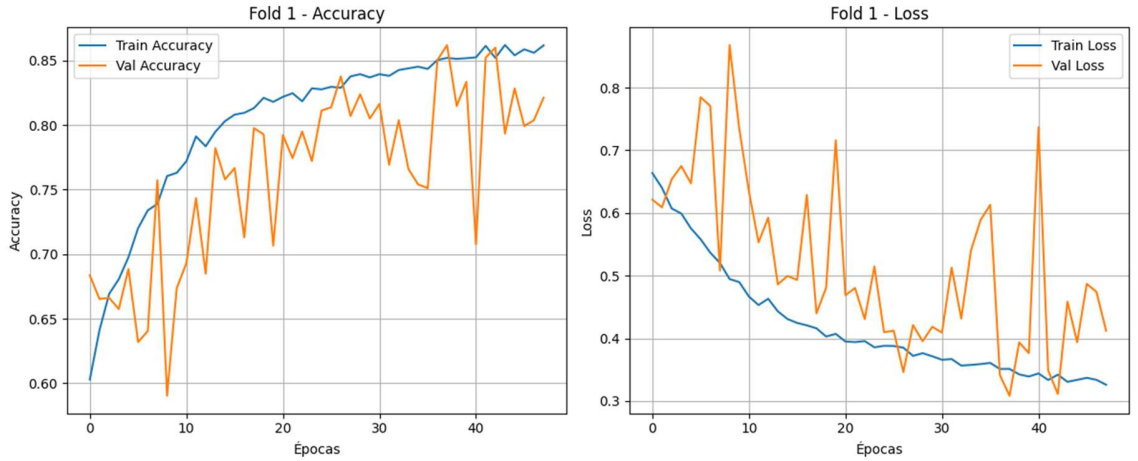


Figura 124. Evolución de la precisión del Modelo 5 durante el Entrenamiento y Validación (fold 1).

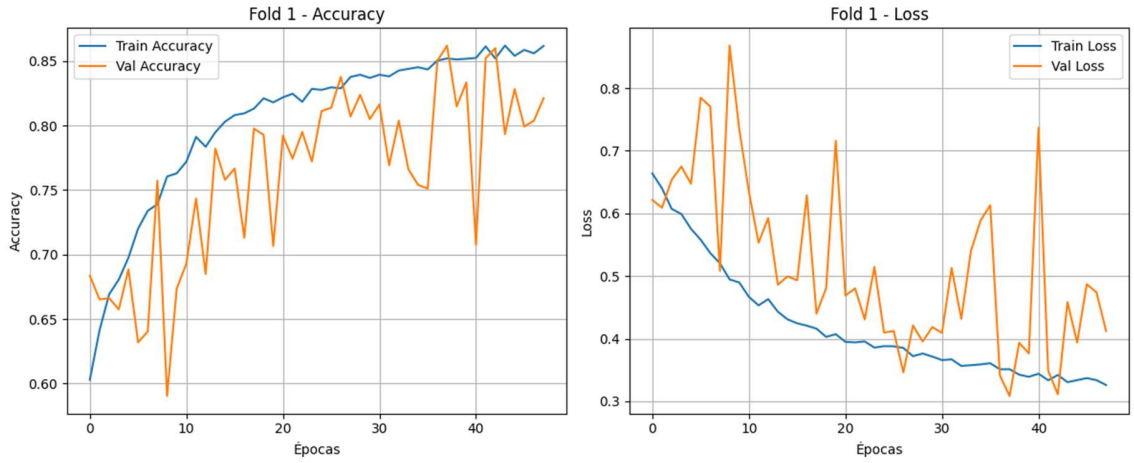


Figura 125. Evolución de la pérdida del Modelo 5 durante el Entrenamiento y Validación (fold 1).

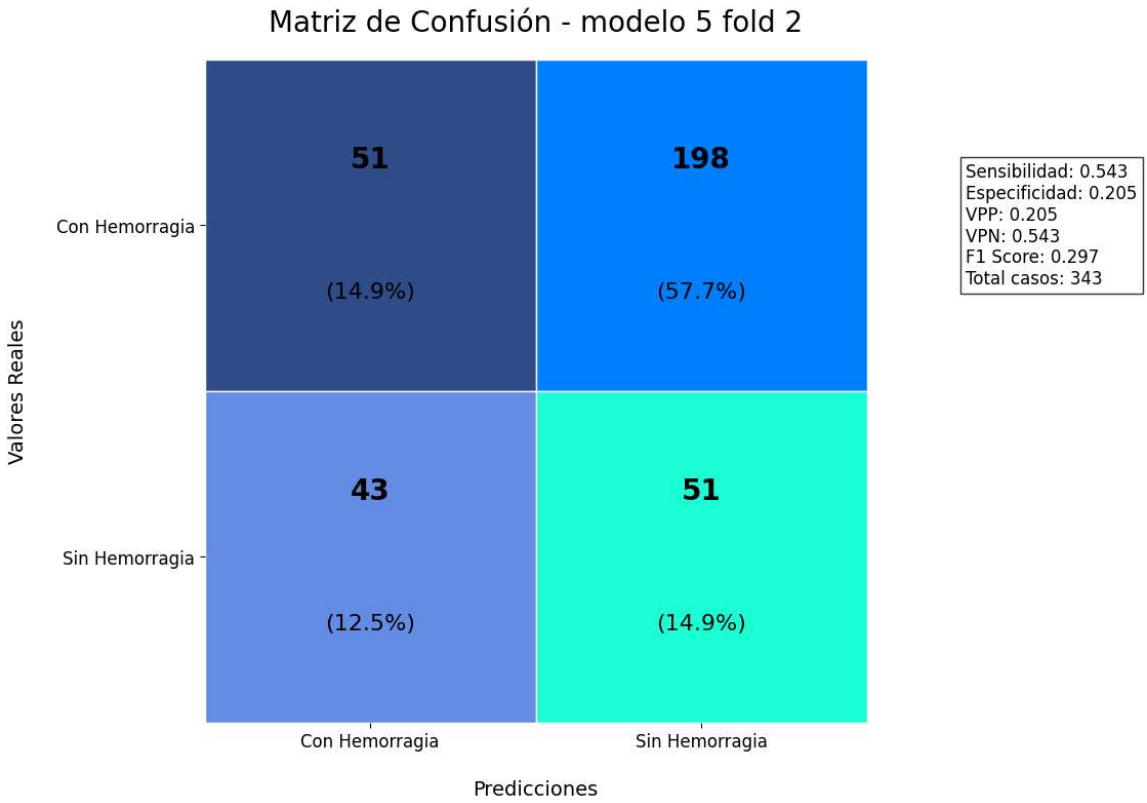


Figura 126. Matriz de confusión de los casos analizados con el modelo 5 (fold 2).

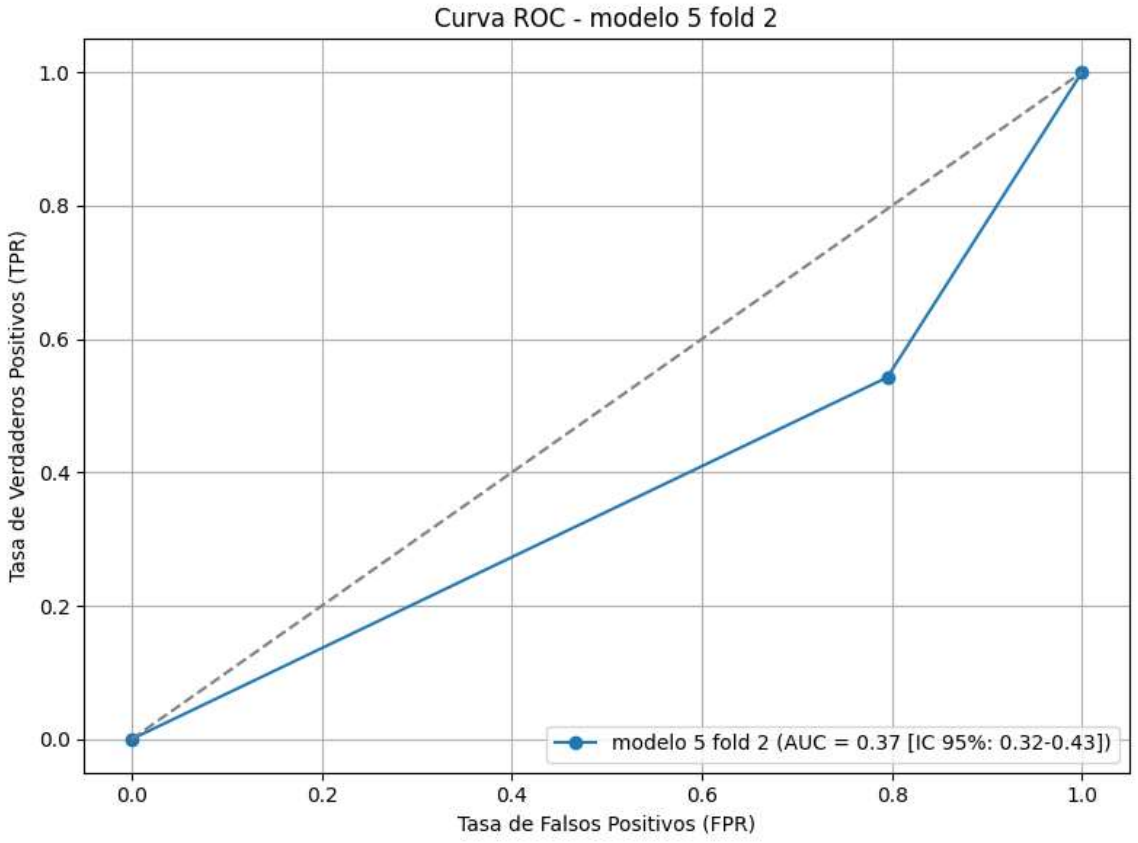


Figura 127. Curva AUC-ROC evaluando el desempeño del modelo 5 de clasificación (fold 2).

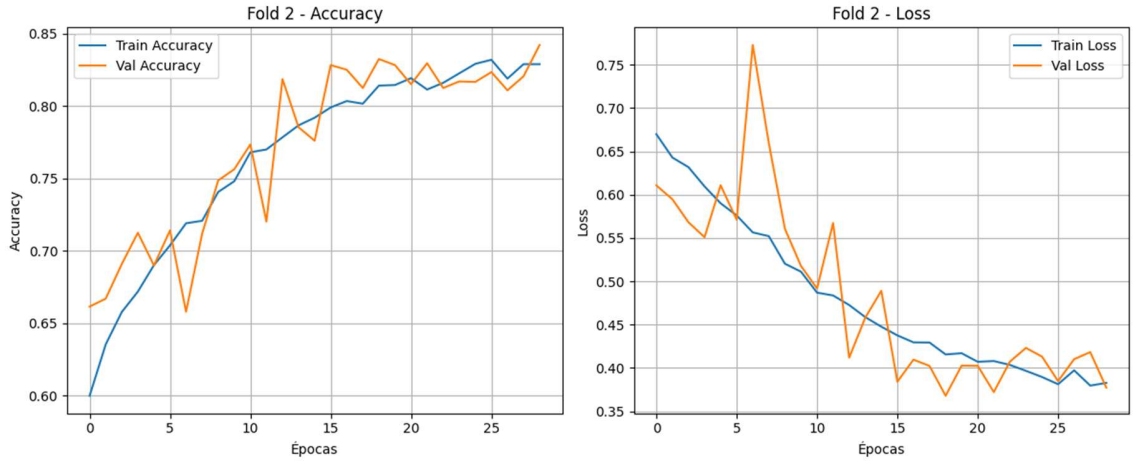


Figura 128. Evolución de la precisión del Modelo 5 durante el Entrenamiento y Validación (fold 2).

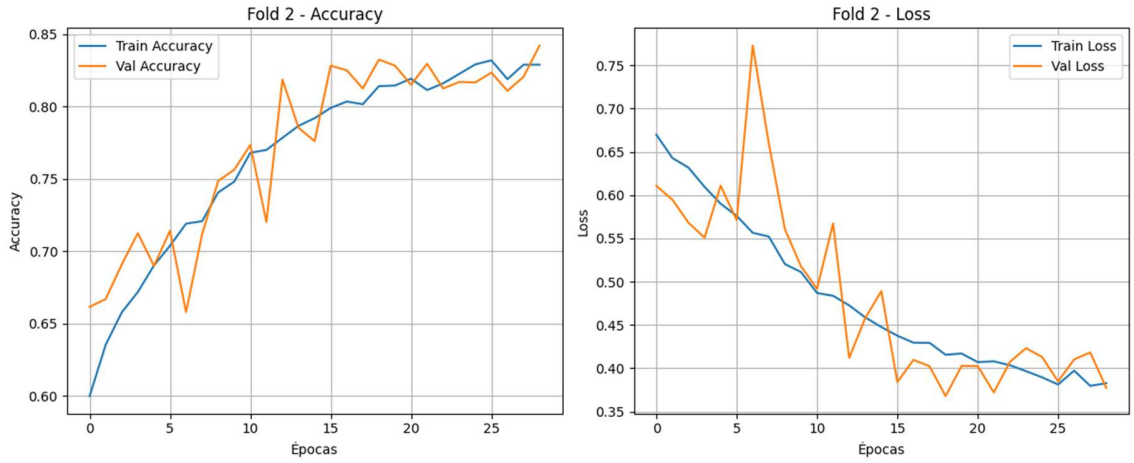


Figura 129. Evolución de la pérdida del Modelo 5 durante el Entrenamiento y Validación (fold 2).

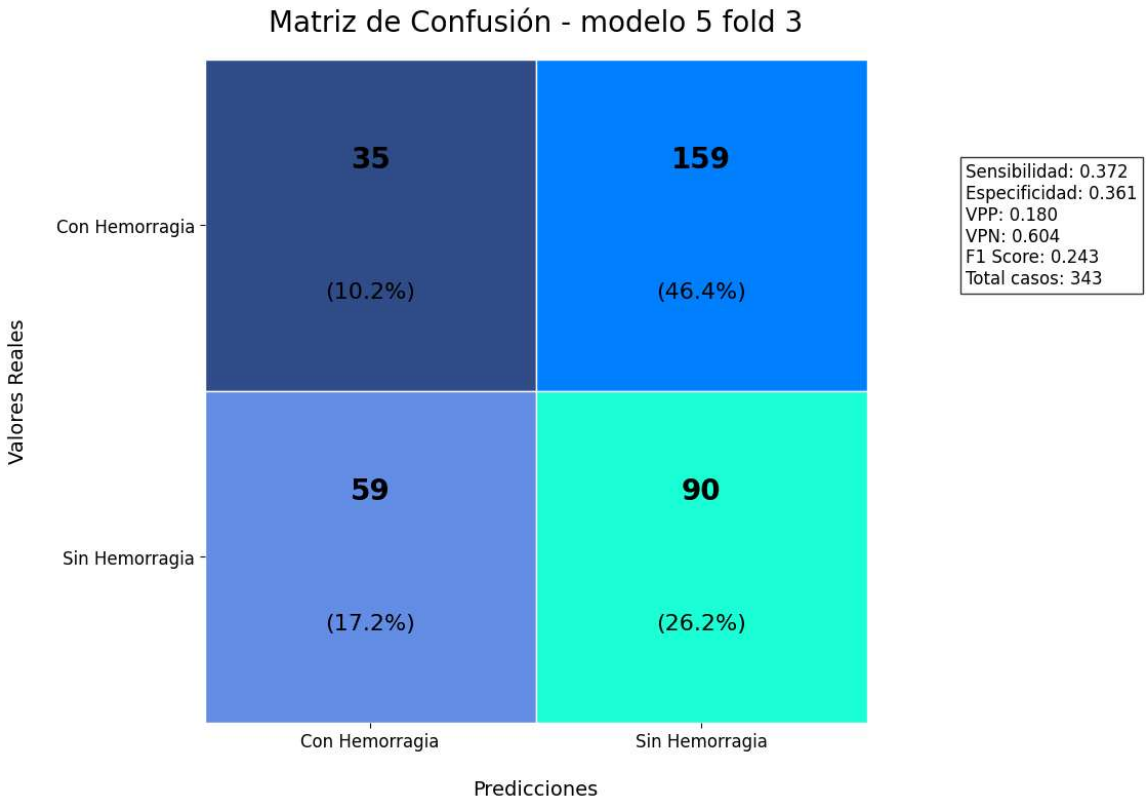


Figura 130. Matriz de confusión de los casos analizados con el modelo 5 (fold 3).

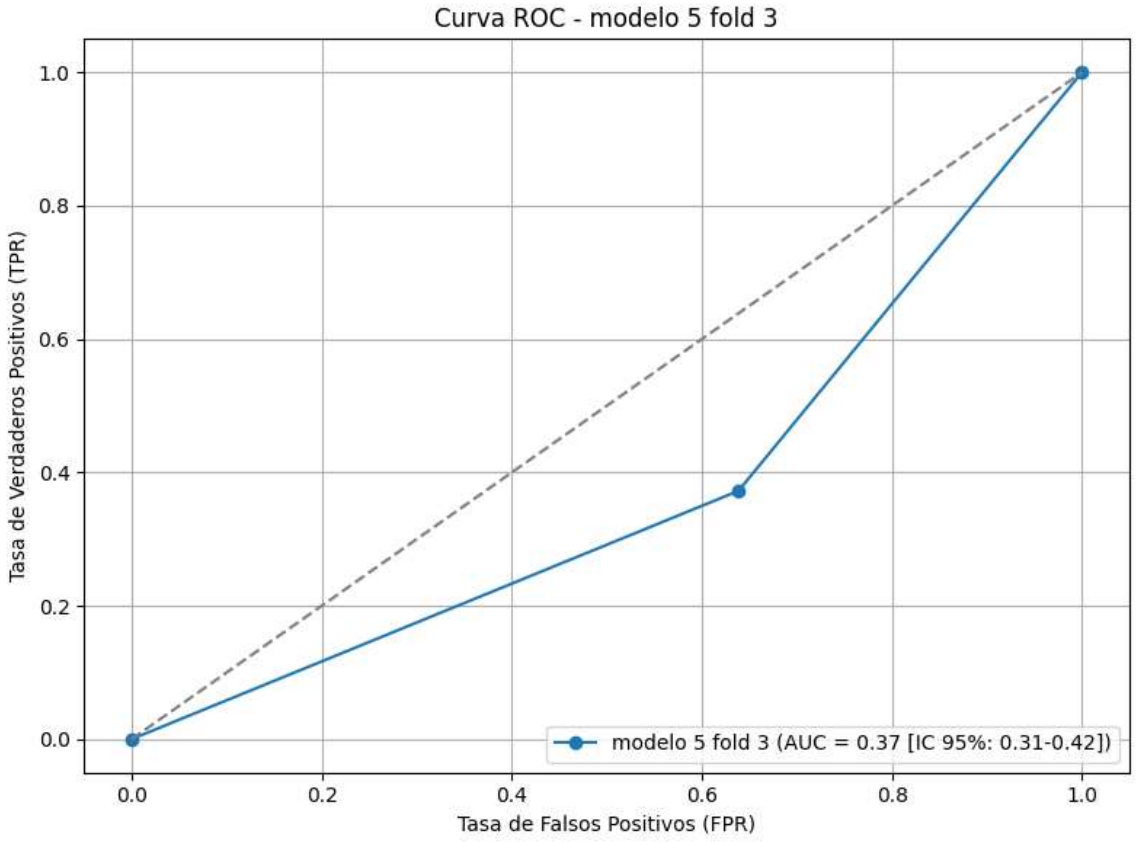


Figura 131. Curva AUC-ROC evaluando el desempeño del modelo 5 de clasificación (fold 3).

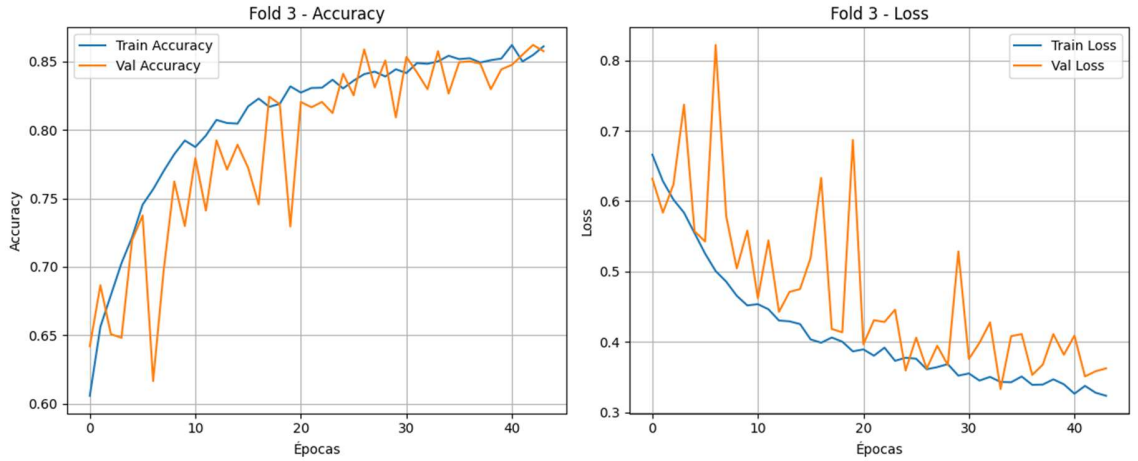


Figura 132. Evolución de la precisión del Modelo 5 durante el Entrenamiento y Validación (fold 3).

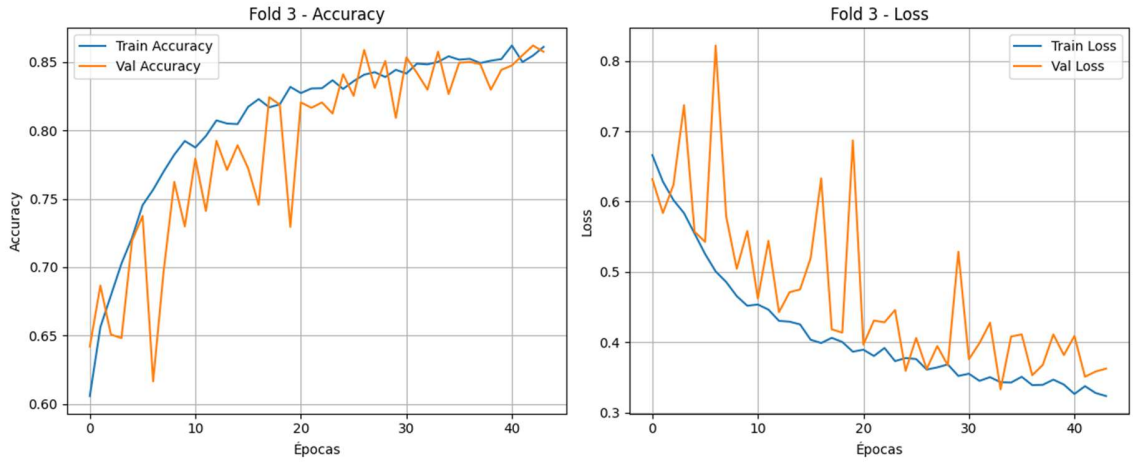


Figura 133. Evolución de la pérdida del Modelo 5 durante el Entrenamiento y Validación (fold 3).

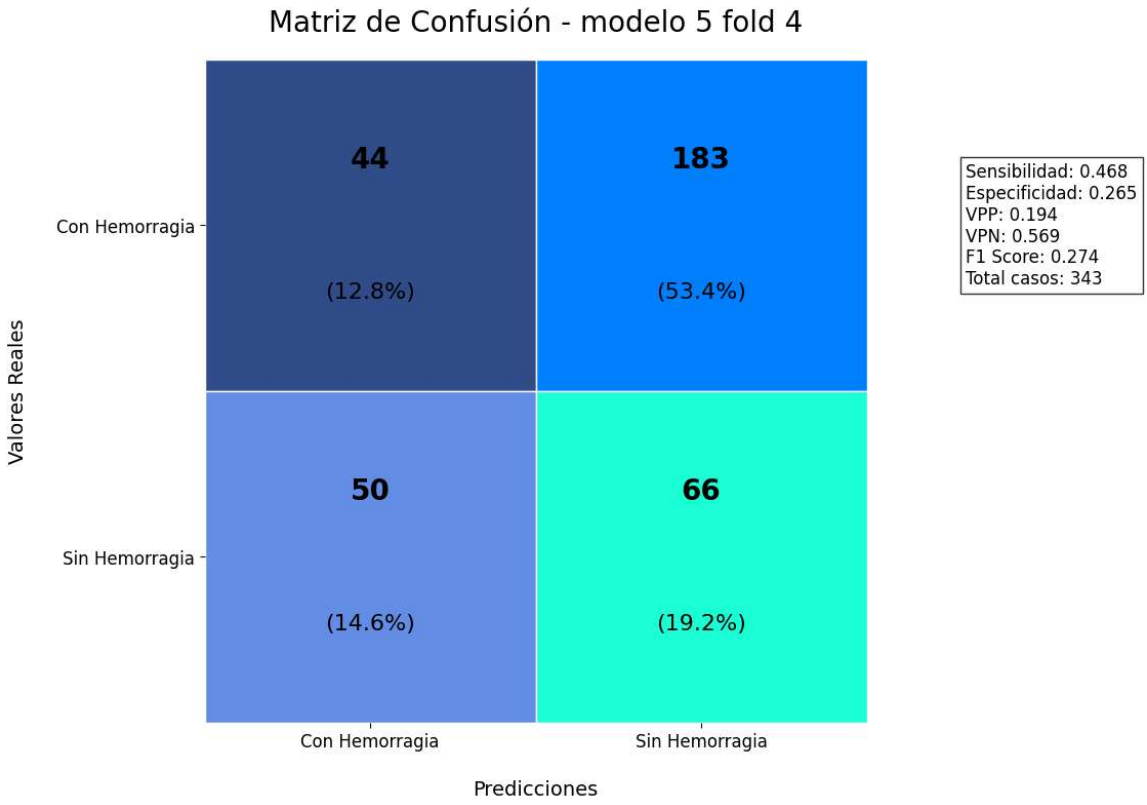


Figura 134. Matriz de confusión de los casos analizados con el modelo 5 (fold 4).

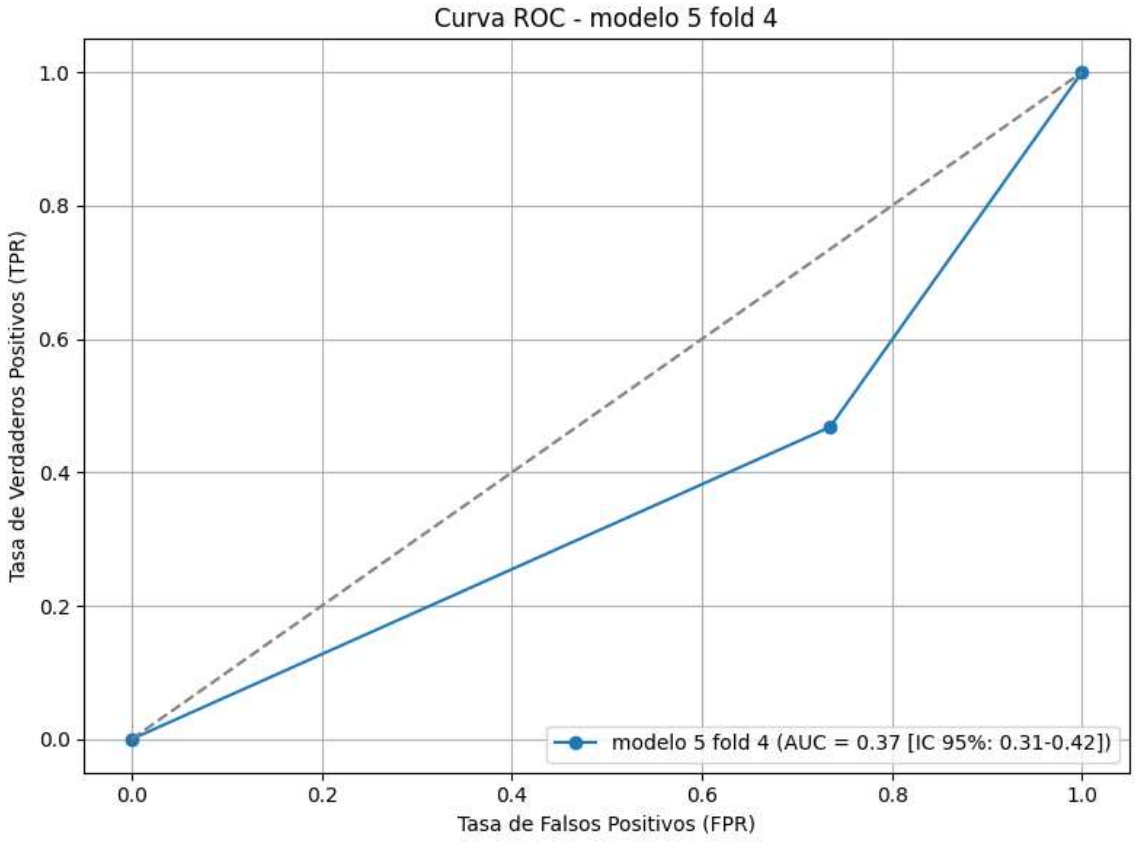


Figura 135. Curva AUC-ROC evaluando el desempeño del modelo 5 de clasificación (fold 4).

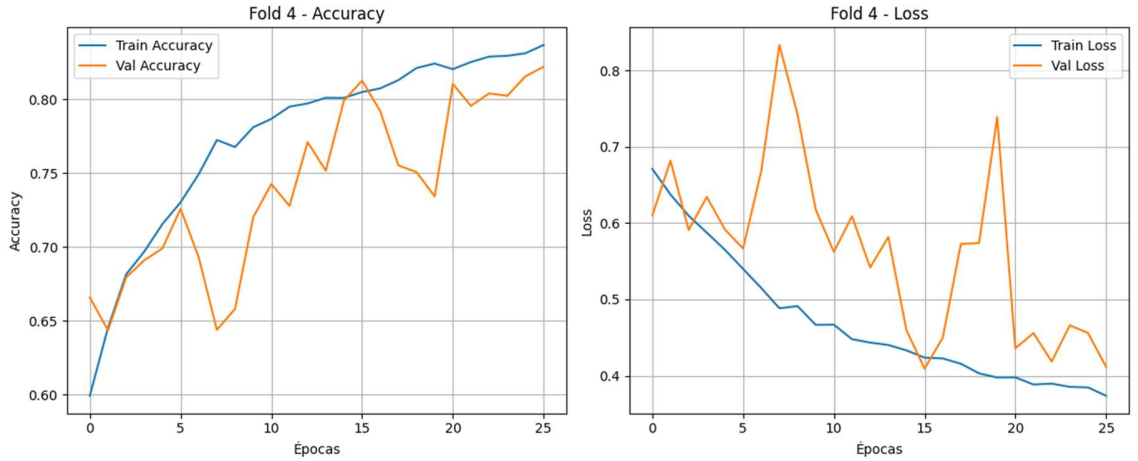


Figura 136. Evolución de la precisión del Modelo 5 durante el Entrenamiento y Validación (fold 4).

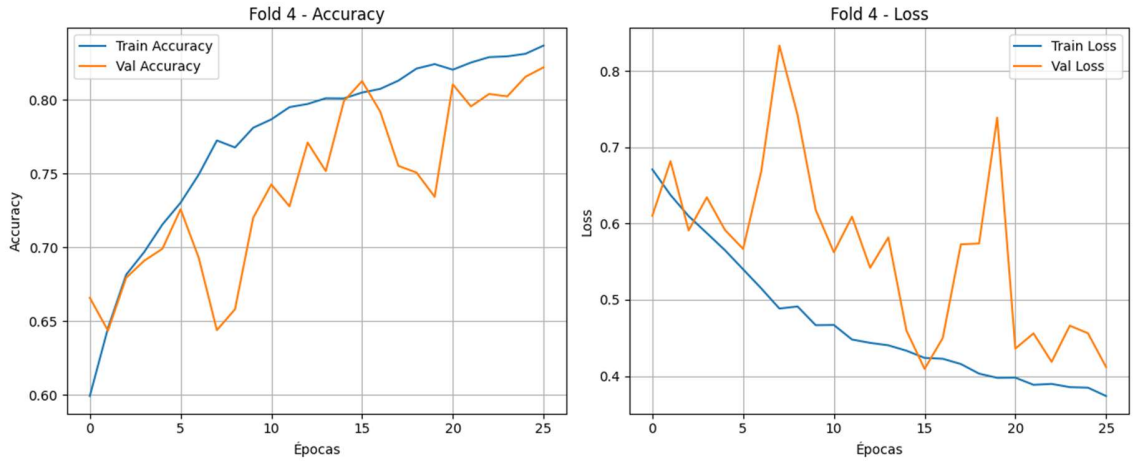


Figura 137. Evolución de la pérdida del Modelo 5 durante el Entrenamiento y Validación (fold 4).

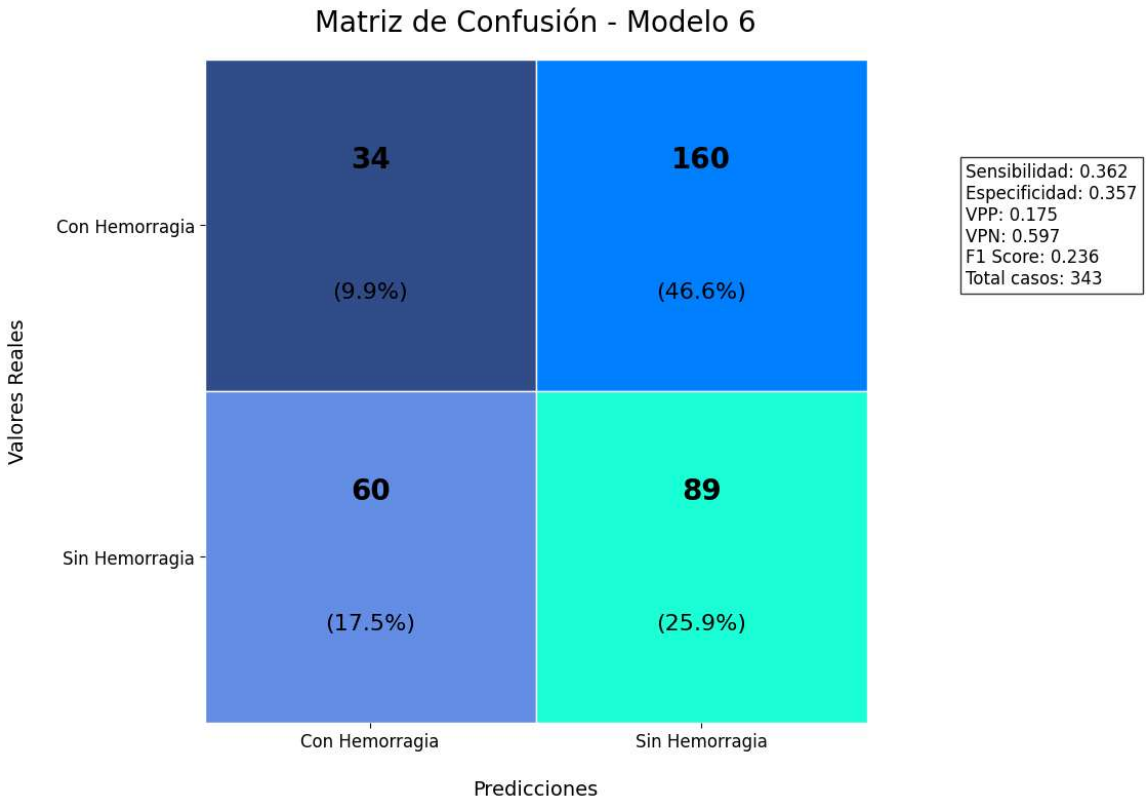


Figura 138. Matriz de confusión de los casos analizados con el modelo 6.

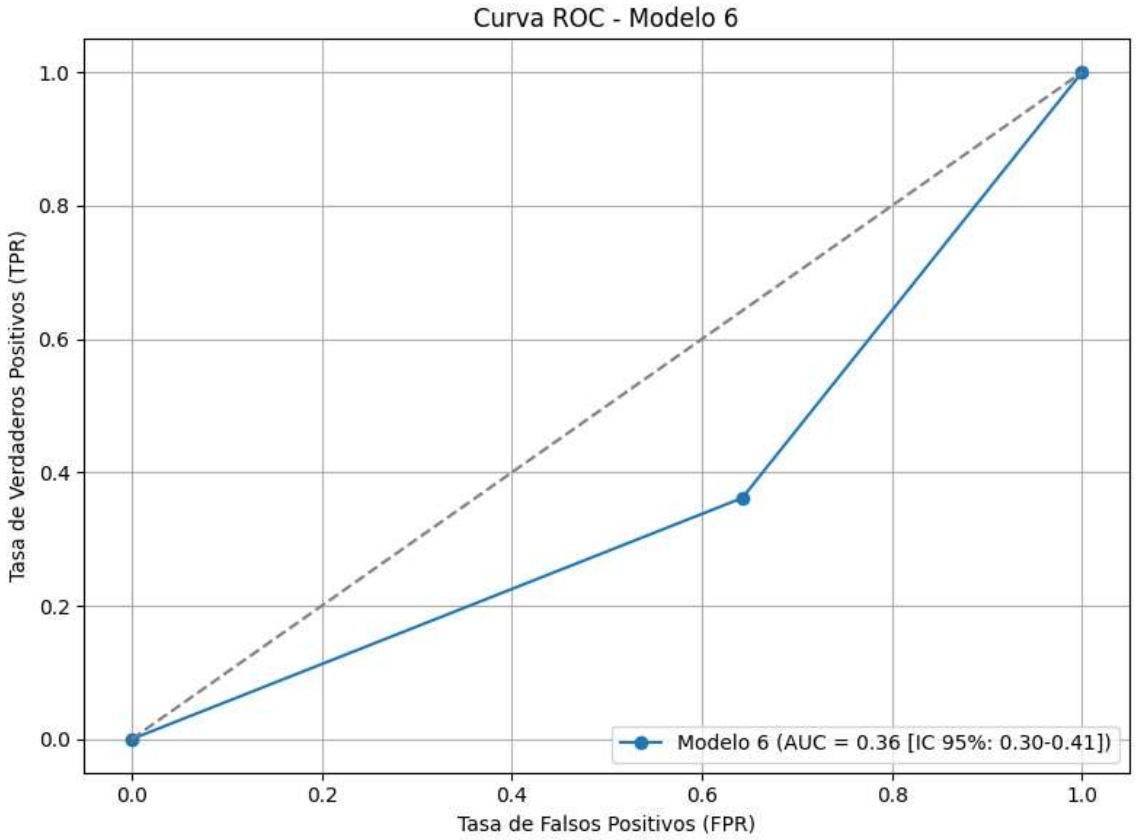


Figura 139. Curva AUC-ROC evaluando el desempeño del modelo 6 de clasificación.

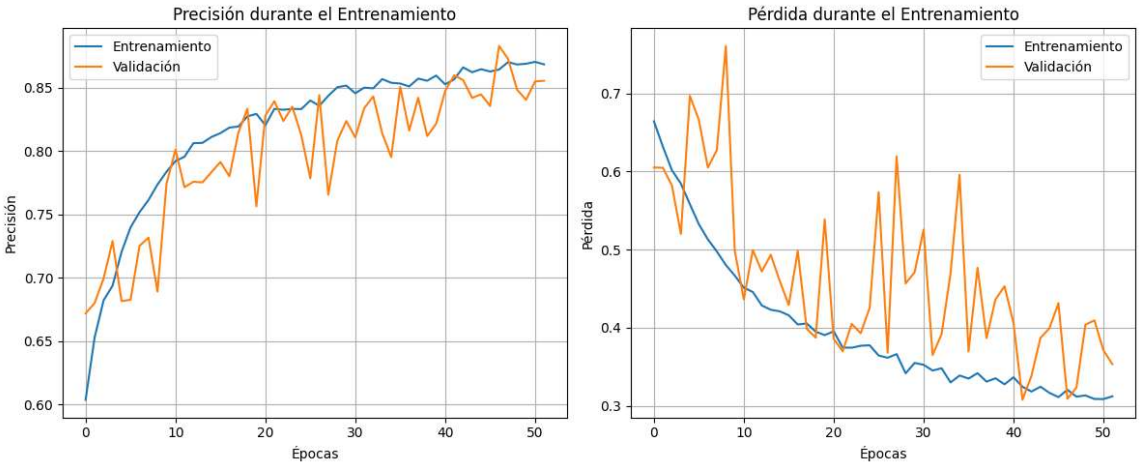


Figura 140. Evolución de la precisión del Modelo 6 durante el Entrenamiento y Validación.

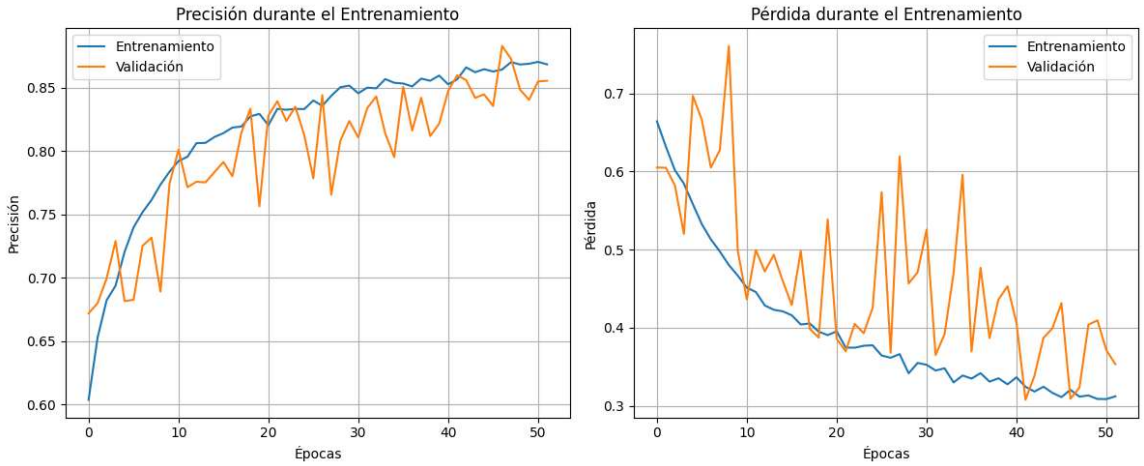


Figura 141. Evolución de la pérdida del Modelo 6 durante el Entrenamiento y Validación.

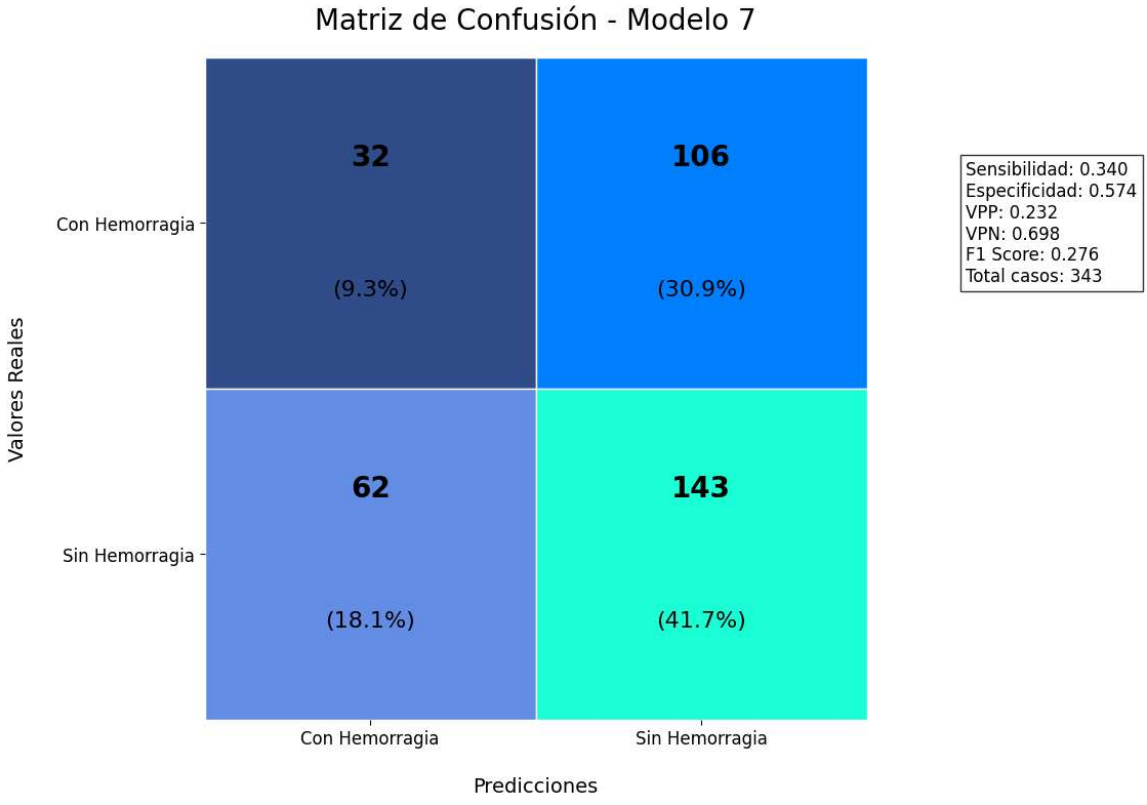


Figura 142. Matriz de confusión de los casos analizados con el modelo 7.

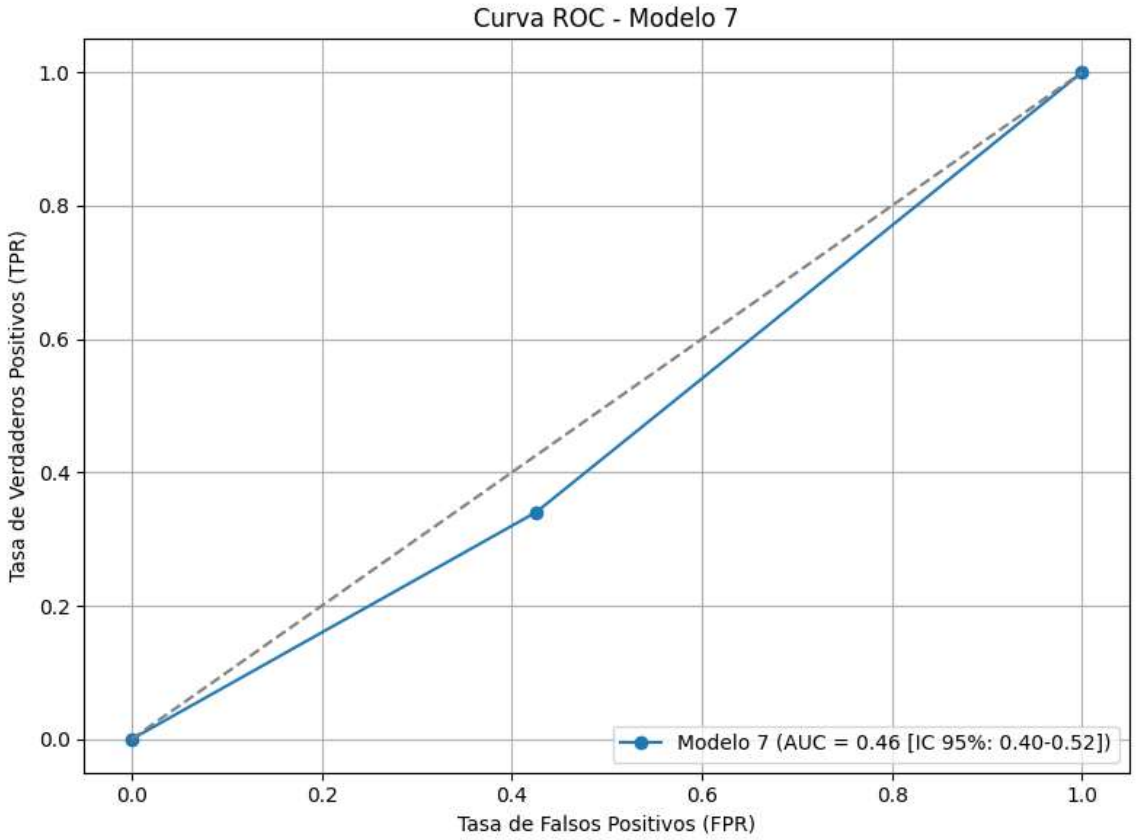


Figura 143. Curva AUC-ROC evaluando el desempeño del modelo 7 de clasificación.

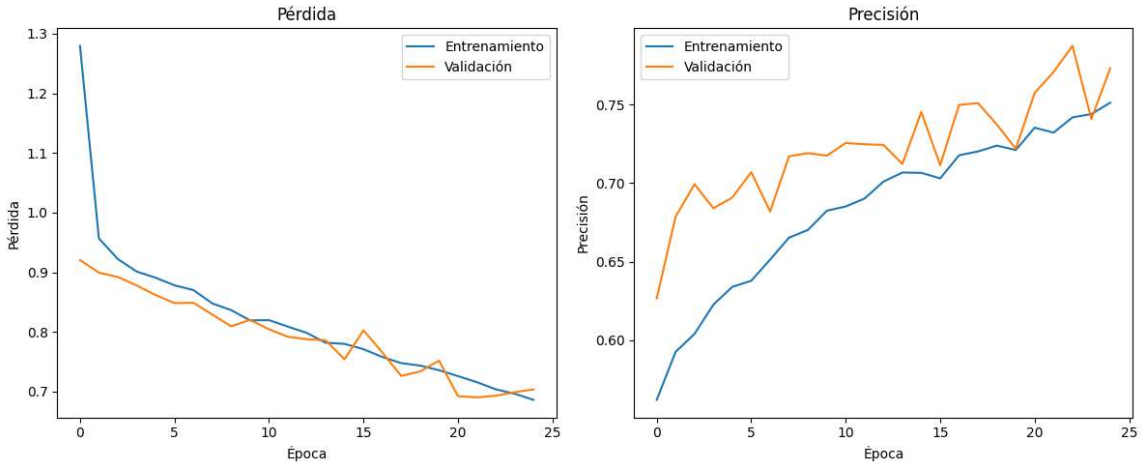


Figura 144. Evolución de la precisión del Modelo 7 durante el Entrenamiento y Validación.

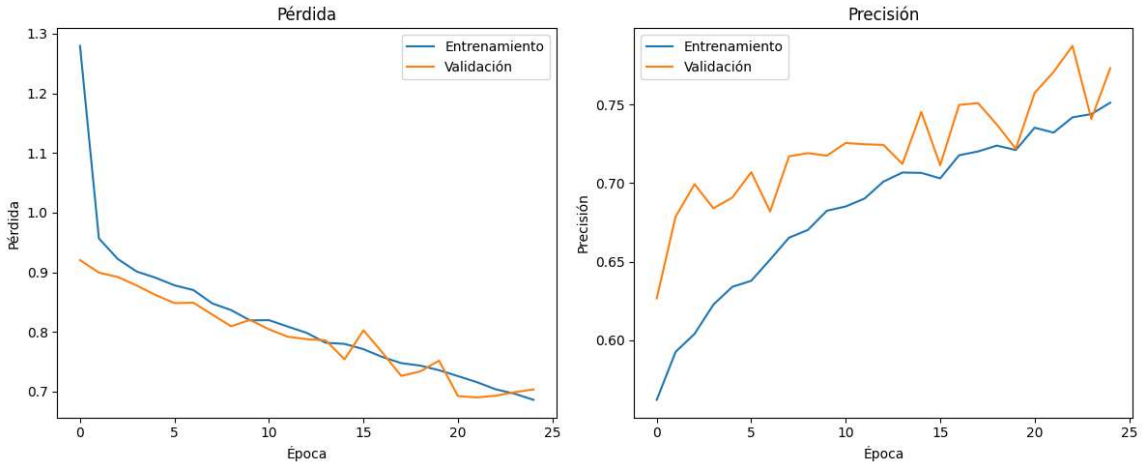


Figura 145. Evolución de la pérdida del Modelo 7 durante el Entrenamiento y Validación.

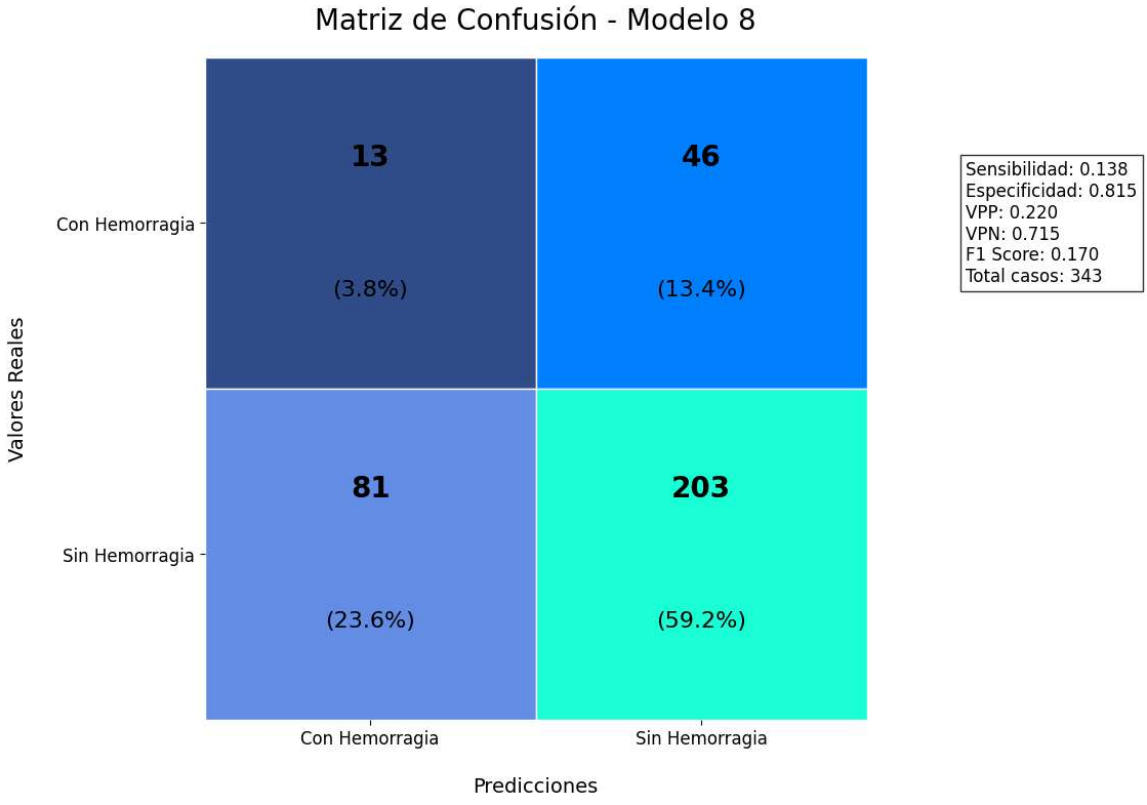


Figura 146. Matriz de confusión de los casos analizados con el modelo 8.

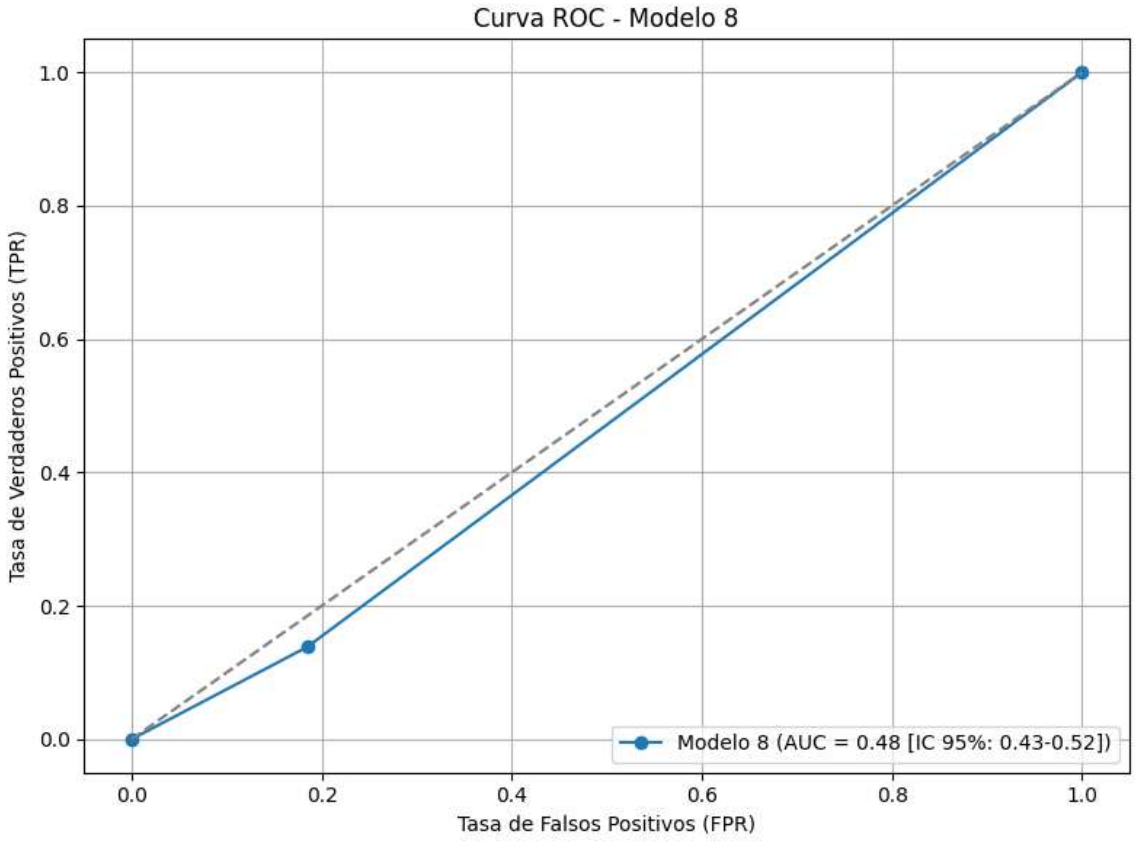


Figura 147. Curva AUC-ROC evaluando el desempeño del modelo 8 de clasificación.

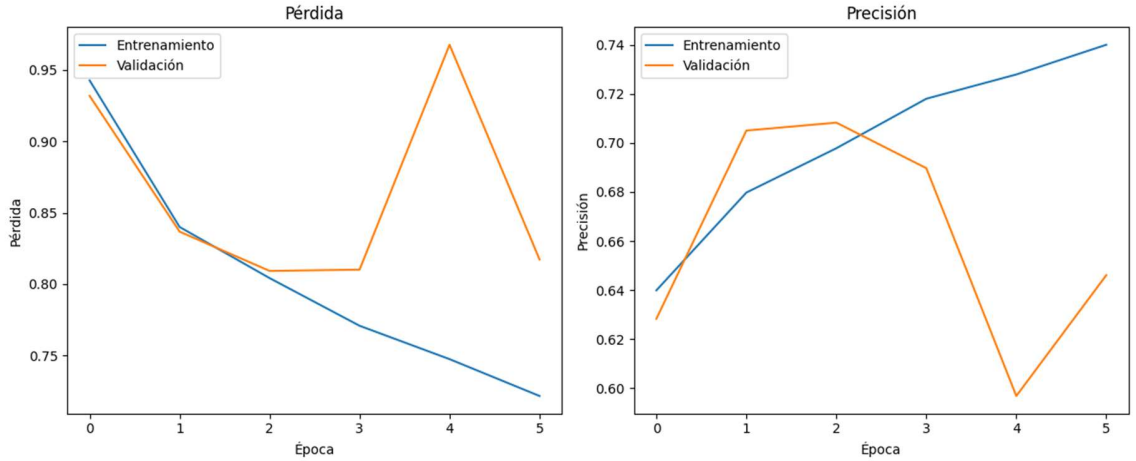


Figura 148. Evolución de la precisión del Modelo 8 durante el Entrenamiento y Validación.

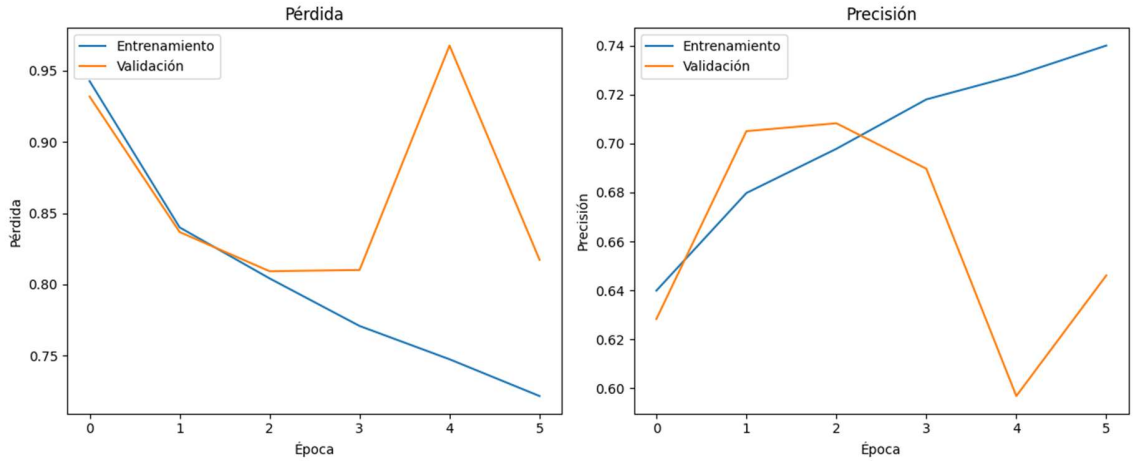


Figura 149. Evolución de la pérdida del Modelo 8 durante el Entrenamiento y Validación.

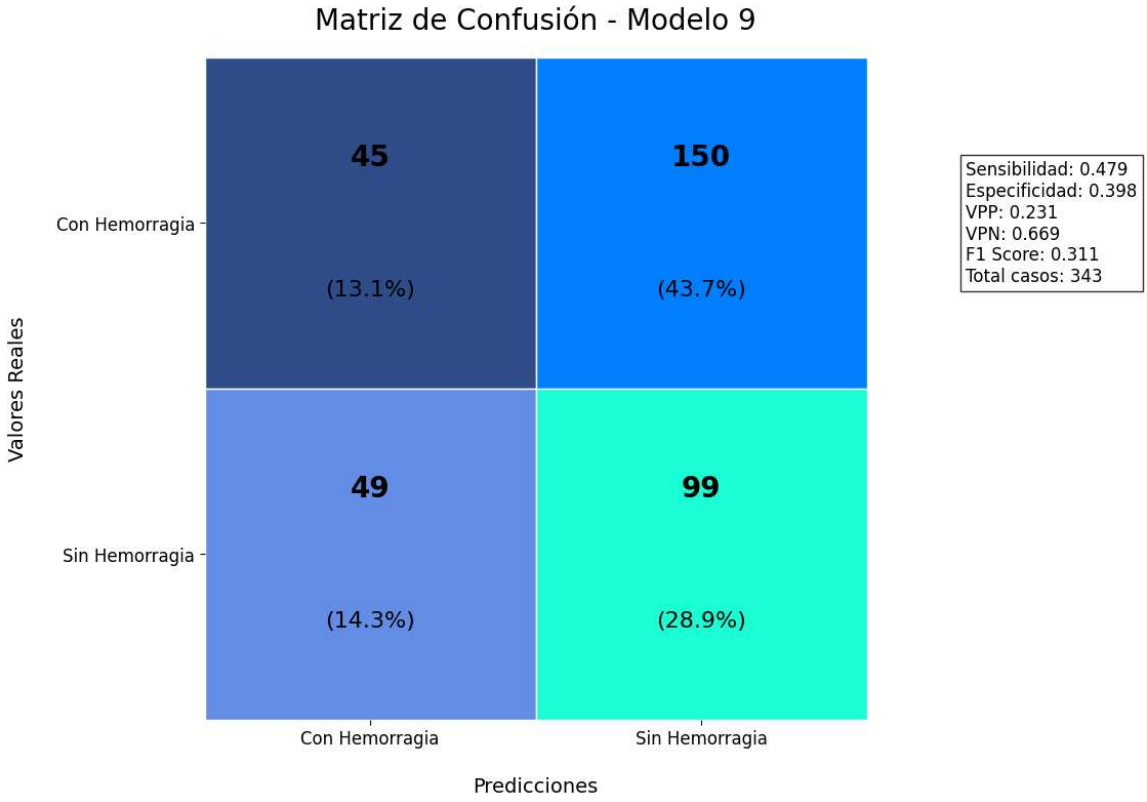


Figura 150. Matriz de confusión de los casos analizados con el modelo 9.

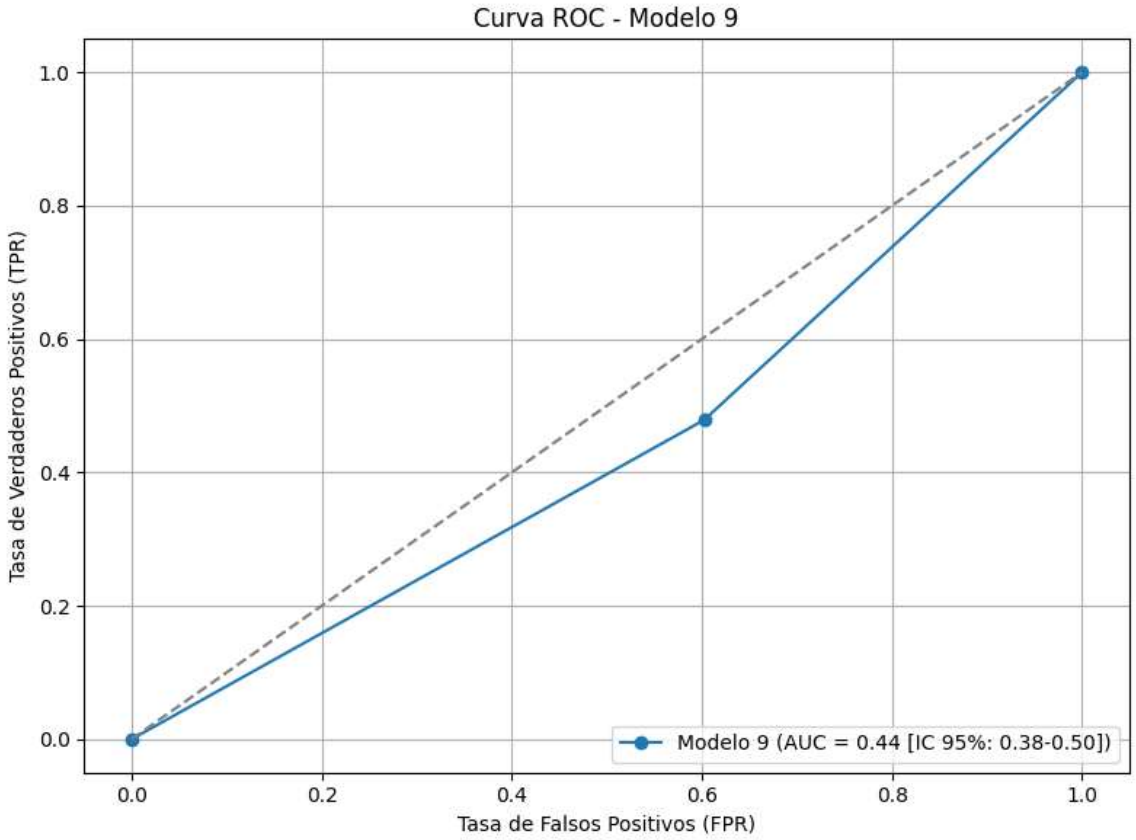


Figura 151. Curva AUC-ROC evaluando el desempeño del modelo 9 de clasificación.

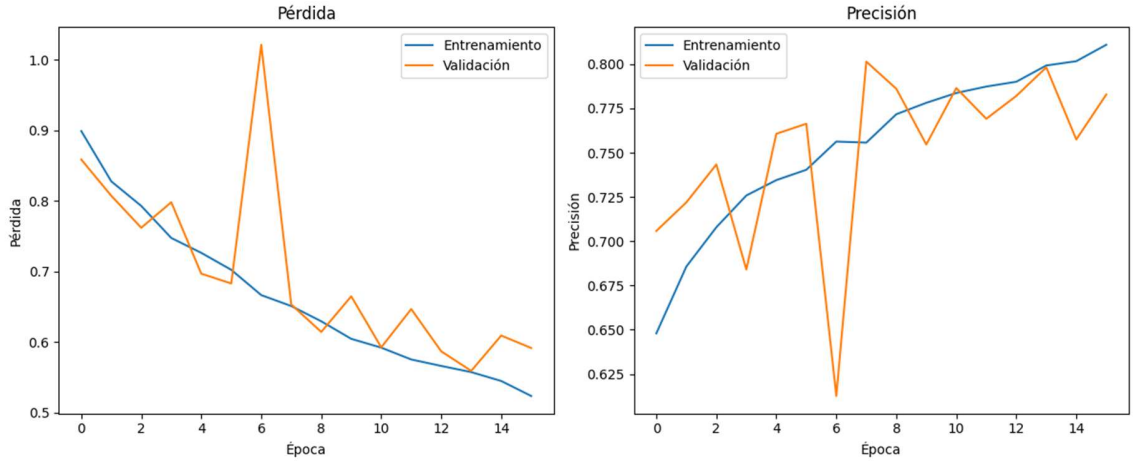


Figura 152. Evolución de la precisión del Modelo 9 durante el Entrenamiento y Validación.

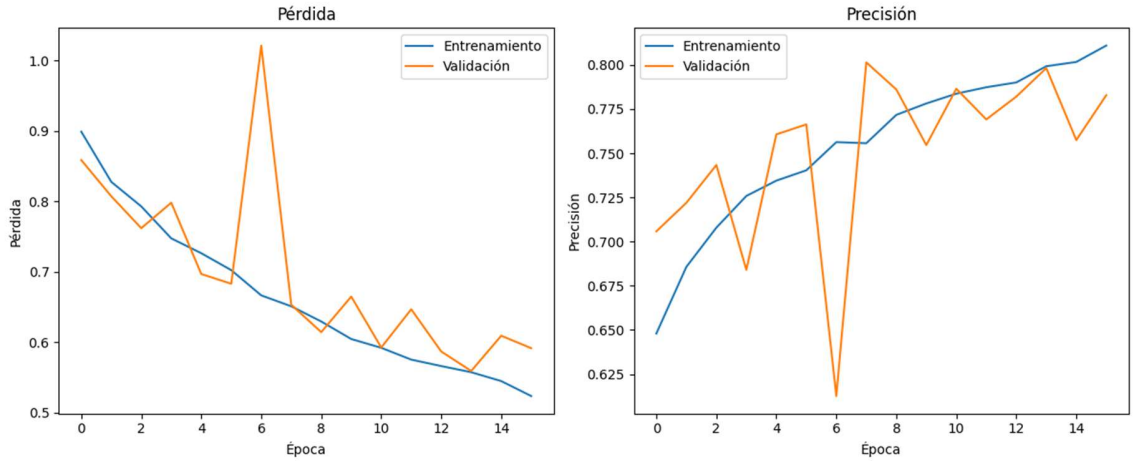


Figura 153. Evolución de la pérdida del Modelo 9 durante el Entrenamiento y Validación.

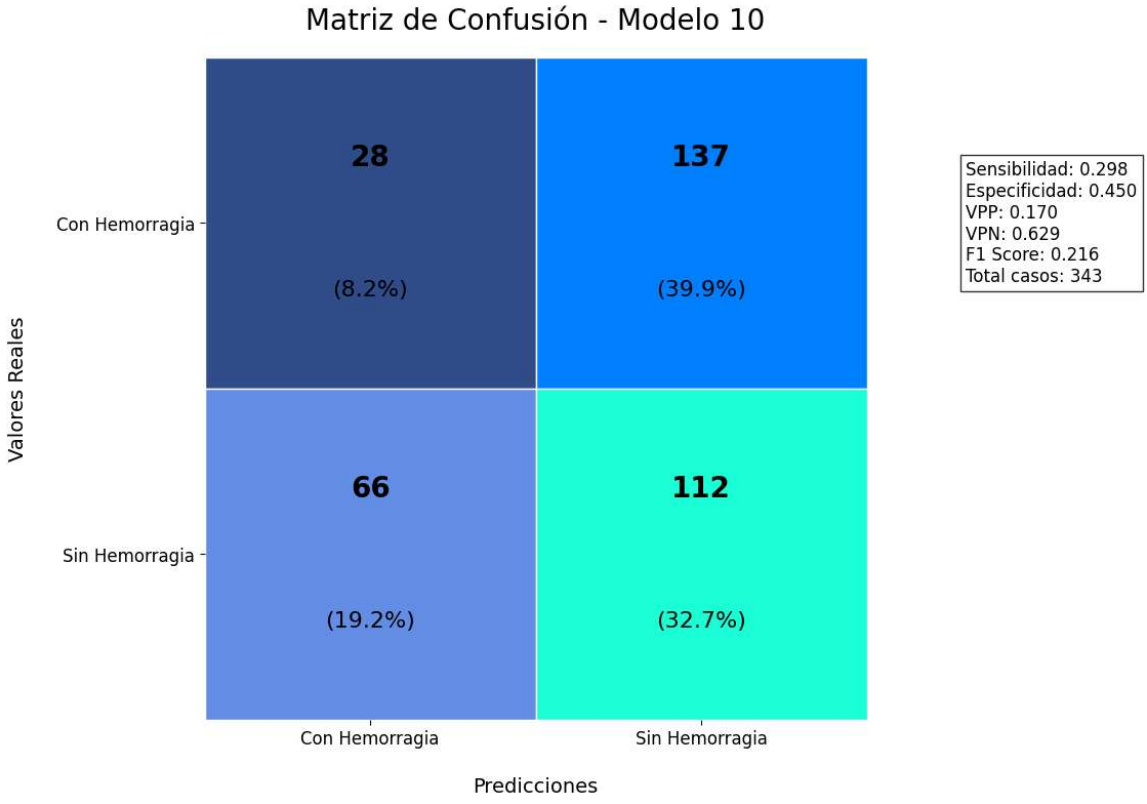


Figura 154. Matriz de confusión de los casos analizados con el modelo 10.

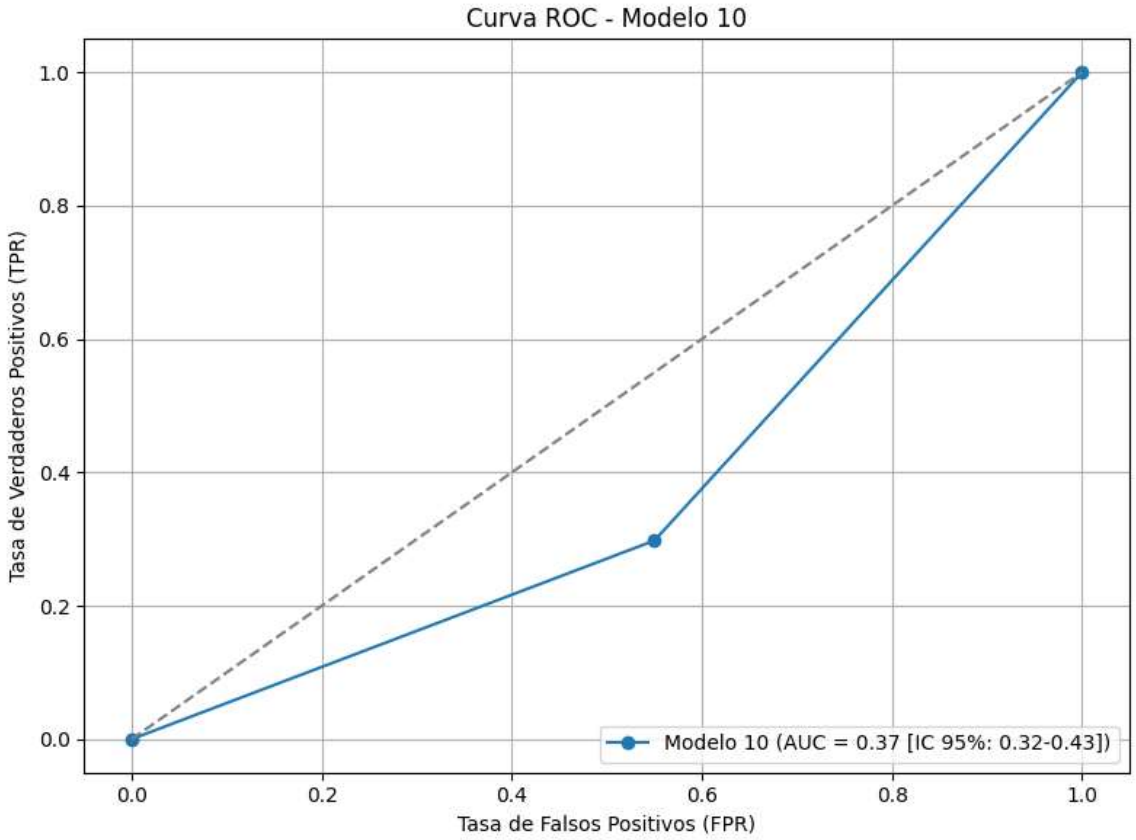


Figura 155. Curva AUC-ROC evaluando el desempeño del modelo 10 de clasificación.

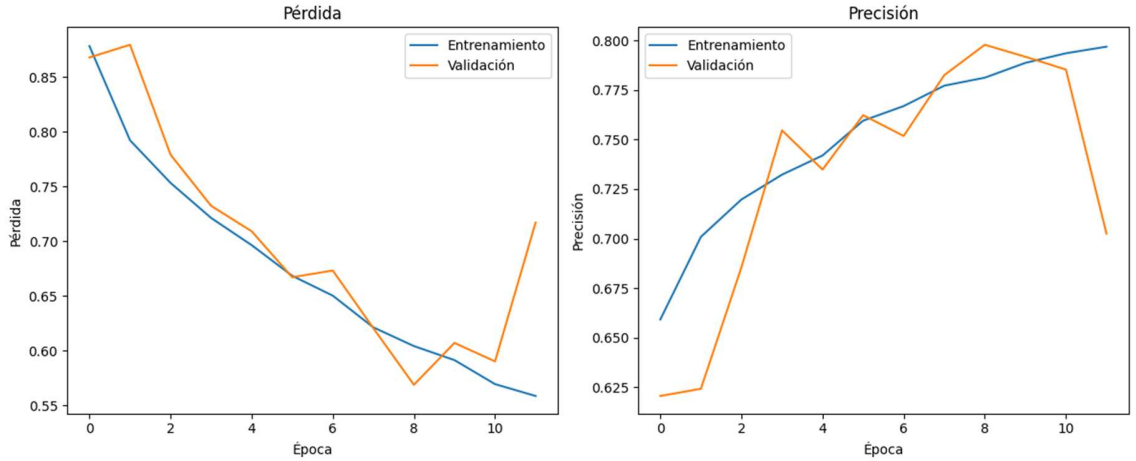


Figura 156. Evolución de la precisión del Modelo 10 durante el Entrenamiento y Validación.

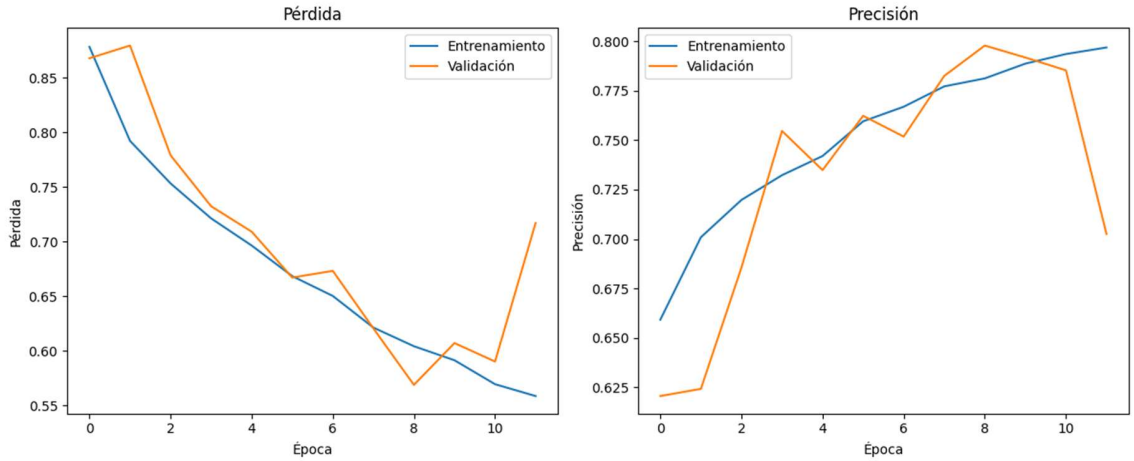


Figura 157. Evolución de la pérdida del Modelo 10 durante el Entrenamiento y Validación.

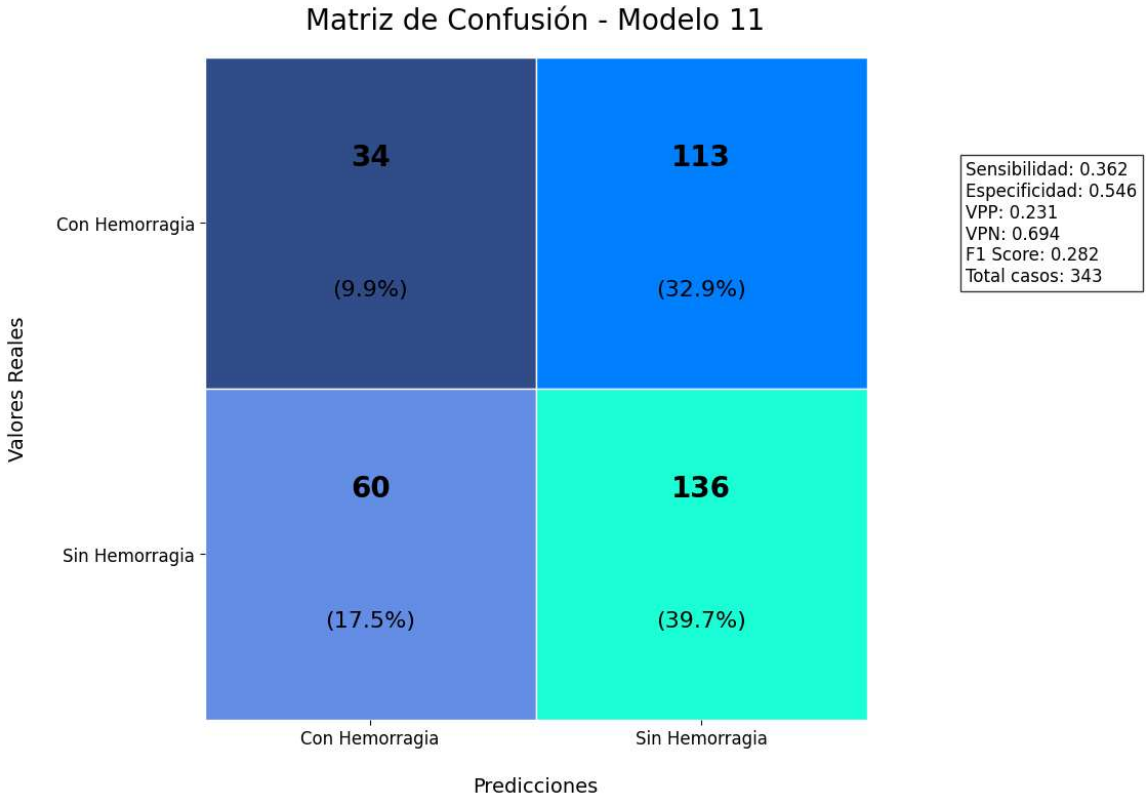


Figura 158. Matriz de confusión de los casos analizados con el modelo 11.

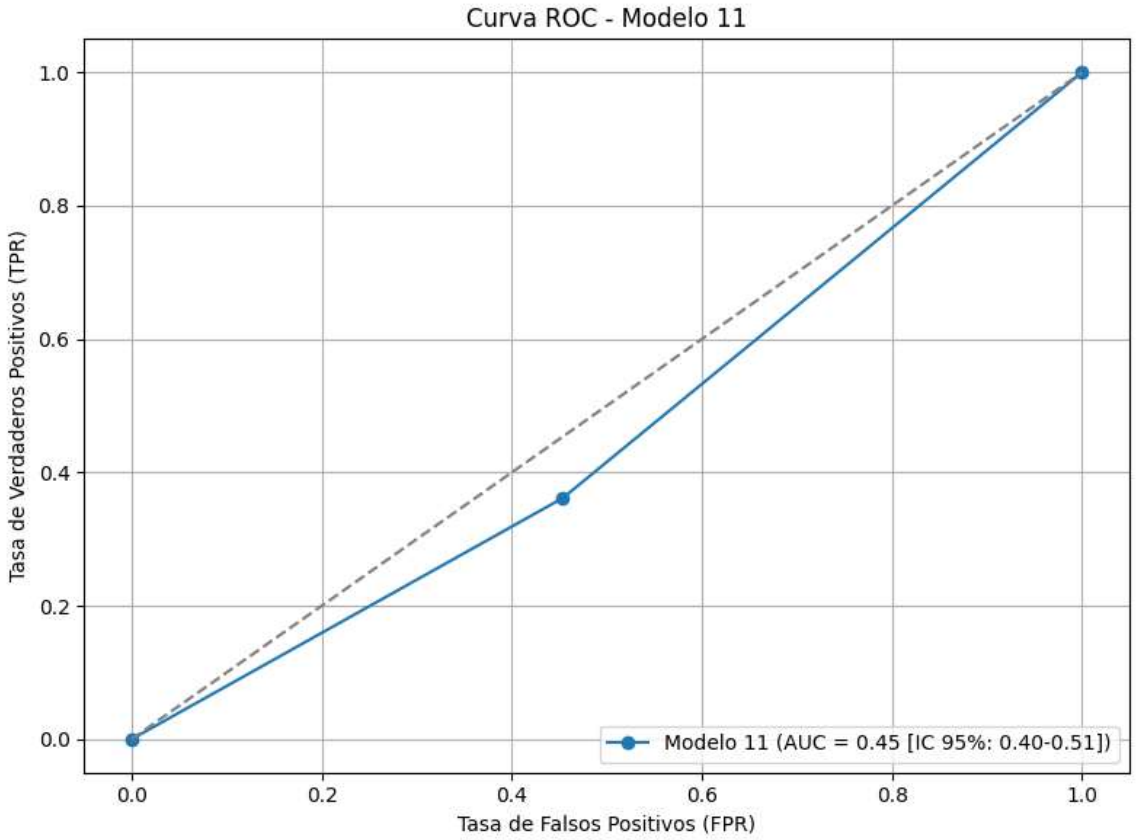


Figura 159. Curva AUC-ROC evaluando el desempeño del modelo 11 de clasificación.

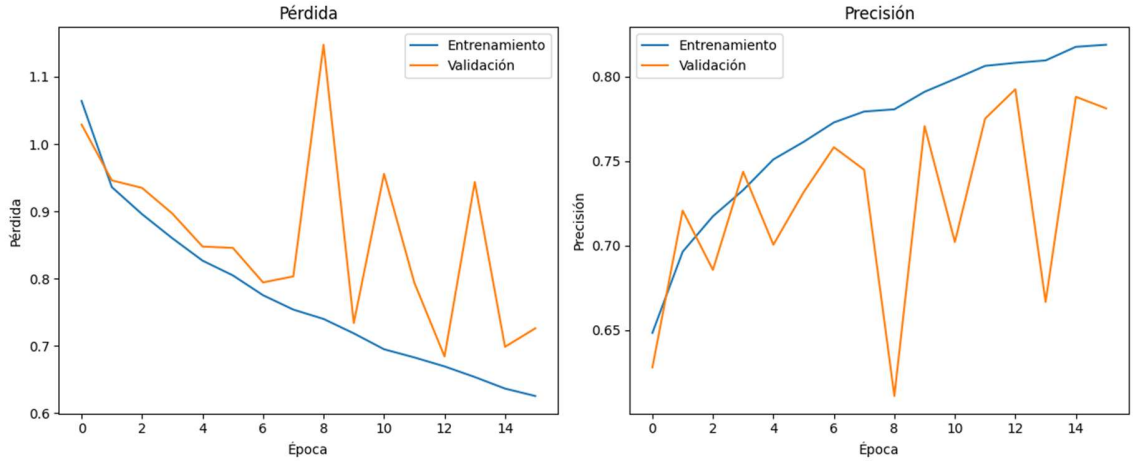


Figura 160. Evolución de la precisión del Modelo 11 durante el Entrenamiento y Validación.

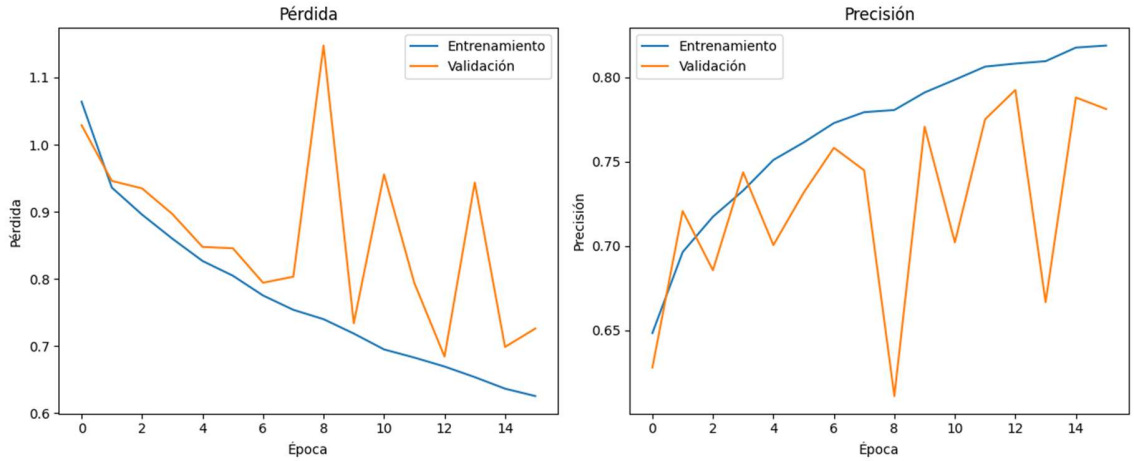


Figura 161. Evolución de la pérdida del Modelo 11 durante el Entrenamiento y Validación.

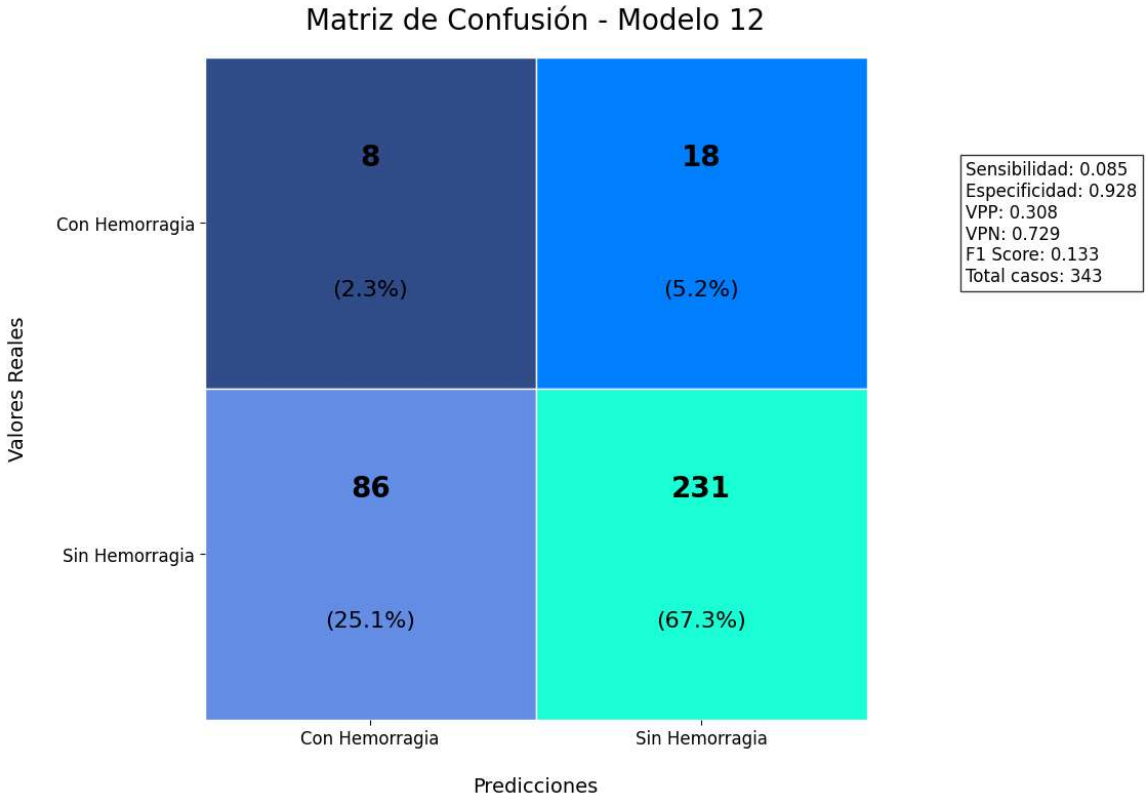


Figura 162. Matriz de confusión de los casos analizados con el modelo 12.

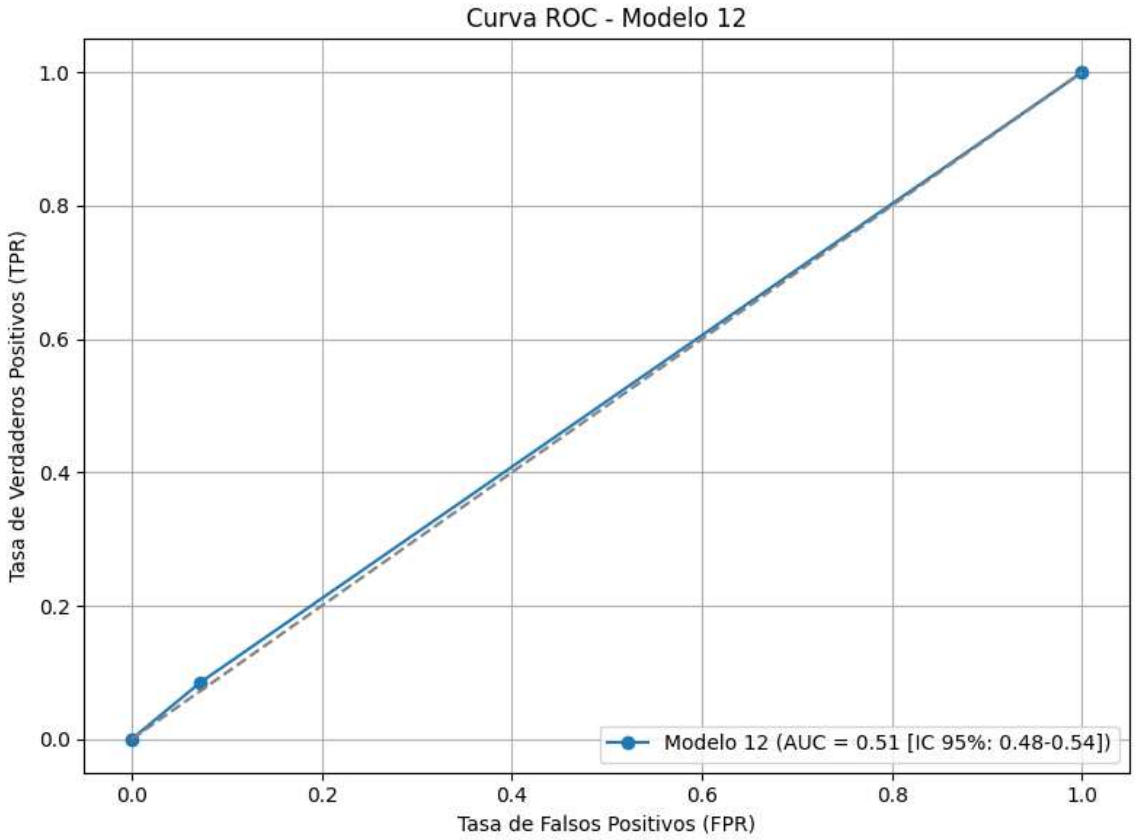


Figura 163. Curva AUC-ROC evaluando el desempeño del modelo 12 de clasificación.

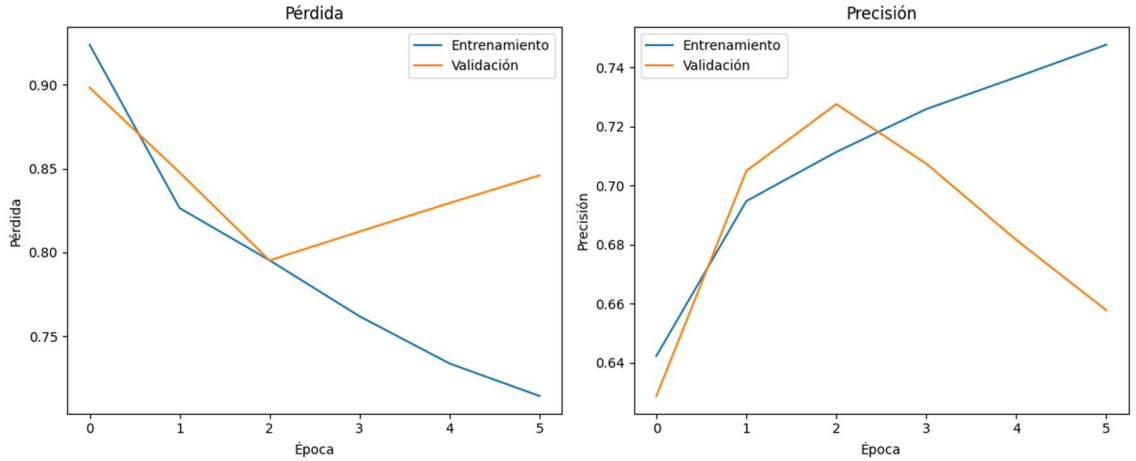


Figura 164. Evolución de la precisión del Modelo 12 durante el Entrenamiento y Validación.

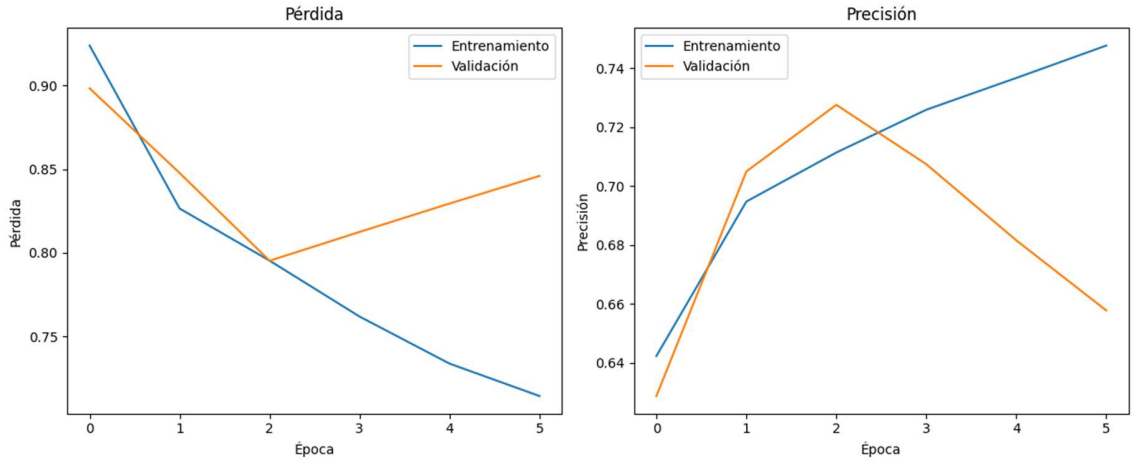


Figura 165. Evolución de la pérdida del Modelo 12 durante el Entrenamiento y Validación.

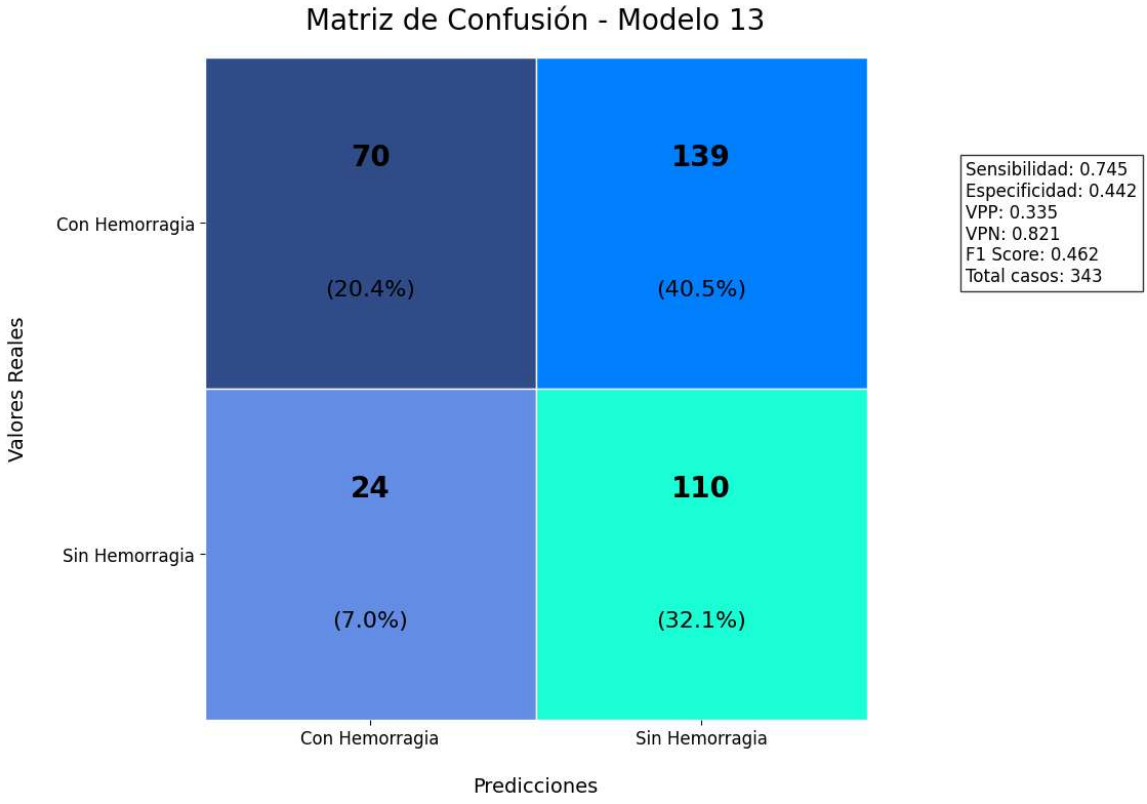


Figura 166. Matriz de confusión de los casos analizados con el modelo 13.

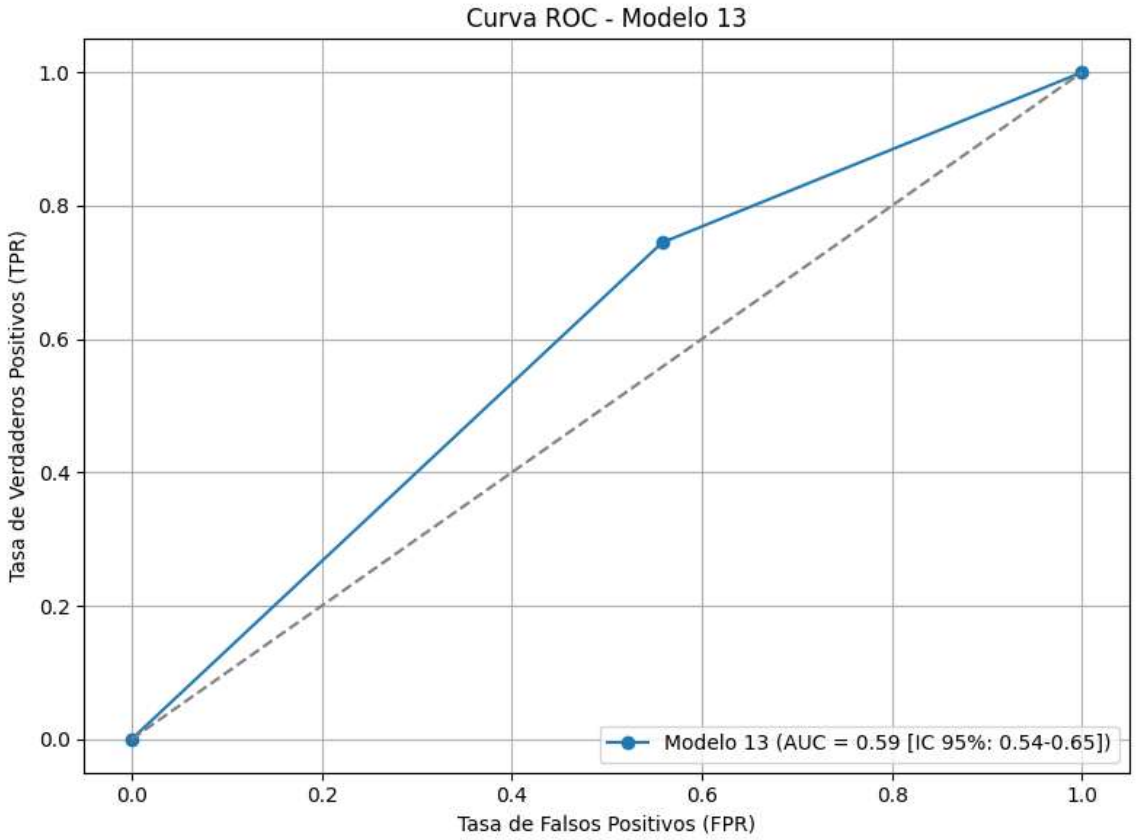


Figura 167. Curva AUC-ROC evaluando el desempeño del modelo 13 de clasificación.

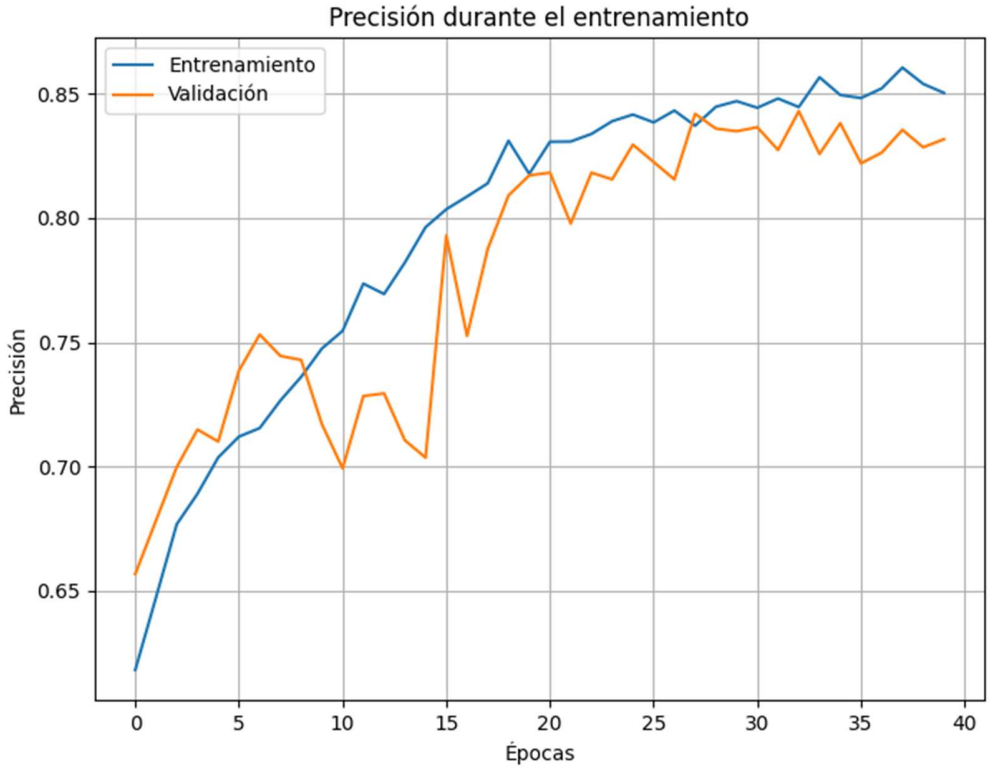


Figura 168. Evolución de la precisión del Modelo 13 durante el Entrenamiento y Validación.

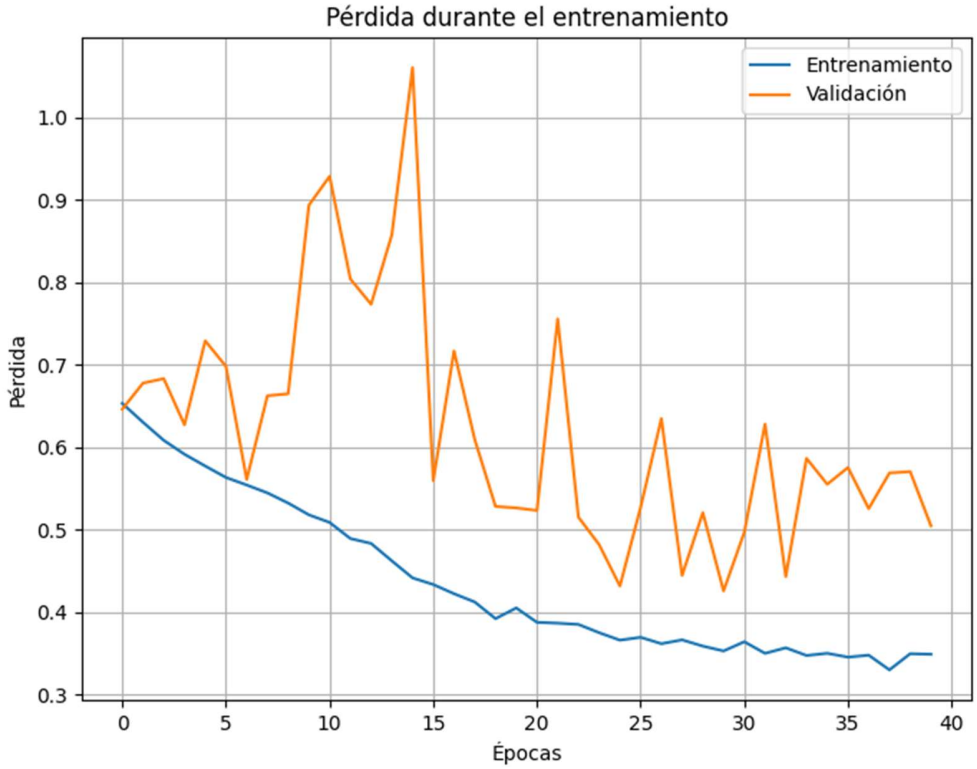


Figura 169. Evolución de la pérdida del Modelo 13 durante el Entrenamiento y Validación.

Anexo I. Características de la arquitectura de cada modelo desarrollado.

Modelo CNN-1.

In [ ]:
```
data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS2"
```

In [ ]:
```
import numpy as np
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image

data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS2"

def load_images_from_directory(directory):
    images = []
    labels = []

    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)


        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)


            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)

    return np.array(images), np.array(labels)
X, y = load_images_from_directory(data_dir)
X = X / 255.0
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15, random_stat
print(f"Tamaño de X_train: {X_train.shape}")
print(f"Tamaño de X_val: {X_val.shape}")
```

```
Tamaño de X_train: (6764, 100, 100, 3)
Tamaño de X_val: (1194, 100, 100, 3)
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        896

 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 47, 47, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 23, 23, 64)        0
 g2D)

 flatten (Flatten)           (None, 33856)             0

 dense (Dense)               (None, 64)                2166848

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2186305 (8.34 MB)
Trainable params: 2186305 (8.34 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)
```

In [ ]:
```python
early_stopping = EarlyStopping(monitor='val_loss', patience=30, restore_best_weight
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping]
)
```

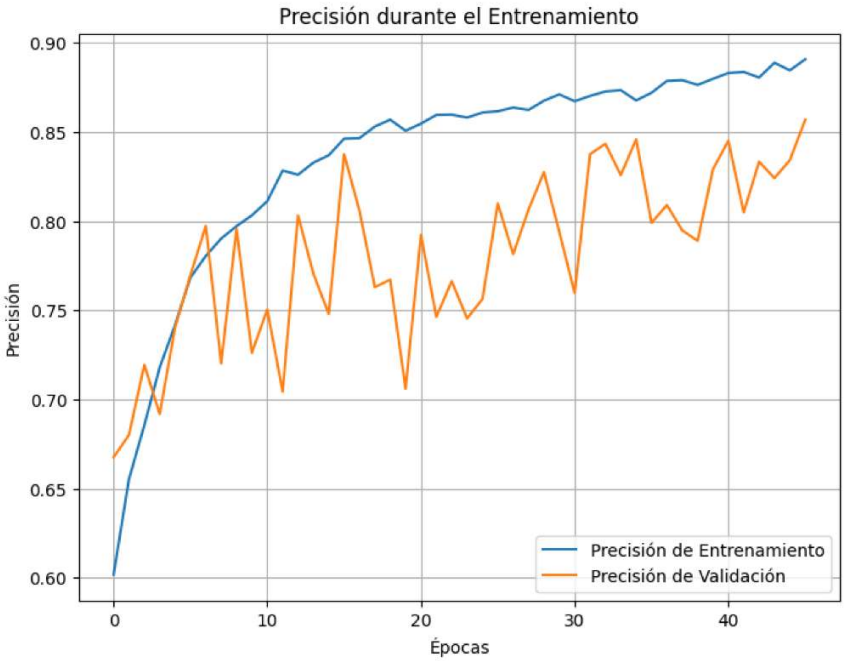
```
Epoch 1/100
212/212 [==============================] - 68s 308ms/step - loss: 0.6675 - accuracy:
0.6017 - val_loss: 0.6180 - val_accuracy: 0.6675
Epoch 2/100
212/212 [==============================] - 70s 329ms/step - loss: 0.6265 - accuracy:
0.6552 - val_loss: 0.6091 - val_accuracy: 0.6801
Epoch 3/100
212/212 [==============================] - 68s 320ms/step - loss: 0.5902 - accuracy:
0.6855 - val_loss: 0.5721 - val_accuracy: 0.7194
Epoch 4/100
212/212 [==============================] - 68s 322ms/step - loss: 0.5556 - accuracy:
0.7179 - val_loss: 0.8349 - val_accuracy: 0.6918
Epoch 5/100
212/212 [==============================] - 71s 334ms/step - loss: 0.5213 - accuracy:
0.7417 - val_loss: 0.6327 - val_accuracy: 0.7404
Epoch 6/100
212/212 [==============================] - 72s 341ms/step - loss: 0.4882 - accuracy:
0.7683 - val_loss: 0.5978 - val_accuracy: 0.7697
Epoch 7/100
212/212 [==============================] - 70s 329ms/step - loss: 0.4687 - accuracy:
0.7805 - val_loss: 0.4598 - val_accuracy: 0.7973
Epoch 8/100
212/212 [==============================] - 69s 326ms/step - loss: 0.4551 - accuracy:
0.7902 - val_loss: 0.6661 - val_accuracy: 0.7203
Epoch 9/100
212/212 [==============================] - 69s 325ms/step - loss: 0.4441 - accuracy:
0.7972 - val_loss: 0.4688 - val_accuracy: 0.7956
Epoch 10/100
212/212 [==============================] - 69s 325ms/step - loss: 0.4267 - accuracy:
0.8032 - val_loss: 0.7778 - val_accuracy: 0.7261
Epoch 11/100
212/212 [==============================] - 69s 326ms/step - loss: 0.4196 - accuracy:
0.8112 - val_loss: 0.6496 - val_accuracy: 0.7504
Epoch 12/100
212/212 [==============================] - 70s 328ms/step - loss: 0.3963 - accuracy:
0.8284 - val_loss: 0.9335 - val_accuracy: 0.7044
Epoch 13/100
212/212 [==============================] - 70s 328ms/step - loss: 0.3917 - accuracy:
0.8260 - val_loss: 0.4915 - val_accuracy: 0.8032
Epoch 14/100
212/212 [==============================] - 68s 320ms/step - loss: 0.3838 - accuracy:
0.8328 - val_loss: 0.5543 - val_accuracy: 0.7705
Epoch 15/100
212/212 [==============================] - 70s 331ms/step - loss: 0.3790 - accuracy:
0.8369 - val_loss: 0.6280 - val_accuracy: 0.7479
Epoch 16/100
212/212 [==============================] - 69s 325ms/step - loss: 0.3575 - accuracy:
0.8462 - val_loss: 0.3964 - val_accuracy: 0.8375
Epoch 17/100
212/212 [==============================] - 69s 327ms/step - loss: 0.3572 - accuracy:
0.8465 - val_loss: 0.5100 - val_accuracy: 0.8057
Epoch 18/100
212/212 [==============================] - 67s 318ms/step - loss: 0.3482 - accuracy:
0.8530 - val_loss: 0.6671 - val_accuracy: 0.7630
Epoch 19/100
212/212 [==============================] - 70s 329ms/step - loss: 0.3337 - accuracy:
```

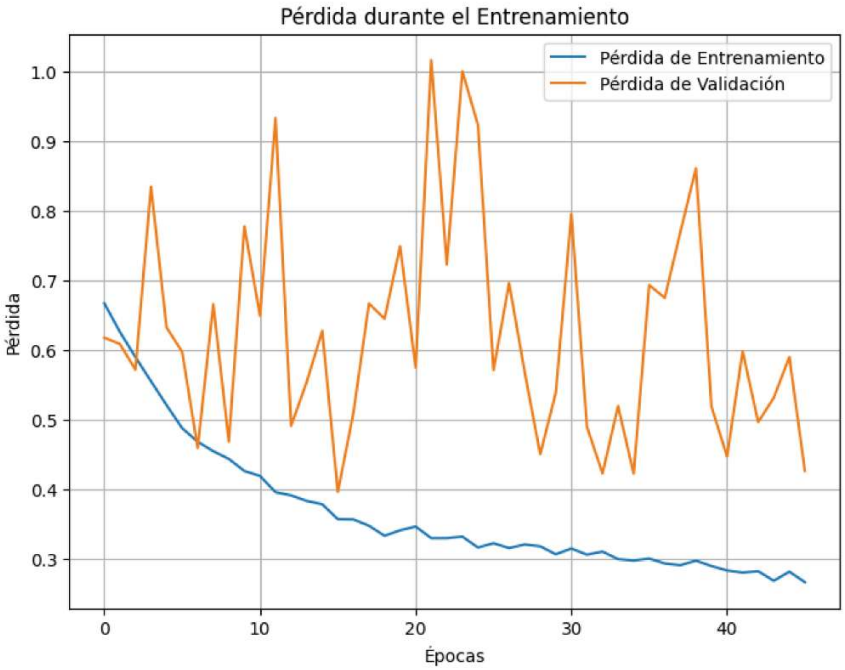
```
0.8569 - val_loss: 0.6453 - val_accuracy: 0.7672
Epoch 20/100
212/212 [==============================] - 72s 337ms/step - loss: 0.3416 - accuracy:
0.8507 - val_loss: 0.7494 - val_accuracy: 0.7060
Epoch 21/100
212/212 [==============================] - 71s 336ms/step - loss: 0.3470 - accuracy:
0.8547 - val_loss: 0.5753 - val_accuracy: 0.7923
Epoch 22/100
212/212 [==============================] - 68s 322ms/step - loss: 0.3303 - accuracy:
0.8596 - val_loss: 1.0166 - val_accuracy: 0.7462
Epoch 23/100
212/212 [==============================] - 68s 322ms/step - loss: 0.3304 - accuracy:
0.8597 - val_loss: 0.7233 - val_accuracy: 0.7663
Epoch 24/100
212/212 [==============================] - 69s 325ms/step - loss: 0.3325 - accuracy:
0.8581 - val_loss: 1.0006 - val_accuracy: 0.7454
Epoch 25/100
212/212 [==============================] - 69s 325ms/step - loss: 0.3169 - accuracy:
0.8609 - val_loss: 0.9233 - val_accuracy: 0.7563
Epoch 26/100
212/212 [==============================] - 68s 321ms/step - loss: 0.3230 - accuracy:
0.8616 - val_loss: 0.5715 - val_accuracy: 0.8099
Epoch 27/100
212/212 [==============================] - 68s 318ms/step - loss: 0.3161 - accuracy:
0.8637 - val_loss: 0.6964 - val_accuracy: 0.7814
Epoch 28/100
212/212 [==============================] - 69s 326ms/step - loss: 0.3212 - accuracy:
0.8624 - val_loss: 0.5691 - val_accuracy: 0.8065
Epoch 29/100
212/212 [==============================] - 69s 324ms/step - loss: 0.3186 - accuracy:
0.8675 - val_loss: 0.4507 - val_accuracy: 0.8275
Epoch 30/100
212/212 [==============================] - 69s 324ms/step - loss: 0.3071 - accuracy:
0.8711 - val_loss: 0.5391 - val_accuracy: 0.7940
Epoch 31/100
212/212 [==============================] - 68s 322ms/step - loss: 0.3155 - accuracy:
0.8672 - val_loss: 0.7958 - val_accuracy: 0.7596
Epoch 32/100
212/212 [==============================] - 69s 327ms/step - loss: 0.3066 - accuracy:
0.8702 - val_loss: 0.4902 - val_accuracy: 0.8375
Epoch 33/100
212/212 [==============================] - 70s 329ms/step - loss: 0.3110 - accuracy:
0.8727 - val_loss: 0.4231 - val_accuracy: 0.8434
Epoch 34/100
212/212 [==============================] - 69s 324ms/step - loss: 0.3002 - accuracy:
0.8734 - val_loss: 0.5198 - val_accuracy: 0.8258
Epoch 35/100
212/212 [==============================] - 68s 319ms/step - loss: 0.2982 - accuracy:
0.8677 - val_loss: 0.4229 - val_accuracy: 0.8459
Epoch 36/100
212/212 [==============================] - 68s 321ms/step - loss: 0.3010 - accuracy:
0.8720 - val_loss: 0.6935 - val_accuracy: 0.7990
Epoch 37/100
212/212 [==============================] - 69s 324ms/step - loss: 0.2940 - accuracy:
0.8786 - val_loss: 0.6753 - val_accuracy: 0.8090
Epoch 38/100
```

```
212/212 [==============================] - 70s 328ms/step - loss: 0.2915 - accuracy:
0.8791 - val_loss: 0.7696 - val_accuracy: 0.7948
Epoch 39/100
212/212 [==============================] - 68s 321ms/step - loss: 0.2980 - accuracy:
0.8764 - val_loss: 0.8611 - val_accuracy: 0.7889
Epoch 40/100
212/212 [==============================] - 68s 321ms/step - loss: 0.2903 - accuracy:
0.8798 - val_loss: 0.5190 - val_accuracy: 0.8291
Epoch 41/100
212/212 [==============================] - 67s 314ms/step - loss: 0.2839 - accuracy:
0.8831 - val_loss: 0.4475 - val_accuracy: 0.8451
Epoch 42/100
212/212 [==============================] - 67s 317ms/step - loss: 0.2809 - accuracy:
0.8836 - val_loss: 0.5985 - val_accuracy: 0.8049
Epoch 43/100
212/212 [==============================] - 67s 314ms/step - loss: 0.2828 - accuracy:
0.8805 - val_loss: 0.4969 - val_accuracy: 0.8333
Epoch 44/100
212/212 [==============================] - 67s 315ms/step - loss: 0.2691 - accuracy:
0.8888 - val_loss: 0.5318 - val_accuracy: 0.8241
Epoch 45/100
212/212 [==============================] - 67s 313ms/step - loss: 0.2822 - accuracy:
0.8845 - val_loss: 0.5904 - val_accuracy: 0.8342
Epoch 46/100
212/212 [==============================] - 67s 314ms/step - loss: 0.2671 - accuracy:
0.8907 - val_loss: 0.4267 - val_accuracy: 0.8568
```

In [ ]:

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
plt.title('Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 6))

plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```


Precisión durante el Entrenamiento
Precisión
Épocas
Precisión de Entrenamiento
Precisión de Validación
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
10
20
30
40



In [ ]:
```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validación - Pérdida: {val_loss:.4f}, Exactitud: {val_accuracy:.4f}")
```

```
38/38 [==============================] - 3s 69ms/step - loss: 0.3964 - accuracy: 0.8
375
Validación - Pérdida: 0.3964, Exactitud: 0.8375
```

In [18]:
```
import json

# Guardar la arquitectura del modelo en un archivo JSON
model_json = model.to_json()
model_json_path = "D:/PROYECTO 1/modelo_2.json"  # Especifica la ruta y el nombre a

with open(model_json_path, "w") as json_file:
    json.dump(model_json, json_file)

print(f"Modelo guardado como JSON en {model_json_path}")
```

```
Modelo guardado como JSON en D:/PROYECTO 1/modelo_2.json
```

In [ ]:
```
model.save('modelo_completo1.h5')
model.save_weights('pesos_modelo1.h5')
```

Modelo CNN-2.

In [ ]:
```
import os
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import Callback
import os
import numpy as np
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS2"
```

In [ ]:
```
import numpy as np
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image
data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS2"

def load_images_from_directory(directory):
    images = []
    labels = []

    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)

        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)

            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)

    return np.array(images), np.array(labels)
X, y = load_images_from_directory(data_dir)

X = X / 255.0

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15, random_stat

print(f"Tamaño de X_train: {X_train.shape}")
print(f"Tamaño de X_val: {X_val.shape}")
```

```
Tamaño de X_train: (6764, 100, 100, 3)
Tamaño de X_val: (1194, 100, 100, 3)
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()
```

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        896

 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 47, 47, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 23, 23, 64)        0
 g2D)

 flatten (Flatten)           (None, 33856)             0

 dense (Dense)               (None, 64)                2166848

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2186305 (8.34 MB)
Trainable params: 2186305 (8.34 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

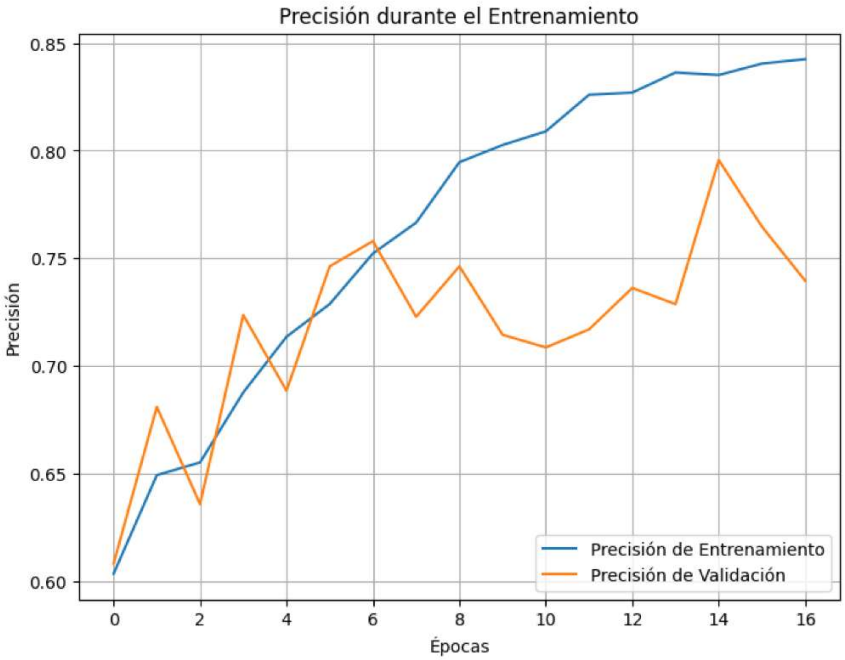
In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)
```

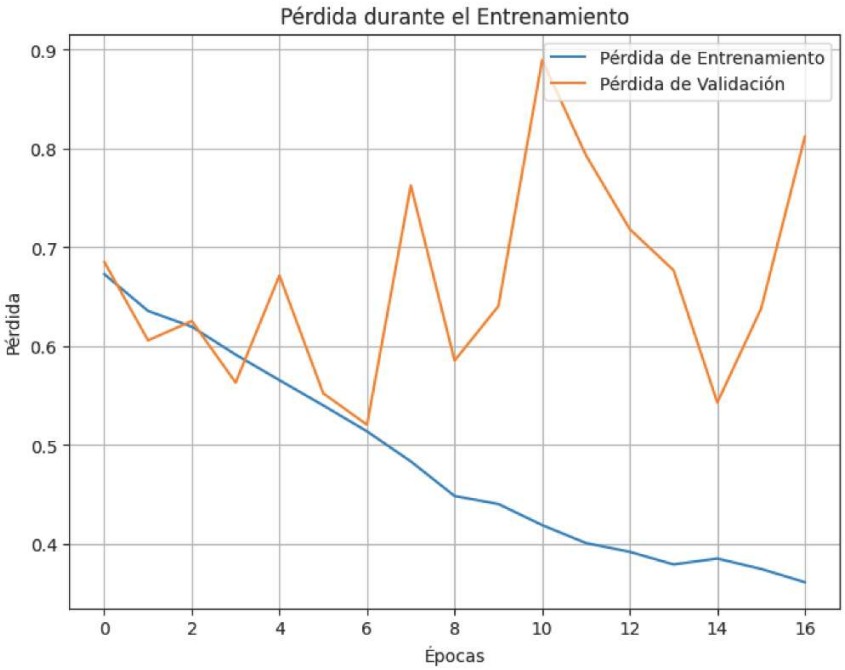
In [ ]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weight

history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping]
)
```

```
Epoch 1/100
212/212 [==============================] - 65s 292ms/step - loss: 0.6730 - accuracy:
0.6033 - val_loss: 0.6853 - val_accuracy: 0.6080
Epoch 2/100
212/212 [==============================] - 62s 293ms/step - loss: 0.6357 - accuracy:
0.6492 - val_loss: 0.6060 - val_accuracy: 0.6809
Epoch 3/100
212/212 [==============================] - 66s 309ms/step - loss: 0.6198 - accuracy:
0.6551 - val_loss: 0.6255 - val_accuracy: 0.6357
Epoch 4/100
212/212 [==============================] - 70s 328ms/step - loss: 0.5916 - accuracy:
0.6876 - val_loss: 0.5633 - val_accuracy: 0.7236
Epoch 5/100
212/212 [==============================] - 70s 330ms/step - loss: 0.5657 - accuracy:
0.7135 - val_loss: 0.6717 - val_accuracy: 0.6884
Epoch 6/100
212/212 [==============================] - 67s 314ms/step - loss: 0.5402 - accuracy:
0.7287 - val_loss: 0.5524 - val_accuracy: 0.7462
Epoch 7/100
212/212 [==============================] - 66s 309ms/step - loss: 0.5139 - accuracy:
0.7522 - val_loss: 0.5206 - val_accuracy: 0.7580
Epoch 8/100
212/212 [==============================] - 71s 335ms/step - loss: 0.4835 - accuracy:
0.7666 - val_loss: 0.7627 - val_accuracy: 0.7228
Epoch 9/100
212/212 [==============================] - 70s 330ms/step - loss: 0.4485 - accuracy:
0.7946 - val_loss: 0.5855 - val_accuracy: 0.7462
Epoch 10/100
212/212 [==============================] - 69s 327ms/step - loss: 0.4404 - accuracy:
0.8026 - val_loss: 0.6406 - val_accuracy: 0.7144
Epoch 11/100
212/212 [==============================] - 71s 333ms/step - loss: 0.4191 - accuracy:
0.8090 - val_loss: 0.8895 - val_accuracy: 0.7085
Epoch 12/100
212/212 [==============================] - 73s 345ms/step - loss: 0.4009 - accuracy:
0.8260 - val_loss: 0.7933 - val_accuracy: 0.7169
Epoch 13/100
212/212 [==============================] - 70s 329ms/step - loss: 0.3921 - accuracy:
0.8270 - val_loss: 0.7185 - val_accuracy: 0.7362
Epoch 14/100
212/212 [==============================] - 70s 328ms/step - loss: 0.3793 - accuracy:
0.8363 - val_loss: 0.6769 - val_accuracy: 0.7286
Epoch 15/100
212/212 [==============================] - 70s 329ms/step - loss: 0.3853 - accuracy:
0.8352 - val_loss: 0.5431 - val_accuracy: 0.7956
Epoch 16/100
212/212 [==============================] - 71s 335ms/step - loss: 0.3747 - accuracy:
0.8405 - val_loss: 0.6381 - val_accuracy: 0.7647
Epoch 17/100
212/212 [==============================] - 70s 328ms/step - loss: 0.3613 - accuracy:
0.8425 - val_loss: 0.8121 - val_accuracy: 0.7395
```

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
```

```python
plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
plt.title('Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 6))

plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```





In [ ]:
```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validación - Pérdida: {val_loss:.4f}, Exactitud: {val_accuracy:.4f}")
```

```
38/38 [==============================] - 3s 65ms/step - loss: 0.5206 - accuracy: 0.7
580
Validación - Pérdida: 0.5206, Exactitud: 0.7580
```

In [24]:
```
model.save('modelo_completo2.h5')
# Guardar solo pesos
model.save_weights('pesos_modelo2.h5')
```

```
d:\Anaconda\envs\tesis38\lib\site-packages\keras\src\engine\training.py:3000: UserWa
rning: You are saving your model as an HDF5 file via `model.save()`. This file forma
t is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')`.
  saving_api.save_model(
```

Modelo CNN-3.

In [ ]:

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import KFold

data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"

def load_images_from_directory(directory):
    images = []
    labels = []
    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)
        for img_name in os.listdir(folder_path):
            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img_path = os.path.join(folder_path, img_name)
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)
    return np.array(images), np.array(labels)

X, y = load_images_from_directory(data_dir)
X = X / 255.0

k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

os.makedirs("modelos_kfold", exist_ok=True)
os.makedirs("graficas_kfold", exist_ok=True)

fold_accuracies = []
fold_losses = []

for fold, (train_idx, val_idx) in enumerate(kf.split(X, y)):
    print(f"\n--- Fold {fold + 1}/{k} ---")
    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]
```

```
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_we

    history = model.fit(
        datagen.flow(X_train, y_train, batch_size=32),
        epochs=100,
        validation_data=(X_val, y_val),
        callbacks=[early_stopping],
        verbose=1
    )

    model.save(f"modelos_kfold/modelo_fold_{fold+1}.h5")

    val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=0)
    fold_losses.append(val_loss)
    fold_accuracies.append(val_accuracy)

    print(f"Fold {fold+1} - Val Loss: {val_loss:.4f}, Val Accuracy: {val_accuracy:.

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.title(f'Fold {fold+1} - Accuracy')
    plt.xlabel('Epochs'); plt.ylabel('Accuracy'); plt.legend(); plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f'Fold {fold+1} - Loss')
    plt.xlabel('Epochs'); plt.ylabel('Loss'); plt.legend(); plt.grid(True)

    plt.tight_layout()
    plt.savefig(f'graficas_kfold/curvas_fold_{fold+1}.png')
    plt.close()

print("\n--- Resultados de Validación Cruzada ---")
print(f"Accuracy promedio: {np.mean(fold_accuracies):.4f} ± {np.std(fold_accuracies
print(f"Loss promedio: {np.mean(fold_losses):.4f} ± {np.std(fold_losses):.4f}")
```

```
--- Fold 1/5 ---
Epoch 1/100
310/310 [==============================] - 139s 439ms/step - loss: 0.6684 - accurac
y: 0.6020 - val_loss: 0.6286 - val_accuracy: 0.6644
Epoch 2/100
310/310 [==============================] - 133s 428ms/step - loss: 0.6414 - accurac
y: 0.6485 - val_loss: 0.6403 - val_accuracy: 0.6406
Epoch 3/100
310/310 [==============================] - 145s 466ms/step - loss: 0.6238 - accurac
y: 0.6623 - val_loss: 0.6272 - val_accuracy: 0.6559
Epoch 4/100
310/310 [==============================] - 150s 484ms/step - loss: 0.5982 - accurac
y: 0.6881 - val_loss: 0.7397 - val_accuracy: 0.6519
Epoch 5/100
310/310 [==============================] - 165s 532ms/step - loss: 0.5787 - accurac
y: 0.7037 - val_loss: 0.5378 - val_accuracy: 0.7200
Epoch 6/100
310/310 [==============================] - 138s 444ms/step - loss: 0.5464 - accurac
y: 0.7274 - val_loss: 0.8006 - val_accuracy: 0.6555
Epoch 7/100
310/310 [==============================] - 129s 415ms/step - loss: 0.5141 - accurac
y: 0.7546 - val_loss: 0.6542 - val_accuracy: 0.6902
Epoch 8/100
310/310 [==============================] - 132s 425ms/step - loss: 0.4941 - accurac
y: 0.7692 - val_loss: 0.5007 - val_accuracy: 0.7547
Epoch 9/100
310/310 [==============================] - 132s 425ms/step - loss: 0.4753 - accurac
y: 0.7780 - val_loss: 0.5725 - val_accuracy: 0.7430
Epoch 10/100
310/310 [==============================] - 132s 424ms/step - loss: 0.4695 - accurac
y: 0.7786 - val_loss: 0.5404 - val_accuracy: 0.7455
Epoch 11/100
310/310 [==============================] - 130s 420ms/step - loss: 0.4469 - accurac
y: 0.7945 - val_loss: 0.4969 - val_accuracy: 0.7822
Epoch 12/100
310/310 [==============================] - 132s 424ms/step - loss: 0.4360 - accurac
y: 0.7997 - val_loss: 0.6503 - val_accuracy: 0.7100
Epoch 13/100
310/310 [==============================] - 133s 429ms/step - loss: 0.4307 - accurac
y: 0.8037 - val_loss: 0.5052 - val_accuracy: 0.7668
Epoch 14/100
310/310 [==============================] - 135s 435ms/step - loss: 0.4294 - accurac
y: 0.8068 - val_loss: 0.4727 - val_accuracy: 0.7793
Epoch 15/100
310/310 [==============================] - 136s 438ms/step - loss: 0.4119 - accurac
y: 0.8160 - val_loss: 0.4343 - val_accuracy: 0.7979
Epoch 16/100
310/310 [==============================] - 134s 432ms/step - loss: 0.4092 - accurac
y: 0.8197 - val_loss: 0.4228 - val_accuracy: 0.7975
Epoch 17/100
310/310 [==============================] - 142s 458ms/step - loss: 0.4112 - accurac
y: 0.8204 - val_loss: 0.5873 - val_accuracy: 0.7326
Epoch 18/100
310/310 [==============================] - 136s 437ms/step - loss: 0.4045 - accurac
y: 0.8192 - val_loss: 0.5225 - val_accuracy: 0.7850
Epoch 19/100
```

```
310/310 [==============================] - 136s 437ms/step - loss: 0.3920 - accurac
y: 0.8295 - val_loss: 0.4390 - val_accuracy: 0.7991
Epoch 20/100
310/310 [==============================] - 132s 425ms/step - loss: 0.3890 - accurac
y: 0.8275 - val_loss: 0.3912 - val_accuracy: 0.8286
Epoch 21/100
310/310 [==============================] - 135s 436ms/step - loss: 0.3817 - accurac
y: 0.8307 - val_loss: 0.4260 - val_accuracy: 0.8104
Epoch 22/100
310/310 [==============================] - 135s 435ms/step - loss: 0.3811 - accurac
y: 0.8359 - val_loss: 0.3181 - val_accuracy: 0.8641
Epoch 23/100
310/310 [==============================] - 142s 458ms/step - loss: 0.3742 - accurac
y: 0.8377 - val_loss: 0.3341 - val_accuracy: 0.8528
Epoch 24/100
310/310 [==============================] - 140s 452ms/step - loss: 0.3722 - accurac
y: 0.8335 - val_loss: 0.4610 - val_accuracy: 0.7967
Epoch 25/100
310/310 [==============================] - 136s 440ms/step - loss: 0.3720 - accurac
y: 0.8360 - val_loss: 0.3839 - val_accuracy: 0.8419
Epoch 26/100
310/310 [==============================] - 140s 451ms/step - loss: 0.3739 - accurac
y: 0.8370 - val_loss: 0.5160 - val_accuracy: 0.7870
Epoch 27/100
310/310 [==============================] - 135s 434ms/step - loss: 0.3669 - accurac
y: 0.8419 - val_loss: 0.3786 - val_accuracy: 0.8342
Epoch 28/100
310/310 [==============================] - 130s 419ms/step - loss: 0.3667 - accurac
y: 0.8411 - val_loss: 0.4814 - val_accuracy: 0.7890
Epoch 29/100
310/310 [==============================] - 126s 405ms/step - loss: 0.3534 - accurac
y: 0.8473 - val_loss: 0.3954 - val_accuracy: 0.8354
Epoch 30/100
310/310 [==============================] - 124s 399ms/step - loss: 0.3533 - accurac
y: 0.8465 - val_loss: 0.3472 - val_accuracy: 0.8544
Epoch 31/100
310/310 [==============================] - 123s 397ms/step - loss: 0.3528 - accurac
y: 0.8495 - val_loss: 0.4540 - val_accuracy: 0.7971
Epoch 32/100
310/310 [==============================] - 140s 450ms/step - loss: 0.3488 - accurac
y: 0.8466 - val_loss: 0.4888 - val_accuracy: 0.7947
Fold 1 - Val Loss: 0.3181, Val Accuracy: 0.8641

--- Fold 2/5 ---
Epoch 1/100
310/310 [==============================] - 126s 395ms/step - loss: 0.6710 - accurac
y: 0.5953 - val_loss: 0.6384 - val_accuracy: 0.6454
Epoch 2/100
310/310 [==============================] - 121s 390ms/step - loss: 0.6443 - accurac
y: 0.6462 - val_loss: 0.6927 - val_accuracy: 0.6410
Epoch 3/100
310/310 [==============================] - 122s 393ms/step - loss: 0.6152 - accurac
y: 0.6626 - val_loss: 1.0331 - val_accuracy: 0.6200
Epoch 4/100
310/310 [==============================] - 125s 404ms/step - loss: 0.5881 - accurac
y: 0.6913 - val_loss: 0.7271 - val_accuracy: 0.6644
```

```
Epoch 5/100
310/310 [==============================] - 114s 368ms/step - loss: 0.5673 - accurac
y: 0.7115 - val_loss: 0.6834 - val_accuracy: 0.7047
Epoch 6/100
310/310 [==============================] - 113s 365ms/step - loss: 0.5479 - accurac
y: 0.7275 - val_loss: 0.7668 - val_accuracy: 0.6563
Epoch 7/100
310/310 [==============================] - 125s 402ms/step - loss: 0.5211 - accurac
y: 0.7454 - val_loss: 0.6533 - val_accuracy: 0.7152
Epoch 8/100
310/310 [==============================] - 114s 367ms/step - loss: 0.5086 - accurac
y: 0.7496 - val_loss: 0.7337 - val_accuracy: 0.7055
Epoch 9/100
310/310 [==============================] - 113s 365ms/step - loss: 0.4884 - accurac
y: 0.7699 - val_loss: 0.6274 - val_accuracy: 0.7096
Epoch 10/100
310/310 [==============================] - 115s 370ms/step - loss: 0.4750 - accurac
y: 0.7794 - val_loss: 0.4547 - val_accuracy: 0.8076
Epoch 11/100
310/310 [==============================] - 116s 374ms/step - loss: 0.4705 - accurac
y: 0.7801 - val_loss: 0.4574 - val_accuracy: 0.8148
Epoch 12/100
310/310 [==============================] - 111s 359ms/step - loss: 0.4516 - accurac
y: 0.7896 - val_loss: 0.4294 - val_accuracy: 0.8080
Epoch 13/100
310/310 [==============================] - 114s 366ms/step - loss: 0.4409 - accurac
y: 0.7974 - val_loss: 0.4653 - val_accuracy: 0.7939
Epoch 14/100
310/310 [==============================] - 113s 364ms/step - loss: 0.4290 - accurac
y: 0.8039 - val_loss: 0.4934 - val_accuracy: 0.7668
Epoch 15/100
310/310 [==============================] - 111s 358ms/step - loss: 0.4284 - accurac
y: 0.8010 - val_loss: 0.5089 - val_accuracy: 0.7886
Epoch 16/100
310/310 [==============================] - 112s 361ms/step - loss: 0.4213 - accurac
y: 0.8164 - val_loss: 0.4149 - val_accuracy: 0.8193
Epoch 17/100
310/310 [==============================] - 106s 342ms/step - loss: 0.4169 - accurac
y: 0.8115 - val_loss: 0.4160 - val_accuracy: 0.8108
Epoch 18/100
310/310 [==============================] - 112s 362ms/step - loss: 0.4031 - accurac
y: 0.8189 - val_loss: 0.4070 - val_accuracy: 0.8213
Epoch 19/100
310/310 [==============================] - 111s 359ms/step - loss: 0.3996 - accurac
y: 0.8247 - val_loss: 0.4738 - val_accuracy: 0.7975
Epoch 20/100
310/310 [==============================] - 111s 357ms/step - loss: 0.4030 - accurac
y: 0.8224 - val_loss: 0.3884 - val_accuracy: 0.8354
Epoch 21/100
310/310 [==============================] - 111s 357ms/step - loss: 0.4037 - accurac
y: 0.8180 - val_loss: 0.4101 - val_accuracy: 0.8189
Epoch 22/100
310/310 [==============================] - 113s 363ms/step - loss: 0.3997 - accurac
y: 0.8180 - val_loss: 0.3747 - val_accuracy: 0.8415
Epoch 23/100
310/310 [==============================] - 114s 366ms/step - loss: 0.3847 - accurac
```

```
y: 0.8297 - val_loss: 0.4083 - val_accuracy: 0.8265
Epoch 24/100
310/310 [==============================] - 115s 372ms/step - loss: 0.3742 - accurac
y: 0.8317 - val_loss: 0.4314 - val_accuracy: 0.8157
Epoch 25/100
310/310 [==============================] - 110s 356ms/step - loss: 0.3776 - accurac
y: 0.8290 - val_loss: 0.3874 - val_accuracy: 0.8463
Epoch 26/100
310/310 [==============================] - 110s 354ms/step - loss: 0.3804 - accurac
y: 0.8326 - val_loss: 0.3677 - val_accuracy: 0.8435
Epoch 27/100
310/310 [==============================] - 110s 356ms/step - loss: 0.3751 - accurac
y: 0.8317 - val_loss: 0.3656 - val_accuracy: 0.8467
Epoch 28/100
310/310 [==============================] - 113s 364ms/step - loss: 0.3725 - accurac
y: 0.8352 - val_loss: 0.4009 - val_accuracy: 0.8278
Epoch 29/100
310/310 [==============================] - 113s 366ms/step - loss: 0.3789 - accurac
y: 0.8365 - val_loss: 0.3572 - val_accuracy: 0.8378
Epoch 30/100
310/310 [==============================] - 111s 358ms/step - loss: 0.3655 - accurac
y: 0.8406 - val_loss: 0.3363 - val_accuracy: 0.8459
Epoch 31/100
310/310 [==============================] - 112s 361ms/step - loss: 0.3647 - accurac
y: 0.8418 - val_loss: 0.3804 - val_accuracy: 0.8511
Epoch 32/100
310/310 [==============================] - 118s 379ms/step - loss: 0.3625 - accurac
y: 0.8413 - val_loss: 0.4371 - val_accuracy: 0.8350
Epoch 33/100
310/310 [==============================] - 101s 327ms/step - loss: 0.3555 - accurac
y: 0.8459 - val_loss: 0.3728 - val_accuracy: 0.8399
Epoch 34/100
310/310 [==============================] - 113s 363ms/step - loss: 0.3547 - accurac
y: 0.8436 - val_loss: 0.4326 - val_accuracy: 0.8225
Epoch 35/100
310/310 [==============================] - 109s 351ms/step - loss: 0.3632 - accurac
y: 0.8437 - val_loss: 0.3916 - val_accuracy: 0.8520
Epoch 36/100
310/310 [==============================] - 109s 351ms/step - loss: 0.3563 - accurac
y: 0.8487 - val_loss: 0.3705 - val_accuracy: 0.8516
Epoch 37/100
310/310 [==============================] - 109s 350ms/step - loss: 0.3513 - accurac
y: 0.8476 - val_loss: 0.4131 - val_accuracy: 0.8338
Epoch 38/100
310/310 [==============================] - 108s 348ms/step - loss: 0.3502 - accurac
y: 0.8484 - val_loss: 0.6293 - val_accuracy: 0.7939
Epoch 39/100
310/310 [==============================] - 120s 388ms/step - loss: 0.3436 - accurac
y: 0.8497 - val_loss: 0.4422 - val_accuracy: 0.8427
Epoch 40/100
310/310 [==============================] - 131s 421ms/step - loss: 0.3457 - accurac
y: 0.8525 - val_loss: 0.4083 - val_accuracy: 0.8511
Fold 2 - Val Loss: 0.3363, Val Accuracy: 0.8459

--- Fold 3/5 ---
Epoch 1/100
```

```
310/310 [==============================] - 123s 389ms/step - loss: 0.6646 - accurac
y: 0.6066 - val_loss: 0.6039 - val_accuracy: 0.6764
Epoch 2/100
310/310 [==============================] - 121s 391ms/step - loss: 0.6381 - accurac
y: 0.6434 - val_loss: 0.6658 - val_accuracy: 0.6764
Epoch 3/100
310/310 [==============================] - 122s 394ms/step - loss: 0.6137 - accurac
y: 0.6681 - val_loss: 0.5933 - val_accuracy: 0.7111
Epoch 4/100
310/310 [==============================] - 122s 394ms/step - loss: 0.5847 - accurac
y: 0.6902 - val_loss: 0.6236 - val_accuracy: 0.6957
Epoch 5/100
310/310 [==============================] - 119s 382ms/step - loss: 0.5676 - accurac
y: 0.7085 - val_loss: 0.7029 - val_accuracy: 0.6715
Epoch 6/100
310/310 [==============================] - 116s 373ms/step - loss: 0.5478 - accurac
y: 0.7233 - val_loss: 0.7917 - val_accuracy: 0.6626
Epoch 7/100
310/310 [==============================] - 126s 405ms/step - loss: 0.5334 - accurac
y: 0.7375 - val_loss: 0.5258 - val_accuracy: 0.7385
Epoch 8/100
310/310 [==============================] - 119s 384ms/step - loss: 0.5037 - accurac
y: 0.7537 - val_loss: 0.7038 - val_accuracy: 0.6965
Epoch 9/100
310/310 [==============================] - 125s 402ms/step - loss: 0.5051 - accurac
y: 0.7506 - val_loss: 0.6129 - val_accuracy: 0.7038
Epoch 10/100
310/310 [==============================] - 108s 348ms/step - loss: 0.4739 - accurac
y: 0.7715 - val_loss: 0.5276 - val_accuracy: 0.7429
Epoch 11/100
310/310 [==============================] - 110s 354ms/step - loss: 0.4620 - accurac
y: 0.7853 - val_loss: 0.7013 - val_accuracy: 0.6816
Epoch 12/100
310/310 [==============================] - 115s 370ms/step - loss: 0.4528 - accurac
y: 0.7912 - val_loss: 0.6196 - val_accuracy: 0.7260
Epoch 13/100
310/310 [==============================] - 124s 399ms/step - loss: 0.4402 - accurac
y: 0.7960 - val_loss: 0.4699 - val_accuracy: 0.7817
Epoch 14/100
310/310 [==============================] - 117s 379ms/step - loss: 0.4324 - accurac
y: 0.7997 - val_loss: 0.4631 - val_accuracy: 0.7736
Epoch 15/100
310/310 [==============================] - 120s 386ms/step - loss: 0.4266 - accurac
y: 0.8063 - val_loss: 0.7032 - val_accuracy: 0.7082
Epoch 16/100
310/310 [==============================] - 119s 382ms/step - loss: 0.4130 - accurac
y: 0.8110 - val_loss: 0.5222 - val_accuracy: 0.7651
Epoch 17/100
310/310 [==============================] - 117s 376ms/step - loss: 0.4206 - accurac
y: 0.8086 - val_loss: 0.4906 - val_accuracy: 0.7744
Epoch 18/100
310/310 [==============================] - 134s 432ms/step - loss: 0.4035 - accurac
y: 0.8209 - val_loss: 0.4480 - val_accuracy: 0.7906
Epoch 19/100
310/310 [==============================] - 111s 358ms/step - loss: 0.3965 - accurac
y: 0.8232 - val_loss: 0.4689 - val_accuracy: 0.7825
```

```
Epoch 20/100
310/310 [==============================] - 112s 362ms/step - loss: 0.3882 - accurac
y: 0.8261 - val_loss: 0.5218 - val_accuracy: 0.7672
Epoch 21/100
310/310 [==============================] - 112s 360ms/step - loss: 0.3841 - accurac
y: 0.8303 - val_loss: 0.4941 - val_accuracy: 0.7926
Epoch 22/100
310/310 [==============================] - 108s 348ms/step - loss: 0.3736 - accurac
y: 0.8319 - val_loss: 0.4506 - val_accuracy: 0.7881
Epoch 23/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3786 - accurac
y: 0.8357 - val_loss: 0.4590 - val_accuracy: 0.7906
Epoch 24/100
310/310 [==============================] - 114s 367ms/step - loss: 0.3737 - accurac
y: 0.8357 - val_loss: 0.3322 - val_accuracy: 0.8527
Epoch 25/100
310/310 [==============================] - 109s 352ms/step - loss: 0.3610 - accurac
y: 0.8397 - val_loss: 0.4987 - val_accuracy: 0.7785
Epoch 26/100
310/310 [==============================] - 110s 353ms/step - loss: 0.3602 - accurac
y: 0.8402 - val_loss: 0.3595 - val_accuracy: 0.8345
Epoch 27/100
310/310 [==============================] - 122s 393ms/step - loss: 0.3627 - accurac
y: 0.8408 - val_loss: 0.4491 - val_accuracy: 0.8067
Epoch 28/100
310/310 [==============================] - 122s 392ms/step - loss: 0.3503 - accurac
y: 0.8484 - val_loss: 0.4651 - val_accuracy: 0.7877
Epoch 29/100
310/310 [==============================] - 122s 393ms/step - loss: 0.3499 - accurac
y: 0.8439 - val_loss: 0.3511 - val_accuracy: 0.8406
Epoch 30/100
310/310 [==============================] - 111s 357ms/step - loss: 0.3453 - accurac
y: 0.8497 - val_loss: 0.3905 - val_accuracy: 0.8200
Epoch 31/100
310/310 [==============================] - 112s 361ms/step - loss: 0.3538 - accurac
y: 0.8502 - val_loss: 0.3591 - val_accuracy: 0.8313
Epoch 32/100
310/310 [==============================] - 143s 461ms/step - loss: 0.3402 - accurac
y: 0.8515 - val_loss: 0.4170 - val_accuracy: 0.8184
Epoch 33/100
310/310 [==============================] - 125s 404ms/step - loss: 0.3425 - accurac
y: 0.8546 - val_loss: 0.4869 - val_accuracy: 0.8006
Epoch 34/100
310/310 [==============================] - 124s 399ms/step - loss: 0.3379 - accurac
y: 0.8523 - val_loss: 0.3942 - val_accuracy: 0.8245
Fold 3 - Val Loss: 0.3322, Val Accuracy: 0.8527

--- Fold 4/5 ---
Epoch 1/100
310/310 [==============================] - 126s 393ms/step - loss: 0.6698 - accurac
y: 0.6051 - val_loss: 0.6529 - val_accuracy: 0.6384
Epoch 2/100
310/310 [==============================] - 131s 423ms/step - loss: 0.6380 - accurac
y: 0.6479 - val_loss: 0.6690 - val_accuracy: 0.6489
Epoch 3/100
310/310 [==============================] - 123s 398ms/step - loss: 0.6135 - accurac
```

```
y: 0.6703 - val_loss: 0.5774 - val_accuracy: 0.6812
Epoch 4/100
310/310 [==============================] - 119s 385ms/step - loss: 0.5905 - accurac
y: 0.6916 - val_loss: 0.5712 - val_accuracy: 0.7014
Epoch 5/100
310/310 [==============================] - 121s 390ms/step - loss: 0.5603 - accurac
y: 0.7187 - val_loss: 0.7823 - val_accuracy: 0.6691
Epoch 6/100
310/310 [==============================] - 122s 394ms/step - loss: 0.5395 - accurac
y: 0.7312 - val_loss: 0.6999 - val_accuracy: 0.6759
Epoch 7/100
310/310 [==============================] - 117s 376ms/step - loss: 0.5243 - accurac
y: 0.7420 - val_loss: 0.7902 - val_accuracy: 0.6723
Epoch 8/100
310/310 [==============================] - 114s 367ms/step - loss: 0.5076 - accurac
y: 0.7528 - val_loss: 0.7819 - val_accuracy: 0.6695
Epoch 9/100
310/310 [==============================] - 114s 367ms/step - loss: 0.4983 - accurac
y: 0.7612 - val_loss: 0.6618 - val_accuracy: 0.7042
Epoch 10/100
310/310 [==============================] - 117s 377ms/step - loss: 0.4822 - accurac
y: 0.7706 - val_loss: 0.4744 - val_accuracy: 0.7736
Epoch 11/100
310/310 [==============================] - 120s 387ms/step - loss: 0.4665 - accurac
y: 0.7820 - val_loss: 0.5382 - val_accuracy: 0.7421
Epoch 12/100
310/310 [==============================] - 123s 395ms/step - loss: 0.4610 - accurac
y: 0.7835 - val_loss: 0.5650 - val_accuracy: 0.7421
Epoch 13/100
310/310 [==============================] - 117s 378ms/step - loss: 0.4530 - accurac
y: 0.7905 - val_loss: 0.6874 - val_accuracy: 0.7002
Epoch 14/100
310/310 [==============================] - 117s 377ms/step - loss: 0.4425 - accurac
y: 0.7979 - val_loss: 0.5613 - val_accuracy: 0.7470
Epoch 15/100
310/310 [==============================] - 119s 384ms/step - loss: 0.4282 - accurac
y: 0.7980 - val_loss: 0.4825 - val_accuracy: 0.7873
Epoch 16/100
310/310 [==============================] - 117s 379ms/step - loss: 0.4233 - accurac
y: 0.8077 - val_loss: 0.4596 - val_accuracy: 0.7926
Epoch 17/100
310/310 [==============================] - 116s 374ms/step - loss: 0.4238 - accurac
y: 0.8100 - val_loss: 0.4950 - val_accuracy: 0.7873
Epoch 18/100
310/310 [==============================] - 129s 416ms/step - loss: 0.4117 - accurac
y: 0.8162 - val_loss: 0.4961 - val_accuracy: 0.7954
Epoch 19/100
310/310 [==============================] - 123s 397ms/step - loss: 0.4048 - accurac
y: 0.8170 - val_loss: 0.4626 - val_accuracy: 0.8006
Epoch 20/100
310/310 [==============================] - 128s 411ms/step - loss: 0.4032 - accurac
y: 0.8180 - val_loss: 0.4200 - val_accuracy: 0.8152
Epoch 21/100
310/310 [==============================] - 146s 469ms/step - loss: 0.3988 - accurac
y: 0.8203 - val_loss: 0.4610 - val_accuracy: 0.7950
Epoch 22/100
```

```
310/310 [==============================] - 121s 390ms/step - loss: 0.3881 - accurac
y: 0.8284 - val_loss: 0.4203 - val_accuracy: 0.8204
Epoch 23/100
310/310 [==============================] - 132s 425ms/step - loss: 0.4026 - accurac
y: 0.8245 - val_loss: 0.4369 - val_accuracy: 0.8059
Epoch 24/100
310/310 [==============================] - 125s 403ms/step - loss: 0.3830 - accurac
y: 0.8316 - val_loss: 0.4326 - val_accuracy: 0.8148
Epoch 25/100
310/310 [==============================] - 122s 392ms/step - loss: 0.3779 - accurac
y: 0.8360 - val_loss: 0.4643 - val_accuracy: 0.8111
Epoch 26/100
310/310 [==============================] - 108s 347ms/step - loss: 0.3819 - accurac
y: 0.8307 - val_loss: 0.5370 - val_accuracy: 0.7740
Epoch 27/100
310/310 [==============================] - 104s 335ms/step - loss: 0.3748 - accurac
y: 0.8370 - val_loss: 0.4498 - val_accuracy: 0.7974
Epoch 28/100
310/310 [==============================] - 107s 343ms/step - loss: 0.3686 - accurac
y: 0.8386 - val_loss: 0.5516 - val_accuracy: 0.7776
Epoch 29/100
310/310 [==============================] - 113s 363ms/step - loss: 0.3640 - accurac
y: 0.8424 - val_loss: 0.4982 - val_accuracy: 0.7861
Epoch 30/100
310/310 [==============================] - 110s 353ms/step - loss: 0.3687 - accurac
y: 0.8389 - val_loss: 0.3854 - val_accuracy: 0.8245
Epoch 31/100
310/310 [==============================] - 111s 357ms/step - loss: 0.3649 - accurac
y: 0.8364 - val_loss: 0.4397 - val_accuracy: 0.8152
Epoch 32/100
310/310 [==============================] - 110s 356ms/step - loss: 0.3549 - accurac
y: 0.8467 - val_loss: 0.4281 - val_accuracy: 0.8111
Epoch 33/100
310/310 [==============================] - 111s 359ms/step - loss: 0.3575 - accurac
y: 0.8394 - val_loss: 0.4630 - val_accuracy: 0.8111
Epoch 34/100
310/310 [==============================] - 118s 380ms/step - loss: 0.3556 - accurac
y: 0.8452 - val_loss: 0.3804 - val_accuracy: 0.8370
Epoch 35/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3538 - accurac
y: 0.8458 - val_loss: 0.4617 - val_accuracy: 0.8192
Epoch 36/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3570 - accurac
y: 0.8406 - val_loss: 0.3730 - val_accuracy: 0.8378
Epoch 37/100
310/310 [==============================] - 118s 381ms/step - loss: 0.3487 - accurac
y: 0.8474 - val_loss: 0.4522 - val_accuracy: 0.8091
Epoch 38/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3452 - accurac
y: 0.8518 - val_loss: 0.3715 - val_accuracy: 0.8426
Epoch 39/100
310/310 [==============================] - 115s 372ms/step - loss: 0.3489 - accurac
y: 0.8525 - val_loss: 0.3970 - val_accuracy: 0.8305
Epoch 40/100
310/310 [==============================] - 116s 375ms/step - loss: 0.3412 - accurac
y: 0.8503 - val_loss: 0.4044 - val_accuracy: 0.8341
```

```
Epoch 41/100
310/310 [==============================] - 116s 373ms/step - loss: 0.3430 - accurac
y: 0.8516 - val_loss: 0.3592 - val_accuracy: 0.8527
Epoch 42/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3388 - accurac
y: 0.8504 - val_loss: 0.4627 - val_accuracy: 0.8224
Epoch 43/100
310/310 [==============================] - 117s 376ms/step - loss: 0.3389 - accurac
y: 0.8529 - val_loss: 0.4303 - val_accuracy: 0.8257
Epoch 44/100
310/310 [==============================] - 117s 376ms/step - loss: 0.3380 - accurac
y: 0.8548 - val_loss: 0.3146 - val_accuracy: 0.8721
Epoch 45/100
310/310 [==============================] - 117s 378ms/step - loss: 0.3323 - accurac
y: 0.8548 - val_loss: 0.4638 - val_accuracy: 0.8236
Epoch 46/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3363 - accurac
y: 0.8543 - val_loss: 0.4566 - val_accuracy: 0.8269
Epoch 47/100
310/310 [==============================] - 118s 381ms/step - loss: 0.3303 - accurac
y: 0.8600 - val_loss: 0.4415 - val_accuracy: 0.8236
Epoch 48/100
310/310 [==============================] - 118s 379ms/step - loss: 0.3346 - accurac
y: 0.8573 - val_loss: 0.3645 - val_accuracy: 0.8499
Epoch 49/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3265 - accurac
y: 0.8608 - val_loss: 0.3695 - val_accuracy: 0.8333
Epoch 50/100
310/310 [==============================] - 117s 377ms/step - loss: 0.3146 - accurac
y: 0.8642 - val_loss: 0.4146 - val_accuracy: 0.8418
Epoch 51/100
310/310 [==============================] - 124s 399ms/step - loss: 0.3224 - accurac
y: 0.8654 - val_loss: 0.3804 - val_accuracy: 0.8398
Epoch 52/100
310/310 [==============================] - 116s 375ms/step - loss: 0.3229 - accurac
y: 0.8602 - val_loss: 0.4169 - val_accuracy: 0.8430
Epoch 53/100
310/310 [==============================] - 117s 376ms/step - loss: 0.3216 - accurac
y: 0.8656 - val_loss: 0.3949 - val_accuracy: 0.8329
Epoch 54/100
310/310 [==============================] - 119s 383ms/step - loss: 0.3162 - accurac
y: 0.8647 - val_loss: 0.4686 - val_accuracy: 0.8047
Fold 4 - Val Loss: 0.3146, Val Accuracy: 0.8721

--- Fold 5/5 ---
Epoch 1/100
310/310 [==============================] - 122s 382ms/step - loss: 0.6626 - accurac
y: 0.6080 - val_loss: 0.6238 - val_accuracy: 0.6509
Epoch 2/100
310/310 [==============================] - 120s 386ms/step - loss: 0.6361 - accurac
y: 0.6530 - val_loss: 0.5961 - val_accuracy: 0.6703
Epoch 3/100
310/310 [==============================] - 118s 381ms/step - loss: 0.6095 - accurac
y: 0.6764 - val_loss: 0.7129 - val_accuracy: 0.6457
Epoch 4/100
310/310 [==============================] - 122s 394ms/step - loss: 0.5868 - accurac
```

```
y: 0.6890 - val_loss: 0.6232 - val_accuracy: 0.6836
Epoch 5/100
310/310 [==============================] - 121s 389ms/step - loss: 0.5549 - accurac
y: 0.7181 - val_loss: 0.6184 - val_accuracy: 0.6860
Epoch 6/100
310/310 [==============================] - 118s 379ms/step - loss: 0.5321 - accurac
y: 0.7415 - val_loss: 0.5782 - val_accuracy: 0.7304
Epoch 7/100
310/310 [==============================] - 121s 390ms/step - loss: 0.5079 - accurac
y: 0.7566 - val_loss: 0.5853 - val_accuracy: 0.7401
Epoch 8/100
310/310 [==============================] - 118s 379ms/step - loss: 0.4955 - accurac
y: 0.7591 - val_loss: 0.5873 - val_accuracy: 0.7349
Epoch 9/100
310/310 [==============================] - 120s 386ms/step - loss: 0.4743 - accurac
y: 0.7749 - val_loss: 0.5431 - val_accuracy: 0.7478
Epoch 10/100
310/310 [==============================] - 110s 355ms/step - loss: 0.4693 - accurac
y: 0.7819 - val_loss: 0.4870 - val_accuracy: 0.7833
Epoch 11/100
310/310 [==============================] - 118s 380ms/step - loss: 0.4578 - accurac
y: 0.7904 - val_loss: 0.4469 - val_accuracy: 0.7889
Epoch 12/100
310/310 [==============================] - 118s 381ms/step - loss: 0.4454 - accurac
y: 0.7972 - val_loss: 0.5377 - val_accuracy: 0.7530
Epoch 13/100
310/310 [==============================] - 119s 384ms/step - loss: 0.4375 - accurac
y: 0.7982 - val_loss: 0.4876 - val_accuracy: 0.7809
Epoch 14/100
310/310 [==============================] - 119s 385ms/step - loss: 0.4338 - accurac
y: 0.8020 - val_loss: 0.4452 - val_accuracy: 0.7970
Epoch 15/100
310/310 [==============================] - 121s 391ms/step - loss: 0.4238 - accurac
y: 0.8049 - val_loss: 0.4126 - val_accuracy: 0.8087
Epoch 16/100
310/310 [==============================] - 122s 394ms/step - loss: 0.4172 - accurac
y: 0.8108 - val_loss: 0.4878 - val_accuracy: 0.7785
Epoch 17/100
310/310 [==============================] - 118s 381ms/step - loss: 0.4136 - accurac
y: 0.8131 - val_loss: 0.5116 - val_accuracy: 0.7772
Epoch 18/100
310/310 [==============================] - 121s 389ms/step - loss: 0.4039 - accurac
y: 0.8221 - val_loss: 0.3956 - val_accuracy: 0.7978
Epoch 19/100
310/310 [==============================] - 118s 380ms/step - loss: 0.4031 - accurac
y: 0.8178 - val_loss: 0.5274 - val_accuracy: 0.7587
Epoch 20/100
310/310 [==============================] - 121s 391ms/step - loss: 0.3861 - accurac
y: 0.8260 - val_loss: 0.3973 - val_accuracy: 0.8345
Epoch 21/100
310/310 [==============================] - 119s 382ms/step - loss: 0.3941 - accurac
y: 0.8251 - val_loss: 0.4801 - val_accuracy: 0.7910
Epoch 22/100
310/310 [==============================] - 118s 380ms/step - loss: 0.3885 - accurac
y: 0.8297 - val_loss: 0.4872 - val_accuracy: 0.7849
Epoch 23/100
```

```
310/310 [==============================] - 119s 383ms/step - loss: 0.3869 - accurac
y: 0.8287 - val_loss: 0.4243 - val_accuracy: 0.8212
Epoch 24/100
310/310 [==============================] - 118s 382ms/step - loss: 0.3784 - accurac
y: 0.8326 - val_loss: 0.3858 - val_accuracy: 0.8265
Epoch 25/100
310/310 [==============================] - 118s 381ms/step - loss: 0.3757 - accurac
y: 0.8373 - val_loss: 0.4411 - val_accuracy: 0.8132
Epoch 26/100
310/310 [==============================] - 126s 405ms/step - loss: 0.3694 - accurac
y: 0.8434 - val_loss: 0.5223 - val_accuracy: 0.7841
Epoch 27/100
310/310 [==============================] - 116s 375ms/step - loss: 0.3643 - accurac
y: 0.8406 - val_loss: 0.4659 - val_accuracy: 0.8184
Epoch 28/100
310/310 [==============================] - 115s 372ms/step - loss: 0.3617 - accurac
y: 0.8396 - val_loss: 0.5299 - val_accuracy: 0.7684
Epoch 29/100
310/310 [==============================] - 107s 346ms/step - loss: 0.3475 - accurac
y: 0.8508 - val_loss: 0.6086 - val_accuracy: 0.7631
Epoch 30/100
310/310 [==============================] - 155s 499ms/step - loss: 0.3686 - accurac
y: 0.8408 - val_loss: 0.4168 - val_accuracy: 0.8164
Epoch 31/100
310/310 [==============================] - 138s 446ms/step - loss: 0.3517 - accurac
y: 0.8466 - val_loss: 0.4318 - val_accuracy: 0.8216
Epoch 32/100
310/310 [==============================] - 134s 431ms/step - loss: 0.3442 - accurac
y: 0.8501 - val_loss: 0.4043 - val_accuracy: 0.8236
Epoch 33/100
310/310 [==============================] - 118s 380ms/step - loss: 0.3552 - accurac
y: 0.8440 - val_loss: 0.4683 - val_accuracy: 0.8055
Epoch 34/100
310/310 [==============================] - 121s 388ms/step - loss: 0.3442 - accurac
y: 0.8512 - val_loss: 0.4203 - val_accuracy: 0.8362
Fold 5 - Val Loss: 0.3858, Val Accuracy: 0.8265

--- Resultados de Validación Cruzada ---
Accuracy promedio: 0.8522 ± 0.0157
Loss promedio: 0.3374 ± 0.0255
```

Modelo CNN-4.

In [ ]:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import KFold

data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"

def load_images_from_directory(directory):
    images = []
    labels = []

    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)

        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)

            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)

    return np.array(images), np.array(labels)

X, y = load_images_from_directory(data_dir)
X = X / 255.0

k = 20
kf = KFold(n_splits=k, shuffle=True, random_state=42)

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

fold_accuracies = []
fold_losses = []

os.makedirs("graficas_kfold", exist_ok=True)

for fold, (train_idx, val_idx) in enumerate(kf.split(X, y)):
    print(f"\n--- Fold {fold + 1}/{k} ---")
```

```
X_train, X_val = X[train_idx], X[val_idx]
y_train, y_val = y[train_idx], y[val_idx]

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_we

history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping],
    verbose=1
)

val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=0)
fold_accuracies.append(val_accuracy)
fold_losses.append(val_loss)

print(f"Fold {fold + 1} - Val Loss: {val_loss:.4f}, Val Accuracy: {val_accuracy

model.save(f'modelo_fold_{fold+1}.h5')

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title(f'Fold {fold+1} - Accuracy')
plt.xlabel('Épocas')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title(f'Fold {fold+1} - Loss')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
```

```
    plt.savefig(f'graficas_kfold/curvas_fold_{fold+1}.png')
    plt.close()

print("\n--- Resultados de Validación Cruzada ---")
print(f"Accuracy promedio: {np.mean(fold_accuracies):.4f} ± {np.std(fold_accuracies
print(f"Loss promedio: {np.mean(fold_losses):.4f} ± {np.std(fold_losses):.4f}")

def predict_image(model, img_path):
    img = image.load_img(img_path, target_size=(100, 100))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = "CON HEMORRAGIA" if prediction[0] > 0.5 else "SIN HEMORRAGIA"

    plt.imshow(img)
    plt.title(f"Predicción: {predicted_class} ({prediction[0][0]:.4f})")
    plt.axis('off')
    plt.show()

test_image_path = "D:/PROYECTO 1/prueba/prueba.png"
predict_image(model, test_image_path)
```

```
--- Fold 1/20 ---
Epoch 1/100
368/368 [==============================] - 164s 434ms/step - loss: 0.6767 - accurac
y: 0.5978 - val_loss: 0.6472 - val_accuracy: 0.6597
Epoch 2/100
368/368 [==============================] - 128s 347ms/step - loss: 0.6451 - accurac
y: 0.6384 - val_loss: 0.6481 - val_accuracy: 0.6677
Epoch 3/100
368/368 [==============================] - 126s 341ms/step - loss: 0.6181 - accurac
y: 0.6640 - val_loss: 0.7668 - val_accuracy: 0.6242
Epoch 4/100
368/368 [==============================] - 191s 519ms/step - loss: 0.5933 - accurac
y: 0.6926 - val_loss: 0.6072 - val_accuracy: 0.6790
Epoch 5/100
368/368 [==============================] - 143s 388ms/step - loss: 0.5648 - accurac
y: 0.7138 - val_loss: 0.6566 - val_accuracy: 0.6661
Epoch 6/100
368/368 [==============================] - 141s 382ms/step - loss: 0.5385 - accurac
y: 0.7387 - val_loss: 0.7981 - val_accuracy: 0.6194
Epoch 7/100
368/368 [==============================] - 130s 353ms/step - loss: 0.5119 - accurac
y: 0.7560 - val_loss: 0.8186 - val_accuracy: 0.6306
Epoch 8/100
368/368 [==============================] - 132s 359ms/step - loss: 0.4927 - accurac
y: 0.7665 - val_loss: 0.5763 - val_accuracy: 0.7210
Epoch 9/100
368/368 [==============================] - 140s 381ms/step - loss: 0.4822 - accurac
y: 0.7825 - val_loss: 0.6649 - val_accuracy: 0.6742
Epoch 10/100
368/368 [==============================] - 131s 355ms/step - loss: 0.4657 - accurac
y: 0.7856 - val_loss: 0.6527 - val_accuracy: 0.6758
Epoch 11/100
368/368 [==============================] - 145s 395ms/step - loss: 0.4502 - accurac
y: 0.7940 - val_loss: 0.6260 - val_accuracy: 0.7274
Epoch 12/100
368/368 [==============================] - 142s 386ms/step - loss: 0.4462 - accurac
y: 0.7954 - val_loss: 0.5920 - val_accuracy: 0.7484
Epoch 13/100
368/368 [==============================] - 162s 441ms/step - loss: 0.4350 - accurac
y: 0.8016 - val_loss: 0.5546 - val_accuracy: 0.7371
Epoch 14/100
368/368 [==============================] - 170s 462ms/step - loss: 0.4279 - accurac
y: 0.8089 - val_loss: 0.6223 - val_accuracy: 0.7177
Epoch 15/100
368/368 [==============================] - 186s 506ms/step - loss: 0.4299 - accurac
y: 0.8082 - val_loss: 0.5666 - val_accuracy: 0.7565
Epoch 16/100
368/368 [==============================] - 134s 365ms/step - loss: 0.4202 - accurac
y: 0.8124 - val_loss: 0.5484 - val_accuracy: 0.7306
Epoch 17/100
368/368 [==============================] - 120s 325ms/step - loss: 0.4097 - accurac
y: 0.8142 - val_loss: 0.4857 - val_accuracy: 0.8000
Epoch 18/100
368/368 [==============================] - 118s 321ms/step - loss: 0.4101 - accurac
y: 0.8171 - val_loss: 0.5565 - val_accuracy: 0.7468
Epoch 19/100
```

```
368/368 [==============================] - 118s 320ms/step - loss: 0.4043 - accurac
y: 0.8245 - val_loss: 0.3827 - val_accuracy: 0.8452
Epoch 20/100
368/368 [==============================] - 129s 351ms/step - loss: 0.3915 - accurac
y: 0.8307 - val_loss: 0.4781 - val_accuracy: 0.7968
Epoch 21/100
368/368 [==============================] - 128s 346ms/step - loss: 0.3891 - accurac
y: 0.8280 - val_loss: 0.5305 - val_accuracy: 0.7710
Epoch 22/100
368/368 [==============================] - 128s 348ms/step - loss: 0.3828 - accurac
y: 0.8319 - val_loss: 0.5400 - val_accuracy: 0.7887
Epoch 23/100
368/368 [==============================] - 130s 353ms/step - loss: 0.3872 - accurac
y: 0.8342 - val_loss: 0.4344 - val_accuracy: 0.8016
Epoch 24/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3795 - accurac
y: 0.8351 - val_loss: 0.4709 - val_accuracy: 0.8113
Epoch 25/100
368/368 [==============================] - 125s 338ms/step - loss: 0.3669 - accurac
y: 0.8434 - val_loss: 0.4486 - val_accuracy: 0.8274
Epoch 26/100
368/368 [==============================] - 129s 350ms/step - loss: 0.3699 - accurac
y: 0.8391 - val_loss: 0.4009 - val_accuracy: 0.8242
Epoch 27/100
368/368 [==============================] - 126s 341ms/step - loss: 0.3640 - accurac
y: 0.8463 - val_loss: 0.3180 - val_accuracy: 0.8661
Epoch 28/100
368/368 [==============================] - 115s 311ms/step - loss: 0.3597 - accurac
y: 0.8449 - val_loss: 0.4868 - val_accuracy: 0.7839
Epoch 29/100
368/368 [==============================] - 122s 331ms/step - loss: 0.3602 - accurac
y: 0.8457 - val_loss: 0.3734 - val_accuracy: 0.8306
Epoch 30/100
368/368 [==============================] - 126s 343ms/step - loss: 0.3538 - accurac
y: 0.8475 - val_loss: 0.4250 - val_accuracy: 0.8161
Epoch 31/100
368/368 [==============================] - 129s 351ms/step - loss: 0.3455 - accurac
y: 0.8534 - val_loss: 0.3763 - val_accuracy: 0.8194
Epoch 32/100
368/368 [==============================] - 128s 347ms/step - loss: 0.3427 - accurac
y: 0.8541 - val_loss: 0.4442 - val_accuracy: 0.8097
Epoch 33/100
368/368 [==============================] - 128s 348ms/step - loss: 0.3436 - accurac
y: 0.8574 - val_loss: 0.4860 - val_accuracy: 0.7887
Epoch 34/100
368/368 [==============================] - 127s 344ms/step - loss: 0.3485 - accurac
y: 0.8518 - val_loss: 0.5212 - val_accuracy: 0.7887
Epoch 35/100
368/368 [==============================] - 128s 347ms/step - loss: 0.3395 - accurac
y: 0.8547 - val_loss: 0.4216 - val_accuracy: 0.8113
Epoch 36/100
368/368 [==============================] - 127s 345ms/step - loss: 0.3402 - accurac
y: 0.8559 - val_loss: 0.4308 - val_accuracy: 0.8226
Epoch 37/100
368/368 [==============================] - 127s 345ms/step - loss: 0.3360 - accurac

y: 0.8560 - val_loss: 0.5578 - val_accuracy: 0.8065
Fold 1 - Val Loss: 0.3180, Val Accuracy: 0.8661
d:\Anaconda\envs\tesis38\lib\site-packages\keras\src\engine\training.py:3000: UserWa
rning: You are saving your model as an HDF5 file via `model.save()`. This file forma
t is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')`.
  saving_api.save_model(
```

```
--- Fold 2/20 ---
Epoch 1/100
368/368 [==============================] - 135s 356ms/step - loss: 0.6686 - accurac
y: 0.6038 - val_loss: 0.6553 - val_accuracy: 0.6306
Epoch 2/100
368/368 [==============================] - 127s 344ms/step - loss: 0.6360 - accurac
y: 0.6521 - val_loss: 0.7147 - val_accuracy: 0.6306
Epoch 3/100
368/368 [==============================] - 127s 346ms/step - loss: 0.5987 - accurac
y: 0.6879 - val_loss: 0.6334 - val_accuracy: 0.6919
Epoch 4/100
368/368 [==============================] - 124s 338ms/step - loss: 0.5663 - accurac
y: 0.7133 - val_loss: 0.7169 - val_accuracy: 0.6806
Epoch 5/100
368/368 [==============================] - 126s 343ms/step - loss: 0.5371 - accurac
y: 0.7380 - val_loss: 0.6645 - val_accuracy: 0.6758
Epoch 6/100
368/368 [==============================] - 125s 339ms/step - loss: 0.5014 - accurac
y: 0.7597 - val_loss: 0.6251 - val_accuracy: 0.6871
Epoch 7/100
368/368 [==============================] - 125s 340ms/step - loss: 0.4789 - accurac
y: 0.7755 - val_loss: 0.5022 - val_accuracy: 0.7532
Epoch 8/100
368/368 [==============================] - 124s 338ms/step - loss: 0.4687 - accurac
y: 0.7764 - val_loss: 0.5070 - val_accuracy: 0.7645
Epoch 9/100
368/368 [==============================] - 127s 344ms/step - loss: 0.4521 - accurac
y: 0.7853 - val_loss: 0.4377 - val_accuracy: 0.8065
Epoch 10/100
368/368 [==============================] - 124s 337ms/step - loss: 0.4433 - accurac
y: 0.7953 - val_loss: 0.6390 - val_accuracy: 0.6774
Epoch 11/100
368/368 [==============================] - 124s 337ms/step - loss: 0.4365 - accurac
y: 0.7975 - val_loss: 0.5146 - val_accuracy: 0.7581
Epoch 12/100
368/368 [==============================] - 125s 340ms/step - loss: 0.4291 - accurac
y: 0.8012 - val_loss: 0.4394 - val_accuracy: 0.8113
Epoch 13/100
368/368 [==============================] - 126s 341ms/step - loss: 0.4233 - accurac
y: 0.8095 - val_loss: 0.6183 - val_accuracy: 0.7081
Epoch 14/100
368/368 [==============================] - 124s 338ms/step - loss: 0.4208 - accurac
y: 0.8096 - val_loss: 0.5537 - val_accuracy: 0.7500
Epoch 15/100
368/368 [==============================] - 126s 341ms/step - loss: 0.4134 - accurac
y: 0.8157 - val_loss: 0.5074 - val_accuracy: 0.7694
Epoch 16/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3972 - accurac
y: 0.8244 - val_loss: 0.4907 - val_accuracy: 0.7887
Epoch 17/100
368/368 [==============================] - 129s 349ms/step - loss: 0.4004 - accurac
y: 0.8201 - val_loss: 0.3739 - val_accuracy: 0.8306
Epoch 18/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3940 - accurac
y: 0.8242 - val_loss: 0.6026 - val_accuracy: 0.7387
Epoch 19/100
```

```
368/368 [==============================] - 125s 340ms/step - loss: 0.3951 - accurac
y: 0.8243 - val_loss: 0.4254 - val_accuracy: 0.8081
Epoch 20/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3874 - accurac
y: 0.8287 - val_loss: 0.6639 - val_accuracy: 0.7177
Epoch 21/100
368/368 [==============================] - 127s 344ms/step - loss: 0.3820 - accurac
y: 0.8335 - val_loss: 0.4265 - val_accuracy: 0.8161
Epoch 22/100
368/368 [==============================] - 160s 434ms/step - loss: 0.3810 - accurac
y: 0.8331 - val_loss: 0.4104 - val_accuracy: 0.8306
Epoch 23/100
368/368 [==============================] - 126s 341ms/step - loss: 0.3743 - accurac
y: 0.8389 - val_loss: 0.4587 - val_accuracy: 0.7903
Epoch 24/100
368/368 [==============================] - 123s 335ms/step - loss: 0.3711 - accurac
y: 0.8350 - val_loss: 0.3991 - val_accuracy: 0.8323
Epoch 25/100
368/368 [==============================] - 126s 341ms/step - loss: 0.3630 - accurac
y: 0.8439 - val_loss: 0.3732 - val_accuracy: 0.8435
Epoch 26/100
368/368 [==============================] - 125s 338ms/step - loss: 0.3565 - accurac
y: 0.8437 - val_loss: 0.4174 - val_accuracy: 0.8113
Epoch 27/100
368/368 [==============================] - 124s 336ms/step - loss: 0.3654 - accurac
y: 0.8395 - val_loss: 0.3980 - val_accuracy: 0.8306
Epoch 28/100
368/368 [==============================] - 124s 336ms/step - loss: 0.3592 - accurac
y: 0.8464 - val_loss: 0.3876 - val_accuracy: 0.8306
Epoch 29/100
368/368 [==============================] - 124s 337ms/step - loss: 0.3476 - accurac
y: 0.8487 - val_loss: 0.4147 - val_accuracy: 0.8339
Epoch 30/100
368/368 [==============================] - 126s 342ms/step - loss: 0.3542 - accurac
y: 0.8488 - val_loss: 0.4879 - val_accuracy: 0.7984
Epoch 31/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3498 - accurac
y: 0.8490 - val_loss: 0.3558 - val_accuracy: 0.8484
Epoch 32/100
368/368 [==============================] - 125s 338ms/step - loss: 0.3405 - accurac
y: 0.8530 - val_loss: 0.3891 - val_accuracy: 0.8452
Epoch 33/100
368/368 [==============================] - 124s 338ms/step - loss: 0.3457 - accurac
y: 0.8499 - val_loss: 0.4853 - val_accuracy: 0.7919
Epoch 34/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3362 - accurac
y: 0.8535 - val_loss: 0.4443 - val_accuracy: 0.8129
Epoch 35/100
368/368 [==============================] - 125s 338ms/step - loss: 0.3481 - accurac
y: 0.8451 - val_loss: 0.3768 - val_accuracy: 0.8403
Epoch 36/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3383 - accurac
y: 0.8528 - val_loss: 0.3503 - val_accuracy: 0.8468
Epoch 37/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3299 - accurac
y: 0.8579 - val_loss: 0.4232 - val_accuracy: 0.8339
```

```
Epoch 38/100
368/368 [==============================] - 125s 341ms/step - loss: 0.3308 - accurac
y: 0.8580 - val_loss: 0.4250 - val_accuracy: 0.8339
Epoch 39/100
368/368 [==============================] - 126s 341ms/step - loss: 0.3296 - accurac
y: 0.8579 - val_loss: 0.3920 - val_accuracy: 0.8581
Epoch 40/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3319 - accurac
y: 0.8586 - val_loss: 0.3691 - val_accuracy: 0.8661
Epoch 41/100
368/368 [==============================] - 124s 337ms/step - loss: 0.3301 - accurac
y: 0.8620 - val_loss: 0.3559 - val_accuracy: 0.8484
Epoch 42/100
368/368 [==============================] - 124s 337ms/step - loss: 0.3232 - accurac
y: 0.8641 - val_loss: 0.3751 - val_accuracy: 0.8548
Epoch 43/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3237 - accurac
y: 0.8642 - val_loss: 0.3766 - val_accuracy: 0.8339
Epoch 44/100
368/368 [==============================] - 123s 335ms/step - loss: 0.3157 - accurac
y: 0.8634 - val_loss: 0.3209 - val_accuracy: 0.8677
Epoch 45/100
368/368 [==============================] - 125s 338ms/step - loss: 0.3268 - accurac
y: 0.8595 - val_loss: 0.3921 - val_accuracy: 0.8435
Epoch 46/100
368/368 [==============================] - 126s 342ms/step - loss: 0.3147 - accurac
y: 0.8646 - val_loss: 0.3407 - val_accuracy: 0.8597
Epoch 47/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3157 - accurac
y: 0.8635 - val_loss: 0.3806 - val_accuracy: 0.8468
Epoch 48/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3178 - accurac
y: 0.8654 - val_loss: 0.3455 - val_accuracy: 0.8419
Epoch 49/100
368/368 [==============================] - 125s 338ms/step - loss: 0.3136 - accurac
y: 0.8676 - val_loss: 0.4191 - val_accuracy: 0.8177
Epoch 50/100
368/368 [==============================] - 126s 341ms/step - loss: 0.3174 - accurac
y: 0.8634 - val_loss: 0.4552 - val_accuracy: 0.8177
Epoch 51/100
368/368 [==============================] - 125s 339ms/step - loss: 0.3049 - accurac
y: 0.8702 - val_loss: 0.3455 - val_accuracy: 0.8532
Epoch 52/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3035 - accurac
y: 0.8706 - val_loss: 0.4800 - val_accuracy: 0.8161
Epoch 53/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3124 - accurac
y: 0.8698 - val_loss: 0.3570 - val_accuracy: 0.8500
Epoch 54/100
368/368 [==============================] - 126s 342ms/step - loss: 0.3093 - accurac
y: 0.8689 - val_loss: 0.5502 - val_accuracy: 0.7839
Fold 2 - Val Loss: 0.3209, Val Accuracy: 0.8677

--- Fold 3/20 ---
Epoch 1/100
368/368 [==============================] - 128s 340ms/step - loss: 0.6726 - accurac
```

```
y: 0.6066 - val_loss: 0.5920 - val_accuracy: 0.6919
Epoch 2/100
368/368 [==============================] - 125s 340ms/step - loss: 0.6357 - accurac
y: 0.6487 - val_loss: 0.6522 - val_accuracy: 0.6597
Epoch 3/100
368/368 [==============================] - 124s 337ms/step - loss: 0.6141 - accurac
y: 0.6725 - val_loss: 0.6753 - val_accuracy: 0.6565
Epoch 4/100
368/368 [==============================] - 124s 336ms/step - loss: 0.5885 - accurac
y: 0.6901 - val_loss: 0.9268 - val_accuracy: 0.6548
Epoch 5/100
368/368 [==============================] - 127s 344ms/step - loss: 0.5715 - accurac
y: 0.7094 - val_loss: 0.6141 - val_accuracy: 0.7065
Epoch 6/100
368/368 [==============================] - 133s 362ms/step - loss: 0.5518 - accurac
y: 0.7192 - val_loss: 0.5747 - val_accuracy: 0.7290
Epoch 7/100
368/368 [==============================] - 136s 370ms/step - loss: 0.5284 - accurac
y: 0.7357 - val_loss: 0.7535 - val_accuracy: 0.6887
Epoch 8/100
368/368 [==============================] - 137s 372ms/step - loss: 0.5002 - accurac
y: 0.7568 - val_loss: 0.7177 - val_accuracy: 0.7129
Epoch 9/100
368/368 [==============================] - 136s 369ms/step - loss: 0.4844 - accurac
y: 0.7650 - val_loss: 0.6293 - val_accuracy: 0.7468
Epoch 10/100
368/368 [==============================] - 135s 367ms/step - loss: 0.4671 - accurac
y: 0.7811 - val_loss: 0.6874 - val_accuracy: 0.7129
Epoch 11/100
368/368 [==============================] - 136s 369ms/step - loss: 0.4632 - accurac
y: 0.7855 - val_loss: 0.6849 - val_accuracy: 0.6984
Epoch 12/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4516 - accurac
y: 0.7884 - val_loss: 0.5575 - val_accuracy: 0.7355
Epoch 13/100
368/368 [==============================] - 143s 388ms/step - loss: 0.4403 - accurac
y: 0.7985 - val_loss: 0.4178 - val_accuracy: 0.8032
Epoch 14/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4346 - accurac
y: 0.8021 - val_loss: 0.5881 - val_accuracy: 0.7613
Epoch 15/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4314 - accurac
y: 0.8060 - val_loss: 0.4372 - val_accuracy: 0.8081
Epoch 16/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4203 - accurac
y: 0.8118 - val_loss: 0.5929 - val_accuracy: 0.7500
Epoch 17/100
368/368 [==============================] - 135s 368ms/step - loss: 0.4121 - accurac
y: 0.8139 - val_loss: 0.4680 - val_accuracy: 0.8161
Epoch 18/100
368/368 [==============================] - 133s 361ms/step - loss: 0.4079 - accurac
y: 0.8214 - val_loss: 0.4286 - val_accuracy: 0.8081
Epoch 19/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4048 - accurac
y: 0.8180 - val_loss: 0.4213 - val_accuracy: 0.8387
Epoch 20/100
```

```
368/368 [==============================] - 142s 385ms/step - loss: 0.4044 - accurac
y: 0.8157 - val_loss: 0.3638 - val_accuracy: 0.8468
Epoch 21/100
368/368 [==============================] - 144s 391ms/step - loss: 0.3881 - accurac
y: 0.8276 - val_loss: 0.3434 - val_accuracy: 0.8484
Epoch 22/100
368/368 [==============================] - 144s 390ms/step - loss: 0.3882 - accurac
y: 0.8287 - val_loss: 0.4318 - val_accuracy: 0.8145
Epoch 23/100
368/368 [==============================] - 143s 388ms/step - loss: 0.3900 - accurac
y: 0.8268 - val_loss: 0.4942 - val_accuracy: 0.7726
Epoch 24/100
368/368 [==============================] - 135s 366ms/step - loss: 0.3924 - accurac
y: 0.8286 - val_loss: 0.3608 - val_accuracy: 0.8403
Epoch 25/100
368/368 [==============================] - 141s 382ms/step - loss: 0.3805 - accurac
y: 0.8332 - val_loss: 0.6836 - val_accuracy: 0.7629
Epoch 26/100
368/368 [==============================] - 145s 393ms/step - loss: 0.3728 - accurac
y: 0.8353 - val_loss: 0.4962 - val_accuracy: 0.8194
Epoch 27/100
368/368 [==============================] - 151s 409ms/step - loss: 0.3722 - accurac
y: 0.8355 - val_loss: 0.4348 - val_accuracy: 0.8161
Epoch 28/100
368/368 [==============================] - 159s 433ms/step - loss: 0.3653 - accurac
y: 0.8395 - val_loss: 0.3506 - val_accuracy: 0.8516
Epoch 29/100
368/368 [==============================] - 143s 387ms/step - loss: 0.3729 - accurac
y: 0.8345 - val_loss: 0.4547 - val_accuracy: 0.8339
Epoch 30/100
368/368 [==============================] - 143s 387ms/step - loss: 0.3701 - accurac
y: 0.8404 - val_loss: 0.4333 - val_accuracy: 0.8274
Epoch 31/100
368/368 [==============================] - 149s 404ms/step - loss: 0.3574 - accurac
y: 0.8448 - val_loss: 0.4089 - val_accuracy: 0.8226
Fold 3 - Val Loss: 0.3434, Val Accuracy: 0.8484

--- Fold 4/20 ---
Epoch 1/100
368/368 [==============================] - 147s 391ms/step - loss: 0.6674 - accurac
y: 0.6036 - val_loss: 0.6523 - val_accuracy: 0.6290
Epoch 2/100
368/368 [==============================] - 146s 397ms/step - loss: 0.6250 - accurac
y: 0.6598 - val_loss: 0.8063 - val_accuracy: 0.6532
Epoch 3/100
368/368 [==============================] - 131s 357ms/step - loss: 0.5918 - accurac
y: 0.6914 - val_loss: 0.5960 - val_accuracy: 0.6984
Epoch 4/100
368/368 [==============================] - 138s 376ms/step - loss: 0.5592 - accurac
y: 0.7167 - val_loss: 0.5840 - val_accuracy: 0.7032
Epoch 5/100
368/368 [==============================] - 136s 368ms/step - loss: 0.5351 - accurac
y: 0.7336 - val_loss: 0.5922 - val_accuracy: 0.7371
Epoch 6/100
368/368 [==============================] - 137s 372ms/step - loss: 0.5128 - accurac
y: 0.7497 - val_loss: 0.5852 - val_accuracy: 0.7065
```

```
Epoch 7/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4928 - accurac
y: 0.7627 - val_loss: 0.9579 - val_accuracy: 0.6016
Epoch 8/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4780 - accurac
y: 0.7745 - val_loss: 0.4778 - val_accuracy: 0.7806
Epoch 9/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4583 - accurac
y: 0.7876 - val_loss: 0.5520 - val_accuracy: 0.7177
Epoch 10/100
368/368 [==============================] - 150s 408ms/step - loss: 0.4447 - accurac
y: 0.7938 - val_loss: 0.4202 - val_accuracy: 0.8000
Epoch 11/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4347 - accurac
y: 0.7983 - val_loss: 0.5926 - val_accuracy: 0.7145
Epoch 12/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4347 - accurac
y: 0.8067 - val_loss: 0.4369 - val_accuracy: 0.7968
Epoch 13/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4285 - accurac
y: 0.8064 - val_loss: 0.5210 - val_accuracy: 0.7371
Epoch 14/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4158 - accurac
y: 0.8147 - val_loss: 0.3762 - val_accuracy: 0.8435
Epoch 15/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4122 - accurac
y: 0.8160 - val_loss: 0.5983 - val_accuracy: 0.7419
Epoch 16/100
368/368 [==============================] - 139s 376ms/step - loss: 0.4058 - accurac
y: 0.8187 - val_loss: 0.6235 - val_accuracy: 0.7065
Epoch 17/100
368/368 [==============================] - 138s 376ms/step - loss: 0.4020 - accurac
y: 0.8155 - val_loss: 0.4277 - val_accuracy: 0.7839
Epoch 18/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3964 - accurac
y: 0.8236 - val_loss: 0.3928 - val_accuracy: 0.8210
Epoch 19/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3930 - accurac
y: 0.8238 - val_loss: 0.4066 - val_accuracy: 0.8226
Epoch 20/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3839 - accurac
y: 0.8272 - val_loss: 0.3381 - val_accuracy: 0.8597
Epoch 21/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3824 - accurac
y: 0.8285 - val_loss: 0.4084 - val_accuracy: 0.8210
Epoch 22/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3758 - accurac
y: 0.8334 - val_loss: 0.4061 - val_accuracy: 0.8113
Epoch 23/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3714 - accurac
y: 0.8377 - val_loss: 0.4496 - val_accuracy: 0.7984
Epoch 24/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3798 - accurac
y: 0.8304 - val_loss: 0.4126 - val_accuracy: 0.8161
Epoch 25/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3706 - accurac
```

```
y: 0.8344 - val_loss: 0.3447 - val_accuracy: 0.8435
Epoch 26/100
368/368 [==============================] - 140s 379ms/step - loss: 0.3709 - accurac
y: 0.8370 - val_loss: 0.3792 - val_accuracy: 0.8371
Epoch 27/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3668 - accurac
y: 0.8367 - val_loss: 0.4001 - val_accuracy: 0.8145
Epoch 28/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3637 - accurac
y: 0.8394 - val_loss: 0.3900 - val_accuracy: 0.8371
Epoch 29/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3588 - accurac
y: 0.8450 - val_loss: 0.4850 - val_accuracy: 0.7887
Epoch 30/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3521 - accurac
y: 0.8447 - val_loss: 0.4707 - val_accuracy: 0.8032
Fold 4 - Val Loss: 0.3381, Val Accuracy: 0.8597

--- Fold 5/20 ---
Epoch 1/100
368/368 [==============================] - 148s 394ms/step - loss: 0.6655 - accurac
y: 0.6089 - val_loss: 0.6948 - val_accuracy: 0.5726
Epoch 2/100
368/368 [==============================] - 143s 387ms/step - loss: 0.6299 - accurac
y: 0.6504 - val_loss: 0.6493 - val_accuracy: 0.6177
Epoch 3/100
368/368 [==============================] - 141s 382ms/step - loss: 0.5917 - accurac
y: 0.6927 - val_loss: 0.6371 - val_accuracy: 0.6774
Epoch 4/100
368/368 [==============================] - 142s 386ms/step - loss: 0.5587 - accurac
y: 0.7190 - val_loss: 0.8675 - val_accuracy: 0.5952
Epoch 5/100
368/368 [==============================] - 141s 382ms/step - loss: 0.5404 - accurac
y: 0.7300 - val_loss: 0.8676 - val_accuracy: 0.6161
Epoch 6/100
368/368 [==============================] - 142s 385ms/step - loss: 0.5068 - accurac
y: 0.7531 - val_loss: 0.7911 - val_accuracy: 0.6177
Epoch 7/100
368/368 [==============================] - 140s 379ms/step - loss: 0.4893 - accurac
y: 0.7678 - val_loss: 0.5345 - val_accuracy: 0.7484
Epoch 8/100
368/368 [==============================] - 139s 376ms/step - loss: 0.4721 - accurac
y: 0.7785 - val_loss: 0.6647 - val_accuracy: 0.6742
Epoch 9/100
368/368 [==============================] - 140s 379ms/step - loss: 0.4463 - accurac
y: 0.7898 - val_loss: 0.6446 - val_accuracy: 0.6613
Epoch 10/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4366 - accurac
y: 0.8023 - val_loss: 0.6693 - val_accuracy: 0.6984
Epoch 11/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4324 - accurac
y: 0.8022 - val_loss: 0.5278 - val_accuracy: 0.7661
Epoch 12/100
368/368 [==============================] - 140s 381ms/step - loss: 0.4126 - accurac
y: 0.8174 - val_loss: 0.6209 - val_accuracy: 0.7016
Epoch 13/100
```

```
368/368 [==============================] - 140s 379ms/step - loss: 0.4157 - accurac
y: 0.8135 - val_loss: 0.4513 - val_accuracy: 0.8000
Epoch 14/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3956 - accurac
y: 0.8270 - val_loss: 0.4342 - val_accuracy: 0.8129
Epoch 15/100
368/368 [==============================] - 140s 380ms/step - loss: 0.3943 - accurac
y: 0.8210 - val_loss: 0.4326 - val_accuracy: 0.8145
Epoch 16/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3882 - accurac
y: 0.8281 - val_loss: 0.5732 - val_accuracy: 0.7435
Epoch 17/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3785 - accurac
y: 0.8310 - val_loss: 0.5048 - val_accuracy: 0.7597
Epoch 18/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3696 - accurac
y: 0.8367 - val_loss: 0.4779 - val_accuracy: 0.7823
Epoch 19/100
368/368 [==============================] - 140s 381ms/step - loss: 0.3691 - accurac
y: 0.8381 - val_loss: 0.3875 - val_accuracy: 0.8468
Epoch 20/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3602 - accurac
y: 0.8431 - val_loss: 0.4652 - val_accuracy: 0.7984
Epoch 21/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3611 - accurac
y: 0.8452 - val_loss: 0.5148 - val_accuracy: 0.7806
Epoch 22/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3516 - accurac
y: 0.8477 - val_loss: 0.4018 - val_accuracy: 0.8290
Epoch 23/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3554 - accurac
y: 0.8455 - val_loss: 0.4648 - val_accuracy: 0.7952
Epoch 24/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3464 - accurac
y: 0.8512 - val_loss: 0.3453 - val_accuracy: 0.8387
Epoch 25/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3421 - accurac
y: 0.8547 - val_loss: 0.3594 - val_accuracy: 0.8484
Epoch 26/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3452 - accurac
y: 0.8530 - val_loss: 0.3889 - val_accuracy: 0.8387
Epoch 27/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3361 - accurac
y: 0.8576 - val_loss: 0.4812 - val_accuracy: 0.8016
Epoch 28/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3325 - accurac
y: 0.8611 - val_loss: 0.4841 - val_accuracy: 0.8097
Epoch 29/100
368/368 [==============================] - 140s 379ms/step - loss: 0.3279 - accurac
y: 0.8604 - val_loss: 0.4885 - val_accuracy: 0.8097
Epoch 30/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3295 - accurac
y: 0.8576 - val_loss: 0.3621 - val_accuracy: 0.8516
Epoch 31/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3267 - accurac
y: 0.8597 - val_loss: 0.5308 - val_accuracy: 0.8016
```

```
Epoch 32/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3289 - accurac
y: 0.8623 - val_loss: 0.6053 - val_accuracy: 0.7887
Epoch 33/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3186 - accurac
y: 0.8652 - val_loss: 0.3238 - val_accuracy: 0.8629
Epoch 34/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3205 - accurac
y: 0.8669 - val_loss: 0.6244 - val_accuracy: 0.7597
Epoch 35/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3141 - accurac
y: 0.8687 - val_loss: 0.4078 - val_accuracy: 0.8468
Epoch 36/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3206 - accurac
y: 0.8657 - val_loss: 0.4897 - val_accuracy: 0.8339
Epoch 37/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3086 - accurac
y: 0.8706 - val_loss: 0.4045 - val_accuracy: 0.8468
Epoch 38/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3172 - accurac
y: 0.8691 - val_loss: 0.4005 - val_accuracy: 0.8274
Epoch 39/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3158 - accurac
y: 0.8670 - val_loss: 0.4418 - val_accuracy: 0.8355
Epoch 40/100
368/368 [==============================] - 141s 384ms/step - loss: 0.3089 - accurac
y: 0.8702 - val_loss: 0.6410 - val_accuracy: 0.7935
Epoch 41/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3062 - accurac
y: 0.8730 - val_loss: 0.4704 - val_accuracy: 0.8355
Epoch 42/100
368/368 [==============================] - 142s 384ms/step - loss: 0.3107 - accurac
y: 0.8648 - val_loss: 0.4294 - val_accuracy: 0.8371
Epoch 43/100
368/368 [==============================] - 138s 375ms/step - loss: 0.2981 - accurac
y: 0.8751 - val_loss: 0.5276 - val_accuracy: 0.8274
Fold 5 - Val Loss: 0.3238, Val Accuracy: 0.8629

--- Fold 6/20 ---
Epoch 1/100
368/368 [==============================] - 144s 382ms/step - loss: 0.6691 - accurac
y: 0.6039 - val_loss: 0.6347 - val_accuracy: 0.6194
Epoch 2/100
368/368 [==============================] - 138s 375ms/step - loss: 0.6407 - accurac
y: 0.6490 - val_loss: 0.5844 - val_accuracy: 0.6710
Epoch 3/100
368/368 [==============================] - 139s 377ms/step - loss: 0.6244 - accurac
y: 0.6688 - val_loss: 0.5782 - val_accuracy: 0.6661
Epoch 4/100
368/368 [==============================] - 138s 375ms/step - loss: 0.6077 - accurac
y: 0.6799 - val_loss: 0.7131 - val_accuracy: 0.6452
Epoch 5/100
368/368 [==============================] - 140s 380ms/step - loss: 0.5826 - accurac
y: 0.6931 - val_loss: 0.7413 - val_accuracy: 0.6726
Epoch 6/100
368/368 [==============================] - 138s 373ms/step - loss: 0.5575 - accurac
```

```
y: 0.7163 - val_loss: 0.5099 - val_accuracy: 0.7597
Epoch 7/100
368/368 [==============================] - 138s 373ms/step - loss: 0.5316 - accurac
y: 0.7373 - val_loss: 0.6261 - val_accuracy: 0.7242
Epoch 8/100
368/368 [==============================] - 137s 372ms/step - loss: 0.5237 - accurac
y: 0.7442 - val_loss: 0.6957 - val_accuracy: 0.7032
Epoch 9/100
368/368 [==============================] - 137s 371ms/step - loss: 0.5078 - accurac
y: 0.7544 - val_loss: 0.4616 - val_accuracy: 0.7839
Epoch 10/100
368/368 [==============================] - 138s 376ms/step - loss: 0.4985 - accurac
y: 0.7540 - val_loss: 0.4808 - val_accuracy: 0.7774
Epoch 11/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4852 - accurac
y: 0.7668 - val_loss: 0.5446 - val_accuracy: 0.7403
Epoch 12/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4685 - accurac
y: 0.7814 - val_loss: 0.5233 - val_accuracy: 0.7371
Epoch 13/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4629 - accurac
y: 0.7846 - val_loss: 0.5861 - val_accuracy: 0.7274
Epoch 14/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4555 - accurac
y: 0.7846 - val_loss: 0.5023 - val_accuracy: 0.7435
Epoch 15/100
368/368 [==============================] - 138s 373ms/step - loss: 0.4455 - accurac
y: 0.7943 - val_loss: 0.4301 - val_accuracy: 0.7919
Epoch 16/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4318 - accurac
y: 0.8016 - val_loss: 0.4841 - val_accuracy: 0.7806
Epoch 17/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4348 - accurac
y: 0.7987 - val_loss: 0.3949 - val_accuracy: 0.8323
Epoch 18/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4280 - accurac
y: 0.8006 - val_loss: 0.4575 - val_accuracy: 0.7935
Epoch 19/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4264 - accurac
y: 0.8027 - val_loss: 0.4837 - val_accuracy: 0.7919
Epoch 20/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4266 - accurac
y: 0.8065 - val_loss: 0.4358 - val_accuracy: 0.8000
Epoch 21/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4184 - accurac
y: 0.8085 - val_loss: 0.4287 - val_accuracy: 0.8194
Epoch 22/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4102 - accurac
y: 0.8127 - val_loss: 0.3997 - val_accuracy: 0.8339
Epoch 23/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4057 - accurac
y: 0.8136 - val_loss: 0.5278 - val_accuracy: 0.8032
Epoch 24/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4036 - accurac
y: 0.8203 - val_loss: 0.3798 - val_accuracy: 0.8290
Epoch 25/100
```

```
368/368 [==============================] - 137s 372ms/step - loss: 0.4006 - accurac
y: 0.8248 - val_loss: 0.4495 - val_accuracy: 0.8129
Epoch 26/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3960 - accurac
y: 0.8224 - val_loss: 0.5376 - val_accuracy: 0.7790
Epoch 27/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3912 - accurac
y: 0.8247 - val_loss: 0.4937 - val_accuracy: 0.8177
Epoch 28/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3920 - accurac
y: 0.8230 - val_loss: 0.3932 - val_accuracy: 0.8452
Epoch 29/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3856 - accurac
y: 0.8282 - val_loss: 0.5267 - val_accuracy: 0.7887
Epoch 30/100
368/368 [==============================] - 138s 373ms/step - loss: 0.3852 - accurac
y: 0.8296 - val_loss: 0.4309 - val_accuracy: 0.8242
Epoch 31/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3829 - accurac
y: 0.8264 - val_loss: 0.5088 - val_accuracy: 0.7968
Epoch 32/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3758 - accurac
y: 0.8351 - val_loss: 0.4108 - val_accuracy: 0.8355
Epoch 33/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3737 - accurac
y: 0.8323 - val_loss: 0.4524 - val_accuracy: 0.8290
Epoch 34/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3770 - accurac
y: 0.8349 - val_loss: 0.4318 - val_accuracy: 0.8177
Fold 6 - Val Loss: 0.3798, Val Accuracy: 0.8290

--- Fold 7/20 ---
Epoch 1/100
368/368 [==============================] - 141s 374ms/step - loss: 0.6667 - accurac
y: 0.6133 - val_loss: 0.5947 - val_accuracy: 0.6710
Epoch 2/100
368/368 [==============================] - 137s 373ms/step - loss: 0.6348 - accurac
y: 0.6521 - val_loss: 0.5672 - val_accuracy: 0.6952
Epoch 3/100
368/368 [==============================] - 139s 376ms/step - loss: 0.6054 - accurac
y: 0.6795 - val_loss: 0.5331 - val_accuracy: 0.7274
Epoch 4/100
368/368 [==============================] - 136s 370ms/step - loss: 0.5769 - accurac
y: 0.7050 - val_loss: 0.5510 - val_accuracy: 0.7129
Epoch 5/100
368/368 [==============================] - 137s 371ms/step - loss: 0.5496 - accurac
y: 0.7251 - val_loss: 0.8662 - val_accuracy: 0.6613
Epoch 6/100
368/368 [==============================] - 137s 372ms/step - loss: 0.5239 - accurac
y: 0.7440 - val_loss: 0.5882 - val_accuracy: 0.7387
Epoch 7/100
368/368 [==============================] - 137s 373ms/step - loss: 0.5059 - accurac
y: 0.7555 - val_loss: 0.6160 - val_accuracy: 0.7129
Epoch 8/100
368/368 [==============================] - 135s 366ms/step - loss: 0.4801 - accurac
y: 0.7714 - val_loss: 0.7553 - val_accuracy: 0.6661
```

```
Epoch 9/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4763 - accurac
y: 0.7689 - val_loss: 0.6790 - val_accuracy: 0.7097
Epoch 10/100
368/368 [==============================] - 138s 376ms/step - loss: 0.4641 - accurac
y: 0.7834 - val_loss: 0.5770 - val_accuracy: 0.7516
Epoch 11/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4528 - accurac
y: 0.7891 - val_loss: 0.6520 - val_accuracy: 0.7339
Epoch 12/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4436 - accurac
y: 0.7897 - val_loss: 0.5754 - val_accuracy: 0.7613
Epoch 13/100
368/368 [==============================] - 136s 369ms/step - loss: 0.4433 - accurac
y: 0.7973 - val_loss: 0.5029 - val_accuracy: 0.7742
Epoch 14/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4345 - accurac
y: 0.7957 - val_loss: 0.6124 - val_accuracy: 0.7355
Epoch 15/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4166 - accurac
y: 0.8112 - val_loss: 0.5501 - val_accuracy: 0.7532
Epoch 16/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4191 - accurac
y: 0.8065 - val_loss: 0.4388 - val_accuracy: 0.7984
Epoch 17/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4093 - accurac
y: 0.8159 - val_loss: 0.4160 - val_accuracy: 0.8145
Epoch 18/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4149 - accurac
y: 0.8074 - val_loss: 0.4350 - val_accuracy: 0.8113
Epoch 19/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4043 - accurac
y: 0.8186 - val_loss: 0.4040 - val_accuracy: 0.8048
Epoch 20/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4019 - accurac
y: 0.8133 - val_loss: 0.5031 - val_accuracy: 0.7790
Epoch 21/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3898 - accurac
y: 0.8282 - val_loss: 0.4591 - val_accuracy: 0.7968
Epoch 22/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3954 - accurac
y: 0.8242 - val_loss: 0.4635 - val_accuracy: 0.8081
Epoch 23/100
368/368 [==============================] - 140s 381ms/step - loss: 0.3851 - accurac
y: 0.8294 - val_loss: 0.4367 - val_accuracy: 0.7919
Epoch 24/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3785 - accurac
y: 0.8309 - val_loss: 0.4784 - val_accuracy: 0.7790
Epoch 25/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3824 - accurac
y: 0.8312 - val_loss: 0.4024 - val_accuracy: 0.8194
Epoch 26/100
368/368 [==============================] - 136s 371ms/step - loss: 0.3688 - accurac
y: 0.8363 - val_loss: 0.5089 - val_accuracy: 0.7984
Epoch 27/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3695 - accurac
```

```
y: 0.8385 - val_loss: 0.4605 - val_accuracy: 0.8129
Epoch 28/100
368/368 [==============================] - 136s 368ms/step - loss: 0.3603 - accurac
y: 0.8408 - val_loss: 0.4443 - val_accuracy: 0.8242
Epoch 29/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3613 - accurac
y: 0.8368 - val_loss: 0.4578 - val_accuracy: 0.8161
Epoch 30/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3678 - accurac
y: 0.8397 - val_loss: 0.5742 - val_accuracy: 0.7887
Epoch 31/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3655 - accurac
y: 0.8370 - val_loss: 0.4416 - val_accuracy: 0.8194
Epoch 32/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3565 - accurac
y: 0.8428 - val_loss: 0.4361 - val_accuracy: 0.8032
Epoch 33/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3563 - accurac
y: 0.8437 - val_loss: 0.5210 - val_accuracy: 0.7984
Epoch 34/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3516 - accurac
y: 0.8479 - val_loss: 0.5026 - val_accuracy: 0.7887
Epoch 35/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3481 - accurac
y: 0.8492 - val_loss: 0.5050 - val_accuracy: 0.7952
Fold 7 - Val Loss: 0.4024, Val Accuracy: 0.8194

--- Fold 8/20 ---
Epoch 1/100
368/368 [==============================] - 143s 379ms/step - loss: 0.6688 - accurac
y: 0.6014 - val_loss: 0.6456 - val_accuracy: 0.6387
Epoch 2/100
368/368 [==============================] - 139s 378ms/step - loss: 0.6364 - accurac
y: 0.6550 - val_loss: 0.6537 - val_accuracy: 0.6774
Epoch 3/100
368/368 [==============================] - 136s 369ms/step - loss: 0.6048 - accurac
y: 0.6826 - val_loss: 0.6295 - val_accuracy: 0.6581
Epoch 4/100
368/368 [==============================] - 136s 370ms/step - loss: 0.5730 - accurac
y: 0.7063 - val_loss: 0.6686 - val_accuracy: 0.6903
Epoch 5/100
368/368 [==============================] - 136s 369ms/step - loss: 0.5295 - accurac
y: 0.7361 - val_loss: 0.6778 - val_accuracy: 0.7355
Epoch 6/100
368/368 [==============================] - 137s 371ms/step - loss: 0.5075 - accurac
y: 0.7523 - val_loss: 0.5774 - val_accuracy: 0.7435
Epoch 7/100
368/368 [==============================] - 136s 369ms/step - loss: 0.4851 - accurac
y: 0.7730 - val_loss: 0.5093 - val_accuracy: 0.7758
Epoch 8/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4671 - accurac
y: 0.7830 - val_loss: 0.6550 - val_accuracy: 0.7290
Epoch 9/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4513 - accurac
y: 0.7903 - val_loss: 0.4988 - val_accuracy: 0.8016
Epoch 10/100
```

```
368/368 [==============================] - 136s 369ms/step - loss: 0.4460 - accurac
y: 0.7939 - val_loss: 0.4118 - val_accuracy: 0.8323
Epoch 11/100
368/368 [==============================] - 136s 369ms/step - loss: 0.4340 - accurac
y: 0.8032 - val_loss: 0.4896 - val_accuracy: 0.7839
Epoch 12/100
368/368 [==============================] - 135s 367ms/step - loss: 0.4264 - accurac
y: 0.8049 - val_loss: 0.4173 - val_accuracy: 0.8000
Epoch 13/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4222 - accurac
y: 0.8087 - val_loss: 0.3766 - val_accuracy: 0.8419
Epoch 14/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4170 - accurac
y: 0.8108 - val_loss: 0.4220 - val_accuracy: 0.8161
Epoch 15/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4091 - accurac
y: 0.8190 - val_loss: 0.4582 - val_accuracy: 0.7952
Epoch 16/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4028 - accurac
y: 0.8200 - val_loss: 0.4058 - val_accuracy: 0.8194
Epoch 17/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3945 - accurac
y: 0.8242 - val_loss: 0.4579 - val_accuracy: 0.8242
Epoch 18/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3801 - accurac
y: 0.8361 - val_loss: 0.5114 - val_accuracy: 0.7855
Epoch 19/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3861 - accurac
y: 0.8335 - val_loss: 0.4795 - val_accuracy: 0.8161
Epoch 20/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3790 - accurac
y: 0.8340 - val_loss: 0.4023 - val_accuracy: 0.8258
Epoch 21/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3695 - accurac
y: 0.8427 - val_loss: 0.3710 - val_accuracy: 0.8403
Epoch 22/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3748 - accurac
y: 0.8349 - val_loss: 0.4716 - val_accuracy: 0.8145
Epoch 23/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3704 - accurac
y: 0.8402 - val_loss: 0.3711 - val_accuracy: 0.8597
Epoch 24/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3608 - accurac
y: 0.8429 - val_loss: 0.5728 - val_accuracy: 0.7968
Epoch 25/100
368/368 [==============================] - 135s 366ms/step - loss: 0.3616 - accurac
y: 0.8443 - val_loss: 0.4631 - val_accuracy: 0.8355
Epoch 26/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3570 - accurac
y: 0.8469 - val_loss: 0.3378 - val_accuracy: 0.8661
Epoch 27/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3493 - accurac
y: 0.8490 - val_loss: 0.5715 - val_accuracy: 0.7839
Epoch 28/100
368/368 [==============================] - 136s 368ms/step - loss: 0.3535 - accurac
y: 0.8469 - val_loss: 0.3627 - val_accuracy: 0.8661
```

```
Epoch 29/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3395 - accurac
y: 0.8530 - val_loss: 0.6192 - val_accuracy: 0.7984
Epoch 30/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3406 - accurac
y: 0.8533 - val_loss: 0.5185 - val_accuracy: 0.8290
Epoch 31/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3457 - accurac
y: 0.8511 - val_loss: 0.5072 - val_accuracy: 0.8226
Epoch 32/100
368/368 [==============================] - 136s 368ms/step - loss: 0.3325 - accurac
y: 0.8547 - val_loss: 0.5095 - val_accuracy: 0.8065
Epoch 33/100
368/368 [==============================] - 134s 364ms/step - loss: 0.3375 - accurac
y: 0.8565 - val_loss: 0.4832 - val_accuracy: 0.8226
Epoch 34/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3339 - accurac
y: 0.8546 - val_loss: 0.3390 - val_accuracy: 0.8661
Epoch 35/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3166 - accurac
y: 0.8660 - val_loss: 0.3288 - val_accuracy: 0.8726
Epoch 36/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3263 - accurac
y: 0.8625 - val_loss: 0.4240 - val_accuracy: 0.8177
Epoch 37/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3247 - accurac
y: 0.8613 - val_loss: 0.5184 - val_accuracy: 0.8323
Epoch 38/100
368/368 [==============================] - 135s 368ms/step - loss: 0.3215 - accurac
y: 0.8635 - val_loss: 0.4321 - val_accuracy: 0.8371
Epoch 39/100
368/368 [==============================] - 135s 366ms/step - loss: 0.3201 - accurac
y: 0.8664 - val_loss: 0.3707 - val_accuracy: 0.8403
Epoch 40/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3206 - accurac
y: 0.8643 - val_loss: 0.4434 - val_accuracy: 0.8355
Epoch 41/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3117 - accurac
y: 0.8660 - val_loss: 0.4515 - val_accuracy: 0.8242
Epoch 42/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3164 - accurac
y: 0.8689 - val_loss: 0.4303 - val_accuracy: 0.8435
Epoch 43/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3120 - accurac
y: 0.8694 - val_loss: 0.3747 - val_accuracy: 0.8677
Epoch 44/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3103 - accurac
y: 0.8709 - val_loss: 0.4612 - val_accuracy: 0.8403
Epoch 45/100
368/368 [==============================] - 134s 365ms/step - loss: 0.3096 - accurac
y: 0.8724 - val_loss: 0.6108 - val_accuracy: 0.8097
Fold 8 - Val Loss: 0.3288, Val Accuracy: 0.8726

--- Fold 9/20 ---
Epoch 1/100
368/368 [==============================] - 144s 383ms/step - loss: 0.6592 - accurac
```

```
y: 0.6121 - val_loss: 0.6174 - val_accuracy: 0.6500
Epoch 2/100
368/368 [==============================] - 145s 395ms/step - loss: 0.6196 - accurac
y: 0.6617 - val_loss: 0.7009 - val_accuracy: 0.6113
Epoch 3/100
368/368 [==============================] - 144s 392ms/step - loss: 0.5889 - accurac
y: 0.6901 - val_loss: 0.6987 - val_accuracy: 0.6532
Epoch 4/100
368/368 [==============================] - 142s 385ms/step - loss: 0.5639 - accurac
y: 0.7142 - val_loss: 0.8716 - val_accuracy: 0.6403
Epoch 5/100
368/368 [==============================] - 148s 402ms/step - loss: 0.5290 - accurac
y: 0.7387 - val_loss: 0.8745 - val_accuracy: 0.6742
Epoch 6/100
368/368 [==============================] - 144s 391ms/step - loss: 0.5114 - accurac
y: 0.7485 - val_loss: 0.4972 - val_accuracy: 0.7677
Epoch 7/100
368/368 [==============================] - 147s 398ms/step - loss: 0.4918 - accurac
y: 0.7663 - val_loss: 1.0693 - val_accuracy: 0.6452
Epoch 8/100
368/368 [==============================] - 144s 391ms/step - loss: 0.4789 - accurac
y: 0.7723 - val_loss: 0.6717 - val_accuracy: 0.7210
Epoch 9/100
368/368 [==============================] - 144s 392ms/step - loss: 0.4630 - accurac
y: 0.7878 - val_loss: 0.6297 - val_accuracy: 0.7145
Epoch 10/100
368/368 [==============================] - 143s 388ms/step - loss: 0.4553 - accurac
y: 0.7892 - val_loss: 0.7738 - val_accuracy: 0.6742
Epoch 11/100
368/368 [==============================] - 143s 389ms/step - loss: 0.4480 - accurac
y: 0.7925 - val_loss: 0.7298 - val_accuracy: 0.7113
Epoch 12/100
368/368 [==============================] - 145s 392ms/step - loss: 0.4380 - accurac
y: 0.8002 - val_loss: 0.6308 - val_accuracy: 0.7258
Epoch 13/100
368/368 [==============================] - 142s 386ms/step - loss: 0.4229 - accurac
y: 0.8081 - val_loss: 0.7224 - val_accuracy: 0.6984
Epoch 14/100
368/368 [==============================] - 140s 379ms/step - loss: 0.4202 - accurac
y: 0.8092 - val_loss: 0.5118 - val_accuracy: 0.7677
Epoch 15/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4193 - accurac
y: 0.8124 - val_loss: 0.5536 - val_accuracy: 0.7403
Epoch 16/100
368/368 [==============================] - 141s 384ms/step - loss: 0.4104 - accurac
y: 0.8173 - val_loss: 0.6109 - val_accuracy: 0.7500
Fold 9 - Val Loss: 0.4972, Val Accuracy: 0.7677

--- Fold 10/20 ---
Epoch 1/100
368/368 [==============================] - 145s 386ms/step - loss: 0.6641 - accurac
y: 0.6103 - val_loss: 0.5928 - val_accuracy: 0.7016
Epoch 2/100
368/368 [==============================] - 142s 385ms/step - loss: 0.6294 - accurac
y: 0.6485 - val_loss: 0.5977 - val_accuracy: 0.6952
Epoch 3/100
```

```
368/368 [==============================] - 139s 378ms/step - loss: 0.5977 - accurac
y: 0.6888 - val_loss: 0.5724 - val_accuracy: 0.7177
Epoch 4/100
368/368 [==============================] - 144s 391ms/step - loss: 0.5811 - accurac
y: 0.7034 - val_loss: 0.6468 - val_accuracy: 0.7129
Epoch 5/100
368/368 [==============================] - 145s 393ms/step - loss: 0.5527 - accurac
y: 0.7273 - val_loss: 0.6201 - val_accuracy: 0.7065
Epoch 6/100
368/368 [==============================] - 143s 388ms/step - loss: 0.5192 - accurac
y: 0.7448 - val_loss: 0.5022 - val_accuracy: 0.7532
Epoch 7/100
368/368 [==============================] - 142s 386ms/step - loss: 0.5061 - accurac
y: 0.7533 - val_loss: 0.5423 - val_accuracy: 0.7613
Epoch 8/100
368/368 [==============================] - 141s 383ms/step - loss: 0.4798 - accurac
y: 0.7716 - val_loss: 0.4327 - val_accuracy: 0.8016
Epoch 9/100
368/368 [==============================] - 144s 391ms/step - loss: 0.4665 - accurac
y: 0.7754 - val_loss: 0.4296 - val_accuracy: 0.8048
Epoch 10/100
368/368 [==============================] - 143s 389ms/step - loss: 0.4555 - accurac
y: 0.7878 - val_loss: 0.4285 - val_accuracy: 0.8113
Epoch 11/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4453 - accurac
y: 0.7950 - val_loss: 0.3813 - val_accuracy: 0.8242
Epoch 12/100
368/368 [==============================] - 142s 387ms/step - loss: 0.4311 - accurac
y: 0.7990 - val_loss: 0.3810 - val_accuracy: 0.8274
Epoch 13/100
368/368 [==============================] - 143s 387ms/step - loss: 0.4203 - accurac
y: 0.8057 - val_loss: 0.3913 - val_accuracy: 0.8161
Epoch 14/100
368/368 [==============================] - 144s 392ms/step - loss: 0.4187 - accurac
y: 0.8107 - val_loss: 0.4232 - val_accuracy: 0.8145
Epoch 15/100
368/368 [==============================] - 148s 403ms/step - loss: 0.4240 - accurac
y: 0.8073 - val_loss: 0.3925 - val_accuracy: 0.8274
Epoch 16/100
368/368 [==============================] - 146s 395ms/step - loss: 0.4119 - accurac
y: 0.8142 - val_loss: 0.4639 - val_accuracy: 0.7742
Epoch 17/100
368/368 [==============================] - 143s 387ms/step - loss: 0.4060 - accurac
y: 0.8148 - val_loss: 0.3882 - val_accuracy: 0.8274
Epoch 18/100
368/368 [==============================] - 147s 400ms/step - loss: 0.4021 - accurac
y: 0.8216 - val_loss: 0.4176 - val_accuracy: 0.8210
Epoch 19/100
368/368 [==============================] - 135s 366ms/step - loss: 0.3894 - accurac
y: 0.8297 - val_loss: 0.3628 - val_accuracy: 0.8355
Epoch 20/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3974 - accurac
y: 0.8207 - val_loss: 0.3214 - val_accuracy: 0.8565
Epoch 21/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3859 - accurac
y: 0.8276 - val_loss: 0.3997 - val_accuracy: 0.8016
```

```
Epoch 22/100
368/368 [==============================] - 152s 414ms/step - loss: 0.3857 - accurac
y: 0.8282 - val_loss: 0.3825 - val_accuracy: 0.8177
Epoch 23/100
368/368 [==============================] - 154s 420ms/step - loss: 0.3792 - accurac
y: 0.8322 - val_loss: 0.3517 - val_accuracy: 0.8403
Epoch 24/100
368/368 [==============================] - 150s 409ms/step - loss: 0.3735 - accurac
y: 0.8347 - val_loss: 0.3967 - val_accuracy: 0.8274
Epoch 25/100
368/368 [==============================] - 151s 410ms/step - loss: 0.3737 - accurac
y: 0.8370 - val_loss: 0.3675 - val_accuracy: 0.8323
Epoch 26/100
368/368 [==============================] - 149s 405ms/step - loss: 0.3614 - accurac
y: 0.8434 - val_loss: 0.3756 - val_accuracy: 0.8468
Epoch 27/100
368/368 [==============================] - 141s 383ms/step - loss: 0.3623 - accurac
y: 0.8436 - val_loss: 0.3764 - val_accuracy: 0.8194
Epoch 28/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3617 - accurac
y: 0.8419 - val_loss: 0.3453 - val_accuracy: 0.8548
Epoch 29/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3598 - accurac
y: 0.8406 - val_loss: 0.4068 - val_accuracy: 0.8274
Epoch 30/100
368/368 [==============================] - 158s 428ms/step - loss: 0.3499 - accurac
y: 0.8491 - val_loss: 0.3759 - val_accuracy: 0.8323
Fold 10 - Val Loss: 0.3214, Val Accuracy: 0.8565

--- Fold 11/20 ---
Epoch 1/100
368/368 [==============================] - 168s 447ms/step - loss: 0.6700 - accurac
y: 0.5985 - val_loss: 0.6126 - val_accuracy: 0.6806
Epoch 2/100
368/368 [==============================] - 151s 409ms/step - loss: 0.6324 - accurac
y: 0.6535 - val_loss: 0.6763 - val_accuracy: 0.6839
Epoch 3/100
368/368 [==============================] - 147s 397ms/step - loss: 0.6076 - accurac
y: 0.6763 - val_loss: 0.5618 - val_accuracy: 0.7226
Epoch 4/100
368/368 [==============================] - 143s 388ms/step - loss: 0.5700 - accurac
y: 0.7139 - val_loss: 0.6148 - val_accuracy: 0.7161
Epoch 5/100
368/368 [==============================] - 127s 344ms/step - loss: 0.5395 - accurac
y: 0.7312 - val_loss: 0.5778 - val_accuracy: 0.7355
Epoch 6/100
368/368 [==============================] - 126s 343ms/step - loss: 0.5090 - accurac
y: 0.7524 - val_loss: 0.6009 - val_accuracy: 0.7048
Epoch 7/100
368/368 [==============================] - 127s 345ms/step - loss: 0.4897 - accurac
y: 0.7689 - val_loss: 0.6212 - val_accuracy: 0.7129
Epoch 8/100
368/368 [==============================] - 133s 361ms/step - loss: 0.4742 - accurac
y: 0.7747 - val_loss: 0.5698 - val_accuracy: 0.7290
Epoch 9/100
368/368 [==============================] - 142s 386ms/step - loss: 0.4654 - accurac
```

```
y: 0.7830 - val_loss: 0.6056 - val_accuracy: 0.7145
Epoch 10/100
368/368 [==============================] - 146s 396ms/step - loss: 0.4541 - accurac
y: 0.7900 - val_loss: 0.4944 - val_accuracy: 0.7710
Epoch 11/100
368/368 [==============================] - 161s 437ms/step - loss: 0.4438 - accurac
y: 0.7947 - val_loss: 0.6023 - val_accuracy: 0.7129
Epoch 12/100
368/368 [==============================] - 151s 409ms/step - loss: 0.4463 - accurac
y: 0.7983 - val_loss: 0.4777 - val_accuracy: 0.7726
Epoch 13/100
368/368 [==============================] - 151s 409ms/step - loss: 0.4335 - accurac
y: 0.8016 - val_loss: 0.6388 - val_accuracy: 0.7371
Epoch 14/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4278 - accurac
y: 0.8043 - val_loss: 0.4848 - val_accuracy: 0.7742
Epoch 15/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4173 - accurac
y: 0.8083 - val_loss: 0.4981 - val_accuracy: 0.7661
Epoch 16/100
368/368 [==============================] - 164s 445ms/step - loss: 0.4155 - accurac
y: 0.8099 - val_loss: 0.6467 - val_accuracy: 0.7274
Epoch 17/100
368/368 [==============================] - 158s 428ms/step - loss: 0.4143 - accurac
y: 0.8136 - val_loss: 0.5773 - val_accuracy: 0.7565
Epoch 18/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4037 - accurac
y: 0.8197 - val_loss: 0.5655 - val_accuracy: 0.7532
Epoch 19/100
368/368 [==============================] - 136s 368ms/step - loss: 0.4103 - accurac
y: 0.8140 - val_loss: 0.6302 - val_accuracy: 0.7290
Epoch 20/100
368/368 [==============================] - 140s 381ms/step - loss: 0.3945 - accurac
y: 0.8258 - val_loss: 0.4558 - val_accuracy: 0.8129
Epoch 21/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4017 - accurac
y: 0.8226 - val_loss: 0.4254 - val_accuracy: 0.8177
Epoch 22/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3977 - accurac
y: 0.8214 - val_loss: 0.3864 - val_accuracy: 0.8355
Epoch 23/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3831 - accurac
y: 0.8298 - val_loss: 0.4977 - val_accuracy: 0.8016
Epoch 24/100
368/368 [==============================] - 134s 363ms/step - loss: 0.3807 - accurac
y: 0.8313 - val_loss: 0.5106 - val_accuracy: 0.7758
Epoch 25/100
368/368 [==============================] - 140s 380ms/step - loss: 0.3716 - accurac
y: 0.8358 - val_loss: 0.4806 - val_accuracy: 0.7887
Epoch 26/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3678 - accurac
y: 0.8386 - val_loss: 0.4836 - val_accuracy: 0.8016
Epoch 27/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3742 - accurac
y: 0.8390 - val_loss: 0.4700 - val_accuracy: 0.7742
Epoch 28/100
```

```
368/368 [==============================] - 138s 374ms/step - loss: 0.3636 - accurac
y: 0.8407 - val_loss: 0.6649 - val_accuracy: 0.7435
Epoch 29/100
368/368 [==============================] - 141s 384ms/step - loss: 0.3607 - accurac
y: 0.8428 - val_loss: 0.5494 - val_accuracy: 0.7661
Epoch 30/100
368/368 [==============================] - 140s 379ms/step - loss: 0.3587 - accurac
y: 0.8455 - val_loss: 0.4998 - val_accuracy: 0.8065
Epoch 31/100
368/368 [==============================] - 149s 404ms/step - loss: 0.3619 - accurac
y: 0.8399 - val_loss: 0.4704 - val_accuracy: 0.7935
Epoch 32/100
368/368 [==============================] - 147s 401ms/step - loss: 0.3560 - accurac
y: 0.8446 - val_loss: 0.4573 - val_accuracy: 0.8048
Fold 11 - Val Loss: 0.3864, Val Accuracy: 0.8355

--- Fold 12/20 ---
Epoch 1/100
368/368 [==============================] - 156s 413ms/step - loss: 0.6695 - accurac
y: 0.6014 - val_loss: 0.5811 - val_accuracy: 0.7016
Epoch 2/100
368/368 [==============================] - 149s 404ms/step - loss: 0.6398 - accurac
y: 0.6462 - val_loss: 0.5612 - val_accuracy: 0.7306
Epoch 3/100
368/368 [==============================] - 153s 415ms/step - loss: 0.6157 - accurac
y: 0.6666 - val_loss: 0.5462 - val_accuracy: 0.7484
Epoch 4/100
368/368 [==============================] - 162s 441ms/step - loss: 0.5940 - accurac
y: 0.6826 - val_loss: 0.6091 - val_accuracy: 0.7000
Epoch 5/100
368/368 [==============================] - 171s 466ms/step - loss: 0.5729 - accurac
y: 0.7010 - val_loss: 0.5031 - val_accuracy: 0.7452
Epoch 6/100
368/368 [==============================] - 142s 385ms/step - loss: 0.5532 - accurac
y: 0.7208 - val_loss: 0.6203 - val_accuracy: 0.6903
Epoch 7/100
368/368 [==============================] - 128s 348ms/step - loss: 0.5266 - accurac
y: 0.7398 - val_loss: 0.6894 - val_accuracy: 0.6903
Epoch 8/100
368/368 [==============================] - 127s 344ms/step - loss: 0.5034 - accurac
y: 0.7551 - val_loss: 0.5360 - val_accuracy: 0.7532
Epoch 9/100
368/368 [==============================] - 126s 343ms/step - loss: 0.4880 - accurac
y: 0.7669 - val_loss: 0.6133 - val_accuracy: 0.6919
Epoch 10/100
368/368 [==============================] - 126s 343ms/step - loss: 0.4636 - accurac
y: 0.7780 - val_loss: 0.4324 - val_accuracy: 0.8177
Epoch 11/100
368/368 [==============================] - 134s 363ms/step - loss: 0.4544 - accurac
y: 0.7872 - val_loss: 0.5080 - val_accuracy: 0.7581
Epoch 12/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4458 - accurac
y: 0.7960 - val_loss: 0.4595 - val_accuracy: 0.7855
Epoch 13/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4371 - accurac
y: 0.7984 - val_loss: 0.4569 - val_accuracy: 0.7935
```

```
Epoch 14/100
368/368 [==============================] - 133s 362ms/step - loss: 0.4421 - accurac
y: 0.7934 - val_loss: 0.6968 - val_accuracy: 0.7032
Epoch 15/100
368/368 [==============================] - 132s 358ms/step - loss: 0.4234 - accurac
y: 0.8076 - val_loss: 0.5708 - val_accuracy: 0.7419
Epoch 16/100
368/368 [==============================] - 128s 349ms/step - loss: 0.4143 - accurac
y: 0.8130 - val_loss: 0.4740 - val_accuracy: 0.7919
Epoch 17/100
368/368 [==============================] - 136s 369ms/step - loss: 0.4064 - accurac
y: 0.8152 - val_loss: 0.4124 - val_accuracy: 0.8306
Epoch 18/100
368/368 [==============================] - 129s 351ms/step - loss: 0.4043 - accurac
y: 0.8197 - val_loss: 0.3781 - val_accuracy: 0.8371
Epoch 19/100
368/368 [==============================] - 131s 357ms/step - loss: 0.4032 - accurac
y: 0.8199 - val_loss: 0.4799 - val_accuracy: 0.8097
Epoch 20/100
368/368 [==============================] - 131s 356ms/step - loss: 0.4011 - accurac
y: 0.8191 - val_loss: 0.4041 - val_accuracy: 0.8306
Epoch 21/100
368/368 [==============================] - 140s 380ms/step - loss: 0.3886 - accurac
y: 0.8267 - val_loss: 0.3681 - val_accuracy: 0.8484
Epoch 22/100
368/368 [==============================] - 141s 384ms/step - loss: 0.3860 - accurac
y: 0.8291 - val_loss: 0.5015 - val_accuracy: 0.7919
Epoch 23/100
368/368 [==============================] - 155s 421ms/step - loss: 0.3874 - accurac
y: 0.8264 - val_loss: 0.4757 - val_accuracy: 0.7903
Epoch 24/100
368/368 [==============================] - 150s 406ms/step - loss: 0.3775 - accurac
y: 0.8317 - val_loss: 0.4164 - val_accuracy: 0.8194
Epoch 25/100
368/368 [==============================] - 150s 408ms/step - loss: 0.3753 - accurac
y: 0.8357 - val_loss: 0.4987 - val_accuracy: 0.8016
Epoch 26/100
368/368 [==============================] - 144s 391ms/step - loss: 0.3826 - accurac
y: 0.8275 - val_loss: 0.3758 - val_accuracy: 0.8339
Epoch 27/100
368/368 [==============================] - 146s 398ms/step - loss: 0.3717 - accurac
y: 0.8358 - val_loss: 0.5290 - val_accuracy: 0.8016
Epoch 28/100
368/368 [==============================] - 141s 382ms/step - loss: 0.3689 - accurac
y: 0.8385 - val_loss: 0.4242 - val_accuracy: 0.8210
Epoch 29/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3625 - accurac
y: 0.8427 - val_loss: 0.4461 - val_accuracy: 0.8129
Epoch 30/100
368/368 [==============================] - 146s 395ms/step - loss: 0.3663 - accurac
y: 0.8372 - val_loss: 0.4219 - val_accuracy: 0.8081
Epoch 31/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3600 - accurac
y: 0.8453 - val_loss: 0.3855 - val_accuracy: 0.8387
Fold 12 - Val Loss: 0.3681, Val Accuracy: 0.8484
```

```
--- Fold 13/20 ---
Epoch 1/100
368/368 [==============================] - 151s 400ms/step - loss: 0.6561 - accurac
y: 0.6230 - val_loss: 0.6177 - val_accuracy: 0.6494
Epoch 2/100
368/368 [==============================] - 146s 397ms/step - loss: 0.6130 - accurac
y: 0.6749 - val_loss: 0.6249 - val_accuracy: 0.6640
Epoch 3/100
368/368 [==============================] - 150s 406ms/step - loss: 0.5736 - accurac
y: 0.7053 - val_loss: 1.1950 - val_accuracy: 0.6042
Epoch 4/100
368/368 [==============================] - 144s 390ms/step - loss: 0.5431 - accurac
y: 0.7340 - val_loss: 0.6632 - val_accuracy: 0.6721
Epoch 5/100
368/368 [==============================] - 151s 409ms/step - loss: 0.5218 - accurac
y: 0.7441 - val_loss: 0.8537 - val_accuracy: 0.6139
Epoch 6/100
368/368 [==============================] - 152s 413ms/step - loss: 0.4911 - accurac
y: 0.7659 - val_loss: 0.8250 - val_accuracy: 0.6446
Epoch 7/100
368/368 [==============================] - 150s 405ms/step - loss: 0.4770 - accurac
y: 0.7721 - val_loss: 0.6240 - val_accuracy: 0.7205
Epoch 8/100
368/368 [==============================] - 159s 433ms/step - loss: 0.4692 - accurac
y: 0.7797 - val_loss: 0.5311 - val_accuracy: 0.7593
Epoch 9/100
368/368 [==============================] - 136s 370ms/step - loss: 0.4475 - accurac
y: 0.7894 - val_loss: 0.5348 - val_accuracy: 0.7658
Epoch 10/100
368/368 [==============================] - 149s 404ms/step - loss: 0.4415 - accurac
y: 0.7996 - val_loss: 0.5793 - val_accuracy: 0.7447
Epoch 11/100
368/368 [==============================] - 160s 436ms/step - loss: 0.4321 - accurac
y: 0.8035 - val_loss: 0.5652 - val_accuracy: 0.7577
Epoch 12/100
368/368 [==============================] - 160s 433ms/step - loss: 0.4111 - accurac
y: 0.8107 - val_loss: 0.4579 - val_accuracy: 0.7884
Epoch 13/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4141 - accurac
y: 0.8176 - val_loss: 0.5227 - val_accuracy: 0.7593
Epoch 14/100
368/368 [==============================] - 141s 383ms/step - loss: 0.4128 - accurac
y: 0.8133 - val_loss: 0.5094 - val_accuracy: 0.7706
Epoch 15/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3941 - accurac
y: 0.8237 - val_loss: 0.4582 - val_accuracy: 0.7819
Epoch 16/100
368/368 [==============================] - 144s 392ms/step - loss: 0.3959 - accurac
y: 0.8221 - val_loss: 0.5335 - val_accuracy: 0.7641
Epoch 17/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3891 - accurac
y: 0.8237 - val_loss: 0.5940 - val_accuracy: 0.7658
Epoch 18/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3884 - accurac
y: 0.8243 - val_loss: 0.5263 - val_accuracy: 0.7754
Epoch 19/100
```

```
368/368 [==============================] - 136s 369ms/step - loss: 0.3779 - accurac
y: 0.8357 - val_loss: 0.4754 - val_accuracy: 0.7997
Epoch 20/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3716 - accurac
y: 0.8377 - val_loss: 0.5056 - val_accuracy: 0.7835
Epoch 21/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3662 - accurac
y: 0.8378 - val_loss: 0.5270 - val_accuracy: 0.7900
Epoch 22/100
368/368 [==============================] - 140s 380ms/step - loss: 0.3608 - accurac
y: 0.8424 - val_loss: 0.5900 - val_accuracy: 0.7544
Fold 13 - Val Loss: 0.4579, Val Accuracy: 0.7884

--- Fold 14/20 ---
Epoch 1/100
368/368 [==============================] - 141s 375ms/step - loss: 0.6644 - accurac
y: 0.6136 - val_loss: 0.6247 - val_accuracy: 0.6365
Epoch 2/100
368/368 [==============================] - 138s 375ms/step - loss: 0.6222 - accurac
y: 0.6612 - val_loss: 0.6806 - val_accuracy: 0.6624
Epoch 3/100
368/368 [==============================] - 138s 376ms/step - loss: 0.5945 - accurac
y: 0.6875 - val_loss: 0.7023 - val_accuracy: 0.6414
Epoch 4/100
368/368 [==============================] - 138s 375ms/step - loss: 0.5601 - accurac
y: 0.7143 - val_loss: 0.6989 - val_accuracy: 0.6688
Epoch 5/100
368/368 [==============================] - 139s 376ms/step - loss: 0.5270 - accurac
y: 0.7395 - val_loss: 0.6029 - val_accuracy: 0.6801
Epoch 6/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4999 - accurac
y: 0.7617 - val_loss: 0.6818 - val_accuracy: 0.6947
Epoch 7/100
368/368 [==============================] - 142s 385ms/step - loss: 0.4900 - accurac
y: 0.7657 - val_loss: 0.6216 - val_accuracy: 0.6801
Epoch 8/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4686 - accurac
y: 0.7814 - val_loss: 0.7874 - val_accuracy: 0.6575
Epoch 9/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4547 - accurac
y: 0.7863 - val_loss: 0.5311 - val_accuracy: 0.7593
Epoch 10/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4392 - accurac
y: 0.7962 - val_loss: 0.5352 - val_accuracy: 0.7932
Epoch 11/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4352 - accurac
y: 0.7990 - val_loss: 0.5132 - val_accuracy: 0.7415
Epoch 12/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4130 - accurac
y: 0.8081 - val_loss: 0.4813 - val_accuracy: 0.8078
Epoch 13/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4067 - accurac
y: 0.8163 - val_loss: 0.4686 - val_accuracy: 0.7835
Epoch 14/100
368/368 [==============================] - 140s 380ms/step - loss: 0.4093 - accurac
y: 0.8160 - val_loss: 0.4560 - val_accuracy: 0.7900
```

```
Epoch 15/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4039 - accurac
y: 0.8170 - val_loss: 0.5108 - val_accuracy: 0.7803
Epoch 16/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3881 - accurac
y: 0.8259 - val_loss: 0.3830 - val_accuracy: 0.8110
Epoch 17/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3869 - accurac
y: 0.8298 - val_loss: 0.3602 - val_accuracy: 0.8352
Epoch 18/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3899 - accurac
y: 0.8276 - val_loss: 0.3862 - val_accuracy: 0.8191
Epoch 19/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3835 - accurac
y: 0.8300 - val_loss: 0.6122 - val_accuracy: 0.7415
Epoch 20/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3804 - accurac
y: 0.8312 - val_loss: 0.4996 - val_accuracy: 0.7851
Epoch 21/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3630 - accurac
y: 0.8402 - val_loss: 0.5548 - val_accuracy: 0.7658
Epoch 22/100
368/368 [==============================] - 136s 370ms/step - loss: 0.3592 - accurac
y: 0.8434 - val_loss: 0.4619 - val_accuracy: 0.7690
Epoch 23/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3655 - accurac
y: 0.8388 - val_loss: 0.3830 - val_accuracy: 0.8271
Epoch 24/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3518 - accurac
y: 0.8431 - val_loss: 0.6411 - val_accuracy: 0.7641
Epoch 25/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3550 - accurac
y: 0.8452 - val_loss: 0.6835 - val_accuracy: 0.7415
Epoch 26/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3521 - accurac
y: 0.8466 - val_loss: 0.4921 - val_accuracy: 0.7916
Epoch 27/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3389 - accurac
y: 0.8514 - val_loss: 0.5511 - val_accuracy: 0.7819
Fold 14 - Val Loss: 0.3602, Val Accuracy: 0.8352

--- Fold 15/20 ---
Epoch 1/100
368/368 [==============================] - 163s 434ms/step - loss: 0.6598 - accurac
y: 0.6178 - val_loss: 0.6225 - val_accuracy: 0.6624
Epoch 2/100
368/368 [==============================] - 147s 400ms/step - loss: 0.6224 - accurac
y: 0.6647 - val_loss: 0.6225 - val_accuracy: 0.6672
Epoch 3/100
368/368 [==============================] - 123s 335ms/step - loss: 0.5992 - accurac
y: 0.6795 - val_loss: 0.6778 - val_accuracy: 0.6914
Epoch 4/100
368/368 [==============================] - 139s 378ms/step - loss: 0.5702 - accurac
y: 0.7032 - val_loss: 0.5949 - val_accuracy: 0.7044
Epoch 5/100
368/368 [==============================] - 133s 361ms/step - loss: 0.5507 - accurac
```

```
y: 0.7227 - val_loss: 0.7549 - val_accuracy: 0.6769
Epoch 6/100
368/368 [==============================] - 138s 374ms/step - loss: 0.5316 - accurac
y: 0.7389 - val_loss: 0.6325 - val_accuracy: 0.7351
Epoch 7/100
368/368 [==============================] - 140s 380ms/step - loss: 0.5093 - accurac
y: 0.7514 - val_loss: 0.5848 - val_accuracy: 0.7011
Epoch 8/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4902 - accurac
y: 0.7653 - val_loss: 0.5827 - val_accuracy: 0.7464
Epoch 9/100
368/368 [==============================] - 140s 380ms/step - loss: 0.4698 - accurac
y: 0.7775 - val_loss: 0.6713 - val_accuracy: 0.7383
Epoch 10/100
368/368 [==============================] - 129s 350ms/step - loss: 0.4628 - accurac
y: 0.7826 - val_loss: 0.5009 - val_accuracy: 0.7738
Epoch 11/100
368/368 [==============================] - 137s 373ms/step - loss: 0.4514 - accurac
y: 0.7903 - val_loss: 0.6163 - val_accuracy: 0.7221
Epoch 12/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4374 - accurac
y: 0.7990 - val_loss: 0.4421 - val_accuracy: 0.7997
Epoch 13/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4309 - accurac
y: 0.8004 - val_loss: 0.4715 - val_accuracy: 0.7900
Epoch 14/100
368/368 [==============================] - 138s 374ms/step - loss: 0.4198 - accurac
y: 0.8107 - val_loss: 0.6159 - val_accuracy: 0.7415
Epoch 15/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4179 - accurac
y: 0.8084 - val_loss: 0.4367 - val_accuracy: 0.8029
Epoch 16/100
368/368 [==============================] - 137s 372ms/step - loss: 0.4058 - accurac
y: 0.8159 - val_loss: 0.5025 - val_accuracy: 0.7932
Epoch 17/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4065 - accurac
y: 0.8136 - val_loss: 0.4186 - val_accuracy: 0.8191
Epoch 18/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3936 - accurac
y: 0.8221 - val_loss: 0.6812 - val_accuracy: 0.7593
Epoch 19/100
368/368 [==============================] - 137s 372ms/step - loss: 0.3943 - accurac
y: 0.8248 - val_loss: 0.4517 - val_accuracy: 0.8094
Epoch 20/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3827 - accurac
y: 0.8277 - val_loss: 0.4903 - val_accuracy: 0.7787
Epoch 21/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3795 - accurac
y: 0.8321 - val_loss: 0.4399 - val_accuracy: 0.8045
Epoch 22/100
368/368 [==============================] - 137s 371ms/step - loss: 0.3775 - accurac
y: 0.8341 - val_loss: 0.4221 - val_accuracy: 0.8304
Epoch 23/100
368/368 [==============================] - 136s 371ms/step - loss: 0.3780 - accurac
y: 0.8309 - val_loss: 0.4136 - val_accuracy: 0.8061
Epoch 24/100
```

```
368/368 [==============================] - 136s 369ms/step - loss: 0.3699 - accurac
y: 0.8329 - val_loss: 0.4200 - val_accuracy: 0.8255
Epoch 25/100
368/368 [==============================] - 144s 391ms/step - loss: 0.3640 - accurac
y: 0.8382 - val_loss: 0.4187 - val_accuracy: 0.8045
Epoch 26/100
368/368 [==============================] - 137s 373ms/step - loss: 0.3636 - accurac
y: 0.8410 - val_loss: 0.4326 - val_accuracy: 0.8174
Epoch 27/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3612 - accurac
y: 0.8392 - val_loss: 0.4773 - val_accuracy: 0.8207
Epoch 28/100
368/368 [==============================] - 125s 340ms/step - loss: 0.3534 - accurac
y: 0.8441 - val_loss: 0.4597 - val_accuracy: 0.8110
Epoch 29/100
368/368 [==============================] - 119s 324ms/step - loss: 0.3510 - accurac
y: 0.8465 - val_loss: 0.3862 - val_accuracy: 0.8465
Epoch 30/100
368/368 [==============================] - 117s 318ms/step - loss: 0.3419 - accurac
y: 0.8450 - val_loss: 0.4056 - val_accuracy: 0.8223
Epoch 31/100
368/368 [==============================] - 131s 356ms/step - loss: 0.3512 - accurac
y: 0.8446 - val_loss: 0.3549 - val_accuracy: 0.8401
Epoch 32/100
368/368 [==============================] - 121s 330ms/step - loss: 0.3507 - accurac
y: 0.8481 - val_loss: 0.3925 - val_accuracy: 0.8336
Epoch 33/100
368/368 [==============================] - 122s 331ms/step - loss: 0.3368 - accurac
y: 0.8483 - val_loss: 0.4331 - val_accuracy: 0.8191
Epoch 34/100
368/368 [==============================] - 123s 335ms/step - loss: 0.3561 - accurac
y: 0.8442 - val_loss: 0.3838 - val_accuracy: 0.8433
Epoch 35/100
368/368 [==============================] - 121s 328ms/step - loss: 0.3315 - accurac
y: 0.8508 - val_loss: 0.3505 - val_accuracy: 0.8449
Epoch 36/100
368/368 [==============================] - 133s 362ms/step - loss: 0.3424 - accurac
y: 0.8507 - val_loss: 0.6653 - val_accuracy: 0.7803
Epoch 37/100
368/368 [==============================] - 135s 366ms/step - loss: 0.3355 - accurac
y: 0.8549 - val_loss: 0.3905 - val_accuracy: 0.8239
Epoch 38/100
368/368 [==============================] - 134s 365ms/step - loss: 0.3307 - accurac
y: 0.8534 - val_loss: 0.4292 - val_accuracy: 0.8207
Epoch 39/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3432 - accurac
y: 0.8517 - val_loss: 0.5223 - val_accuracy: 0.7868
Epoch 40/100
368/368 [==============================] - 144s 391ms/step - loss: 0.3296 - accurac
y: 0.8555 - val_loss: 0.3930 - val_accuracy: 0.8481
Epoch 41/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3256 - accurac
y: 0.8598 - val_loss: 0.5573 - val_accuracy: 0.8061
Epoch 42/100
368/368 [==============================] - 147s 400ms/step - loss: 0.3269 - accurac
y: 0.8579 - val_loss: 0.3690 - val_accuracy: 0.8465
```

```
Epoch 43/100
368/368 [==============================] - 148s 401ms/step - loss: 0.3263 - accurac
y: 0.8577 - val_loss: 0.3416 - val_accuracy: 0.8465
Epoch 44/100
368/368 [==============================] - 157s 428ms/step - loss: 0.3228 - accurac
y: 0.8610 - val_loss: 0.4310 - val_accuracy: 0.8304
Epoch 45/100
368/368 [==============================] - 149s 404ms/step - loss: 0.3121 - accurac
y: 0.8643 - val_loss: 0.4501 - val_accuracy: 0.8304
Epoch 46/100
368/368 [==============================] - 135s 367ms/step - loss: 0.3224 - accurac
y: 0.8587 - val_loss: 0.3860 - val_accuracy: 0.8498
Epoch 47/100
368/368 [==============================] - 164s 445ms/step - loss: 0.3117 - accurac
y: 0.8658 - val_loss: 0.3915 - val_accuracy: 0.8320
Epoch 48/100
368/368 [==============================] - 155s 422ms/step - loss: 0.3207 - accurac
y: 0.8616 - val_loss: 0.3914 - val_accuracy: 0.8417
Epoch 49/100
368/368 [==============================] - 145s 394ms/step - loss: 0.3134 - accurac
y: 0.8664 - val_loss: 0.3617 - val_accuracy: 0.8498
Epoch 50/100
368/368 [==============================] - 149s 405ms/step - loss: 0.3127 - accurac
y: 0.8643 - val_loss: 0.3013 - val_accuracy: 0.8788
Epoch 51/100
368/368 [==============================] - 146s 397ms/step - loss: 0.3090 - accurac
y: 0.8665 - val_loss: 0.3040 - val_accuracy: 0.8805
Epoch 52/100
368/368 [==============================] - 145s 395ms/step - loss: 0.3023 - accurac
y: 0.8716 - val_loss: 0.4248 - val_accuracy: 0.8481
Epoch 53/100
368/368 [==============================] - 146s 395ms/step - loss: 0.3076 - accurac
y: 0.8700 - val_loss: 0.3582 - val_accuracy: 0.8708
Epoch 54/100
368/368 [==============================] - 145s 393ms/step - loss: 0.3061 - accurac
y: 0.8666 - val_loss: 0.4347 - val_accuracy: 0.8304
Epoch 55/100
368/368 [==============================] - 141s 382ms/step - loss: 0.2996 - accurac
y: 0.8685 - val_loss: 0.3597 - val_accuracy: 0.8595
Epoch 56/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3003 - accurac
y: 0.8701 - val_loss: 0.4120 - val_accuracy: 0.8498
Epoch 57/100
368/368 [==============================] - 138s 376ms/step - loss: 0.2978 - accurac
y: 0.8701 - val_loss: 0.3938 - val_accuracy: 0.8336
Epoch 58/100
368/368 [==============================] - 135s 368ms/step - loss: 0.2913 - accurac
y: 0.8745 - val_loss: 0.4039 - val_accuracy: 0.8675
Epoch 59/100
368/368 [==============================] - 136s 369ms/step - loss: 0.3033 - accurac
y: 0.8703 - val_loss: 0.2651 - val_accuracy: 0.9015
Epoch 60/100
368/368 [==============================] - 135s 367ms/step - loss: 0.2876 - accurac
y: 0.8757 - val_loss: 0.3568 - val_accuracy: 0.8691
Epoch 61/100
368/368 [==============================] - 137s 373ms/step - loss: 0.2896 - accurac
```

```
y: 0.8754 - val_loss: 0.4476 - val_accuracy: 0.8465
Epoch 62/100
368/368 [==============================] - 135s 365ms/step - loss: 0.2848 - accurac
y: 0.8805 - val_loss: 0.5149 - val_accuracy: 0.8336
Epoch 63/100
368/368 [==============================] - 135s 366ms/step - loss: 0.2875 - accurac
y: 0.8789 - val_loss: 0.3376 - val_accuracy: 0.8821
Epoch 64/100
368/368 [==============================] - 136s 370ms/step - loss: 0.2871 - accurac
y: 0.8784 - val_loss: 0.3069 - val_accuracy: 0.8853
Epoch 65/100
368/368 [==============================] - 137s 373ms/step - loss: 0.2865 - accurac
y: 0.8814 - val_loss: 0.3768 - val_accuracy: 0.8740
Epoch 66/100
368/368 [==============================] - 154s 418ms/step - loss: 0.2813 - accurac
y: 0.8822 - val_loss: 0.4237 - val_accuracy: 0.8514
Epoch 67/100
368/368 [==============================] - 140s 379ms/step - loss: 0.2903 - accurac
y: 0.8784 - val_loss: 0.3239 - val_accuracy: 0.8788
Epoch 68/100
368/368 [==============================] - 137s 373ms/step - loss: 0.2899 - accurac
y: 0.8755 - val_loss: 0.3737 - val_accuracy: 0.8756
Epoch 69/100
368/368 [==============================] - 148s 401ms/step - loss: 0.2788 - accurac
y: 0.8813 - val_loss: 0.4312 - val_accuracy: 0.8595
Fold 15 - Val Loss: 0.2651, Val Accuracy: 0.9015

--- Fold 16/20 ---
Epoch 1/100
368/368 [==============================] - 147s 387ms/step - loss: 0.6646 - accurac
y: 0.6057 - val_loss: 0.6107 - val_accuracy: 0.6801
Epoch 2/100
368/368 [==============================] - 146s 395ms/step - loss: 0.6325 - accurac
y: 0.6525 - val_loss: 0.5557 - val_accuracy: 0.6931
Epoch 3/100
368/368 [==============================] - 142s 386ms/step - loss: 0.6020 - accurac
y: 0.6797 - val_loss: 0.5392 - val_accuracy: 0.7092
Epoch 4/100
368/368 [==============================] - 133s 362ms/step - loss: 0.5768 - accurac
y: 0.7031 - val_loss: 0.5048 - val_accuracy: 0.7270
Epoch 5/100
368/368 [==============================] - 142s 386ms/step - loss: 0.5360 - accurac
y: 0.7361 - val_loss: 0.4847 - val_accuracy: 0.7480
Epoch 6/100
368/368 [==============================] - 139s 376ms/step - loss: 0.5108 - accurac
y: 0.7521 - val_loss: 0.5761 - val_accuracy: 0.7221
Epoch 7/100
368/368 [==============================] - 132s 358ms/step - loss: 0.4967 - accurac
y: 0.7631 - val_loss: 0.6063 - val_accuracy: 0.7027
Epoch 8/100
368/368 [==============================] - 135s 368ms/step - loss: 0.4798 - accurac
y: 0.7715 - val_loss: 0.4296 - val_accuracy: 0.8029
Epoch 9/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4617 - accurac
y: 0.7812 - val_loss: 0.4544 - val_accuracy: 0.7884
Epoch 10/100
```

```
368/368 [==============================] - 134s 365ms/step - loss: 0.4601 - accurac
y: 0.7862 - val_loss: 0.5359 - val_accuracy: 0.7415
Epoch 11/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4354 - accurac
y: 0.7984 - val_loss: 0.4699 - val_accuracy: 0.7787
Epoch 12/100
368/368 [==============================] - 137s 371ms/step - loss: 0.4269 - accurac
y: 0.8060 - val_loss: 0.7263 - val_accuracy: 0.7173
Epoch 13/100
368/368 [==============================] - 149s 405ms/step - loss: 0.4261 - accurac
y: 0.8024 - val_loss: 0.4488 - val_accuracy: 0.7997
Epoch 14/100
368/368 [==============================] - 132s 359ms/step - loss: 0.4098 - accurac
y: 0.8134 - val_loss: 0.4187 - val_accuracy: 0.8045
Epoch 15/100
368/368 [==============================] - 133s 360ms/step - loss: 0.4123 - accurac
y: 0.8133 - val_loss: 0.4120 - val_accuracy: 0.8207
Epoch 16/100
368/368 [==============================] - 128s 347ms/step - loss: 0.3975 - accurac
y: 0.8239 - val_loss: 0.5207 - val_accuracy: 0.7722
Epoch 17/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3957 - accurac
y: 0.8203 - val_loss: 0.6121 - val_accuracy: 0.7658
Epoch 18/100
368/368 [==============================] - 145s 393ms/step - loss: 0.3956 - accurac
y: 0.8277 - val_loss: 0.4115 - val_accuracy: 0.8320
Epoch 19/100
368/368 [==============================] - 145s 395ms/step - loss: 0.3885 - accurac
y: 0.8256 - val_loss: 0.5902 - val_accuracy: 0.7577
Epoch 20/100
368/368 [==============================] - 141s 383ms/step - loss: 0.3834 - accurac
y: 0.8313 - val_loss: 0.4357 - val_accuracy: 0.8191
Epoch 21/100
368/368 [==============================] - 141s 383ms/step - loss: 0.3773 - accurac
y: 0.8342 - val_loss: 0.4342 - val_accuracy: 0.8320
Epoch 22/100
368/368 [==============================] - 140s 381ms/step - loss: 0.3715 - accurac
y: 0.8409 - val_loss: 0.4930 - val_accuracy: 0.7964
Epoch 23/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3730 - accurac
y: 0.8394 - val_loss: 0.4807 - val_accuracy: 0.8110
Epoch 24/100
368/368 [==============================] - 142s 387ms/step - loss: 0.3671 - accurac
y: 0.8388 - val_loss: 0.6083 - val_accuracy: 0.7641
Epoch 25/100
368/368 [==============================] - 144s 390ms/step - loss: 0.3584 - accurac
y: 0.8424 - val_loss: 0.5816 - val_accuracy: 0.7819
Epoch 26/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3576 - accurac
y: 0.8442 - val_loss: 0.5502 - val_accuracy: 0.7981
Epoch 27/100
368/368 [==============================] - 140s 381ms/step - loss: 0.3569 - accurac
y: 0.8435 - val_loss: 0.5347 - val_accuracy: 0.8029
Epoch 28/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3565 - accurac
y: 0.8454 - val_loss: 0.6983 - val_accuracy: 0.7851
```

```
Fold 16 - Val Loss: 0.4115, Val Accuracy: 0.8320

--- Fold 17/20 ---
Epoch 1/100
368/368 [==============================] - 179s 468ms/step - loss: 0.6684 - accurac
y: 0.6027 - val_loss: 0.5850 - val_accuracy: 0.6931
Epoch 2/100
368/368 [==============================] - 171s 464ms/step - loss: 0.6367 - accurac
y: 0.6487 - val_loss: 0.5821 - val_accuracy: 0.7044
Epoch 3/100
368/368 [==============================] - 173s 468ms/step - loss: 0.6103 - accurac
y: 0.6739 - val_loss: 0.5786 - val_accuracy: 0.7044
Epoch 4/100
368/368 [==============================] - 179s 487ms/step - loss: 0.5862 - accurac
y: 0.6929 - val_loss: 0.6230 - val_accuracy: 0.6898
Epoch 5/100
368/368 [==============================] - 178s 483ms/step - loss: 0.5641 - accurac
y: 0.7126 - val_loss: 0.5898 - val_accuracy: 0.7318
Epoch 6/100
368/368 [==============================] - 172s 467ms/step - loss: 0.5420 - accurac
y: 0.7315 - val_loss: 0.5030 - val_accuracy: 0.7641
Epoch 7/100
368/368 [==============================] - 174s 472ms/step - loss: 0.5170 - accurac
y: 0.7477 - val_loss: 0.5918 - val_accuracy: 0.7092
Epoch 8/100
368/368 [==============================] - 174s 473ms/step - loss: 0.4906 - accurac
y: 0.7668 - val_loss: 0.5542 - val_accuracy: 0.7528
Epoch 9/100
368/368 [==============================] - 172s 466ms/step - loss: 0.4754 - accurac
y: 0.7768 - val_loss: 0.4606 - val_accuracy: 0.7722
Epoch 10/100
368/368 [==============================] - 170s 460ms/step - loss: 0.4615 - accurac
y: 0.7853 - val_loss: 0.4530 - val_accuracy: 0.7754
Epoch 11/100
368/368 [==============================] - 185s 501ms/step - loss: 0.4508 - accurac
y: 0.7924 - val_loss: 0.3975 - val_accuracy: 0.8320
Epoch 12/100
368/368 [==============================] - 161s 438ms/step - loss: 0.4430 - accurac
y: 0.7963 - val_loss: 0.4554 - val_accuracy: 0.7787
Epoch 13/100
368/368 [==============================] - 165s 449ms/step - loss: 0.4337 - accurac
y: 0.8017 - val_loss: 0.4612 - val_accuracy: 0.7787
Epoch 14/100
368/368 [==============================] - 172s 467ms/step - loss: 0.4318 - accurac
y: 0.8059 - val_loss: 0.4197 - val_accuracy: 0.8207
Epoch 15/100
368/368 [==============================] - 175s 474ms/step - loss: 0.4239 - accurac
y: 0.8079 - val_loss: 0.4511 - val_accuracy: 0.7948
Epoch 16/100
368/368 [==============================] - 174s 472ms/step - loss: 0.4132 - accurac
y: 0.8147 - val_loss: 0.4658 - val_accuracy: 0.7641
Epoch 17/100
368/368 [==============================] - 174s 473ms/step - loss: 0.4032 - accurac
y: 0.8209 - val_loss: 0.3995 - val_accuracy: 0.8174
Epoch 18/100
368/368 [==============================] - 171s 464ms/step - loss: 0.4044 - accurac
```

```
y: 0.8181 - val_loss: 0.3993 - val_accuracy: 0.8158
Epoch 19/100
368/368 [==============================] - 172s 468ms/step - loss: 0.4045 - accurac
y: 0.8232 - val_loss: 0.3359 - val_accuracy: 0.8643
Epoch 20/100
368/368 [==============================] - 170s 461ms/step - loss: 0.3902 - accurac
y: 0.8265 - val_loss: 0.4412 - val_accuracy: 0.8110
Epoch 21/100
368/368 [==============================] - 165s 449ms/step - loss: 0.3896 - accurac
y: 0.8271 - val_loss: 0.4374 - val_accuracy: 0.8110
Epoch 22/100
368/368 [==============================] - 165s 447ms/step - loss: 0.3904 - accurac
y: 0.8236 - val_loss: 0.5542 - val_accuracy: 0.7674
Epoch 23/100
368/368 [==============================] - 167s 454ms/step - loss: 0.3786 - accurac
y: 0.8327 - val_loss: 0.5397 - val_accuracy: 0.7754
Epoch 24/100
368/368 [==============================] - 165s 448ms/step - loss: 0.3680 - accurac
y: 0.8422 - val_loss: 0.3661 - val_accuracy: 0.8320
Epoch 25/100
368/368 [==============================] - 165s 447ms/step - loss: 0.3686 - accurac
y: 0.8414 - val_loss: 0.3939 - val_accuracy: 0.8191
Epoch 26/100
368/368 [==============================] - 164s 446ms/step - loss: 0.3718 - accurac
y: 0.8393 - val_loss: 0.4247 - val_accuracy: 0.8207
Epoch 27/100
368/368 [==============================] - 164s 446ms/step - loss: 0.3695 - accurac
y: 0.8405 - val_loss: 0.3919 - val_accuracy: 0.8255
Epoch 28/100
368/368 [==============================] - 163s 442ms/step - loss: 0.3594 - accurac
y: 0.8425 - val_loss: 0.4162 - val_accuracy: 0.8078
Epoch 29/100
368/368 [==============================] - 164s 444ms/step - loss: 0.3586 - accurac
y: 0.8438 - val_loss: 0.3960 - val_accuracy: 0.8223
Fold 17 - Val Loss: 0.3359, Val Accuracy: 0.8643

--- Fold 18/20 ---
Epoch 1/100
368/368 [==============================] - 158s 421ms/step - loss: 0.6629 - accurac
y: 0.6111 - val_loss: 0.6593 - val_accuracy: 0.6414
Epoch 2/100
368/368 [==============================] - 153s 416ms/step - loss: 0.6319 - accurac
y: 0.6533 - val_loss: 0.7284 - val_accuracy: 0.6543
Epoch 3/100
368/368 [==============================] - 151s 411ms/step - loss: 0.6176 - accurac
y: 0.6679 - val_loss: 0.6313 - val_accuracy: 0.6898
Epoch 4/100
368/368 [==============================] - 151s 409ms/step - loss: 0.5998 - accurac
y: 0.6818 - val_loss: 0.7617 - val_accuracy: 0.6446
Epoch 5/100
368/368 [==============================] - 154s 418ms/step - loss: 0.5652 - accurac
y: 0.7120 - val_loss: 0.5927 - val_accuracy: 0.6963
Epoch 6/100
368/368 [==============================] - 155s 421ms/step - loss: 0.5368 - accurac
y: 0.7341 - val_loss: 0.5938 - val_accuracy: 0.7189
Epoch 7/100
```

```
368/368 [==============================] - 150s 408ms/step - loss: 0.5176 - accurac
y: 0.7470 - val_loss: 0.5771 - val_accuracy: 0.7302
Epoch 8/100
368/368 [==============================] - 152s 412ms/step - loss: 0.5082 - accurac
y: 0.7527 - val_loss: 0.8712 - val_accuracy: 0.6607
Epoch 9/100
368/368 [==============================] - 150s 408ms/step - loss: 0.4960 - accurac
y: 0.7623 - val_loss: 0.5831 - val_accuracy: 0.7334
Epoch 10/100
368/368 [==============================] - 151s 411ms/step - loss: 0.4729 - accurac
y: 0.7803 - val_loss: 0.7795 - val_accuracy: 0.6607
Epoch 11/100
368/368 [==============================] - 152s 412ms/step - loss: 0.4643 - accurac
y: 0.7826 - val_loss: 0.5950 - val_accuracy: 0.7383
Epoch 12/100
368/368 [==============================] - 153s 414ms/step - loss: 0.4584 - accurac
y: 0.7847 - val_loss: 0.9371 - val_accuracy: 0.6284
Epoch 13/100
368/368 [==============================] - 151s 410ms/step - loss: 0.4440 - accurac
y: 0.7973 - val_loss: 0.6186 - val_accuracy: 0.7415
Epoch 14/100
368/368 [==============================] - 150s 407ms/step - loss: 0.4404 - accurac
y: 0.7999 - val_loss: 0.5391 - val_accuracy: 0.7625
Epoch 15/100
368/368 [==============================] - 152s 412ms/step - loss: 0.4370 - accurac
y: 0.8008 - val_loss: 0.5031 - val_accuracy: 0.7835
Epoch 16/100
368/368 [==============================] - 152s 413ms/step - loss: 0.4241 - accurac
y: 0.8081 - val_loss: 0.6767 - val_accuracy: 0.7431
Epoch 17/100
368/368 [==============================] - 151s 410ms/step - loss: 0.4190 - accurac
y: 0.8109 - val_loss: 0.5950 - val_accuracy: 0.7286
Epoch 18/100
368/368 [==============================] - 153s 415ms/step - loss: 0.4115 - accurac
y: 0.8159 - val_loss: 0.6390 - val_accuracy: 0.7334
Epoch 19/100
368/368 [==============================] - 151s 410ms/step - loss: 0.4084 - accurac
y: 0.8176 - val_loss: 0.7653 - val_accuracy: 0.7157
Epoch 20/100
368/368 [==============================] - 151s 409ms/step - loss: 0.4075 - accurac
y: 0.8187 - val_loss: 0.6332 - val_accuracy: 0.7561
Epoch 21/100
368/368 [==============================] - 150s 407ms/step - loss: 0.4002 - accurac
y: 0.8188 - val_loss: 0.4306 - val_accuracy: 0.8126
Epoch 22/100
368/368 [==============================] - 150s 407ms/step - loss: 0.4033 - accurac
y: 0.8218 - val_loss: 0.6555 - val_accuracy: 0.7496
Epoch 23/100
368/368 [==============================] - 150s 408ms/step - loss: 0.3902 - accurac
y: 0.8277 - val_loss: 0.4979 - val_accuracy: 0.7851
Epoch 24/100
368/368 [==============================] - 151s 409ms/step - loss: 0.3907 - accurac
y: 0.8257 - val_loss: 0.4933 - val_accuracy: 0.7964
Epoch 25/100
368/368 [==============================] - 151s 410ms/step - loss: 0.3844 - accurac
y: 0.8288 - val_loss: 0.4756 - val_accuracy: 0.8045
```

```
Epoch 26/100
368/368 [==============================] - 151s 409ms/step - loss: 0.3851 - accurac
y: 0.8333 - val_loss: 0.5698 - val_accuracy: 0.7722
Epoch 27/100
368/368 [==============================] - 150s 406ms/step - loss: 0.3833 - accurac
y: 0.8294 - val_loss: 0.5519 - val_accuracy: 0.8045
Epoch 28/100
368/368 [==============================] - 150s 406ms/step - loss: 0.3809 - accurac
y: 0.8355 - val_loss: 0.4833 - val_accuracy: 0.7916
Epoch 29/100
368/368 [==============================] - 151s 411ms/step - loss: 0.3728 - accurac
y: 0.8390 - val_loss: 0.6623 - val_accuracy: 0.7447
Epoch 30/100
368/368 [==============================] - 153s 414ms/step - loss: 0.3686 - accurac
y: 0.8417 - val_loss: 0.5269 - val_accuracy: 0.7932
Epoch 31/100
368/368 [==============================] - 151s 410ms/step - loss: 0.3635 - accurac
y: 0.8383 - val_loss: 0.5929 - val_accuracy: 0.7658
Fold 18 - Val Loss: 0.4306, Val Accuracy: 0.8126

--- Fold 19/20 ---
Epoch 1/100
368/368 [==============================] - 152s 403ms/step - loss: 0.6587 - accurac
y: 0.6176 - val_loss: 0.6190 - val_accuracy: 0.6672
Epoch 2/100
368/368 [==============================] - 145s 392ms/step - loss: 0.6189 - accurac
y: 0.6659 - val_loss: 0.6027 - val_accuracy: 0.6672
Epoch 3/100
368/368 [==============================] - 145s 394ms/step - loss: 0.5820 - accurac
y: 0.6986 - val_loss: 0.5942 - val_accuracy: 0.7124
Epoch 4/100
368/368 [==============================] - 145s 394ms/step - loss: 0.5521 - accurac
y: 0.7284 - val_loss: 0.6838 - val_accuracy: 0.7060
Epoch 5/100
368/368 [==============================] - 144s 391ms/step - loss: 0.5144 - accurac
y: 0.7510 - val_loss: 0.6691 - val_accuracy: 0.6914
Epoch 6/100
368/368 [==============================] - 147s 399ms/step - loss: 0.4931 - accurac
y: 0.7660 - val_loss: 0.5438 - val_accuracy: 0.7351
Epoch 7/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4792 - accurac
y: 0.7751 - val_loss: 0.5457 - val_accuracy: 0.7561
Epoch 8/100
368/368 [==============================] - 144s 391ms/step - loss: 0.4665 - accurac
y: 0.7844 - val_loss: 0.4644 - val_accuracy: 0.7981
Epoch 9/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4571 - accurac
y: 0.7874 - val_loss: 0.5219 - val_accuracy: 0.7383
Epoch 10/100
368/368 [==============================] - 143s 389ms/step - loss: 0.4506 - accurac
y: 0.7929 - val_loss: 0.4629 - val_accuracy: 0.7835
Epoch 11/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4437 - accurac
y: 0.8011 - val_loss: 0.3872 - val_accuracy: 0.8449
Epoch 12/100
368/368 [==============================] - 144s 392ms/step - loss: 0.4294 - accurac
```

```
y: 0.8047 - val_loss: 0.4248 - val_accuracy: 0.8126
Epoch 13/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4328 - accurac
y: 0.8074 - val_loss: 0.4344 - val_accuracy: 0.8142
Epoch 14/100
368/368 [==============================] - 144s 391ms/step - loss: 0.4218 - accurac
y: 0.8074 - val_loss: 0.4628 - val_accuracy: 0.7771
Epoch 15/100
368/368 [==============================] - 145s 393ms/step - loss: 0.4240 - accurac
y: 0.8079 - val_loss: 0.3775 - val_accuracy: 0.8191
Epoch 16/100
368/368 [==============================] - 143s 389ms/step - loss: 0.4185 - accurac
y: 0.8142 - val_loss: 0.4421 - val_accuracy: 0.8013
Epoch 17/100
368/368 [==============================] - 145s 394ms/step - loss: 0.4064 - accurac
y: 0.8192 - val_loss: 0.4035 - val_accuracy: 0.8384
Epoch 18/100
368/368 [==============================] - 145s 393ms/step - loss: 0.4018 - accurac
y: 0.8207 - val_loss: 0.4920 - val_accuracy: 0.7868
Epoch 19/100
368/368 [==============================] - 145s 395ms/step - loss: 0.3960 - accurac
y: 0.8233 - val_loss: 0.4032 - val_accuracy: 0.8191
Epoch 20/100
368/368 [==============================] - 143s 387ms/step - loss: 0.3945 - accurac
y: 0.8253 - val_loss: 0.3308 - val_accuracy: 0.8449
Epoch 21/100
368/368 [==============================] - 143s 387ms/step - loss: 0.3872 - accurac
y: 0.8243 - val_loss: 0.4267 - val_accuracy: 0.8352
Epoch 22/100
368/368 [==============================] - 145s 393ms/step - loss: 0.3858 - accurac
y: 0.8288 - val_loss: 0.4088 - val_accuracy: 0.8271
Epoch 23/100
368/368 [==============================] - 146s 396ms/step - loss: 0.3804 - accurac
y: 0.8307 - val_loss: 0.3781 - val_accuracy: 0.8465
Epoch 24/100
368/368 [==============================] - 143s 390ms/step - loss: 0.3710 - accurac
y: 0.8317 - val_loss: 0.3459 - val_accuracy: 0.8401
Epoch 25/100
368/368 [==============================] - 144s 391ms/step - loss: 0.3784 - accurac
y: 0.8287 - val_loss: 0.3543 - val_accuracy: 0.8401
Epoch 26/100
368/368 [==============================] - 144s 391ms/step - loss: 0.3757 - accurac
y: 0.8334 - val_loss: 0.3692 - val_accuracy: 0.8417
Epoch 27/100
368/368 [==============================] - 144s 390ms/step - loss: 0.3704 - accurac
y: 0.8353 - val_loss: 0.3870 - val_accuracy: 0.8239
Epoch 28/100
368/368 [==============================] - 145s 395ms/step - loss: 0.3664 - accurac
y: 0.8377 - val_loss: 0.4194 - val_accuracy: 0.8288
Epoch 29/100
368/368 [==============================] - 145s 393ms/step - loss: 0.3624 - accurac
y: 0.8450 - val_loss: 0.3507 - val_accuracy: 0.8530
Epoch 30/100
368/368 [==============================] - 145s 393ms/step - loss: 0.3624 - accurac
y: 0.8362 - val_loss: 0.4302 - val_accuracy: 0.8288
Fold 19 - Val Loss: 0.3308, Val Accuracy: 0.8449
```

```
--- Fold 20/20 ---
Epoch 1/100
368/368 [==============================] - 145s 384ms/step - loss: 0.6697 - accurac
y: 0.6015 - val_loss: 0.6361 - val_accuracy: 0.6414
Epoch 2/100
368/368 [==============================] - 140s 380ms/step - loss: 0.6328 - accurac
y: 0.6525 - val_loss: 0.7125 - val_accuracy: 0.6123
Epoch 3/100
368/368 [==============================] - 138s 376ms/step - loss: 0.6024 - accurac
y: 0.6786 - val_loss: 0.7224 - val_accuracy: 0.6494
Epoch 4/100
368/368 [==============================] - 139s 377ms/step - loss: 0.5707 - accurac
y: 0.7063 - val_loss: 0.8983 - val_accuracy: 0.6462
Epoch 5/100
368/368 [==============================] - 139s 378ms/step - loss: 0.5372 - accurac
y: 0.7345 - val_loss: 0.7897 - val_accuracy: 0.6624
Epoch 6/100
368/368 [==============================] - 138s 376ms/step - loss: 0.5170 - accurac
y: 0.7498 - val_loss: 0.5006 - val_accuracy: 0.7480
Epoch 7/100
368/368 [==============================] - 137s 373ms/step - loss: 0.5037 - accurac
y: 0.7553 - val_loss: 0.5699 - val_accuracy: 0.7480
Epoch 8/100
368/368 [==============================] - 138s 376ms/step - loss: 0.4837 - accurac
y: 0.7674 - val_loss: 0.5891 - val_accuracy: 0.7399
Epoch 9/100
368/368 [==============================] - 140s 381ms/step - loss: 0.4710 - accurac
y: 0.7789 - val_loss: 0.6068 - val_accuracy: 0.7173
Epoch 10/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4608 - accurac
y: 0.7867 - val_loss: 0.5678 - val_accuracy: 0.7431
Epoch 11/100
368/368 [==============================] - 138s 376ms/step - loss: 0.4415 - accurac
y: 0.7964 - val_loss: 0.5361 - val_accuracy: 0.7609
Epoch 12/100
368/368 [==============================] - 140s 381ms/step - loss: 0.4402 - accurac
y: 0.7981 - val_loss: 0.6022 - val_accuracy: 0.7512
Epoch 13/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4336 - accurac
y: 0.8055 - val_loss: 0.6033 - val_accuracy: 0.7254
Epoch 14/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4277 - accurac
y: 0.8048 - val_loss: 0.4609 - val_accuracy: 0.7738
Epoch 15/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4317 - accurac
y: 0.8046 - val_loss: 0.6367 - val_accuracy: 0.7173
Epoch 16/100
368/368 [==============================] - 139s 376ms/step - loss: 0.4130 - accurac
y: 0.8155 - val_loss: 0.5747 - val_accuracy: 0.7480
Epoch 17/100
368/368 [==============================] - 138s 375ms/step - loss: 0.4106 - accurac
y: 0.8168 - val_loss: 0.6206 - val_accuracy: 0.7496
Epoch 18/100
368/368 [==============================] - 139s 378ms/step - loss: 0.4102 - accurac
y: 0.8197 - val_loss: 0.4596 - val_accuracy: 0.7932
```

```
Epoch 19/100
368/368 [==============================] - 139s 377ms/step - loss: 0.4098 - accurac
y: 0.8186 - val_loss: 0.5212 - val_accuracy: 0.7754
Epoch 20/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3965 - accurac
y: 0.8241 - val_loss: 0.4603 - val_accuracy: 0.7932
Epoch 21/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3936 - accurac
y: 0.8243 - val_loss: 0.4876 - val_accuracy: 0.7819
Epoch 22/100
368/368 [==============================] - 139s 376ms/step - loss: 0.3939 - accurac
y: 0.8259 - val_loss: 0.5631 - val_accuracy: 0.7593
Epoch 23/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3838 - accurac
y: 0.8274 - val_loss: 0.4871 - val_accuracy: 0.7997
Epoch 24/100
368/368 [==============================] - 138s 376ms/step - loss: 0.3816 - accurac
y: 0.8310 - val_loss: 0.3871 - val_accuracy: 0.8239
Epoch 25/100
368/368 [==============================] - 140s 380ms/step - loss: 0.3866 - accurac
y: 0.8324 - val_loss: 0.4099 - val_accuracy: 0.8368
Epoch 26/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3742 - accurac
y: 0.8376 - val_loss: 0.5149 - val_accuracy: 0.8061
Epoch 27/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3736 - accurac
y: 0.8357 - val_loss: 0.3706 - val_accuracy: 0.8384
Epoch 28/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3734 - accurac
y: 0.8395 - val_loss: 0.3711 - val_accuracy: 0.8304
Epoch 29/100
368/368 [==============================] - 143s 387ms/step - loss: 0.3651 - accurac
y: 0.8426 - val_loss: 0.3580 - val_accuracy: 0.8417
Epoch 30/100
368/368 [==============================] - 142s 386ms/step - loss: 0.3629 - accurac
y: 0.8423 - val_loss: 0.3737 - val_accuracy: 0.8433
Epoch 31/100
368/368 [==============================] - 141s 382ms/step - loss: 0.3645 - accurac
y: 0.8397 - val_loss: 0.3626 - val_accuracy: 0.8578
Epoch 32/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3680 - accurac
y: 0.8383 - val_loss: 0.4224 - val_accuracy: 0.8336
Epoch 33/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3516 - accurac
y: 0.8488 - val_loss: 0.3554 - val_accuracy: 0.8352
Epoch 34/100
368/368 [==============================] - 139s 376ms/step - loss: 0.3612 - accurac
y: 0.8468 - val_loss: 0.3816 - val_accuracy: 0.8530
Epoch 35/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3556 - accurac
y: 0.8455 - val_loss: 0.3317 - val_accuracy: 0.8595
Epoch 36/100
368/368 [==============================] - 141s 383ms/step - loss: 0.3530 - accurac
y: 0.8469 - val_loss: 0.3434 - val_accuracy: 0.8611
Epoch 37/100
368/368 [==============================] - 143s 389ms/step - loss: 0.3541 - accurac
```

```
y: 0.8486 - val_loss: 0.3978 - val_accuracy: 0.8449
Epoch 38/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3494 - accurac
y: 0.8474 - val_loss: 0.4382 - val_accuracy: 0.8465
Epoch 39/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3501 - accurac
y: 0.8476 - val_loss: 0.3538 - val_accuracy: 0.8546
Epoch 40/100
368/368 [==============================] - 139s 378ms/step - loss: 0.3497 - accurac
y: 0.8518 - val_loss: 0.3948 - val_accuracy: 0.8595
Epoch 41/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3402 - accurac
y: 0.8518 - val_loss: 0.3470 - val_accuracy: 0.8659
Epoch 42/100
368/368 [==============================] - 139s 376ms/step - loss: 0.3494 - accurac
y: 0.8453 - val_loss: 0.3186 - val_accuracy: 0.8659
Epoch 43/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3437 - accurac
y: 0.8519 - val_loss: 0.3276 - val_accuracy: 0.8530
Epoch 44/100
368/368 [==============================] - 139s 379ms/step - loss: 0.3362 - accurac
y: 0.8604 - val_loss: 0.4906 - val_accuracy: 0.8384
Epoch 45/100
368/368 [==============================] - 144s 392ms/step - loss: 0.3336 - accurac
y: 0.8587 - val_loss: 0.4180 - val_accuracy: 0.8352
Epoch 46/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3349 - accurac
y: 0.8560 - val_loss: 0.4392 - val_accuracy: 0.8481
Epoch 47/100
368/368 [==============================] - 138s 375ms/step - loss: 0.3359 - accurac
y: 0.8551 - val_loss: 0.3479 - val_accuracy: 0.8611
Epoch 48/100
368/368 [==============================] - 139s 376ms/step - loss: 0.3386 - accurac
y: 0.8555 - val_loss: 0.3328 - val_accuracy: 0.8675
Epoch 49/100
368/368 [==============================] - 138s 374ms/step - loss: 0.3377 - accurac
y: 0.8524 - val_loss: 0.3576 - val_accuracy: 0.8562
Epoch 50/100
368/368 [==============================] - 139s 377ms/step - loss: 0.3332 - accurac
y: 0.8561 - val_loss: 0.3621 - val_accuracy: 0.8627
Epoch 51/100
368/368 [==============================] - 141s 382ms/step - loss: 0.3220 - accurac
y: 0.8604 - val_loss: 0.5371 - val_accuracy: 0.8271
Epoch 52/100
368/368 [==============================] - 142s 385ms/step - loss: 0.3346 - accurac
y: 0.8553 - val_loss: 0.4723 - val_accuracy: 0.8255
Fold 20 - Val Loss: 0.3186, Val Accuracy: 0.8659

--- Resultados de Validación Cruzada ---
Accuracy promedio: 0.8439 ± 0.0299
Loss promedio: 0.3619 ± 0.0543
1/1 [==============================] - 1s 942ms/step
```

Modelo CNN-5.

In [ ]:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import KFold

data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"

def load_images_from_directory(directory):
    images = []
    labels = []

    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)

        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)

            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)

    return np.array(images), np.array(labels)

X, y = load_images_from_directory(data_dir)
X = X / 255.0

k = 4
kf = KFold(n_splits=k, shuffle=True, random_state=42)

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

os.makedirs("graficas_kfold", exist_ok=True)
os.makedirs("modelos_kfold", exist_ok=True)

fold_accuracies = []
fold_losses = []

for fold, (train_idx, val_idx) in enumerate(kf.split(X, y)):
```

```
    print(f"\n--- Fold {fold + 1}/{k} ---")

    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'

    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_we

    history = model.fit(
        datagen.flow(X_train, y_train, batch_size=32),
        epochs=100,
        validation_data=(X_val, y_val),
        callbacks=[early_stopping],
        verbose=1
    )

    val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=0)
    fold_accuracies.append(val_accuracy)
    fold_losses.append(val_loss)

    print(f"Fold {fold + 1} - Val Loss: {val_loss:.4f}, Val Accuracy: {val_accuracy

    model.save(f'modelos_kfold/modelo_fold_{fold+1}.h5')

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.title(f'Fold {fold+1} - Accuracy')
    plt.xlabel('Épocas')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f'Fold {fold+1} - Loss')
    plt.xlabel('Épocas')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
```

```
    plt.tight_layout()
    plt.savefig(f'graficas_kfold/curvas_fold_{fold+1}.png')
    plt.close()

print("\n--- Resultados de Validación Cruzada ---")
print(f"Accuracy promedio: {np.mean(fold_accuracies):.4f} ± {np.std(fold_accuracies
print(f"Loss promedio: {np.mean(fold_losses):.4f} ± {np.std(fold_losses):.4f}")

def predict_image(model, img_path):
    img = image.load_img(img_path, target_size=(100, 100))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = "CON HEMORRAGIA" if prediction[0] > 0.5 else "SIN HEMORRAGIA"

    plt.imshow(img)
    plt.title(f"Predicción: {predicted_class} ({prediction[0][0]:.4f})")
    plt.axis('off')
    plt.show()

test_image_path = "D:/PROYECTO 1/prueba/prueba.png"
predict_image(model, test_image_path)
```

```
--- Fold 1/4 ---
Epoch 1/100
291/291 [==============================] - 133s 448ms/step - loss: 0.6637 - accurac
y: 0.6030 - val_loss: 0.6213 - val_accuracy: 0.6837
Epoch 2/100
291/291 [==============================] - 117s 401ms/step - loss: 0.6400 - accurac
y: 0.6412 - val_loss: 0.6091 - val_accuracy: 0.6653
Epoch 3/100
291/291 [==============================] - 121s 414ms/step - loss: 0.6073 - accurac
y: 0.6692 - val_loss: 0.6537 - val_accuracy: 0.6662
Epoch 4/100
291/291 [==============================] - 130s 447ms/step - loss: 0.5991 - accurac
y: 0.6807 - val_loss: 0.6748 - val_accuracy: 0.6575
Epoch 5/100
291/291 [==============================] - 126s 433ms/step - loss: 0.5754 - accurac
y: 0.6977 - val_loss: 0.6474 - val_accuracy: 0.6885
Epoch 6/100
291/291 [==============================] - 126s 432ms/step - loss: 0.5581 - accurac
y: 0.7201 - val_loss: 0.7846 - val_accuracy: 0.6320
Epoch 7/100
291/291 [==============================] - 141s 486ms/step - loss: 0.5367 - accurac
y: 0.7339 - val_loss: 0.7707 - val_accuracy: 0.6404
Epoch 8/100
291/291 [==============================] - 140s 479ms/step - loss: 0.5203 - accurac
y: 0.7390 - val_loss: 0.5081 - val_accuracy: 0.7573
Epoch 9/100
291/291 [==============================] - 122s 420ms/step - loss: 0.4946 - accurac
y: 0.7605 - val_loss: 0.8680 - val_accuracy: 0.5904
Epoch 10/100
291/291 [==============================] - 121s 415ms/step - loss: 0.4898 - accurac
y: 0.7630 - val_loss: 0.7349 - val_accuracy: 0.6737
Epoch 11/100
291/291 [==============================] - 124s 427ms/step - loss: 0.4668 - accurac
y: 0.7720 - val_loss: 0.6346 - val_accuracy: 0.6930
Epoch 12/100
291/291 [==============================] - 122s 419ms/step - loss: 0.4532 - accurac
y: 0.7912 - val_loss: 0.5532 - val_accuracy: 0.7434
Epoch 13/100
291/291 [==============================] - 121s 417ms/step - loss: 0.4632 - accurac
y: 0.7835 - val_loss: 0.5926 - val_accuracy: 0.6850
Epoch 14/100
291/291 [==============================] - 129s 441ms/step - loss: 0.4431 - accurac
y: 0.7948 - val_loss: 0.4861 - val_accuracy: 0.7821
Epoch 15/100
291/291 [==============================] - 129s 444ms/step - loss: 0.4309 - accurac
y: 0.8030 - val_loss: 0.4995 - val_accuracy: 0.7579
Epoch 16/100
291/291 [==============================] - 122s 418ms/step - loss: 0.4246 - accurac
y: 0.8080 - val_loss: 0.4933 - val_accuracy: 0.7666
Epoch 17/100
291/291 [==============================] - 121s 416ms/step - loss: 0.4208 - accurac
y: 0.8094 - val_loss: 0.6288 - val_accuracy: 0.7130
Epoch 18/100
291/291 [==============================] - 122s 420ms/step - loss: 0.4160 - accurac
y: 0.8131 - val_loss: 0.4399 - val_accuracy: 0.7976
Epoch 19/100
```

```
291/291 [==============================] - 124s 425ms/step - loss: 0.4029 - accurac
y: 0.8211 - val_loss: 0.4806 - val_accuracy: 0.7928
Epoch 20/100
291/291 [==============================] - 121s 416ms/step - loss: 0.4072 - accurac
y: 0.8179 - val_loss: 0.7159 - val_accuracy: 0.7066
Epoch 21/100
291/291 [==============================] - 122s 419ms/step - loss: 0.3950 - accurac
y: 0.8218 - val_loss: 0.4689 - val_accuracy: 0.7921
Epoch 22/100
291/291 [==============================] - 122s 419ms/step - loss: 0.3941 - accurac
y: 0.8246 - val_loss: 0.4802 - val_accuracy: 0.7744
Epoch 23/100
291/291 [==============================] - 121s 417ms/step - loss: 0.3956 - accurac
y: 0.8184 - val_loss: 0.4308 - val_accuracy: 0.7950
Epoch 24/100
291/291 [==============================] - 122s 420ms/step - loss: 0.3858 - accurac
y: 0.8284 - val_loss: 0.5146 - val_accuracy: 0.7721
Epoch 25/100
291/291 [==============================] - 121s 416ms/step - loss: 0.3881 - accurac
y: 0.8276 - val_loss: 0.4098 - val_accuracy: 0.8112
Epoch 26/100
291/291 [==============================] - 121s 415ms/step - loss: 0.3879 - accurac
y: 0.8296 - val_loss: 0.4123 - val_accuracy: 0.8138
Epoch 27/100
291/291 [==============================] - 121s 416ms/step - loss: 0.3852 - accurac
y: 0.8288 - val_loss: 0.3462 - val_accuracy: 0.8376
Epoch 28/100
291/291 [==============================] - 120s 412ms/step - loss: 0.3721 - accurac
y: 0.8376 - val_loss: 0.4211 - val_accuracy: 0.8070
Epoch 29/100
291/291 [==============================] - 121s 415ms/step - loss: 0.3763 - accurac
y: 0.8394 - val_loss: 0.3954 - val_accuracy: 0.8238
Epoch 30/100
291/291 [==============================] - 122s 419ms/step - loss: 0.3714 - accurac
y: 0.8369 - val_loss: 0.4185 - val_accuracy: 0.8050
Epoch 31/100
291/291 [==============================] - 121s 414ms/step - loss: 0.3659 - accurac
y: 0.8394 - val_loss: 0.4092 - val_accuracy: 0.8163
Epoch 32/100
291/291 [==============================] - 119s 410ms/step - loss: 0.3670 - accurac
y: 0.8381 - val_loss: 0.5128 - val_accuracy: 0.7692
Epoch 33/100
291/291 [==============================] - 120s 412ms/step - loss: 0.3566 - accurac
y: 0.8426 - val_loss: 0.4317 - val_accuracy: 0.8037
Epoch 34/100
291/291 [==============================] - 121s 414ms/step - loss: 0.3576 - accurac
y: 0.8439 - val_loss: 0.5400 - val_accuracy: 0.7660
Epoch 35/100
291/291 [==============================] - 120s 413ms/step - loss: 0.3590 - accurac
y: 0.8452 - val_loss: 0.5888 - val_accuracy: 0.7540
Epoch 36/100
291/291 [==============================] - 119s 409ms/step - loss: 0.3608 - accurac
y: 0.8434 - val_loss: 0.6130 - val_accuracy: 0.7511
Epoch 37/100
291/291 [==============================] - 122s 419ms/step - loss: 0.3512 - accurac
y: 0.8503 - val_loss: 0.3414 - val_accuracy: 0.8502
```

```
Epoch 38/100
291/291 [==============================] - 120s 414ms/step - loss: 0.3513 - accurac
y: 0.8519 - val_loss: 0.3084 - val_accuracy: 0.8618
Epoch 39/100
291/291 [==============================] - 121s 415ms/step - loss: 0.3425 - accurac
y: 0.8512 - val_loss: 0.3935 - val_accuracy: 0.8147
Epoch 40/100
291/291 [==============================] - 119s 410ms/step - loss: 0.3392 - accurac
y: 0.8517 - val_loss: 0.3765 - val_accuracy: 0.8334
Epoch 41/100
291/291 [==============================] - 119s 410ms/step - loss: 0.3441 - accurac
y: 0.8524 - val_loss: 0.7367 - val_accuracy: 0.7076
Epoch 42/100
291/291 [==============================] - 119s 409ms/step - loss: 0.3336 - accurac
y: 0.8613 - val_loss: 0.3487 - val_accuracy: 0.8522
Epoch 43/100
291/291 [==============================] - 120s 411ms/step - loss: 0.3421 - accurac
y: 0.8519 - val_loss: 0.3113 - val_accuracy: 0.8599
Epoch 44/100
291/291 [==============================] - 120s 411ms/step - loss: 0.3307 - accurac
y: 0.8620 - val_loss: 0.4585 - val_accuracy: 0.7934
Epoch 45/100
291/291 [==============================] - 121s 415ms/step - loss: 0.3338 - accurac
y: 0.8540 - val_loss: 0.3938 - val_accuracy: 0.8283
Epoch 46/100
291/291 [==============================] - 121s 415ms/step - loss: 0.3370 - accurac
y: 0.8586 - val_loss: 0.4869 - val_accuracy: 0.7992
Epoch 47/100
291/291 [==============================] - 119s 410ms/step - loss: 0.3339 - accurac
y: 0.8559 - val_loss: 0.4741 - val_accuracy: 0.8037
Epoch 48/100
291/291 [==============================] - 120s 411ms/step - loss: 0.3260 - accurac
y: 0.8616 - val_loss: 0.4122 - val_accuracy: 0.8212
Fold 1 - Val Loss: 0.3084, Val Accuracy: 0.8618
d:\Anaconda\envs\tesis38\lib\site-packages\keras\src\engine\training.py:3000: UserWa
rning: You are saving your model as an HDF5 file via `model.save()`. This file forma
t is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')`.
  saving_api.save_model(
```

```
--- Fold 2/4 ---
Epoch 1/100
291/291 [==============================] - 126s 418ms/step - loss: 0.6698 - accurac
y: 0.5997 - val_loss: 0.6108 - val_accuracy: 0.6614
Epoch 2/100
291/291 [==============================] - 123s 421ms/step - loss: 0.6429 - accurac
y: 0.6352 - val_loss: 0.5947 - val_accuracy: 0.6669
Epoch 3/100
291/291 [==============================] - 122s 417ms/step - loss: 0.6318 - accurac
y: 0.6576 - val_loss: 0.5682 - val_accuracy: 0.6908
Epoch 4/100
291/291 [==============================] - 118s 405ms/step - loss: 0.6095 - accurac
y: 0.6717 - val_loss: 0.5509 - val_accuracy: 0.7124
Epoch 5/100
291/291 [==============================] - 121s 414ms/step - loss: 0.5901 - accurac
y: 0.6900 - val_loss: 0.6109 - val_accuracy: 0.6898
Epoch 6/100
291/291 [==============================] - 126s 434ms/step - loss: 0.5761 - accurac
y: 0.7037 - val_loss: 0.5710 - val_accuracy: 0.7140
Epoch 7/100
291/291 [==============================] - 120s 413ms/step - loss: 0.5565 - accurac
y: 0.7190 - val_loss: 0.7727 - val_accuracy: 0.6578
Epoch 8/100
291/291 [==============================] - 122s 420ms/step - loss: 0.5521 - accurac
y: 0.7207 - val_loss: 0.6594 - val_accuracy: 0.7117
Epoch 9/100
291/291 [==============================] - 152s 522ms/step - loss: 0.5204 - accurac
y: 0.7406 - val_loss: 0.5607 - val_accuracy: 0.7485
Epoch 10/100
291/291 [==============================] - 120s 412ms/step - loss: 0.5111 - accurac
y: 0.7480 - val_loss: 0.5177 - val_accuracy: 0.7563
Epoch 11/100
291/291 [==============================] - 119s 410ms/step - loss: 0.4870 - accurac
y: 0.7680 - val_loss: 0.4918 - val_accuracy: 0.7734
Epoch 12/100
291/291 [==============================] - 134s 459ms/step - loss: 0.4837 - accurac
y: 0.7700 - val_loss: 0.5673 - val_accuracy: 0.7201
Epoch 13/100
291/291 [==============================] - 131s 451ms/step - loss: 0.4727 - accurac
y: 0.7784 - val_loss: 0.4120 - val_accuracy: 0.8186
Epoch 14/100
291/291 [==============================] - 124s 424ms/step - loss: 0.4587 - accurac
y: 0.7866 - val_loss: 0.4580 - val_accuracy: 0.7853
Epoch 15/100
291/291 [==============================] - 122s 420ms/step - loss: 0.4477 - accurac
y: 0.7919 - val_loss: 0.4891 - val_accuracy: 0.7760
Epoch 16/100
291/291 [==============================] - 120s 412ms/step - loss: 0.4376 - accurac
y: 0.7990 - val_loss: 0.3842 - val_accuracy: 0.8283
Epoch 17/100
291/291 [==============================] - 123s 422ms/step - loss: 0.4296 - accurac
y: 0.8034 - val_loss: 0.4096 - val_accuracy: 0.8250
Epoch 18/100
291/291 [==============================] - 123s 422ms/step - loss: 0.4294 - accurac
y: 0.8016 - val_loss: 0.4024 - val_accuracy: 0.8125
Epoch 19/100
```

```
291/291 [==============================] - 119s 410ms/step - loss: 0.4156 - accurac
y: 0.8141 - val_loss: 0.3679 - val_accuracy: 0.8325
Epoch 20/100
291/291 [==============================] - 114s 392ms/step - loss: 0.4170 - accurac
y: 0.8145 - val_loss: 0.4028 - val_accuracy: 0.8283
Epoch 21/100
291/291 [==============================] - 114s 391ms/step - loss: 0.4071 - accurac
y: 0.8192 - val_loss: 0.4025 - val_accuracy: 0.8150
Epoch 22/100
291/291 [==============================] - 114s 390ms/step - loss: 0.4080 - accurac
y: 0.8114 - val_loss: 0.3722 - val_accuracy: 0.8296
Epoch 23/100
291/291 [==============================] - 114s 390ms/step - loss: 0.4035 - accurac
y: 0.8160 - val_loss: 0.4073 - val_accuracy: 0.8125
Epoch 24/100
291/291 [==============================] - 115s 396ms/step - loss: 0.3967 - accurac
y: 0.8226 - val_loss: 0.4232 - val_accuracy: 0.8170
Epoch 25/100
291/291 [==============================] - 116s 398ms/step - loss: 0.3896 - accurac
y: 0.8291 - val_loss: 0.4131 - val_accuracy: 0.8167
Epoch 26/100
291/291 [==============================] - 115s 395ms/step - loss: 0.3813 - accurac
y: 0.8319 - val_loss: 0.3849 - val_accuracy: 0.8234
Epoch 27/100
291/291 [==============================] - 115s 395ms/step - loss: 0.3973 - accurac
y: 0.8189 - val_loss: 0.4102 - val_accuracy: 0.8108
Epoch 28/100
291/291 [==============================] - 114s 393ms/step - loss: 0.3797 - accurac
y: 0.8289 - val_loss: 0.4183 - val_accuracy: 0.8205
Epoch 29/100
291/291 [==============================] - 114s 393ms/step - loss: 0.3828 - accurac
y: 0.8289 - val_loss: 0.3774 - val_accuracy: 0.8422
Fold 2 - Val Loss: 0.3679, Val Accuracy: 0.8325

--- Fold 3/4 ---
Epoch 1/100
291/291 [==============================] - 120s 401ms/step - loss: 0.6661 - accurac
y: 0.6058 - val_loss: 0.6320 - val_accuracy: 0.6420
Epoch 2/100
291/291 [==============================] - 114s 392ms/step - loss: 0.6284 - accurac
y: 0.6562 - val_loss: 0.5837 - val_accuracy: 0.6866
Epoch 3/100
291/291 [==============================] - 117s 401ms/step - loss: 0.6022 - accurac
y: 0.6795 - val_loss: 0.6238 - val_accuracy: 0.6507
Epoch 4/100
291/291 [==============================] - 116s 398ms/step - loss: 0.5837 - accurac
y: 0.7027 - val_loss: 0.7371 - val_accuracy: 0.6482
Epoch 5/100
291/291 [==============================] - 115s 395ms/step - loss: 0.5546 - accurac
y: 0.7215 - val_loss: 0.5572 - val_accuracy: 0.7195
Epoch 6/100
291/291 [==============================] - 115s 393ms/step - loss: 0.5255 - accurac
y: 0.7454 - val_loss: 0.5427 - val_accuracy: 0.7376
Epoch 7/100
291/291 [==============================] - 115s 393ms/step - loss: 0.5007 - accurac
y: 0.7567 - val_loss: 0.8219 - val_accuracy: 0.6165
```

```
Epoch 8/100
291/291 [==============================] - 115s 395ms/step - loss: 0.4856 - accurac
y: 0.7701 - val_loss: 0.5784 - val_accuracy: 0.6975
Epoch 9/100
291/291 [==============================] - 114s 393ms/step - loss: 0.4656 - accurac
y: 0.7823 - val_loss: 0.5045 - val_accuracy: 0.7624
Epoch 10/100
291/291 [==============================] - 115s 393ms/step - loss: 0.4518 - accurac
y: 0.7923 - val_loss: 0.5582 - val_accuracy: 0.7298
Epoch 11/100
291/291 [==============================] - 114s 391ms/step - loss: 0.4537 - accurac
y: 0.7876 - val_loss: 0.4621 - val_accuracy: 0.7795
Epoch 12/100
291/291 [==============================] - 114s 393ms/step - loss: 0.4464 - accurac
y: 0.7960 - val_loss: 0.5442 - val_accuracy: 0.7411
Epoch 13/100
291/291 [==============================] - 114s 393ms/step - loss: 0.4304 - accurac
y: 0.8074 - val_loss: 0.4428 - val_accuracy: 0.7924
Epoch 14/100
291/291 [==============================] - 114s 393ms/step - loss: 0.4293 - accurac
y: 0.8051 - val_loss: 0.4711 - val_accuracy: 0.7711
Epoch 15/100
291/291 [==============================] - 115s 394ms/step - loss: 0.4254 - accurac
y: 0.8047 - val_loss: 0.4752 - val_accuracy: 0.7892
Epoch 16/100
291/291 [==============================] - 113s 389ms/step - loss: 0.4036 - accurac
y: 0.8173 - val_loss: 0.5194 - val_accuracy: 0.7724
Epoch 17/100
291/291 [==============================] - 115s 393ms/step - loss: 0.3989 - accurac
y: 0.8230 - val_loss: 0.6330 - val_accuracy: 0.7456
Epoch 18/100
291/291 [==============================] - 115s 394ms/step - loss: 0.4063 - accurac
y: 0.8170 - val_loss: 0.4184 - val_accuracy: 0.8244
Epoch 19/100
291/291 [==============================] - 114s 392ms/step - loss: 0.4004 - accurac
y: 0.8191 - val_loss: 0.4137 - val_accuracy: 0.8189
Epoch 20/100
291/291 [==============================] - 117s 401ms/step - loss: 0.3866 - accurac
y: 0.8319 - val_loss: 0.6870 - val_accuracy: 0.7295
Epoch 21/100
291/291 [==============================] - 114s 393ms/step - loss: 0.3894 - accurac
y: 0.8274 - val_loss: 0.3967 - val_accuracy: 0.8205
Epoch 22/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3805 - accurac
y: 0.8308 - val_loss: 0.4309 - val_accuracy: 0.8167
Epoch 23/100
291/291 [==============================] - 115s 393ms/step - loss: 0.3919 - accurac
y: 0.8310 - val_loss: 0.4284 - val_accuracy: 0.8205
Epoch 24/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3732 - accurac
y: 0.8368 - val_loss: 0.4458 - val_accuracy: 0.8125
Epoch 25/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3776 - accurac
y: 0.8304 - val_loss: 0.3594 - val_accuracy: 0.8412
Epoch 26/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3761 - accurac
```

```
y: 0.8361 - val_loss: 0.4060 - val_accuracy: 0.8254
Epoch 27/100
291/291 [==============================] - 116s 398ms/step - loss: 0.3612 - accurac
y: 0.8409 - val_loss: 0.3620 - val_accuracy: 0.8589
Epoch 28/100
291/291 [==============================] - 115s 394ms/step - loss: 0.3642 - accurac
y: 0.8427 - val_loss: 0.3946 - val_accuracy: 0.8312
Epoch 29/100
291/291 [==============================] - 113s 389ms/step - loss: 0.3685 - accurac
y: 0.8391 - val_loss: 0.3672 - val_accuracy: 0.8509
Epoch 30/100
291/291 [==============================] - 115s 394ms/step - loss: 0.3521 - accurac
y: 0.8444 - val_loss: 0.5286 - val_accuracy: 0.8092
Epoch 31/100
291/291 [==============================] - 113s 387ms/step - loss: 0.3552 - accurac
y: 0.8417 - val_loss: 0.3759 - val_accuracy: 0.8535
Epoch 32/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3451 - accurac
y: 0.8489 - val_loss: 0.3987 - val_accuracy: 0.8422
Epoch 33/100
291/291 [==============================] - 114s 391ms/step - loss: 0.3504 - accurac
y: 0.8484 - val_loss: 0.4277 - val_accuracy: 0.8299
Epoch 34/100
291/291 [==============================] - 115s 394ms/step - loss: 0.3432 - accurac
y: 0.8502 - val_loss: 0.3330 - val_accuracy: 0.8577
Epoch 35/100
291/291 [==============================] - 115s 396ms/step - loss: 0.3427 - accurac
y: 0.8543 - val_loss: 0.4081 - val_accuracy: 0.8267
Epoch 36/100
291/291 [==============================] - 116s 399ms/step - loss: 0.3509 - accurac
y: 0.8518 - val_loss: 0.4112 - val_accuracy: 0.8496
Epoch 37/100
291/291 [==============================] - 114s 391ms/step - loss: 0.3391 - accurac
y: 0.8525 - val_loss: 0.3531 - val_accuracy: 0.8502
Epoch 38/100
291/291 [==============================] - 114s 391ms/step - loss: 0.3395 - accurac
y: 0.8494 - val_loss: 0.3678 - val_accuracy: 0.8486
Epoch 39/100
291/291 [==============================] - 115s 394ms/step - loss: 0.3469 - accurac
y: 0.8511 - val_loss: 0.4111 - val_accuracy: 0.8299
Epoch 40/100
291/291 [==============================] - 115s 394ms/step - loss: 0.3398 - accurac
y: 0.8523 - val_loss: 0.3818 - val_accuracy: 0.8444
Epoch 41/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3265 - accurac
y: 0.8622 - val_loss: 0.4090 - val_accuracy: 0.8476
Epoch 42/100
291/291 [==============================] - 115s 393ms/step - loss: 0.3376 - accurac
y: 0.8501 - val_loss: 0.3510 - val_accuracy: 0.8551
Epoch 43/100
291/291 [==============================] - 115s 395ms/step - loss: 0.3279 - accurac
y: 0.8549 - val_loss: 0.3584 - val_accuracy: 0.8622
Epoch 44/100
291/291 [==============================] - 115s 394ms/step - loss: 0.3235 - accurac
y: 0.8612 - val_loss: 0.3625 - val_accuracy: 0.8577
Fold 3 - Val Loss: 0.3330, Val Accuracy: 0.8577
```

```
--- Fold 4/4 ---
Epoch 1/100
291/291 [==============================] - 121s 406ms/step - loss: 0.6709 - accurac
y: 0.5993 - val_loss: 0.6101 - val_accuracy: 0.6659
Epoch 2/100
291/291 [==============================] - 116s 399ms/step - loss: 0.6374 - accurac
y: 0.6454 - val_loss: 0.6817 - val_accuracy: 0.6436
Epoch 3/100
291/291 [==============================] - 114s 390ms/step - loss: 0.6096 - accurac
y: 0.6815 - val_loss: 0.5909 - val_accuracy: 0.6795
Epoch 4/100
291/291 [==============================] - 116s 397ms/step - loss: 0.5874 - accurac
y: 0.6970 - val_loss: 0.6343 - val_accuracy: 0.6911
Epoch 5/100
291/291 [==============================] - 115s 396ms/step - loss: 0.5650 - accurac
y: 0.7156 - val_loss: 0.5915 - val_accuracy: 0.6992
Epoch 6/100
291/291 [==============================] - 115s 394ms/step - loss: 0.5399 - accurac
y: 0.7304 - val_loss: 0.5666 - val_accuracy: 0.7260
Epoch 7/100
291/291 [==============================] - 114s 392ms/step - loss: 0.5149 - accurac
y: 0.7494 - val_loss: 0.6682 - val_accuracy: 0.6930
Epoch 8/100
291/291 [==============================] - 117s 403ms/step - loss: 0.4885 - accurac
y: 0.7725 - val_loss: 0.8333 - val_accuracy: 0.6440
Epoch 9/100
291/291 [==============================] - 114s 391ms/step - loss: 0.4912 - accurac
y: 0.7678 - val_loss: 0.7422 - val_accuracy: 0.6582
Epoch 10/100
291/291 [==============================] - 116s 399ms/step - loss: 0.4666 - accurac
y: 0.7811 - val_loss: 0.6179 - val_accuracy: 0.7201
Epoch 11/100
291/291 [==============================] - 113s 389ms/step - loss: 0.4669 - accurac
y: 0.7869 - val_loss: 0.5622 - val_accuracy: 0.7427
Epoch 12/100
291/291 [==============================] - 113s 388ms/step - loss: 0.4478 - accurac
y: 0.7951 - val_loss: 0.6089 - val_accuracy: 0.7279
Epoch 13/100
291/291 [==============================] - 115s 393ms/step - loss: 0.4434 - accurac
y: 0.7973 - val_loss: 0.5419 - val_accuracy: 0.7711
Epoch 14/100
291/291 [==============================] - 114s 392ms/step - loss: 0.4403 - accurac
y: 0.8012 - val_loss: 0.5817 - val_accuracy: 0.7518
Epoch 15/100
291/291 [==============================] - 114s 390ms/step - loss: 0.4332 - accurac
y: 0.8011 - val_loss: 0.4595 - val_accuracy: 0.7992
Epoch 16/100
291/291 [==============================] - 115s 394ms/step - loss: 0.4238 - accurac
y: 0.8050 - val_loss: 0.4093 - val_accuracy: 0.8128
Epoch 17/100
291/291 [==============================] - 116s 397ms/step - loss: 0.4226 - accurac
y: 0.8075 - val_loss: 0.4498 - val_accuracy: 0.7921
Epoch 18/100
291/291 [==============================] - 115s 393ms/step - loss: 0.4157 - accurac
y: 0.8131 - val_loss: 0.5727 - val_accuracy: 0.7553
```

```
Epoch 19/100
291/291 [==============================] - 115s 395ms/step - loss: 0.4031 - accurac
y: 0.8213 - val_loss: 0.5737 - val_accuracy: 0.7508
Epoch 20/100
291/291 [==============================] - 115s 395ms/step - loss: 0.3976 - accurac
y: 0.8243 - val_loss: 0.7388 - val_accuracy: 0.7343
Epoch 21/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3979 - accurac
y: 0.8205 - val_loss: 0.4360 - val_accuracy: 0.8105
Epoch 22/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3885 - accurac
y: 0.8254 - val_loss: 0.4559 - val_accuracy: 0.7957
Epoch 23/100
291/291 [==============================] - 114s 393ms/step - loss: 0.3897 - accurac
y: 0.8290 - val_loss: 0.4186 - val_accuracy: 0.8041
Epoch 24/100
291/291 [==============================] - 115s 395ms/step - loss: 0.3854 - accurac
y: 0.8296 - val_loss: 0.4660 - val_accuracy: 0.8025
Epoch 25/100
291/291 [==============================] - 114s 393ms/step - loss: 0.3847 - accurac
y: 0.8313 - val_loss: 0.4561 - val_accuracy: 0.8157
Epoch 26/100
291/291 [==============================] - 114s 392ms/step - loss: 0.3737 - accurac
y: 0.8369 - val_loss: 0.4117 - val_accuracy: 0.8221
Fold 4 - Val Loss: 0.4093, Val Accuracy: 0.8128

--- Resultados de Validación Cruzada ---
Accuracy promedio: 0.8412 ± 0.0199
Loss promedio: 0.3547 ± 0.0380
1/1 [==============================] - 1s 557ms/step
```

Modelo CNN-6.

In [3]:
```
# Ruta a la carpeta donde están tus imágenes (con_hemorragia y sin_hemorragia)
data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"
```

In [ ]:
```
import numpy as np
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image

data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"

def load_images_from_directory(directory):
    images = []
    labels = []

    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)

        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)

            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)

    return np.array(images), np.array(labels)

X, y = load_images_from_directory(data_dir)

X = X / 255.0

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15, random_stat

print(f"Tamaño de X_train: {X_train.shape}")
print(f"Tamaño de X_val: {X_val.shape}")
```

```
Tamaño de X_train: (10533, 100, 100, 3)
Tamaño de X_val: (1859, 100, 100, 3)
```

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
```

```
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

```
d:\anaconda carpeta de instalacion\envs\trabajo\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
flatten (Flatten)	(None, 33856)	0
dense (Dense)	(None, 64)	2,166,848
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 2,186,305 (8.34 MB)

Trainable params: 2,186,305 (8.34 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)
```

In [ ]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weight

history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping]
)
```

```
Epoch 1/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 128s 371ms/step - accuracy: 0.5707 - loss: 0.6947 - val
_accuracy: 0.6568 - val_loss: 0.6144
Epoch 2/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 125s 378ms/step - accuracy: 0.6433 - loss: 0.6417 - val
_accuracy: 0.6380 - val_loss: 0.6390
Epoch 3/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 130s 393ms/step - accuracy: 0.6646 - loss: 0.6213 - val
_accuracy: 0.7084 - val_loss: 0.5559
Epoch 4/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 124s 375ms/step - accuracy: 0.6614 - loss: 0.6163 - val
_accuracy: 0.7154 - val_loss: 0.5412
Epoch 5/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 124s 377ms/step - accuracy: 0.6933 - loss: 0.5858 - val
_accuracy: 0.6859 - val_loss: 0.6937
Epoch 6/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 134s 405ms/step - accuracy: 0.7120 - loss: 0.5714 - val
_accuracy: 0.6950 - val_loss: 0.6782
Epoch 7/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 123s 373ms/step - accuracy: 0.7289 - loss: 0.5389 - val
_accuracy: 0.7036 - val_loss: 0.6135
Epoch 8/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 122s 369ms/step - accuracy: 0.7308 - loss: 0.5291 - val
_accuracy: 0.6912 - val_loss: 0.7210
Epoch 9/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 126s 380ms/step - accuracy: 0.7447 - loss: 0.5067 - val
_accuracy: 0.7434 - val_loss: 0.6064
Epoch 10/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 125s 377ms/step - accuracy: 0.7681 - loss: 0.4922 - val
_accuracy: 0.7106 - val_loss: 0.6510
Epoch 11/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 123s 371ms/step - accuracy: 0.7864 - loss: 0.4592 - val
_accuracy: 0.7439 - val_loss: 0.5796
Epoch 12/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 124s 374ms/step - accuracy: 0.7817 - loss: 0.4657 - val
_accuracy: 0.6993 - val_loss: 0.6776
Epoch 13/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 123s 371ms/step - accuracy: 0.7850 - loss: 0.4592 - val
_accuracy: 0.7644 - val_loss: 0.4874
Epoch 14/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 122s 369ms/step - accuracy: 0.7961 - loss: 0.4539 - val
_accuracy: 0.7751 - val_loss: 0.4856
Epoch 15/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 123s 372ms/step - accuracy: 0.7994 - loss: 0.4305 - val
_accuracy: 0.7585 - val_loss: 0.5533
Epoch 16/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 120s 362ms/step - accuracy: 0.7921 - loss: 0.4554 - val
_accuracy: 0.7773 - val_loss: 0.4560
Epoch 17/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 121s 365ms/step - accuracy: 0.8069 - loss: 0.4268 - val
_accuracy: 0.7805 - val_loss: 0.4469
Epoch 18/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 177s 535ms/step - accuracy: 0.8150 - loss: 0.4115 - val
_accuracy: 0.8419 - val_loss: 0.3693
Epoch 19/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 168s 509ms/step - accuracy: 0.8164 - loss: 0.4062 - val
```

```
_accuracy: 0.8386 - val_loss: 0.3914
Epoch 20/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 164s 496ms/step - accuracy: 0.8001 - loss: 0.4246 - val
_accuracy: 0.7746 - val_loss: 0.5330
Epoch 21/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 166s 502ms/step - accuracy: 0.8196 - loss: 0.4005 - val
_accuracy: 0.8445 - val_loss: 0.3513
Epoch 22/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 159s 482ms/step - accuracy: 0.8341 - loss: 0.3831 - val
_accuracy: 0.8230 - val_loss: 0.4136
Epoch 23/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 164s 495ms/step - accuracy: 0.8305 - loss: 0.3884 - val
_accuracy: 0.8171 - val_loss: 0.4599
Epoch 24/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 165s 501ms/step - accuracy: 0.8281 - loss: 0.3922 - val
_accuracy: 0.8429 - val_loss: 0.3816
Epoch 25/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 154s 467ms/step - accuracy: 0.8400 - loss: 0.3658 - val
_accuracy: 0.8295 - val_loss: 0.4084
Epoch 26/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 163s 493ms/step - accuracy: 0.8492 - loss: 0.3577 - val
_accuracy: 0.7827 - val_loss: 0.5731
Epoch 27/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 162s 490ms/step - accuracy: 0.8383 - loss: 0.3635 - val
_accuracy: 0.8343 - val_loss: 0.4178
Epoch 28/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 161s 488ms/step - accuracy: 0.8485 - loss: 0.3580 - val
_accuracy: 0.8279 - val_loss: 0.4073
Epoch 29/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 166s 502ms/step - accuracy: 0.8362 - loss: 0.3700 - val
_accuracy: 0.8397 - val_loss: 0.3540
Epoch 30/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 165s 500ms/step - accuracy: 0.8506 - loss: 0.3446 - val
_accuracy: 0.8865 - val_loss: 0.2696
Epoch 31/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 162s 491ms/step - accuracy: 0.8431 - loss: 0.3552 - val
_accuracy: 0.8725 - val_loss: 0.3125
Epoch 32/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 155s 469ms/step - accuracy: 0.8550 - loss: 0.3426 - val
_accuracy: 0.8252 - val_loss: 0.4272
Epoch 33/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 143s 433ms/step - accuracy: 0.8527 - loss: 0.3435 - val
_accuracy: 0.8284 - val_loss: 0.4105
Epoch 34/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 129s 389ms/step - accuracy: 0.8514 - loss: 0.3437 - val
_accuracy: 0.8526 - val_loss: 0.3450
Epoch 35/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 125s 378ms/step - accuracy: 0.8590 - loss: 0.3361 - val
_accuracy: 0.8322 - val_loss: 0.4156
Epoch 36/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 122s 370ms/step - accuracy: 0.8457 - loss: 0.3492 - val
_accuracy: 0.7972 - val_loss: 0.5494
Epoch 37/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 116s 350ms/step - accuracy: 0.8562 - loss: 0.3455 - val
_accuracy: 0.8763 - val_loss: 0.2966
Epoch 38/100
```

```
330/330 ━━━━━━━━━━━━━━━━━━━━ 132s 399ms/step - accuracy: 0.8619 - loss: 0.3316 - val
_accuracy: 0.8289 - val_loss: 0.3979
Epoch 39/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 135s 409ms/step - accuracy: 0.8593 - loss: 0.3341 - val
_accuracy: 0.8252 - val_loss: 0.4397
Epoch 40/100
330/330 ━━━━━━━━━━━━━━━━━━━━ 130s 393ms/step - accuracy: 0.8592 - loss: 0.3258 - val
_accuracy: 0.8166 - val_loss: 0.4128
```

In [ ]:
```python
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validación - Pérdida: {val_loss:.4f}, Exactitud: {val_accuracy:.4f}")
```

```
59/59 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.8783 - loss: 0.2828
Validación - Pérdida: 0.2696, Exactitud: 0.8865
```

In [ ]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
plt.title('Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 6))

plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

In [ ]:
```
json_path = "D:\PROYECTO 1\PRECISION 87 PERDIDA 28.json"
with open(json_path, "w") as json_file:
    json_file.write(model.to_json())

weights_path = "D:\PROYECTO 1\PRECISION 87 PERDIDA 28.weights.h5"
model.save_weights(weights_path)

print(f"Modelo convertido y guardado:\n - Arquitectura: {json_path}\n - Pesos: {wei
```

```
<>:2: SyntaxWarning: invalid escape sequence '\P'
<>:7: SyntaxWarning: invalid escape sequence '\P'
<>:2: SyntaxWarning: invalid escape sequence '\P'
<>:7: SyntaxWarning: invalid escape sequence '\P'
C:\Users\zoqui\AppData\Local\Temp\ipykernel_11232\3427954172.py:2: SyntaxWarning: in
valid escape sequence '\P'
  json_path = "D:\PROYECTO 1\PRECISION 87 PERDIDA 28.json"
C:\Users\zoqui\AppData\Local\Temp\ipykernel_11232\3427954172.py:7: SyntaxWarning: in
valid escape sequence '\P'
  weights_path = "D:\PROYECTO 1\PRECISION 87 PERDIDA 28.weights.h5"  # Aquí está el
cambio
Modelo convertido y guardado:
 - Arquitectura: D:\PROYECTO 1\PRECISION 87 PERDIDA 28.json
 - Pesos: D:\PROYECTO 1\PRECISION 87 PERDIDA 28.weights.h5
```

In [ ]:
```
from tensorflow.keras.models import load_model
import json
```

```
h5_path = "D:/PROYECTO 1/MODELO_FINAL/modelo_cnn.h5"
model = load_model(h5_path)

json_path = "D:/PROYECTO 1/MODELO_FINAL/modelo_cnn.json"
with open(json_path, "w") as json_file:
    json_file.write(model.to_json())

weights_path = "D:/PROYECTO 1/MODELO_FINAL/modelo_cnn_weights.weights.h5"
model.save_weights(weights_path)

print(f"Modelo convertido y guardado:\n - Arquitectura: {json_path}\n - Pesos: {wei
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be buil
t. `model.compile_metrics` will be empty until you train or evaluate the model.
Modelo convertido y guardado:
 - Arquitectura: D:/PROYECTO 1/MODELO_FINAL/modelo_cnn.json
 - Pesos: D:/PROYECTO 1/MODELO_FINAL/modelo_cnn_weights.weights.h5
```

In [ ]:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"

def load_images_from_directory(directory):
    images = []
    labels = []
    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)
        for img_name in os.listdir(folder_path):
            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img_path = os.path.join(folder_path, img_name)
                img = image.load_img(img_path, target_size=(100, 100))
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)
    return np.array(images), np.array(labels)

X, y = load_images_from_directory(data_dir)
X = X / 255.0

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15, random_stat

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
```

```
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weight
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping],
    verbose=1
)

val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=0)
print(f"Validación - Pérdida: {val_loss:.4f}, Precisión: {val_accuracy:.4f}")

output_path = "D:/TESIS/modelo 6/modelo_6.h5"
model.save(output_path)
print(f"✅ Modelo guardado exitosamente en: {output_path}")

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 32)        896

 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 47, 47, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 23, 23, 64)        0
 g2D)

 flatten (Flatten)           (None, 33856)             0

 dense (Dense)               (None, 64)                2166848

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2186305 (8.34 MB)
Trainable params: 2186305 (8.34 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
330/330 [==============================] - 120s 352ms/step - loss: 0.6642 - accurac
y: 0.6037 - val_loss: 0.6050 - val_accuracy: 0.6719
Epoch 2/100
330/330 [==============================] - 151s 456ms/step - loss: 0.6316 - accurac
y: 0.6527 - val_loss: 0.6046 - val_accuracy: 0.6799
Epoch 3/100
330/330 [==============================] - 114s 345ms/step - loss: 0.6015 - accurac
y: 0.6821 - val_loss: 0.5820 - val_accuracy: 0.6993
Epoch 4/100
330/330 [==============================] - 116s 352ms/step - loss: 0.5844 - accurac
y: 0.6937 - val_loss: 0.5200 - val_accuracy: 0.7289
Epoch 5/100
330/330 [==============================] - 115s 349ms/step - loss: 0.5580 - accurac
y: 0.7203 - val_loss: 0.6967 - val_accuracy: 0.6815
Epoch 6/100
330/330 [==============================] - 122s 369ms/step - loss: 0.5324 - accurac
y: 0.7397 - val_loss: 0.6658 - val_accuracy: 0.6826
Epoch 7/100
330/330 [==============================] - 123s 372ms/step - loss: 0.5130 - accurac
y: 0.7516 - val_loss: 0.6049 - val_accuracy: 0.7251
Epoch 8/100
330/330 [==============================] - 126s 382ms/step - loss: 0.4975 - accurac
y: 0.7613 - val_loss: 0.6273 - val_accuracy: 0.7316
Epoch 9/100
330/330 [==============================] - 126s 382ms/step - loss: 0.4802 - accurac
y: 0.7735 - val_loss: 0.7606 - val_accuracy: 0.6891
Epoch 10/100
330/330 [==============================] - 115s 349ms/step - loss: 0.4665 - accurac
```

```
y: 0.7835 - val_loss: 0.4978 - val_accuracy: 0.7741
Epoch 11/100
330/330 [==============================] - 123s 371ms/step - loss: 0.4511 - accurac
y: 0.7920 - val_loss: 0.4362 - val_accuracy: 0.8010
Epoch 12/100
330/330 [==============================] - 121s 366ms/step - loss: 0.4457 - accurac
y: 0.7954 - val_loss: 0.4994 - val_accuracy: 0.7714
Epoch 13/100
330/330 [==============================] - 118s 356ms/step - loss: 0.4284 - accurac
y: 0.8061 - val_loss: 0.4719 - val_accuracy: 0.7757
Epoch 14/100
330/330 [==============================] - 139s 421ms/step - loss: 0.4229 - accurac
y: 0.8063 - val_loss: 0.4934 - val_accuracy: 0.7751
Epoch 15/100
330/330 [==============================] - 139s 420ms/step - loss: 0.4209 - accurac
y: 0.8110 - val_loss: 0.4596 - val_accuracy: 0.7832
Epoch 16/100
330/330 [==============================] - 118s 358ms/step - loss: 0.4158 - accurac
y: 0.8140 - val_loss: 0.4290 - val_accuracy: 0.7913
Epoch 17/100
330/330 [==============================] - 145s 438ms/step - loss: 0.4040 - accurac
y: 0.8184 - val_loss: 0.4981 - val_accuracy: 0.7800
Epoch 18/100
330/330 [==============================] - 139s 421ms/step - loss: 0.4054 - accurac
y: 0.8192 - val_loss: 0.3985 - val_accuracy: 0.8144
Epoch 19/100
330/330 [==============================] - 137s 414ms/step - loss: 0.3949 - accurac
y: 0.8272 - val_loss: 0.3872 - val_accuracy: 0.8332
Epoch 20/100
330/330 [==============================] - 137s 414ms/step - loss: 0.3904 - accurac
y: 0.8292 - val_loss: 0.5387 - val_accuracy: 0.7563
Epoch 21/100
330/330 [==============================] - 138s 416ms/step - loss: 0.3952 - accurac
y: 0.8203 - val_loss: 0.3856 - val_accuracy: 0.8279
Epoch 22/100
330/330 [==============================] - 136s 412ms/step - loss: 0.3745 - accurac
y: 0.8332 - val_loss: 0.3698 - val_accuracy: 0.8392
Epoch 23/100
330/330 [==============================] - 134s 406ms/step - loss: 0.3743 - accurac
y: 0.8324 - val_loss: 0.4047 - val_accuracy: 0.8236
Epoch 24/100
330/330 [==============================] - 134s 407ms/step - loss: 0.3768 - accurac
y: 0.8331 - val_loss: 0.3928 - val_accuracy: 0.8349
Epoch 25/100
330/330 [==============================] - 134s 406ms/step - loss: 0.3774 - accurac
y: 0.8330 - val_loss: 0.4253 - val_accuracy: 0.8123
Epoch 26/100
330/330 [==============================] - 138s 418ms/step - loss: 0.3643 - accurac
y: 0.8397 - val_loss: 0.5735 - val_accuracy: 0.7784
Epoch 27/100
330/330 [==============================] - 137s 414ms/step - loss: 0.3615 - accurac
y: 0.8356 - val_loss: 0.3678 - val_accuracy: 0.8440
Epoch 28/100
330/330 [==============================] - 138s 419ms/step - loss: 0.3661 - accurac
y: 0.8434 - val_loss: 0.6194 - val_accuracy: 0.7655
Epoch 29/100
```

```
330/330 [==============================] - 138s 418ms/step - loss: 0.3415 - accurac
y: 0.8502 - val_loss: 0.4565 - val_accuracy: 0.8080
Epoch 30/100
330/330 [==============================] - 138s 418ms/step - loss: 0.3548 - accurac
y: 0.8515 - val_loss: 0.4710 - val_accuracy: 0.8236
Epoch 31/100
330/330 [==============================] - 140s 423ms/step - loss: 0.3525 - accurac
y: 0.8454 - val_loss: 0.5255 - val_accuracy: 0.8107
Epoch 32/100
330/330 [==============================] - 143s 432ms/step - loss: 0.3451 - accurac
y: 0.8499 - val_loss: 0.3649 - val_accuracy: 0.8338
Epoch 33/100
330/330 [==============================] - 153s 463ms/step - loss: 0.3480 - accurac
y: 0.8493 - val_loss: 0.3917 - val_accuracy: 0.8429
Epoch 34/100
330/330 [==============================] - 144s 436ms/step - loss: 0.3298 - accurac
y: 0.8566 - val_loss: 0.4697 - val_accuracy: 0.8139
Epoch 35/100
330/330 [==============================] - 141s 427ms/step - loss: 0.3387 - accurac
y: 0.8537 - val_loss: 0.5957 - val_accuracy: 0.7951
Epoch 36/100
330/330 [==============================] - 144s 438ms/step - loss: 0.3348 - accurac
y: 0.8531 - val_loss: 0.3695 - val_accuracy: 0.8505
Epoch 37/100
330/330 [==============================] - 143s 432ms/step - loss: 0.3418 - accurac
y: 0.8508 - val_loss: 0.4766 - val_accuracy: 0.8160
Epoch 38/100
330/330 [==============================] - 139s 422ms/step - loss: 0.3309 - accurac
y: 0.8570 - val_loss: 0.3867 - val_accuracy: 0.8419
Epoch 39/100
330/330 [==============================] - 131s 397ms/step - loss: 0.3352 - accurac
y: 0.8553 - val_loss: 0.4362 - val_accuracy: 0.8117
Epoch 40/100
330/330 [==============================] - 133s 404ms/step - loss: 0.3275 - accurac
y: 0.8594 - val_loss: 0.4529 - val_accuracy: 0.8214
Epoch 41/100
330/330 [==============================] - 207s 626ms/step - loss: 0.3364 - accurac
y: 0.8525 - val_loss: 0.4060 - val_accuracy: 0.8472
Epoch 42/100
330/330 [==============================] - 145s 439ms/step - loss: 0.3242 - accurac
y: 0.8565 - val_loss: 0.3076 - val_accuracy: 0.8596
Epoch 43/100
330/330 [==============================] - 125s 378ms/step - loss: 0.3182 - accurac
y: 0.8659 - val_loss: 0.3384 - val_accuracy: 0.8558
Epoch 44/100
330/330 [==============================] - 130s 393ms/step - loss: 0.3244 - accurac
y: 0.8621 - val_loss: 0.3868 - val_accuracy: 0.8419
Epoch 45/100
330/330 [==============================] - 132s 399ms/step - loss: 0.3165 - accurac
y: 0.8644 - val_loss: 0.3993 - val_accuracy: 0.8445
Epoch 46/100
330/330 [==============================] - 137s 414ms/step - loss: 0.3111 - accurac
y: 0.8626 - val_loss: 0.4315 - val_accuracy: 0.8354
Epoch 47/100
330/330 [==============================] - 133s 402ms/step - loss: 0.3204 - accurac
y: 0.8641 - val_loss: 0.3089 - val_accuracy: 0.8827
```

```
Epoch 48/100
330/330 [==============================] - 133s 402ms/step - loss: 0.3117 - accurac
y: 0.8700 - val_loss: 0.3232 - val_accuracy: 0.8725
Epoch 49/100
330/330 [==============================] - 142s 430ms/step - loss: 0.3132 - accurac
y: 0.8681 - val_loss: 0.4037 - val_accuracy: 0.8483
Epoch 50/100
330/330 [==============================] - 131s 396ms/step - loss: 0.3087 - accurac
y: 0.8687 - val_loss: 0.4094 - val_accuracy: 0.8402
Epoch 51/100
330/330 [==============================] - 135s 410ms/step - loss: 0.3085 - accurac
y: 0.8702 - val_loss: 0.3710 - val_accuracy: 0.8548
Epoch 52/100
330/330 [==============================] - 132s 400ms/step - loss: 0.3121 - accurac
y: 0.8682 - val_loss: 0.3533 - val_accuracy: 0.8553
Validación - Pérdida: 0.3076, Precisión: 0.8596
```

Modelo CNN-7.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_height, img_width = 224, 224
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=True
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def segmentar_craneo(img_path, size=(100, 100), plot=False):
    img_color = cv2.imread(img_path)
    img_color = cv2.resize(img_color, size)
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
    mask = np.zeros_like(img_gray)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=-1)

    img_segmentado = cv2.bitwise_and(img_color, img_color, mask=mask)

    if plot:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')
        plt.subplot(1, 3, 2)
        plt.imshow(mask, cmap='gray')
        plt.title("Máscara cráneo")
        plt.axis('off')
        plt.subplot(1, 3, 3)
        plt.imshow(cv2.cvtColor(img_segmentado, cv2.COLOR_BGR2RGB))
        plt.title("Segmentado")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return img_segmentado
```

In [ ]:
```
import os

ruta_entrada = 'D:/PROYECTO 1/SET DE DATOS RECORTADOS'
ruta_salida = 'D:/PROYECTO 1/DATA_LIMPIA2'
img_size = (100, 100)

for subfolder in os.listdir(ruta_entrada):
    ruta_subcarpeta = os.path.join(ruta_entrada, subfolder)
    ruta_salida_sub = os.path.join(ruta_salida, subfolder)
    os.makedirs(ruta_salida_sub, exist_ok=True)

    for filename in os.listdir(ruta_subcarpeta):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            ruta_img = os.path.join(ruta_subcarpeta, filename)
            img_seg = segmentar_craneo(ruta_img, size=img_size, plot=False)

            salida_path = os.path.join(ruta_salida_sub, filename)
            cv2.imwrite(salida_path, img_seg)

print("Todas las imágenes fueron procesadas y guardadas en:", ruta_salida)
```

```
Todas las imágenes fueron procesadas y guardadas en: D:/PROYECTO 1/DATA_LIMPIA2
```

In [ ]:
```
def aplicar_mascara_craneo(img_path, size=(100, 100)):
    import cv2
    img = cv2.imread(img_path)
    img = cv2.resize(img, size)
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    center = (size[0] // 2, size[1] // 2)
    radius = int(min(size) * 0.45)
    cv2.circle(mask, center, radius, 255, thickness=-1)
```

```
    masked_img = cv2.bitwise_and(img, img, mask=mask)
    return masked_img
```

In [ ]:
```
img_size = (100, 100)
batch_size = 32
epochs = 100
input_shape = (100, 100, 3)

print('Configuración definida.')
```

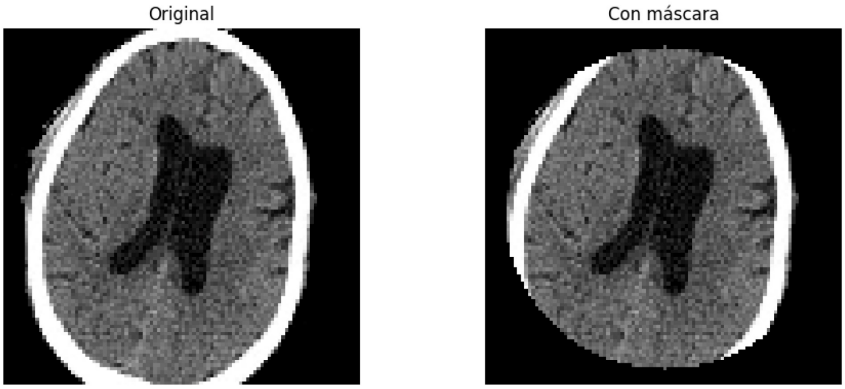
```
Configuración definida.
```

In [ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode="nearest"
)

train_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
img_path = r"D:\PROYECTO 1\DATA_LIMPIA2\CON HEMORRAGIA\1.2.840.113704.1.111.416.172
img_original = cv2.imread(img_path)
img_original = cv2.resize(img_original, img_size)
img_mascara = aplicar_mascara_craneo(img_path)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_mascara, cv2.COLOR_BGR2RGB))
plt.title('Con máscara')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



In [ ]:
```
from tensorflow.keras import regularizers

model = Sequential()
model.add(Conv2D(70, (3, 3), activation='relu', input_shape=input_shape,
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Flatten())
model.add(Dense(70, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.8))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_48 (Conv2D)          (None, 98, 98, 70)        1960

 batch_normalization_48 (Ba  (None, 98, 98, 70)        280
 tchNormalization)

 max_pooling2d_48 (MaxPooli  (None, 32, 32, 70)        0
 ng2D)

 conv2d_49 (Conv2D)          (None, 30, 30, 70)        44170

 batch_normalization_49 (Ba  (None, 30, 30, 70)        280
 tchNormalization)

 max_pooling2d_49 (MaxPooli  (None, 10, 10, 70)        0
 ng2D)

 conv2d_50 (Conv2D)          (None, 8, 8, 70)          44170

 batch_normalization_50 (Ba  (None, 8, 8, 70)          280
 tchNormalization)

 max_pooling2d_50 (MaxPooli  (None, 2, 2, 70)          0
 ng2D)

 flatten_16 (Flatten)        (None, 280)               0

 dense_32 (Dense)            (None, 70)                19670

 dropout_16 (Dropout)        (None, 70)                0

 dense_33 (Dense)            (None, 1)                 71

=================================================================
Total params: 110881 (433.13 KB)
Trainable params: 110461 (431.49 KB)
Non-trainable params: 420 (1.64 KB)
_________________________________________________________________
```

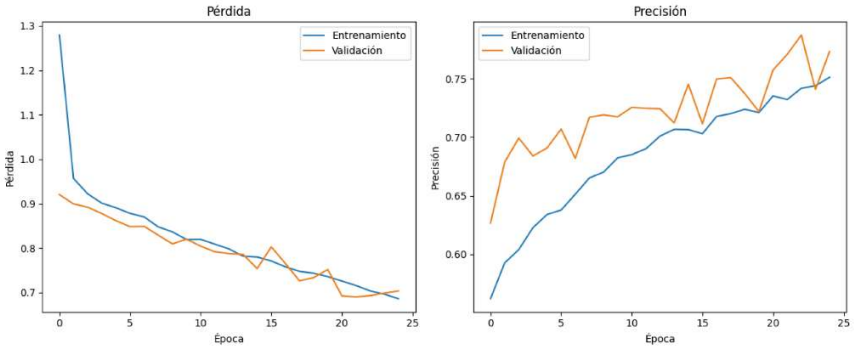
In [ ]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=30, factor=0.5)

history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stop, reduce_lr]
)

print('Entrenamiento completado.')
```

```
Epoch 1/100
310/310 [==============================] - 206s 646ms/step - loss: 1.2793 - accurac
y: 0.5621 - val_loss: 0.9204 - val_accuracy: 0.6267 - lr: 1.0000e-04
Epoch 2/100
310/310 [==============================] - 203s 656ms/step - loss: 0.9567 - accurac
y: 0.5927 - val_loss: 0.8994 - val_accuracy: 0.6788 - lr: 1.0000e-04
Epoch 3/100
310/310 [==============================] - 194s 626ms/step - loss: 0.9222 - accurac
y: 0.6041 - val_loss: 0.8921 - val_accuracy: 0.6994 - lr: 1.0000e-04
Epoch 4/100
310/310 [==============================] - 193s 622ms/step - loss: 0.9012 - accurac
y: 0.6228 - val_loss: 0.8778 - val_accuracy: 0.6840 - lr: 1.0000e-04
Epoch 5/100
310/310 [==============================] - 197s 635ms/step - loss: 0.8908 - accurac
y: 0.6341 - val_loss: 0.8618 - val_accuracy: 0.6909 - lr: 1.0000e-04
Epoch 6/100
310/310 [==============================] - 198s 639ms/step - loss: 0.8781 - accurac
y: 0.6379 - val_loss: 0.8483 - val_accuracy: 0.7070 - lr: 1.0000e-04
Epoch 7/100
310/310 [==============================] - 202s 650ms/step - loss: 0.8702 - accurac
y: 0.6514 - val_loss: 0.8490 - val_accuracy: 0.6820 - lr: 1.0000e-04
Epoch 8/100
310/310 [==============================] - 202s 650ms/step - loss: 0.8478 - accurac
y: 0.6653 - val_loss: 0.8291 - val_accuracy: 0.7171 - lr: 1.0000e-04
Epoch 9/100
310/310 [==============================] - 198s 640ms/step - loss: 0.8365 - accurac
y: 0.6703 - val_loss: 0.8094 - val_accuracy: 0.7191 - lr: 1.0000e-04
Epoch 10/100
310/310 [==============================] - 189s 610ms/step - loss: 0.8193 - accurac
y: 0.6825 - val_loss: 0.8202 - val_accuracy: 0.7175 - lr: 1.0000e-04
Epoch 11/100
310/310 [==============================] - 195s 627ms/step - loss: 0.8197 - accurac
y: 0.6852 - val_loss: 0.8044 - val_accuracy: 0.7256 - lr: 1.0000e-04
Epoch 12/100
310/310 [==============================] - 203s 656ms/step - loss: 0.8089 - accurac
y: 0.6902 - val_loss: 0.7919 - val_accuracy: 0.7248 - lr: 1.0000e-04
Epoch 13/100
310/310 [==============================] - 200s 645ms/step - loss: 0.7984 - accurac
y: 0.7010 - val_loss: 0.7878 - val_accuracy: 0.7244 - lr: 1.0000e-04
Epoch 14/100
310/310 [==============================] - 203s 654ms/step - loss: 0.7818 - accurac
y: 0.7068 - val_loss: 0.7859 - val_accuracy: 0.7123 - lr: 1.0000e-04
Epoch 15/100
310/310 [==============================] - 192s 619ms/step - loss: 0.7800 - accurac
y: 0.7066 - val_loss: 0.7541 - val_accuracy: 0.7454 - lr: 1.0000e-04
Epoch 16/100
310/310 [==============================] - 203s 655ms/step - loss: 0.7711 - accurac
y: 0.7030 - val_loss: 0.8028 - val_accuracy: 0.7115 - lr: 1.0000e-04
Epoch 17/100
310/310 [==============================] - 183s 588ms/step - loss: 0.7580 - accurac
y: 0.7178 - val_loss: 0.7653 - val_accuracy: 0.7498 - lr: 1.0000e-04
Epoch 18/100
310/310 [==============================] - 214s 689ms/step - loss: 0.7477 - accurac
y: 0.7202 - val_loss: 0.7263 - val_accuracy: 0.7510 - lr: 1.0000e-04
Epoch 19/100
310/310 [==============================] - 231s 744ms/step - loss: 0.7436 - accurac
```

```
y: 0.7239 - val_loss: 0.7338 - val_accuracy: 0.7373 - lr: 1.0000e-04
Epoch 20/100
310/310 [==============================] - 195s 629ms/step - loss: 0.7358 - accurac
y: 0.7211 - val_loss: 0.7518 - val_accuracy: 0.7220 - lr: 1.0000e-04
Epoch 21/100
310/310 [==============================] - 274s 883ms/step - loss: 0.7259 - accurac
y: 0.7353 - val_loss: 0.6925 - val_accuracy: 0.7575 - lr: 1.0000e-04
Epoch 22/100
310/310 [==============================] - 223s 718ms/step - loss: 0.7158 - accurac
y: 0.7322 - val_loss: 0.6902 - val_accuracy: 0.7708 - lr: 1.0000e-04
Epoch 23/100
310/310 [==============================] - 214s 692ms/step - loss: 0.7038 - accurac
y: 0.7419 - val_loss: 0.6931 - val_accuracy: 0.7873 - lr: 1.0000e-04
Epoch 24/100
310/310 [==============================] - 196s 631ms/step - loss: 0.6965 - accurac
y: 0.7440 - val_loss: 0.6989 - val_accuracy: 0.7409 - lr: 1.0000e-04
Epoch 25/100
310/310 [==============================] - 203s 653ms/step - loss: 0.6862 - accurac
y: 0.7513 - val_loss: 0.7037 - val_accuracy: 0.7732 - lr: 1.0000e-04
Entrenamiento completado.
```

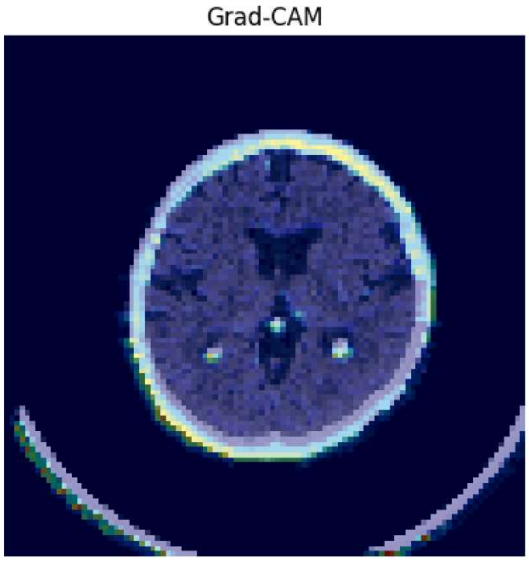
In [ ]:

```python
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.savefig("curvas_entrenamiento.png")
plt.show()
```

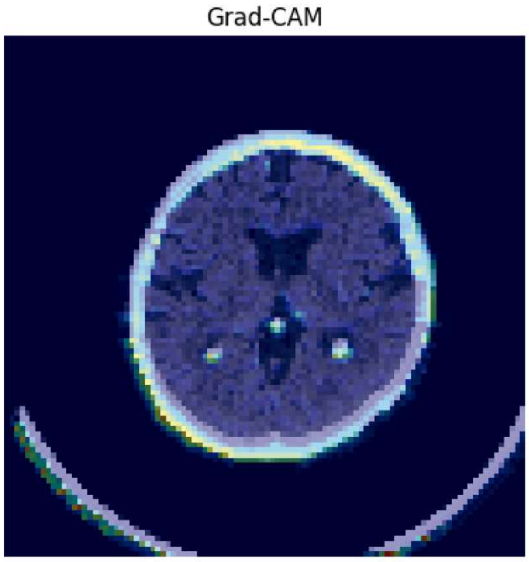


In [ ]:
```
def apply_gradcam(model, img_path, layer_name='conv2d_2'):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=img_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    grad_model = Model([model.inputs], [model.get_layer(layer_name).output, model.o

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[0]

    grads = tape.gradient(loss, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=(0, 1))
    cam = np.zeros(conv_outputs.shape[1:3], dtype=np.float32)

    for i, w in enumerate(weights):
        cam += w * conv_outputs[0, :, :, i]

    cam = np.maximum(cam, 0)
    cam = cam / np.max(cam)
    cam = cv2.resize(cam, img_size)
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)


    img_original = cv2.imread(img_path)
    img_original = cv2.resize(img_original, img_size)
    superimposed_img = cv2.addWeighted(img_original, 0.6, heatmap, 0.4, 0)

    plt.imshow(cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Grad-CAM")
    plt.show()
```

In [ ]:
```
test_img = r"D:\DOCUMENTOS COMPU\PROTOCOLOS\MACHINE LEARNING YO\MUESTRA TESIS PNGS\
apply_gradcam(model, test_img, layer_name='conv2d_48')
```



In [ ]:
```
test_img = r"D:\DOCUMENTOS COMPU\PROTOCOLOS\MACHINE LEARNING YO\MUESTRA TESIS PNGS\
apply_gradcam(model, test_img, layer_name='conv2d_48')
```



In [ ]:
```
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]

print(f"Pérdida en entrenamiento final: {final_loss:.4f}")
print(f"Pérdida en validación final: {final_val_loss:.4f}")
print(f"Precisión en entrenamiento final: {final_accuracy:.4f}")
print(f"Precisión en validación final: {final_val_accuracy:.4f}")
```

```
Pérdida en entrenamiento final: 0.6862
Pérdida en validación final: 0.7037
Precisión en entrenamiento final: 0.7513
Precisión en validación final: 0.7732
```

Modelo CNN-8.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_height, img_width = 224, 224
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=True
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def segmentar_craneo(img_path, size=(100, 100), plot=False):
    img_color = cv2.imread(img_path)
    img_color = cv2.resize(img_color, size)
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
    mask = np.zeros_like(img_gray)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=-1)

    img_segmentado = cv2.bitwise_and(img_color, img_color, mask=mask)

    if plot:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')
        plt.subplot(1, 3, 2)
        plt.imshow(mask, cmap='gray')
        plt.title("Máscara cráneo")
        plt.axis('off')
        plt.subplot(1, 3, 3)
        plt.imshow(cv2.cvtColor(img_segmentado, cv2.COLOR_BGR2RGB))
        plt.title("Segmentado")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return img_segmentado
```

In [ ]:
```
import os

ruta_entrada = 'D:/PROYECTO 1/SET DE DATOS RECORTADOS'
ruta_salida = 'D:/PROYECTO 1/DATA_LIMPIA2'
img_size = (100, 100)

for subfolder in os.listdir(ruta_entrada):
    ruta_subcarpeta = os.path.join(ruta_entrada, subfolder)
    ruta_salida_sub = os.path.join(ruta_salida, subfolder)
    os.makedirs(ruta_salida_sub, exist_ok=True)

    for filename in os.listdir(ruta_subcarpeta):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            ruta_img = os.path.join(ruta_subcarpeta, filename)
            img_seg = segmentar_craneo(ruta_img, size=img_size, plot=False)

            salida_path = os.path.join(ruta_salida_sub, filename)
            cv2.imwrite(salida_path, img_seg)

print("Todas las imágenes fueron procesadas y guardadas en:", ruta_salida)
```

```
Todas las imágenes fueron procesadas y guardadas en: D:/PROYECTO 1/DATA_LIMPIA2
```

In [ ]:
```
def aplicar_mascara_craneo(img_path, size=(100, 100)):
    import cv2
    img = cv2.imread(img_path)
    img = cv2.resize(img, size)
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    center = (size[0] // 2, size[1] // 2)
    radius = int(min(size) * 0.45)
    cv2.circle(mask, center, radius, 255, thickness=-1)
```

```
    masked_img = cv2.bitwise_and(img, img, mask=mask)
    return masked_img
```

In [ ]:
```
img_size = (100, 100)
batch_size = 32
epochs = 100
input_shape = (100, 100, 3)

print('Configuración definida.')
```

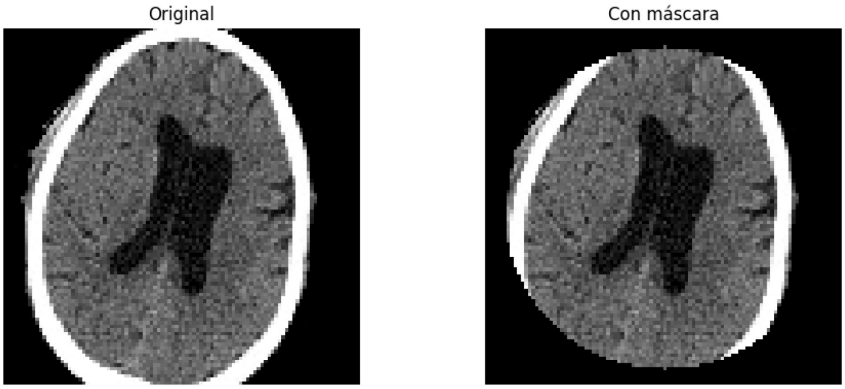
```
Configuración definida.
```

In [ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode="nearest"
)

train_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
img_path = r"D:\PROYECTO 1\DATA_LIMPIA2\CON HEMORRAGIA\1.2.840.113704.1.111.416.172
img_original = cv2.imread(img_path)
img_original = cv2.resize(img_original, img_size)
img_mascara = aplicar_mascara_craneo(img_path)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_mascara, cv2.COLOR_BGR2RGB))
plt.title('Con máscara')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



In [ ]:
```
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Conv2D(70, (3, 3), activation='relu', input_shape=input_shape,
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Flatten())
model.add(Dense(70, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.3377))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Adam(learning_rate=0.0002309686020041116),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

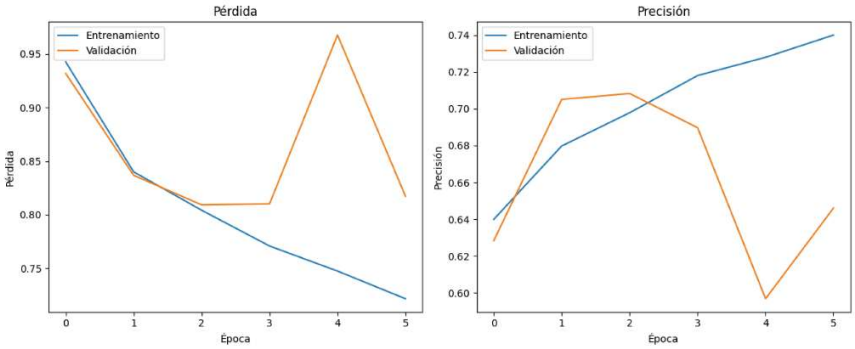
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 70)        1960

 batch_normalization (Batch  (None, 98, 98, 70)        280
 Normalization)

 max_pooling2d (MaxPooling2  (None, 32, 32, 70)        0
 D)

 conv2d_1 (Conv2D)           (None, 30, 30, 70)        44170

 batch_normalization_1 (Bat  (None, 30, 30, 70)        280
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 70)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 70)          44170

 batch_normalization_2 (Bat  (None, 8, 8, 70)          280
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 70)          0
 g2D)

 flatten (Flatten)           (None, 280)               0

 dense (Dense)               (None, 70)                19670

 dropout (Dropout)           (None, 70)                0

 dense_1 (Dense)             (None, 1)                 71

=================================================================
Total params: 110881 (433.13 KB)
Trainable params: 110461 (431.49 KB)
Non-trainable params: 420 (1.64 KB)
_________________________________________________________________
```

In [ ]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=30, factor=0.5)

history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stop, reduce_lr]
)

print('Entrenamiento completado.')
```

```
Epoch 1/100
310/310 [==============================] - 190s 596ms/step - loss: 0.9425 - accurac
y: 0.6399 - val_loss: 0.9317 - val_accuracy: 0.6283 - lr: 2.3097e-04
Epoch 2/100
310/310 [==============================] - 167s 538ms/step - loss: 0.8400 - accurac
y: 0.6797 - val_loss: 0.8367 - val_accuracy: 0.7050 - lr: 2.3097e-04
Epoch 3/100
310/310 [==============================] - 179s 576ms/step - loss: 0.8043 - accurac
y: 0.6978 - val_loss: 0.8093 - val_accuracy: 0.7082 - lr: 2.3097e-04
Epoch 4/100
310/310 [==============================] - 169s 544ms/step - loss: 0.7709 - accurac
y: 0.7180 - val_loss: 0.8102 - val_accuracy: 0.6897 - lr: 2.3097e-04
Epoch 5/100
310/310 [==============================] - 189s 608ms/step - loss: 0.7476 - accurac
y: 0.7279 - val_loss: 0.9674 - val_accuracy: 0.5969 - lr: 2.3097e-04
Epoch 6/100
310/310 [==============================] - 189s 607ms/step - loss: 0.7218 - accurac
y: 0.7400 - val_loss: 0.8173 - val_accuracy: 0.6461 - lr: 2.3097e-04
Entrenamiento completado.
```

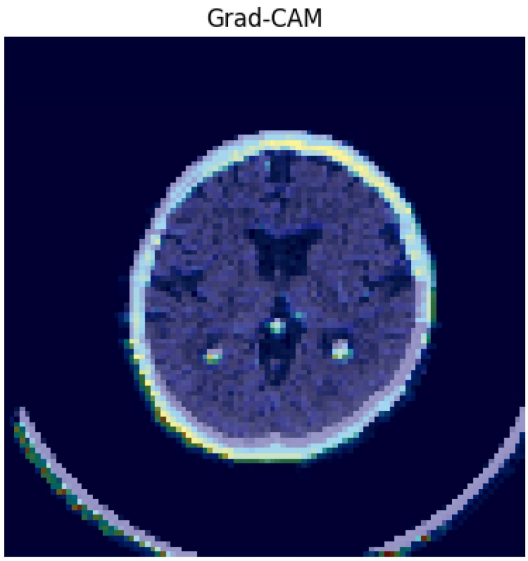
In [ ]:

```python
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.savefig("curvas_entrenamiento.png")
plt.show()
```



In [ ]:
```
def apply_gradcam(model, img_path, layer_name='conv2d_2'):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=img_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    grad_model = Model([model.inputs], [model.get_layer(layer_name).output, model.o

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[0]

    grads = tape.gradient(loss, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=(0, 1))
    cam = np.zeros(conv_outputs.shape[1:3], dtype=np.float32)

    for i, w in enumerate(weights):
        cam += w * conv_outputs[0, :, :, i]

    cam = np.maximum(cam, 0)
    cam = cam / np.max(cam)
    cam = cv2.resize(cam, img_size)
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)


    img_original = cv2.imread(img_path)
    img_original = cv2.resize(img_original, img_size)
    superimposed_img = cv2.addWeighted(img_original, 0.6, heatmap, 0.4, 0)

    plt.imshow(cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Grad-CAM")
    plt.show()
```

In [ ]:
```
test_img = r"D:\DOCUMENTOS COMPU\PROTOCOLOS\MACHINE LEARNING YO\MUESTRA TESIS PNGS\
apply_gradcam(model, test_img, layer_name='conv2d_48')
```



In [ ]:
```
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]

print(f"Pérdida en entrenamiento final: {final_loss:.4f}")
print(f"Pérdida en validación final: {final_val_loss:.4f}")
print(f"Precisión en entrenamiento final: {final_accuracy:.4f}")
print(f"Precisión en validación final: {final_val_accuracy:.4f}")
```

```
Pérdida en entrenamiento final: 0.7218
Pérdida en validación final: 0.8173
Precisión en entrenamiento final: 0.7400
Precisión en validación final: 0.6461
```

In [ ]:
```
model.save('modelo_completo2.h5')
model.save_weights('pesos_modelo2.h5')
```

Modelo CNN-9.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_height, img_width = 224, 224
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=True
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def segmentar_craneo(img_path, size=(100, 100), plot=False):
    img_color = cv2.imread(img_path)
    img_color = cv2.resize(img_color, size)
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
    mask = np.zeros_like(img_gray)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=-1)

    img_segmentado = cv2.bitwise_and(img_color, img_color, mask=mask)

    if plot:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')
        plt.subplot(1, 3, 2)
        plt.imshow(mask, cmap='gray')
        plt.title("Máscara cráneo")
        plt.axis('off')
        plt.subplot(1, 3, 3)
        plt.imshow(cv2.cvtColor(img_segmentado, cv2.COLOR_BGR2RGB))
        plt.title("Segmentado")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return img_segmentado
```

In [ ]:
```
import os

ruta_entrada = 'D:/PROYECTO 1/SET DE DATOS RECORTADOS'
ruta_salida = 'D:/PROYECTO 1/DATA_LIMPIA2'
img_size = (100, 100)

for subfolder in os.listdir(ruta_entrada):
    ruta_subcarpeta = os.path.join(ruta_entrada, subfolder)
    ruta_salida_sub = os.path.join(ruta_salida, subfolder)
    os.makedirs(ruta_salida_sub, exist_ok=True)

    for filename in os.listdir(ruta_subcarpeta):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            ruta_img = os.path.join(ruta_subcarpeta, filename)
            img_seg = segmentar_craneo(ruta_img, size=img_size, plot=False)

            salida_path = os.path.join(ruta_salida_sub, filename)
            cv2.imwrite(salida_path, img_seg)

print("Todas las imágenes fueron procesadas y guardadas en:", ruta_salida)
```

```
Todas las imágenes fueron procesadas y guardadas en: D:/PROYECTO 1/DATA_LIMPIA2
```

In [ ]:
```
def aplicar_mascara_craneo(img_path, size=(100, 100)):
    import cv2
    img = cv2.imread(img_path)
    img = cv2.resize(img, size)
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    center = (size[0] // 2, size[1] // 2)
    radius = int(min(size) * 0.45)
    cv2.circle(mask, center, radius, 255, thickness=-1)
```

```
    masked_img = cv2.bitwise_and(img, img, mask=mask)
    return masked_img
```

In [ ]:
```
img_size = (100, 100)
batch_size = 16
epochs = 100
input_shape = (100, 100, 3)

print('Configuración definida.')
```

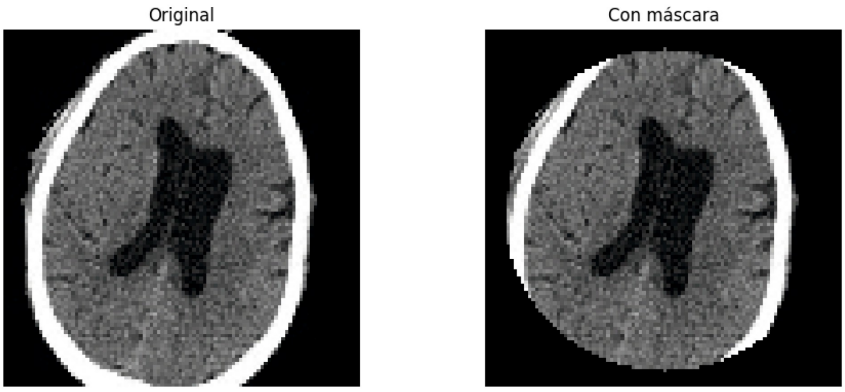
```
Configuración definida.
```

In [ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    brightness_range=[0.8, 1.2],
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode="nearest"
)

train_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
img_path = r"D:\PROYECTO 1\DATA_LIMPIA2\CON HEMORRAGIA\1.2.840.113704.1.111.416.172
img_original = cv2.imread(img_path)
img_original = cv2.resize(img_original, img_size)
img_mascara = aplicar_mascara_craneo(img_path)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_mascara, cv2.COLOR_BGR2RGB))
plt.title('Con máscara')
```

```
plt.axis('off')
plt.tight_layout()
plt.show()
```



In [ ]:
```
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Conv2D(70, (3, 3), activation='relu', input_shape=input_shape,
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Flatten())
model.add(Dense(70, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Adam(learning_rate=0.0002309686020041116),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

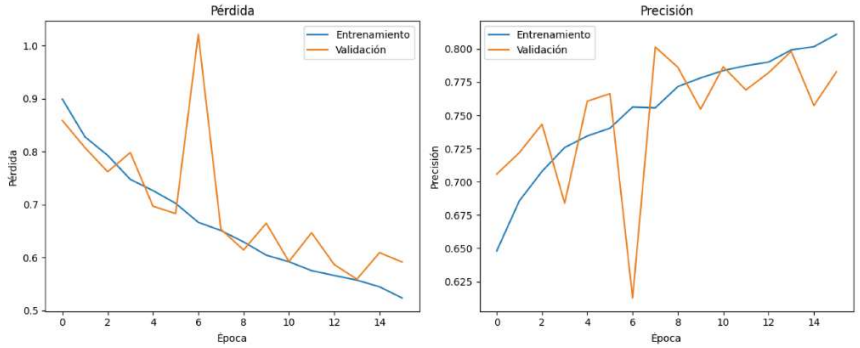
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 70)        1960

 batch_normalization (Batch  (None, 98, 98, 70)        280
 Normalization)

 max_pooling2d (MaxPooling2  (None, 32, 32, 70)        0
 D)

 conv2d_1 (Conv2D)           (None, 30, 30, 70)        44170

 batch_normalization_1 (Bat  (None, 30, 30, 70)        280
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 70)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 70)          44170

 batch_normalization_2 (Bat  (None, 8, 8, 70)          280
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 70)          0
 g2D)

 flatten (Flatten)           (None, 280)               0

 dense (Dense)               (None, 70)                19670

 dropout (Dropout)           (None, 70)                0

 dense_1 (Dense)             (None, 1)                 71

=================================================================
Total params: 110881 (433.13 KB)
Trainable params: 110461 (431.49 KB)
Non-trainable params: 420 (1.64 KB)
_________________________________________________________________
```

In [ ]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=30, factor=0.5)

history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stop, reduce_lr]
)

print('Entrenamiento completado.')
```

```
Epoch 1/100
620/620 [==============================] - 202s 319ms/step - loss: 0.8987 - accurac
y: 0.6480 - val_loss: 0.8585 - val_accuracy: 0.7058 - lr: 2.3097e-04
Epoch 2/100
620/620 [==============================] - 212s 341ms/step - loss: 0.8275 - accurac
y: 0.6857 - val_loss: 0.8071 - val_accuracy: 0.7220 - lr: 2.3097e-04
Epoch 3/100
620/620 [==============================] - 199s 320ms/step - loss: 0.7926 - accurac
y: 0.7079 - val_loss: 0.7617 - val_accuracy: 0.7433 - lr: 2.3097e-04
Epoch 4/100
620/620 [==============================] - 199s 320ms/step - loss: 0.7473 - accurac
y: 0.7258 - val_loss: 0.7979 - val_accuracy: 0.6840 - lr: 2.3097e-04
Epoch 5/100
620/620 [==============================] - 197s 317ms/step - loss: 0.7262 - accurac
y: 0.7345 - val_loss: 0.6967 - val_accuracy: 0.7607 - lr: 2.3097e-04
Epoch 6/100
620/620 [==============================] - 199s 321ms/step - loss: 0.7021 - accurac
y: 0.7404 - val_loss: 0.6828 - val_accuracy: 0.7663 - lr: 2.3097e-04
Epoch 7/100
620/620 [==============================] - 193s 311ms/step - loss: 0.6663 - accurac
y: 0.7563 - val_loss: 1.0210 - val_accuracy: 0.6126 - lr: 2.3097e-04
Epoch 8/100
620/620 [==============================] - 214s 345ms/step - loss: 0.6508 - accurac
y: 0.7557 - val_loss: 0.6531 - val_accuracy: 0.8015 - lr: 2.3097e-04
Epoch 9/100
620/620 [==============================] - 207s 334ms/step - loss: 0.6292 - accurac
y: 0.7717 - val_loss: 0.6142 - val_accuracy: 0.7861 - lr: 2.3097e-04
Epoch 10/100
620/620 [==============================] - 224s 361ms/step - loss: 0.6044 - accurac
y: 0.7782 - val_loss: 0.6646 - val_accuracy: 0.7546 - lr: 2.3097e-04
Epoch 11/100
620/620 [==============================] - 229s 368ms/step - loss: 0.5919 - accurac
y: 0.7837 - val_loss: 0.5923 - val_accuracy: 0.7865 - lr: 2.3097e-04
Epoch 12/100
620/620 [==============================] - 229s 370ms/step - loss: 0.5751 - accurac
y: 0.7874 - val_loss: 0.6466 - val_accuracy: 0.7692 - lr: 2.3097e-04
Epoch 13/100
620/620 [==============================] - 225s 363ms/step - loss: 0.5660 - accurac
y: 0.7901 - val_loss: 0.5867 - val_accuracy: 0.7821 - lr: 2.3097e-04
Epoch 14/100
620/620 [==============================] - 243s 392ms/step - loss: 0.5572 - accurac
y: 0.7993 - val_loss: 0.5590 - val_accuracy: 0.7982 - lr: 2.3097e-04
Epoch 15/100
620/620 [==============================] - 215s 347ms/step - loss: 0.5447 - accurac
y: 0.8017 - val_loss: 0.6090 - val_accuracy: 0.7575 - lr: 2.3097e-04
Epoch 16/100
620/620 [==============================] - 212s 341ms/step - loss: 0.5234 - accurac
y: 0.8110 - val_loss: 0.5914 - val_accuracy: 0.7829 - lr: 2.3097e-04
Entrenamiento completado.
```

In [ ]:
```python
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
```

```
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.savefig("curvas_entrenamiento.png")
plt.show()
```



In [ ]:
```
def apply_gradcam(model, img_path, layer_name='conv2d_2'):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=img_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    grad_model = Model([model.inputs], [model.get_layer(layer_name).output, model.o

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[0]

    grads = tape.gradient(loss, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=(0, 1))
    cam = np.zeros(conv_outputs.shape[1:3], dtype=np.float32)

    for i, w in enumerate(weights):
        cam += w * conv_outputs[0, :, :, i]

    cam = np.maximum(cam, 0)
    cam = cam / np.max(cam)
    cam = cv2.resize(cam, img_size)
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)
```

```
    img_original = cv2.imread(img_path)
    img_original = cv2.resize(img_original, img_size)
    superimposed_img = cv2.addWeighted(img_original, 0.6, heatmap, 0.4, 0)

    plt.imshow(cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Grad-CAM")
    plt.show()
```

In [ ]:
```
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]

print(f"Pérdida en entrenamiento final: {final_loss:.4f}")
print(f"Pérdida en validación final: {final_val_loss:.4f}")
print(f"Precisión en entrenamiento final: {final_accuracy:.4f}")
print(f"Precisión en validación final: {final_val_accuracy:.4f}")
```

```
Pérdida en entrenamiento final: 0.5234
Pérdida en validación final: 0.5914
Precisión en entrenamiento final: 0.8110
Precisión en validación final: 0.7829
```

In [ ]:
```
model.save('modelo_completo3.h5')
model.save_weights('pesos_modelo3.h5')
```

Modelo CNN-10.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_height, img_width = 224, 224
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=True
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def segmentar_craneo(img_path, size=(100, 100), plot=False):
    img_color = cv2.imread(img_path)
    img_color = cv2.resize(img_color, size)
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
    mask = np.zeros_like(img_gray)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=-1)

    img_segmentado = cv2.bitwise_and(img_color, img_color, mask=mask)

    if plot:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')
        plt.subplot(1, 3, 2)
        plt.imshow(mask, cmap='gray')
        plt.title("Máscara cráneo")
        plt.axis('off')
        plt.subplot(1, 3, 3)
        plt.imshow(cv2.cvtColor(img_segmentado, cv2.COLOR_BGR2RGB))
        plt.title("Segmentado")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return img_segmentado
```

In [ ]:
```
import os

ruta_entrada = 'D:/PROYECTO 1/SET DE DATOS RECORTADOS'
ruta_salida = 'D:/PROYECTO 1/DATA_LIMPIA2'
img_size = (100, 100)

for subfolder in os.listdir(ruta_entrada):
    ruta_subcarpeta = os.path.join(ruta_entrada, subfolder)
    ruta_salida_sub = os.path.join(ruta_salida, subfolder)
    os.makedirs(ruta_salida_sub, exist_ok=True)

    for filename in os.listdir(ruta_subcarpeta):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            ruta_img = os.path.join(ruta_subcarpeta, filename)
            img_seg = segmentar_craneo(ruta_img, size=img_size, plot=False)

            salida_path = os.path.join(ruta_salida_sub, filename)
            cv2.imwrite(salida_path, img_seg)

print("Todas las imágenes fueron procesadas y guardadas en:", ruta_salida)
```

```
Todas las imágenes fueron procesadas y guardadas en: D:/PROYECTO 1/DATA_LIMPIA2
```

In [ ]:
```
def aplicar_mascara_craneo(img_path, size=(100, 100)):
    import cv2
    img = cv2.imread(img_path)
    img = cv2.resize(img, size)
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    center = (size[0] // 2, size[1] // 2)
    radius = int(min(size) * 0.45)
    cv2.circle(mask, center, radius, 255, thickness=-1)
```

```
    masked_img = cv2.bitwise_and(img, img, mask=mask)
    return masked_img
```

In [ ]:
```
img_size = (100, 100)
batch_size = 32
epochs = 100
input_shape = (100, 100, 3)

print('Configuración definida.')
```

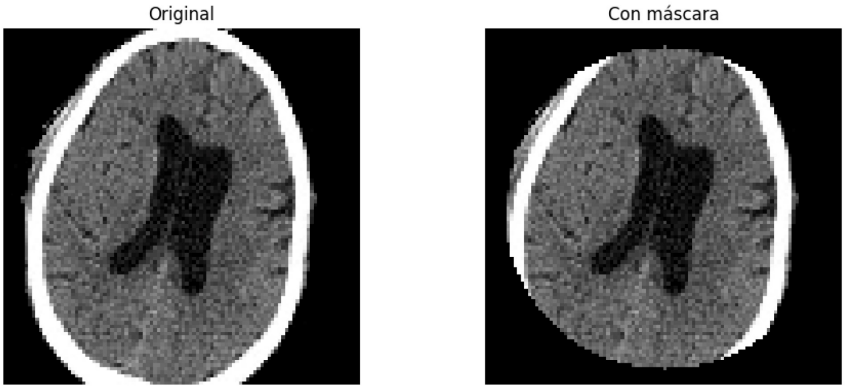
```
Configuración definida.
```

In [ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode="nearest"
)

train_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
img_path = r"D:\PROYECTO 1\DATA_LIMPIA2\CON HEMORRAGIA\1.2.840.113704.1.111.416.172
img_original = cv2.imread(img_path)
img_original = cv2.resize(img_original, img_size)
img_mascara = aplicar_mascara_craneo(img_path)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_mascara, cv2.COLOR_BGR2RGB))
plt.title('Con máscara')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



In [ ]:
```
from tensorflow.keras import regularizers

model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=input_shape,
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(64, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(64, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Flatten())
model.add(Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.31315315703867835))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000337164615819957
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

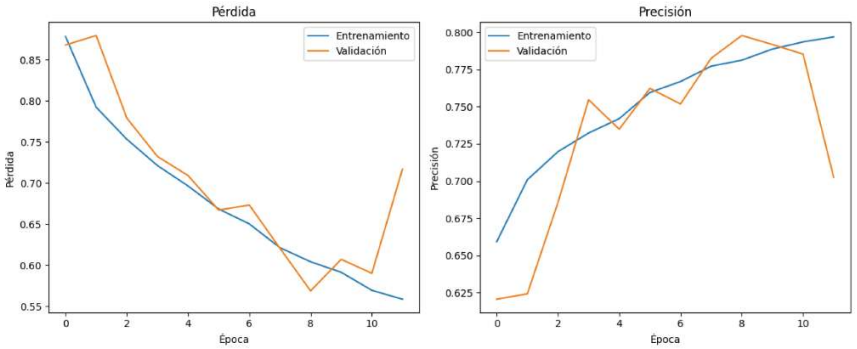
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 64)        1792

 batch_normalization (Batch  (None, 98, 98, 64)        256
 Normalization)

 max_pooling2d (MaxPooling2  (None, 32, 32, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 30, 30, 64)        36928

 batch_normalization_1 (Bat  (None, 30, 30, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          36928

 batch_normalization_2 (Bat  (None, 8, 8, 64)          256
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 64)          0
 g2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 64)                16448

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 92929 (363.00 KB)
Trainable params: 92545 (361.50 KB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

In [ ]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=30, factor=0.5)

history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stop, reduce_lr]
)

print('Entrenamiento completado.')
```

```
Epoch 1/100
310/310 [==============================] - 186s 586ms/step - loss: 0.8782 - accurac
y: 0.6593 - val_loss: 0.8679 - val_accuracy: 0.6207 - lr: 3.3716e-04
Epoch 2/100
310/310 [==============================] - 169s 546ms/step - loss: 0.7922 - accurac
y: 0.7009 - val_loss: 0.8795 - val_accuracy: 0.6243 - lr: 3.3716e-04
Epoch 3/100
310/310 [==============================] - 178s 574ms/step - loss: 0.7532 - accurac
y: 0.7198 - val_loss: 0.7790 - val_accuracy: 0.6856 - lr: 3.3716e-04
Epoch 4/100
310/310 [==============================] - 180s 580ms/step - loss: 0.7212 - accurac
y: 0.7323 - val_loss: 0.7323 - val_accuracy: 0.7546 - lr: 3.3716e-04
Epoch 5/100
310/310 [==============================] - 155s 498ms/step - loss: 0.6963 - accurac
y: 0.7420 - val_loss: 0.7090 - val_accuracy: 0.7349 - lr: 3.3716e-04
Epoch 6/100
310/310 [==============================] - 155s 499ms/step - loss: 0.6683 - accurac
y: 0.7595 - val_loss: 0.6670 - val_accuracy: 0.7623 - lr: 3.3716e-04
Epoch 7/100
310/310 [==============================] - 158s 508ms/step - loss: 0.6502 - accurac
y: 0.7669 - val_loss: 0.6732 - val_accuracy: 0.7518 - lr: 3.3716e-04
Epoch 8/100
310/310 [==============================] - 156s 503ms/step - loss: 0.6212 - accurac
y: 0.7772 - val_loss: 0.6205 - val_accuracy: 0.7825 - lr: 3.3716e-04
Epoch 9/100
310/310 [==============================] - 161s 518ms/step - loss: 0.6041 - accurac
y: 0.7812 - val_loss: 0.5687 - val_accuracy: 0.7978 - lr: 3.3716e-04
Epoch 10/100
310/310 [==============================] - 178s 573ms/step - loss: 0.5913 - accurac
y: 0.7887 - val_loss: 0.6070 - val_accuracy: 0.7918 - lr: 3.3716e-04
Epoch 11/100
310/310 [==============================] - 178s 575ms/step - loss: 0.5694 - accurac
y: 0.7935 - val_loss: 0.5901 - val_accuracy: 0.7853 - lr: 3.3716e-04
Epoch 12/100
310/310 [==============================] - 179s 578ms/step - loss: 0.5585 - accurac
y: 0.7969 - val_loss: 0.7170 - val_accuracy: 0.7026 - lr: 3.3716e-04
Entrenamiento completado.
```

In [ ]:
```python
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()
```

```
plt.tight_layout()
plt.savefig("curvas_entrenamiento.png")
plt.show()
```



In [ ]:
```
def apply_gradcam(model, img_path, layer_name='conv2d_2'):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=img_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    grad_model = Model([model.inputs], [model.get_layer(layer_name).output, model.o

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[0]

    grads = tape.gradient(loss, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=(0, 1))
    cam = np.zeros(conv_outputs.shape[1:3], dtype=np.float32)

    for i, w in enumerate(weights):
        cam += w * conv_outputs[0, :, :, i]

    cam = np.maximum(cam, 0)
    cam = cam / np.max(cam)
    cam = cv2.resize(cam, img_size)
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)


    img_original = cv2.imread(img_path)
    img_original = cv2.resize(img_original, img_size)
    superimposed_img = cv2.addWeighted(img_original, 0.6, heatmap, 0.4, 0)

    plt.imshow(cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Grad-CAM")
    plt.show()
```

In [ ]:
```
test_img = r"D:\DOCUMENTOS COMPU\PROTOCOLOS\MACHINE LEARNING YO\MUESTRA TESIS PNGS\
apply_gradcam(model, test_img, layer_name='conv2d_2')
```

In [ ]:
```
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
```

```
final_val_accuracy = history.history['val_accuracy'][-1]

print(f"Pérdida en entrenamiento final: {final_loss:.4f}")
print(f"Pérdida en validación final: {final_val_loss:.4f}")
print(f"Precisión en entrenamiento final: {final_accuracy:.4f}")
print(f"Precisión en validación final: {final_val_accuracy:.4f}")
```

```
Pérdida en entrenamiento final: 0.5585
Pérdida en validación final: 0.7170
Precisión en entrenamiento final: 0.7969
Precisión en validación final: 0.7026
```

In [ ]:
```
model.save('modelo_completo10.h5')
model.save_weights('pesos_modelo10.h5')
```

Modelo CNN-11.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_height, img_width = 224, 224
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=True
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def segmentar_craneo(img_path, size=(100, 100), plot=False):
    img_color = cv2.imread(img_path)
    img_color = cv2.resize(img_color, size)
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
    mask = np.zeros_like(img_gray)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=-1)

    img_segmentado = cv2.bitwise_and(img_color, img_color, mask=mask)

    if plot:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')
        plt.subplot(1, 3, 2)
        plt.imshow(mask, cmap='gray')
        plt.title("Máscara cráneo")
        plt.axis('off')
        plt.subplot(1, 3, 3)
        plt.imshow(cv2.cvtColor(img_segmentado, cv2.COLOR_BGR2RGB))
        plt.title("Segmentado")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return img_segmentado
```

In [ ]:
```
import os

ruta_entrada = 'D:/PROYECTO 1/SET DE DATOS RECORTADOS'
ruta_salida = 'D:/PROYECTO 1/DATA_LIMPIA2's
img_size = (100, 100)

for subfolder in os.listdir(ruta_entrada):
    ruta_subcarpeta = os.path.join(ruta_entrada, subfolder)
    ruta_salida_sub = os.path.join(ruta_salida, subfolder)
    os.makedirs(ruta_salida_sub, exist_ok=True)

    for filename in os.listdir(ruta_subcarpeta):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            ruta_img = os.path.join(ruta_subcarpeta, filename)
            img_seg = segmentar_craneo(ruta_img, size=img_size, plot=False)

            salida_path = os.path.join(ruta_salida_sub, filename)
            cv2.imwrite(salida_path, img_seg)

print("Todas las imágenes fueron procesadas y guardadas en:", ruta_salida)
```

```
Todas las imágenes fueron procesadas y guardadas en: D:/PROYECTO 1/DATA_LIMPIA2
```

In [ ]:
```
def aplicar_mascara_craneo(img_path, size=(100, 100)):
    import cv2
    img = cv2.imread(img_path)
    img = cv2.resize(img, size)
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    center = (size[0] // 2, size[1] // 2)
    radius = int(min(size) * 0.45)
    cv2.circle(mask, center, radius, 255, thickness=-1)
```

```
    masked_img = cv2.bitwise_and(img, img, mask=mask)
    return masked_img
```

In [ ]:
```
img_size = (100, 100)
batch_size = 32
epochs = 100
input_shape = (100, 100, 3)

print('Configuración definida.')
```

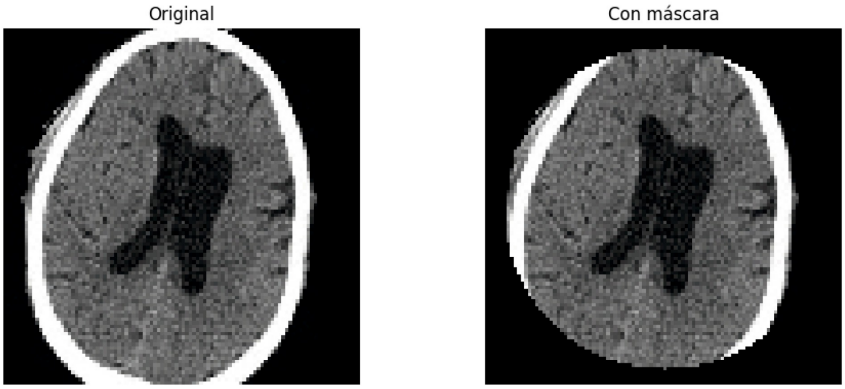
```
Configuración definida.
```

In [ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode="nearest"
)

train_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
img_path = r"D:\PROYECTO 1\DATA_LIMPIA2\CON HEMORRAGIA\1.2.840.113704.1.111.416.172
img_original = cv2.imread(img_path)
img_original = cv2.resize(img_original, img_size)
img_mascara = aplicar_mascara_craneo(img_path)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_mascara, cv2.COLOR_BGR2RGB))
plt.title('Con máscara')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



In [ ]:
```
from tensorflow.keras import regularizers

model = Sequential()
model.add(Conv2D(100, (3, 3), activation='relu', input_shape=input_shape,
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(100, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(100, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Flatten())
model.add(Dense(100, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.3043120680453069))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000151809425428442
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

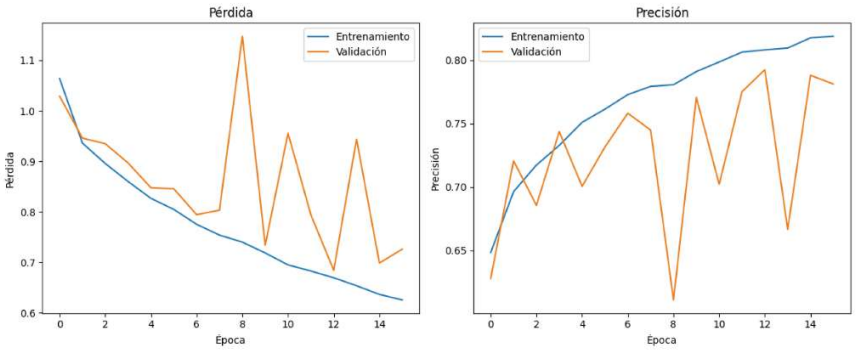
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 100)       2800

 batch_normalization (Batch  (None, 98, 98, 100)       400
 Normalization)

 max_pooling2d (MaxPooling2  (None, 32, 32, 100)       0
 D)

 conv2d_1 (Conv2D)           (None, 30, 30, 100)       90100

 batch_normalization_1 (Bat  (None, 30, 30, 100)       400
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 100)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 100)         90100

 batch_normalization_2 (Bat  (None, 8, 8, 100)         400
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 100)         0
 g2D)

 flatten (Flatten)           (None, 400)               0

 dense (Dense)               (None, 100)               40100

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 224401 (876.57 KB)
Trainable params: 223801 (874.22 KB)
Non-trainable params: 600 (2.34 KB)
_________________________________________________________________
```

In [ ]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=30, factor=0.5)

history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stop, reduce_lr]
)

print('Entrenamiento completado.')
```

```
Epoch 1/100
310/310 [==============================] - 267s 845ms/step - loss: 1.0640 - accurac
y: 0.6484 - val_loss: 1.0291 - val_accuracy: 0.6279 - lr: 1.5181e-04
Epoch 2/100
310/310 [==============================] - 237s 765ms/step - loss: 0.9361 - accurac
y: 0.6964 - val_loss: 0.9461 - val_accuracy: 0.7207 - lr: 1.5181e-04
Epoch 3/100
310/310 [==============================] - 226s 729ms/step - loss: 0.8959 - accurac
y: 0.7174 - val_loss: 0.9349 - val_accuracy: 0.6856 - lr: 1.5181e-04
Epoch 4/100
310/310 [==============================] - 233s 752ms/step - loss: 0.8601 - accurac
y: 0.7329 - val_loss: 0.8968 - val_accuracy: 0.7437 - lr: 1.5181e-04
Epoch 5/100
310/310 [==============================] - 254s 820ms/step - loss: 0.8268 - accurac
y: 0.7511 - val_loss: 0.8478 - val_accuracy: 0.7006 - lr: 1.5181e-04
Epoch 6/100
310/310 [==============================] - 243s 785ms/step - loss: 0.8050 - accurac
y: 0.7614 - val_loss: 0.8458 - val_accuracy: 0.7316 - lr: 1.5181e-04
Epoch 7/100
310/310 [==============================] - 234s 754ms/step - loss: 0.7755 - accurac
y: 0.7729 - val_loss: 0.7945 - val_accuracy: 0.7583 - lr: 1.5181e-04
Epoch 8/100
310/310 [==============================] - 246s 792ms/step - loss: 0.7541 - accurac
y: 0.7794 - val_loss: 0.8035 - val_accuracy: 0.7450 - lr: 1.5181e-04
Epoch 9/100
310/310 [==============================] - 244s 787ms/step - loss: 0.7402 - accurac
y: 0.7807 - val_loss: 1.1476 - val_accuracy: 0.6110 - lr: 1.5181e-04
Epoch 10/100
310/310 [==============================] - 246s 793ms/step - loss: 0.7188 - accurac
y: 0.7911 - val_loss: 0.7342 - val_accuracy: 0.7708 - lr: 1.5181e-04
Epoch 11/100
310/310 [==============================] - 234s 755ms/step - loss: 0.6952 - accurac
y: 0.7987 - val_loss: 0.9557 - val_accuracy: 0.7022 - lr: 1.5181e-04
Epoch 12/100
310/310 [==============================] - 238s 767ms/step - loss: 0.6831 - accurac
y: 0.8064 - val_loss: 0.7943 - val_accuracy: 0.7752 - lr: 1.5181e-04
Epoch 13/100
310/310 [==============================] - 215s 692ms/step - loss: 0.6697 - accurac
y: 0.8083 - val_loss: 0.6845 - val_accuracy: 0.7926 - lr: 1.5181e-04
Epoch 14/100
310/310 [==============================] - 203s 654ms/step - loss: 0.6539 - accurac
y: 0.8097 - val_loss: 0.9437 - val_accuracy: 0.6667 - lr: 1.5181e-04
Epoch 15/100
310/310 [==============================] - 190s 612ms/step - loss: 0.6368 - accurac
y: 0.8177 - val_loss: 0.6988 - val_accuracy: 0.7881 - lr: 1.5181e-04
Epoch 16/100
310/310 [==============================] - 184s 595ms/step - loss: 0.6258 - accurac
y: 0.8189 - val_loss: 0.7263 - val_accuracy: 0.7813 - lr: 1.5181e-04
Entrenamiento completado.
```

In [ ]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
```

```
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

# Precisión
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.savefig("curvas_entrenamiento.png")
plt.show()
```



In [ ]:
```
def apply_gradcam(model, img_path, layer_name='conv2d_2'):
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=img_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    grad_model = Model([model.inputs], [model.get_layer(layer_name).output, model.o

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[0]

    grads = tape.gradient(loss, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=(0, 1))
    cam = np.zeros(conv_outputs.shape[1:3], dtype=np.float32)

    for i, w in enumerate(weights):
        cam += w * conv_outputs[0, :, :, i]

    cam = np.maximum(cam, 0)
    cam = cam / np.max(cam)
    cam = cv2.resize(cam, img_size)  # <--- CORREGIDO
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)
```

```
    img_original = cv2.imread(img_path)
    img_original = cv2.resize(img_original, img_size)
    superimposed_img = cv2.addWeighted(img_original, 0.6, heatmap, 0.4, 0)

    plt.imshow(cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title("Grad-CAM")
    plt.show()
```

In [ ]:
```
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]

print(f"Pérdida en entrenamiento final: {final_loss:.4f}")
print(f"Pérdida en validación final: {final_val_loss:.4f}")
print(f"Precisión en entrenamiento final: {final_accuracy:.4f}")
print(f"Precisión en validación final: {final_val_accuracy:.4f}")
```

```
Pérdida en entrenamiento final: 0.6258
Pérdida en validación final: 0.7263
Precisión en entrenamiento final: 0.8189
Precisión en validación final: 0.7813
```

In [ ]:
```
model.save('modelo_completo11.h5')
model.save_weights('pesos_modelo11.h5')
```

Modelo CNN-12.

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

In [ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_height, img_width = 224, 224
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=True
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def segmentar_craneo(img_path, size=(100, 100), plot=False):
    img_color = cv2.imread(img_path)
    img_color = cv2.resize(img_color, size)
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
    mask = np.zeros_like(img_gray)
```

```
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        cv2.drawContours(mask, [largest_contour], -1, 255, thickness=-1)

    img_segmentado = cv2.bitwise_and(img_color, img_color, mask=mask)

    if plot:
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')
        plt.subplot(1, 3, 2)
        plt.imshow(mask, cmap='gray')
        plt.title("Máscara cráneo")
        plt.axis('off')
        plt.subplot(1, 3, 3)
        plt.imshow(cv2.cvtColor(img_segmentado, cv2.COLOR_BGR2RGB))
        plt.title("Segmentado")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return img_segmentado
```

In [ ]:
```
import os

ruta_entrada = 'D:/PROYECTO 1/SET DE DATOS RECORTADOS'
ruta_salida = 'D:/PROYECTO 1/DATA_LIMPIA2'
img_size = (100, 100)

for subfolder in os.listdir(ruta_entrada):
    ruta_subcarpeta = os.path.join(ruta_entrada, subfolder)
    ruta_salida_sub = os.path.join(ruta_salida, subfolder)
    os.makedirs(ruta_salida_sub, exist_ok=True)

    for filename in os.listdir(ruta_subcarpeta):
        if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
            ruta_img = os.path.join(ruta_subcarpeta, filename)
            img_seg = segmentar_craneo(ruta_img, size=img_size, plot=False)

            salida_path = os.path.join(ruta_salida_sub, filename)
            cv2.imwrite(salida_path, img_seg)

print("Todas las imágenes fueron procesadas y guardadas en:", ruta_salida)
```

```
Todas las imágenes fueron procesadas y guardadas en: D:/PROYECTO 1/DATA_LIMPIA2
```

In [ ]:
```
def aplicar_mascara_craneo(img_path, size=(100, 100)):
    import cv2
    img = cv2.imread(img_path)
    img = cv2.resize(img, size)
    mask = np.zeros(img.shape[:2], dtype=np.uint8)
    center = (size[0] // 2, size[1] // 2)
    radius = int(min(size) * 0.45)
    cv2.circle(mask, center, radius, 255, thickness=-1)
```

```
    masked_img = cv2.bitwise_and(img, img, mask=mask)
    return masked_img
```

In [ ]:
```
img_size = (100, 100)
batch_size = 32
epochs = 100
input_shape = (100, 100, 3)

print('Configuración definida.')
```

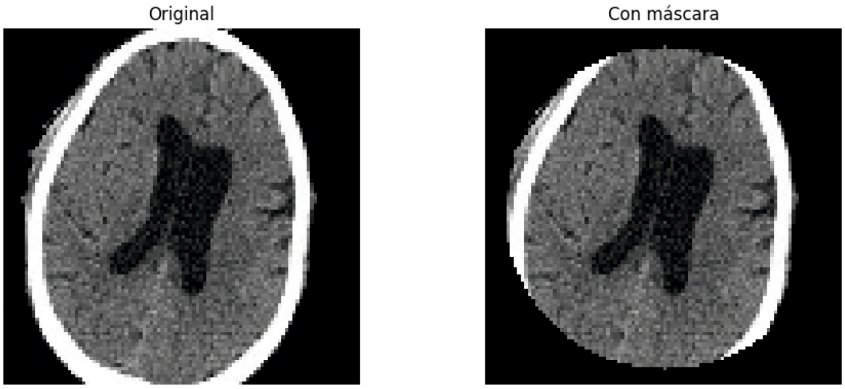
```
Configuración definida.
```

In [ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode="nearest"
)

train_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    'D:/PROYECTO 1/DATA_LIMPIA2',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

```
Found 9914 images belonging to 2 classes.
Found 2478 images belonging to 2 classes.
```

In [ ]:
```
img_path = r"D:\PROYECTO 1\DATA_LIMPIA2\CON HEMORRAGIA\1.2.840.113704.1.111.416.172
img_original = cv2.imread(img_path)
img_original = cv2.resize(img_original, img_size)
img_mascara = aplicar_mascara_craneo(img_path)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_mascara, cv2.COLOR_BGR2RGB))
plt.title('Con máscara')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



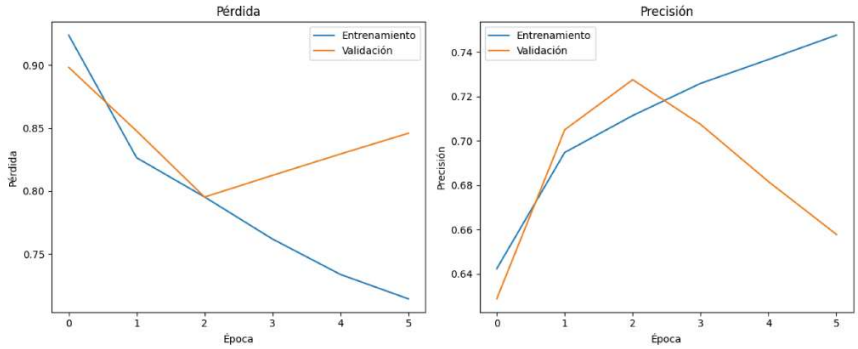
In [ ]:
```
from tensorflow.keras import regularizers

model = Sequential()
model.add(Conv2D(70, (3, 3), activation='relu', input_shape=input_shape,
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Conv2D(70, (3, 3), activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(BatchNormalization())
model.add(MaxPooling2D(3, 3))

model.add(Flatten())
model.add(Dense(70, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.3377373637135522))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.000230906860200411
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 98, 70)        1960

 batch_normalization (Batch  (None, 98, 98, 70)        280
 Normalization)

 max_pooling2d (MaxPooling2  (None, 32, 32, 70)        0
 D)

 conv2d_1 (Conv2D)           (None, 30, 30, 70)        44170

 batch_normalization_1 (Bat  (None, 30, 30, 70)        280
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 70)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 70)          44170

 batch_normalization_2 (Bat  (None, 8, 8, 70)          280
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 2, 2, 70)          0
 g2D)

 flatten (Flatten)           (None, 280)               0

 dense (Dense)               (None, 70)                19670

 dropout (Dropout)           (None, 70)                0

 dense_1 (Dense)             (None, 1)                 71

=================================================================
Total params: 110881 (433.13 KB)
Trainable params: 110461 (431.49 KB)
Non-trainable params: 420 (1.64 KB)
_________________________________________________________________
```

In [ ]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=30, factor=0.5)

history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stop, reduce_lr]
)

print('Entrenamiento completado.')
```

```
Epoch 1/100
310/310 [==============================] - 164s 513ms/step - loss: 0.9237 - accurac
y: 0.6423 - val_loss: 0.8982 - val_accuracy: 0.6287 - lr: 2.3091e-04
Epoch 2/100
310/310 [==============================] - 148s 476ms/step - loss: 0.8265 - accurac
y: 0.6948 - val_loss: 0.8477 - val_accuracy: 0.7050 - lr: 2.3091e-04
Epoch 3/100
310/310 [==============================] - 157s 506ms/step - loss: 0.7954 - accurac
y: 0.7114 - val_loss: 0.7953 - val_accuracy: 0.7276 - lr: 2.3091e-04
Epoch 4/100
310/310 [==============================] - 158s 511ms/step - loss: 0.7620 - accurac
y: 0.7259 - val_loss: 0.8125 - val_accuracy: 0.7074 - lr: 2.3091e-04
Epoch 5/100
310/310 [==============================] - 152s 490ms/step - loss: 0.7339 - accurac
y: 0.7367 - val_loss: 0.8295 - val_accuracy: 0.6816 - lr: 2.3091e-04
Epoch 6/100
310/310 [==============================] - 148s 476ms/step - loss: 0.7145 - accurac
y: 0.7477 - val_loss: 0.8459 - val_accuracy: 0.6578 - lr: 2.3091e-04
Entrenamiento completado.
```

In [ ]:
```python
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida')
plt.xlabel('Época')
plt.ylabel('Pérdida')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.savefig("curvas_entrenamiento.png")
plt.show()
```



In [ ]:
```
final_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]

print(f"Pérdida en entrenamiento final: {final_loss:.4f}")
print(f"Pérdida en validación final: {final_val_loss:.4f}")
print(f"Precisión en entrenamiento final: {final_accuracy:.4f}")
print(f"Precisión en validación final: {final_val_accuracy:.4f}")
```

```
Pérdida en entrenamiento final: 0.7145
Pérdida en validación final: 0.8459
Precisión en entrenamiento final: 0.7477
Precisión en validación final: 0.6578
```

In [ ]:
```
model.save('modelo_completo12.h5')
model.save_weights('pesos_modelo12.h5')
```

Modelo CNN-13.

In [ ]:
```python
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image


original_data_dir = "D:/PROYECTO 1/SET DE DATOS RECORTADOS"
filtered_data_dir = "D:/PROYECTO 1/SET DE DATOS FILTRADOS"

def remove_skull(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        return None
    img = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX)
    img = np.uint8(img)
    _, mask = cv2.threshold(img, 180, 255, cv2.THRESH_BINARY)
    mask_inv = cv2.bitwise_not(mask)
    brain_only = cv2.bitwise_and(img, img, mask=mask_inv)
    return brain_only

def filtrar_dataset(origen_base, destino_base):
    os.makedirs(destino_base, exist_ok=True)
    for clase in ["CON HEMORRAGIA", "SIN HEMORRAGIA"]:
        origen_clase = os.path.join(origen_base, clase)
        destino_clase = os.path.join(destino_base, clase)
        os.makedirs(destino_clase, exist_ok=True)
        for nombre_img in tqdm(os.listdir(origen_clase), desc=f"Procesando {clase}"
            ruta_img = os.path.join(origen_clase, nombre_img)
            salida_img = os.path.join(destino_clase, nombre_img)
            if not nombre_img.lower().endswith(('.png', '.jpg', '.jpeg')):
                continue
            resultado = remove_skull(ruta_img)
            if resultado is not None:
                cv2.imwrite(salida_img, resultado)

def load_images_from_directory(directory):
    images, labels = [], []
    for label, folder in enumerate(["CON HEMORRAGIA", "SIN HEMORRAGIA"]):
        folder_path = os.path.join(directory, folder)
        for img_name in os.listdir(folder_path):
            if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img_path = os.path.join(folder_path, img_name)
                img = image.load_img(img_path, target_size=(100, 100), color_mode='
                img_array = image.img_to_array(img)
                images.append(img_array)
                labels.append(label)
```

```
    return np.array(images), np.array(labels)

filtrar_dataset(original_data_dir, filtered_data_dir)
X, y = load_images_from_directory(filtered_data_dir)
X = X / 255.0

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15, random_stat
print(f"Tamaño de X_train: {X_train.shape}, X_val: {X_val.shape}")

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weight
history = model.fit(
    datagen.flow(X_train, y_train, batch_size=32),
    epochs=100,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping]
)

val_loss, val_acc = model.evaluate(X_val, y_val)
print(f"Validación - Pérdida: {val_loss:.4f}, Precisión: {val_acc:.4f}")

plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Precisión de Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión de Validación')
plt.title('Precisión durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento')
```

```
plt.plot(history.history['val_loss'], label='Pérdida de Validación')
plt.title('Pérdida durante el Entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.grid(True)
plt.show()

model.save("D:/PROYECTO 1/MODELO_FINAL/modelo_filtrado.h5")
print("✅ Modelo guardado en: MODELO_FINAL/modelo_filtrado.h5")
```

```
Tamaño de X_train: (10533, 100, 100, 1), X_val: (1859, 100, 100, 1)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	320
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
flatten (Flatten)	(None, 33856)	0
dense (Dense)	(None, 64)	2166848
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

```
Total params: 2185729 (8.34 MB)
Trainable params: 2185729 (8.34 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
Epoch 1/100
330/330 [==============================] - 74s 216ms/step - loss: 0.6531 - accuracy:
0.6182 - val_loss: 0.6461 - val_accuracy: 0.6568
Epoch 2/100
330/330 [==============================] - 76s 230ms/step - loss: 0.6303 - accuracy:
0.6474 - val_loss: 0.6778 - val_accuracy: 0.6783
Epoch 3/100
330/330 [==============================] - 75s 227ms/step - loss: 0.6084 - accuracy:
0.6768 - val_loss: 0.6832 - val_accuracy: 0.6998
Epoch 4/100
330/330 [==============================] - 83s 250ms/step - loss: 0.5915 - accuracy:
0.6891 - val_loss: 0.6272 - val_accuracy: 0.7149
Epoch 5/100
330/330 [==============================] - 78s 237ms/step - loss: 0.5773 - accuracy:
0.7037 - val_loss: 0.7290 - val_accuracy: 0.7101
Epoch 6/100
330/330 [==============================] - 82s 247ms/step - loss: 0.5634 - accuracy:
0.7120 - val_loss: 0.6981 - val_accuracy: 0.7386
Epoch 7/100
330/330 [==============================] - 75s 228ms/step - loss: 0.5542 - accuracy:
0.7155 - val_loss: 0.5610 - val_accuracy: 0.7531
Epoch 8/100
330/330 [==============================] - 75s 226ms/step - loss: 0.5447 - accuracy:
0.7266 - val_loss: 0.6624 - val_accuracy: 0.7445
Epoch 9/100
330/330 [==============================] - 79s 238ms/step - loss: 0.5323 - accuracy:
0.7361 - val_loss: 0.6649 - val_accuracy: 0.7429
Epoch 10/100
```

```
330/330 [==============================] - 79s 238ms/step - loss: 0.5181 - accuracy:
0.7475 - val_loss: 0.8934 - val_accuracy: 0.7171
Epoch 11/100
330/330 [==============================] - 74s 225ms/step - loss: 0.5089 - accuracy:
0.7546 - val_loss: 0.9284 - val_accuracy: 0.6993
Epoch 12/100
330/330 [==============================] - 93s 281ms/step - loss: 0.4893 - accuracy:
0.7736 - val_loss: 0.8040 - val_accuracy: 0.7283
Epoch 13/100
330/330 [==============================] - 83s 250ms/step - loss: 0.4834 - accuracy:
0.7694 - val_loss: 0.7734 - val_accuracy: 0.7294
Epoch 14/100
330/330 [==============================] - 83s 250ms/step - loss: 0.4623 - accuracy:
0.7820 - val_loss: 0.8574 - val_accuracy: 0.7106
Epoch 15/100
330/330 [==============================] - 93s 281ms/step - loss: 0.4416 - accuracy:
0.7963 - val_loss: 1.0602 - val_accuracy: 0.7036
Epoch 16/100
330/330 [==============================] - 78s 237ms/step - loss: 0.4336 - accuracy:
0.8035 - val_loss: 0.5597 - val_accuracy: 0.7929
Epoch 17/100
330/330 [==============================] - 82s 247ms/step - loss: 0.4225 - accuracy:
0.8086 - val_loss: 0.7168 - val_accuracy: 0.7526
Epoch 18/100
330/330 [==============================] - 93s 281ms/step - loss: 0.4124 - accuracy:
0.8139 - val_loss: 0.6094 - val_accuracy: 0.7875
Epoch 19/100
330/330 [==============================] - 82s 249ms/step - loss: 0.3922 - accuracy:
0.8310 - val_loss: 0.5283 - val_accuracy: 0.8090
Epoch 20/100
330/330 [==============================] - 80s 241ms/step - loss: 0.4052 - accuracy:
0.8177 - val_loss: 0.5265 - val_accuracy: 0.8171
Epoch 21/100
330/330 [==============================] - 92s 278ms/step - loss: 0.3878 - accuracy:
0.8306 - val_loss: 0.5234 - val_accuracy: 0.8182
Epoch 22/100
330/330 [==============================] - 84s 253ms/step - loss: 0.3867 - accuracy:
0.8307 - val_loss: 0.7556 - val_accuracy: 0.7977
Epoch 23/100
330/330 [==============================] - 83s 252ms/step - loss: 0.3852 - accuracy:
0.8338 - val_loss: 0.5153 - val_accuracy: 0.8182
Epoch 24/100
330/330 [==============================] - 82s 248ms/step - loss: 0.3752 - accuracy:
0.8389 - val_loss: 0.4819 - val_accuracy: 0.8155
Epoch 25/100
330/330 [==============================] - 86s 261ms/step - loss: 0.3662 - accuracy:
0.8415 - val_loss: 0.4318 - val_accuracy: 0.8295
Epoch 26/100
330/330 [==============================] - 97s 295ms/step - loss: 0.3696 - accuracy:
0.8384 - val_loss: 0.5272 - val_accuracy: 0.8225
Epoch 27/100
330/330 [==============================] - 93s 281ms/step - loss: 0.3619 - accuracy:
0.8432 - val_loss: 0.6348 - val_accuracy: 0.8155
Epoch 28/100
330/330 [==============================] - 98s 295ms/step - loss: 0.3665 - accuracy:
0.8371 - val_loss: 0.4446 - val_accuracy: 0.8419
```

```
Epoch 29/100
330/330 [==============================] - 114s 344ms/step - loss: 0.3589 - accurac
y: 0.8447 - val_loss: 0.5207 - val_accuracy: 0.8359
Epoch 30/100
330/330 [==============================] - 121s 367ms/step - loss: 0.3531 - accurac
y: 0.8470 - val_loss: 0.4258 - val_accuracy: 0.8349
Epoch 31/100
330/330 [==============================] - 119s 360ms/step - loss: 0.3642 - accurac
y: 0.8443 - val_loss: 0.4969 - val_accuracy: 0.8365
Epoch 32/100
330/330 [==============================] - 113s 342ms/step - loss: 0.3502 - accurac
y: 0.8480 - val_loss: 0.6281 - val_accuracy: 0.8273
Epoch 33/100
330/330 [==============================] - 114s 345ms/step - loss: 0.3569 - accurac
y: 0.8446 - val_loss: 0.4433 - val_accuracy: 0.8429
Epoch 34/100
330/330 [==============================] - 115s 349ms/step - loss: 0.3476 - accurac
y: 0.8565 - val_loss: 0.5865 - val_accuracy: 0.8257
Epoch 35/100
330/330 [==============================] - 115s 348ms/step - loss: 0.3502 - accurac
y: 0.8494 - val_loss: 0.5552 - val_accuracy: 0.8381
Epoch 36/100
330/330 [==============================] - 113s 341ms/step - loss: 0.3456 - accurac
y: 0.8482 - val_loss: 0.5755 - val_accuracy: 0.8219
Epoch 37/100
330/330 [==============================] - 115s 349ms/step - loss: 0.3479 - accurac
y: 0.8521 - val_loss: 0.5256 - val_accuracy: 0.8263
Epoch 38/100
330/330 [==============================] - 111s 335ms/step - loss: 0.3301 - accurac
y: 0.8604 - val_loss: 0.5689 - val_accuracy: 0.8354
Epoch 39/100
330/330 [==============================] - 114s 345ms/step - loss: 0.3497 - accurac
y: 0.8539 - val_loss: 0.5704 - val_accuracy: 0.8284
Epoch 40/100
330/330 [==============================] - 114s 346ms/step - loss: 0.3491 - accurac
y: 0.8503 - val_loss: 0.5049 - val_accuracy: 0.8316
59/59 [==============================] - 5s 86ms/step - loss: 0.4258 - accuracy: 0.8
349
Validación - Pérdida: 0.4258, Precisión: 0.8349
✅ Modelo guardado en: MODELO_FINAL/modelo_filtrado.h5
```

In [ ]:
```
model.save("D:\TESIS\modelo 13\modelo 13.h5")
print("✅ Modelo completo guardado como .h5")
```

```
✅ Modelo completo guardado como .h5
d:\Anaconda\envs\tesis38\lib\site-packages\keras\src\engine\training.py:3000: UserWa
rning: You are saving your model as an HDF5 file via `model.save()`. This file forma
t is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')`.
  saving_api.save_model(
```

In [ ]:
```
import matplotlib.pyplot as plt
import os

output_dir = "D:/TESIS/modelo 13"
os.makedirs(output_dir, exist_ok=True)

plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión durante el entrenamiento')
```

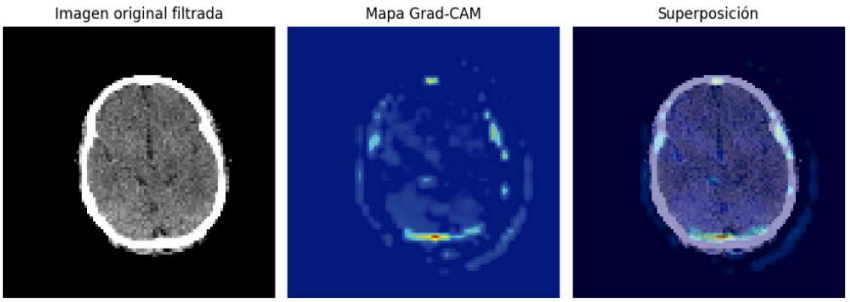
```
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(output_dir, "precision_entrenamiento.png"))
plt.close()

plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida durante el entrenamiento')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(output_dir, "perdida_entrenamiento.png"))
plt.close()
```

In [ ]:

```
import tensorflow as tf
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os

def extraer_craneo(imagen):
    if imagen.shape[-1] == 3:
        imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    _, umbral = cv2.threshold(imagen, 30, 255, cv2.THRESH_BINARY)
    contornos, _ = cv2.findContours(umbral, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM
    mascara = np.zeros_like(imagen)
    if contornos:
        contorno_principal = max(contornos, key=cv2.contourArea)
        cv2.drawContours(mascara, [contorno_principal], -1, 255, thickness=cv2.FILL
        imagen_filtrada = cv2.bitwise_and(imagen, imagen, mask=mascara)
    else:
        imagen_filtrada = imagen
    return imagen_filtrada

def cargar_imagen_filtrada(ruta_imagen):
    imagen = cv2.imread(ruta_imagen, cv2.IMREAD_COLOR)
    if imagen is None:
        raise ValueError(f"No se pudo cargar la imagen: {ruta_imagen}")

    imagen_filtrada = extraer_craneo(imagen)
    imagen_filtrada = cv2.resize(imagen_filtrada, (100, 100))

    if len(imagen_filtrada.shape) == 3:
        imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_BGR2GRAY)

    imagen_filtrada = imagen_filtrada.astype(np.float32) / 255.0
    imagen_filtrada = np.expand_dims(imagen_filtrada, axis=-1)
    imagen_batch = np.expand_dims(imagen_filtrada, axis=0)

    return imagen_batch, imagen_filtrada

def aplicar_gradcam(modelo, ruta_imagen, nombre_capa_conv):
```

```
    entrada, imagen_filtrada = cargar_imagen_filtrada(ruta_imagen)

    grad_model = tf.keras.models.Model(
        [modelo.inputs],
        [modelo.get_layer(nombre_capa_conv).output, modelo.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(entrada)
        class_output = predictions[:, 0]

    grads = tape.gradient(class_output, conv_outputs)[0]
    weights = tf.reduce_mean(grads, axis=(0, 1))
    cam = tf.reduce_sum(tf.multiply(weights, conv_outputs[0]), axis=-1)
    cam = np.maximum(cam, 0)
    cam = cam / (np.max(cam) + 1e-8)
    cam = cv2.resize(cam, (100, 100))

    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)
    heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)

    original_rgb = cv2.cvtColor(np.uint8(imagen_filtrada * 255), cv2.COLOR_GRAY2RGB
    superpuesta = np.uint8(heatmap * 0.4 + original_rgb * 0.6)

    plt.figure(figsize=(10, 4))
    plt.subplot(1, 3, 1)
    plt.imshow(original_rgb)
    plt.title("Imagen original filtrada")
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.imshow(heatmap)
    plt.title("Mapa Grad-CAM")
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(superpuesta)
    plt.title("Superposición")
    plt.axis('off')
    plt.tight_layout()
    plt.show()

    pred_valor = predictions.numpy()[0][0]
    clase = "Con hemorragia" if pred_valor >= 0.5 else "Sin hemorragia"
    print(f"\n🧠 Predicción: {pred_valor:.4f}")
    print(f"📌 Clasificación: {clase}")

modelo = tf.keras.models.load_model(r"D:\TESIS\modelo 13\modelo 13.h5")

ultima_capa_conv = None
for layer in reversed(modelo.layers):
    if isinstance(layer, tf.keras.layers.Conv2D):
        ultima_capa_conv = layer.name
        break
if not ultima_capa_conv:
    raise ValueError("No se encontró capa convolucional para Grad-CAM.")
```

```
ruta_imagen = r"D:\DOCUMENTOS COMPU\PROTOCOLOS\MACHINE LEARNING YO\MUESTRA TESIS PN
aplicar_gradcam(modelo, ruta_imagen, ultima_capa_conv)
```



🧠 Predicción: 0.0000
📌 Clasificación: Sin hemorragia