UNIVERSIDAD ANDRÉS BELLO
FACULTAD DE INGENIERÍA

INGENIERÍA CIVÍL INFORMÁTICA

Implementación de Arquitectura Mixture of Experts Basada en Redes Neuronales Convolucionales para la Detección de Glaucoma en Imágenes del Fondo de Ojo

CHRISTIAN MATÍAS GARRIDO MENESES

Profesor guía: Billy Marck Peralta Márquez

SANTIAGO - CHILE

Diciembre - 2024

Índice

1. Resumen

El glaucoma representa un grave problema de salud pública, afectando a millones de personas a nivel mundial y posicionándose como una de las principales causas de ceguera irreversible. Esta enfermedad, cuya prevalencia aumenta con la edad y otros factores de riesgo, requiere métodos eficientes para su diagnóstico temprano, a fin de mitigar su impacto negativo en la calidad de vida de quienes lo padecen.

Este trabajo presenta una solución basada en el uso de una red de mezcla de expertos (`Mixture of Experts, MoE`), evaluada con el conjunto de datos `ACRIMA`. Se llevó a cabo un análisis exhaustivo de rendimiento utilizando validación cruzada de 10 folds y variaciones en la cantidad de expertos seleccionados, explorando el impacto en métricas clave como *Accuracy*, *Recall*, *Precision*, *Specificity*, *F1-Score* y *Error_rate*.

Los resultados obtenidos reflejan un *Accuracy* del 82.84 % con seleccionando 1 experto, 88.51 % seleccionando 2 expertos y 94.18 % con 3 expertos, alcanzando un *Accuracy* del 95.05 % para el modelo `VGG19`, que fue el mejor de los modelos individuales. El valor de pérdida fue de un 0.4898 para el modelo `MoE` con 1 experto, 0.3525 para el otro con 2 expertos y 0.4677 para el de 3 expertos. Además, el modelo `MoE` con 3 expertos alcanzó un *Recall* de 92.72 %, un *Precision* de 93.67 %, y un *F1-score* de 93.09 %, destacándose en métricas clave para la detección temprana del glaucoma.

Estos hallazgos resaltan la eficacia del enfoque `MoE`, mostrando un incremento significativo en el rendimiento a medida que se aumenta el número de expertos, lo que sugiere que esta estrategia tiene un gran potencial para mejorar las capacidades diagnósticas en tareas relacionadas con la salud visual o incluso en otras áreas.

2. Abstract

Glaucoma represents a serious public health issue, affecting millions of people worldwide and positioning itself as one of the primary causes of irreversible blindness. This disease, whose prevalence increases with age and other risk factors, requires efficient methods for early diagnosis to mitigate its negative impact on the quality of life of those who suffer from it.

This work presents a solution based on the use of a Mixture of Experts (MoE) network, evaluated with the `ACRIMA` dataset. An extensive performance analysis was conducted using 10-fold cross-validation and variations in the number of selected experts, exploring the impact on key metrics such as *Accuracy*, *Recall*, *Precision*, *Specificity*, *F1-Score*, and *Error_rate*.

The obtained results reflect an *Accuracy* of 82.84 % when selecting 1 expert, 88.51 % when selecting 2 experts, and 94.18 % with 3 experts, reaching an *Accuracy* of 95.05 % for the `VGG19` model, which was the best individual model. The loss value was 0.4898 for the `MoE` model with 1 expert, 0.3525 for the model with 2 experts, and 0.4677 for the model with 3 experts. Additionally, the `MoE` model with 3 experts achieved a *Recall* of 92.72 %, a *Precision* of 93.67 %, and an *F1-score* of 93.09 %, standing out in key metrics for early glaucoma detection.

These findings highlight the effectiveness of the `MoE` approach, showing a significant performance increase as the number of experts increases, suggesting that this strategy has great potential for improving diagnostic capabilities in tasks related to visual health or even in other areas.

3. Introducción

El glaucoma es una enfermedad degenerativa del nervio óptico que afecta los ojos y puede convertirse en la principal causa de ceguera irreversible si no se diagnostica y trata a tiempo [1]. Aunque es principalmente una afección de personas mayores, puede presentarse a cualquier edad, lo que hace que la detección temprana sea crucial para prevenir el daño que causa. Según estimaciones de la Organización Mundial de la Salud (OMS), el número global de personas con glaucoma aumentará de 76 millones en 2020 a 95.4 millones en 2030 [2], un incremento de 1.3 veces. Esto destaca la importancia de estrategias de diagnóstico como la identificación de signos de glaucoma en las imágenes del fondo de ojo, una tarea que tradicionalmente ha dependido de la interpretación manual de oftalmólogos experimentados [3].

En las últimas décadas, los avances en inteligencia artificial y aprendizaje profundo han revolucionado los métodos de diagnóstico médico, ofreciendo herramientas cada vez más precisas para la detección precoz de enfermedades. Sin embargo, la complejidad y variabilidad de las manifestaciones del glaucoma representan desafíos importantes para los sistemas de diagnóstico automático. En este contexto, las arquitecturas de mezcla de expertos (`Mixture of Experts, MoE`) emergen como una solución prometedora. Esta arquitectura permite combinar múltiples modelos especializados, con el potencial de adaptarse mejor a la diversidad de características del glaucoma y mejorar la precisión y robustez de los sistemas de apoyo al diagnóstico.

El presente trabajo propone una arquitectura de mezcla de expertos basada en redes neuronales convolucionales para la detección automatizada de glaucoma en imágenes de fondo de ojo, utilizando el conjunto de datos `ACRIMA`. La implementación aprovecha una combinación de modelos convolucionales preentrenados (`VGG19, ResNet50, DenseNet121`) en la base de datos de `Imagenet` para extraer características relevantes de las imágenes, donde una red neuronal convolucional adicional (`gate network`) actúa como selector de expertos. Este enfoque permite que la red mezcla de expertos aprenda y asigne la importancia de diferentes expertos según la entrada específica, mejorando el

rendimiento mediante la selección de modelos en comparación con el uso de un modelo individual.

La motivación de este trabajo es contribuir con hallazgos relevantes que puedan ser de utilidad para futuras investigaciones que implementen enfoques similares o aborden tareas afines al diagnóstico asistido por inteligencia artificial.

4. Marco teórico

4.1. Definiciones importantes

4.1.1. Redes Neuronales Artificiales

Según el libro [4], Las redes neuronales artificiales constituyen una técnica de aprendizaje automático ampliamente utilizada que emula el mecanismo de aprendizaje observado en organismos biológicos. El sistema nervioso de los seres vivos está compuesto por células conocidas como neuronas, las cuales se interconectan a través de estructuras denominadas axones y dendritas. Los puntos donde estas neuronas se conectan entre sí reciben el nombre de sinapsis, y la fortaleza de dichas conexiones sinápticas puede variar en respuesta a estímulos externos, lo cual se considera el fundamento del aprendizaje biológico.

Este proceso de aprendizaje biológico es simulado en las redes neuronales artificiales, donde las unidades computacionales similares a las neuronas biológicas se encuentran interconectadas mediante conexiones ponderadas, similares a la fuerza de las conexiones sinápticas. La red neuronal artificial aprende ajustando los pesos de estas conexiones, conocidos como parámetros entrenables del modelo, en función de los datos de entrenamiento proporcionados.

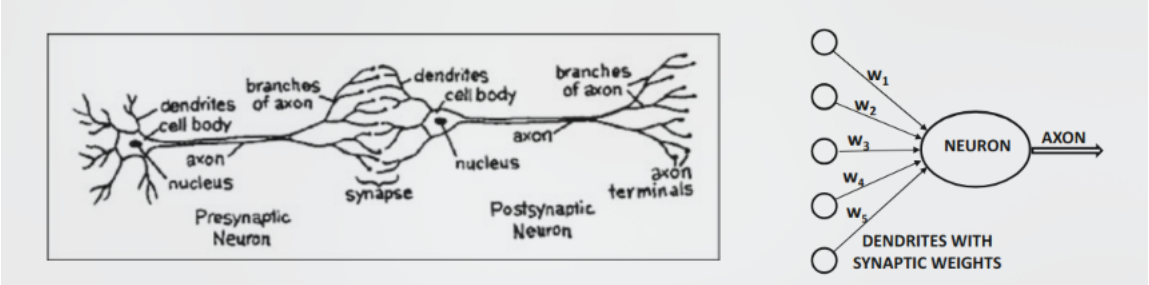

Figura 1: Red Neuronal Biologica y Red Neuronal artificial [4]

Las redes neuronales artificiales se estructuran principalmente en capas interconectadas, donde se tiene al principio una capa de entrada la cual recibe los datos, capas ocultas que realizan la extracción y transformación de características mediante funciones de activación, y al final una capa de salida que genera las predicciones. Esta arquitectura

dispuesta por capas o estratificada, junto con el aprendizaje adaptativo de los pesos de conexión, permite que las redes neuronales modelen relaciones complejas y no lineales presentes en los datos, convirtiéndolas en una herramienta poderosa para tareas como reconocimiento de imágenes, procesamiento de lenguaje natural, tareas de regresión, tareas de clasificación y análisis predictivo, entre otros, superando en muchos casos a los enfoques tradicionales de aprendizaje automático.

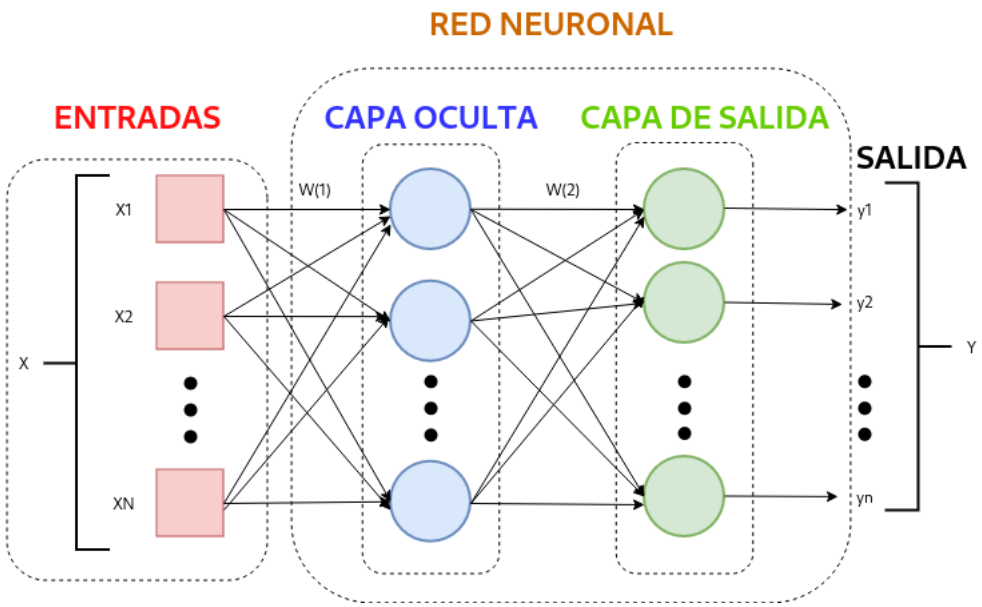

Figura 2: Arquitectura simple red neuronal artificial. Elaboración Propia

Por lo tanto, las redes neuronales artificiales representan un paradigma de aprendizaje automático inspirado en los principios del aprendizaje biológico, lo cual las dota de una gran capacidad de adaptación y modelado de fenómenos complejos, con amplias aplicaciones en diversos dominios del conocimiento.

4.1.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) representan un paradigma revolucionario en el campo de la inteligencia artificial, particularmente en visión por computadora. Al igual que las redes neuronales artificiales, las CNN son modelos computacionales inspirados biológicamente, diseñados para imitar los procesos de percepción visual del cerebro.

El origen de esta innovadora arquitectura se remonta a los experimentos de David Hubel y Torsten Wiesel en la década de 1960. Mediante un estudio de la corteza visual de un gato, estos neurocientíficos descubrieron que neuronas específicas en el cerebro responden selectivamente a estímulos visuales en regiones particulares del campo visual. Este descubrimiento fundamental reveló que el procesamiento visual no es un proceso uniforme, sino que ocurre mediante la activación de neuronas especializadas que detectan características específicas como bordes, orientaciones y formas.

La arquitectura de las CNN imita directamente este principio. A diferencia de las redes neuronales artificiales tradicionales, las CNN incorporan capas especializadas que extraen automáticamente características jerárquicas de las imágenes, simulando el procesamiento visual biológico, las capas convolucionales y las capas de pooling.

Las capas convolucionales constituyen el núcleo de la red, aplicando filtros (kernels) que se desplazan sobre la imagen para generar mapas de características. Estos filtros detectan patrones locales como bordes, texturas y formas, de manera similar a como las neuronas de la corteza visual.

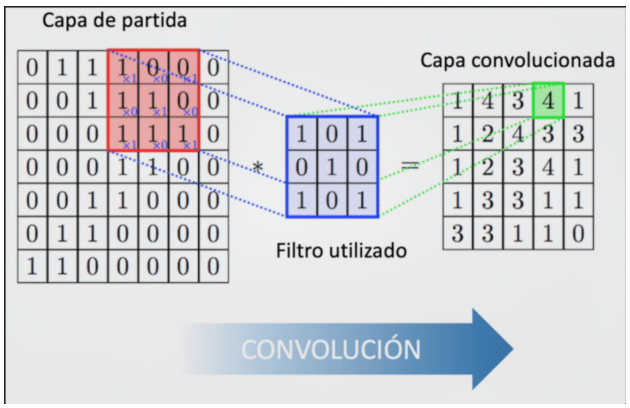

Figura 3: Proceso de convolución [5]

Por el otro lado, las capas de pooling (submuestreo) reducen las dimensiones espaciales de los mapas de características, preservando la información más relevante. Esta operación imita el proceso de abstracción y generalización que ocurre en el sistema visual biológico, permitiendo que la red se enfoque en características invariantes a pequeñas variaciones.

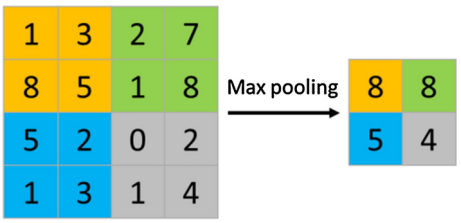

Figura 4: Proceso de Max-Pooling [6]

El resultado es una arquitectura profundamente eficiente para tareas de visión artificial, capaz de realizar desde clasificación de imágenes hasta detección de objetos con una precisión sorprendente. Las CNN han demostrado ser especialmente poderosas porque aprenden representaciones jerárquicas de características, comenzando desde bordes simples hasta características más complejas y abstractas en capas más profundas de la red.

La inspiración neurobiológica de Hubel y Wiesel no solo proporcionó un modelo conceptual para las CNN, sino que fundamentalmente transformó nuestra comprensión de cómo los sistemas inteligentes pueden procesar información visual, tendiendo un puente entre la neurociencia y la inteligencia artificial.

4.1.3. Mezcla de Expertos (MoE)

En el Paper [7] se introduce la estrategia de mezcla de expertos (`Mixture of Experts, MoE`); esta surge como una innovadora aproximación en el desarrollo de modelos de procesamiento. Consiste básicamente en la integración de varios sub-modelos especializados, cada uno centrado en aspectos particulares de una tarea compleja.

Un ejemplo sería un equipo de profesionales, donde cada uno tiene una habilidad específica. En este enfoque, un mecanismo central decide dinámicamente qué profesional (experto) es más adecuado para abordar una tarea específica según sus características únicas, asignándoles un nivel de importancia.

Esta arquitectura se compone de 2 elementos fundamentales:

- **Expertos individuales:** Son modelos que se especializan en segmentos concretos del problema. Por ejemplo:
  - En análisis de imágenes, un experto podría centrarse en texturas.
  - El segundo experto podría enfocarse en formas geométricas.
  - El últmio experto podría detectar bordes.
- **Sistema de asignación (router o gate):** Un componente inteligente que evalúa la entrada y distribuye la carga de trabajo entre los expertos más adecuados. Utiliza técnicas probabilísticas para determinar la relevancia de cada especialista.

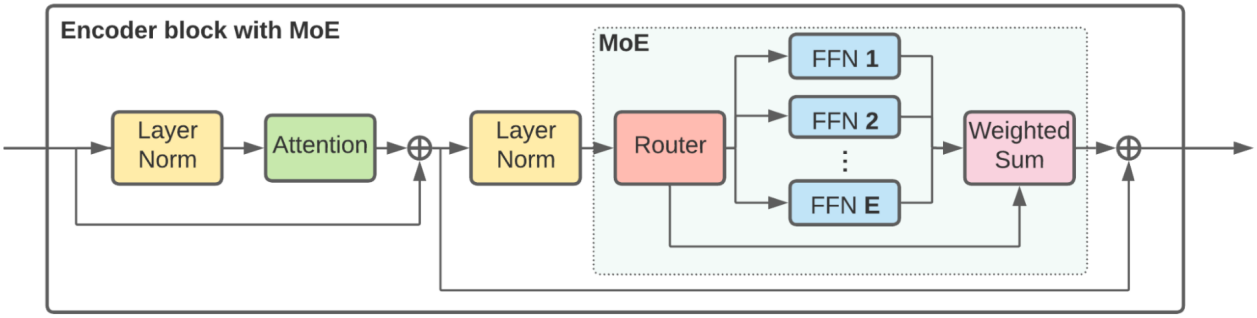

Figura 5: Arquitectura de un bloque codificador con mezcla de expertos (`Mixture of Experts, MoE`). El sistema se compone de un enrutador (router o gate), que distribuye dinámicamente las tareas entre los expertos individuales (FFNs), y una suma ponderada (weighted sum), que combina sus contribuciones según la relevancia asignada.[7].

La ventaja fundamental de esta estrategia radica en su capacidad para manejar tareas complejas de manera eficiente, activando solo los recursos necesarios en cada momento. Esto permite una utilización más inteligente de los recursos computacionales, reduciendo la sobrecarga y mejorando la precisión. Este enfoque ha demostrado ser efectivo en diversas aplicaciones, incluyendo modelos de lenguaje y visión por computadora con transformers visuales. Sin embargo, la evidencia de este método en redes neuronales convolucionales es limitada; por esto, este trabajo busca explorar el potencial de las redes neuronales convolucionales utilizando el enfoque de mezcla de expertos, contribuyendo así a la expansión del conocimiento en este campo.

4.2. Trabajos relacionados

4.2.1. Glaucoma Detection Using Explainable AI and Deep Learning

En este artículo [8], se presenta una estrategia para la detección de glaucoma utilizando inteligencia artificial explicable (Explainable AI, XAI) y aprendizaje profundo. El objetivo es abordar las limitaciones de los métodos tradicionales en la detección temprana del glaucoma, una enfermedad ocular que puede provocar ceguera irreversible si no se diagnostica a tiempo.

La metodología combina modelos de aprendizaje profundo como VGG19, AlexNet, ResNet y MobileNet con sistemas basados en lógica difusa, como ANFIS (Sistema Adaptativo de Inferencia Neuro-Difusa) y SNN (Redes Neuronales de Disparo), integrando técnicas de XAI para ofrecer explicaciones claras sobre las decisiones tomadas por los modelos. Se emplearon imágenes de fondo de ojo (fundus images) provenientes de diversas bases de datos para el entrenamiento y la validación.

Los resultados muestran que el modelo ANFIS, al combinar técnicas de XAI y redes profundas, alcanza altos niveles de precisión en la clasificación, superando a modelos tradicionales. Además, se destaca que el uso de XAI mejora la interpretabilidad de los resultados, permitiendo a los profesionales de la salud comprender mejor las decisiones del sistema automatizado. Esto representa un paso significativo hacia la integración de herramientas de inteligencia artificial confiables en el ámbito clínico.

4.2.2. 3-LbNets: Tri-Labeling Deep Convolutional Neural Network for the Automated Screening of Glaucoma, Glaucoma Suspect, and No Glaucoma in Fundus Images

El presente estudio [9] aborda un desafío crítico en oftalmología, la identificación temprana de alteraciones del nervio óptico que pueden conducir a pérdida visual irreversible. Considerando las limitaciones actuales en el diagnóstico oftalmológico, como la escasez de especialistas y los costos asociados a evaluaciones exhaustivas, se propuso un modelo computacional innovador para el análisis de imágenes del fondo ocular.

La metodología desarrollada integra un sistema de clasificación multimodal que diferencia tres estados fundamentales: condición normal, sospecha de glaucoma y glaucoma detectado. Mediante un algoritmo de análisis de imagen que procesa características estructurales específicas, se logró un rendimiento diagnóstico significativamente superior a los métodos tradicionales.

Los resultados preliminares muestran una precisión diagnóstica destacable. En un conjunto de 206 imágenes evaluadas con consenso completo de especialistas, el modelo alcanzó tasas de precisión superiores al 97 % para cada categoría de clasificación. Adicionalmente, en un subconjunto de 178 imágenes con mayor complejidad diagnóstica, se mantuvieron rendimientos consistentes, con intervalos de precisión entre 69 % y 83 %.

La contribución fundamental de esta investigación radica en su potencial para optimizar los procesos de detección temprana, reduciendo la carga asistencial y permitiendo intervenciones más oportunas en padecimientos potencialmente progresivos.

4.2.3. Deep Learning Ensemble Method for Classifying Glaucoma Stages Using Fundus Photographs and Convolutional Neural Networks

La investigación [10] exploró una estrategia para la clasificación de estados de una condición oftalmológica degenerativa mediante ensamble de modelos de redes neuronales convolucionales. El trabajo se sustentó en un riguroso proceso de validación que involucró la participación de especialistas expertos en glaucoma.

El estudio comprendió un conjunto de 3,460 registros fotográficos correspondientes a 2,204 individuos, categorizados cuidadosamente en tres grupos fundamentales: sujetos sin alteraciones, casos en fase inicial de progresión y manifestaciones avanzadas de la patología. La clasificación se realizó utilizando parámetros clínicos estandarizados, específicamente el promedio de desviación en pruebas de campo visual.

La metodología contempló el desarrollo de 56 modelos de redes neuronales convolucionales con arquitecturas diferenciadas, implementando una estrategia de ensamble que permitiera optimizar el rendimiento diagnóstico mediante la combinación de sus múltiples resultados.

Los hallazgos resultaron particularmente prometedores. El sistema propuesto logró una

precisión del 88.1 %, superando significativamente los rendimientos de modelos individuales. La capacidad de discriminación alcanzó niveles de eficacia superiores, reduciendo de manera sustancial los márgenes de error, especialmente en la identificación de casos de progresión temprana.
La aproximación de esta metodología demuestra la viabilidad de ensamblar modelos de redes neuronales convolucionales en la detección de condiciones oftalmológicas progresivas. Los resultados sugieren que las estrategias de ensamblaje de redes neuronales convolucionales representan una vía prometedora para mejorar la precisión y confiabilidad de los sistemas de diagnóstico asistido.

4.2.4. Scaling Vision with Sparse Mixture of Experts

Esta investigación [7] introduce un paradigma de diseño de red neuronal que reimagina fundamentalmente la asignación de recursos computacionales durante tareas de procesamiento de imágenes.
El marco propuesto aprovecha un diseño de red neuronal especializado en el que módulos computacionales se especializan dinámicamente en procesar distintas características de imagen. A diferencia de arquitecturas tradicionales anteriormente mencionadas que procesan uniformemente todos los datos de entrada, este enfoque permite vías computacionales selectivas y dirigidas, reduciendo significativamente los requisitos computacionales generales.
Mediante un mecanismo de enrutamiento dinámico, el modelo puede asignar recursos computacionales de manera adaptativa en lotes de imágenes. Este método permite una flexibilidad sin precedentes, permitiendo que cada imagen sea procesada a través de vías neuronales únicamente optimizadas. El resultado es un sistema altamente eficiente que mantiene una alta precisión mientras reduce significativamente la sobrecarga computacional.

Los resultados experimentales son convincentes. Mediante la optimización estratégica de parámetros, se demostró el potencial de crear modelos de visión a gran escala con más de 15 mil millones de parámetros, alcanzando niveles de precisión notables cercanos al

90.35 % en benchmarks estandarizados de reconocimiento de imágenes.

4.3. Redes Neuronales Utilizadas

4.3.1. VGG19

VGG (Visual Geometry Group) [11], desarrollada por investigadores de la Universidad de Oxford, introdujo una arquitectura de red neuronal convolucional caracterizada por su simplicidad y profundidad. La versión `VGG19`, en particular, se distingue por utilizar exclusivamente filtros convolucionales de 3x3 con stride 1 y padding para mantener la dimensionalidad espacial, y capas de max-pooling de 2x2 con stride 2 para reducir progresivamente el tamaño de los mapas de características.

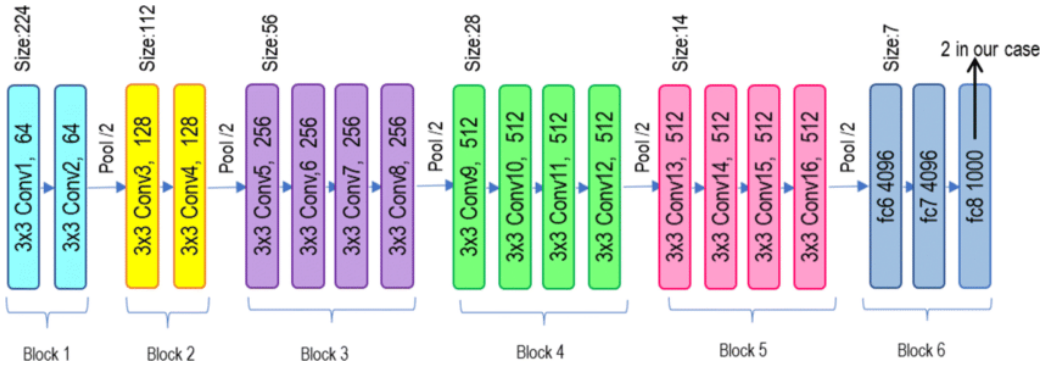

Figura 6: El modelo `VGG19` cuenta con 16 capas convolucionales organizadas en 5 bloques. Después de cada bloque, se incluye una capa de MaxPooling que reduce las dimensiones de la imagen de entrada a la mitad y, al mismo tiempo, duplica la cantidad de filtros en las capas convolucionales. [12]

La arquitectura VGG se destaca por su enfoque de construcción sistemática, donde la profundidad de la red se incrementa mediante la adición secuencial de bloques convolucionales, cada uno seguido de una capa de MaxPooling. Esta estructura permite crear redes neuronales con hasta 19 capas de peso (de ahí su nombre `VGG19`), lo que en su momento representó un salto significativo en la complejidad de las redes neuronales convolucionales para reconocimiento de imágenes.

4.3.2. Resnet50

`ResNet50` [13] representa un punto de inflexión en el diseño de redes neuronales convolucionales profundas, introduciendo el concepto de bloques residuales con "skip connections"que permiten el flujo directo de información a través de capas más profundas. Su diseño se basa en bloques de bottleneck con tres capas convolucionales (1x1, 3x3, 1x1) que reducen, procesan y expanden la dimensionalidad de manera eficiente.

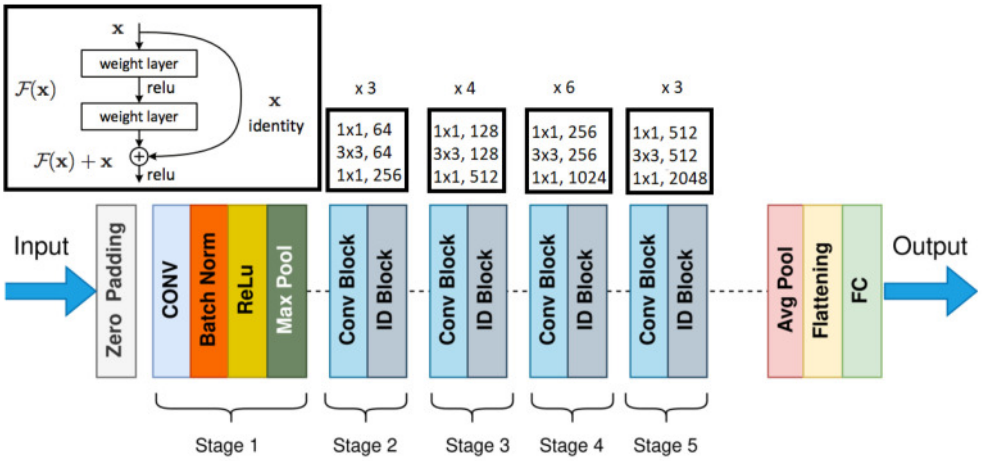

Figura 7: Arquitectura de `ResNet50`, con su diseño basado en bloques residuales con conexiones de salto (skip connections). La red se organiza en cinco etapas principales, cada una compuesta por bloques residuales (Conv Block e ID Block) que utilizan convoluciones de tipo bottleneck (1x1, 3x3, 1x1).[14]

La innovación principal de `ResNet50` radica en su capacidad para entrenar redes neuronales significativamente más profundas sin sufrir el problema del "Vanishing Gradient.º desvanecimiento del gradiente, demostrando que la profundidad puede ser un recurso valioso para mejorar el rendimiento en tareas de reconocimiento de imágenes. Las conexiones residuales permiten que la información de capas anteriores se propague directamente a capas más profundas, facilitando el entrenamiento de redes con 50 capas y abriendo nuevos horizontes en el diseño de arquitecturas neuronales.

4.3.3. DenseNet121

`DenseNet121` [15] introduce una arquitectura de red neuronal convolucional basada en bloques densos, donde cada capa dentro de un bloque está conectada directamente con todas las capas posteriores, permitiendo un flujo de información significativamente mejorado. Esta arquitectura, al igual que `ResNet`, supera el problema del "Vanishing Gradient.º desvanecimiento del gradiente, pero esta red lo logra mediante conexiones que concatenan características de todas las capas precedentes, lo que resulta en un uso más eficiente de los parámetros de la red.

El diseño de `DenseNet121` se caracteriza por dividir la red en cuatro bloques densos, cada uno con una transición de compresión que reduce el número de características, manteniendo una estructura computacionalmente eficiente. A diferencia de arquitecturas tradicionales, `DenseNet121` logra un rendimiento superior con una profundidad considerablemente menor, demostrando que la reutilización de características a través de conexiones densas puede ser más efectiva que la simple acumulación de capas.

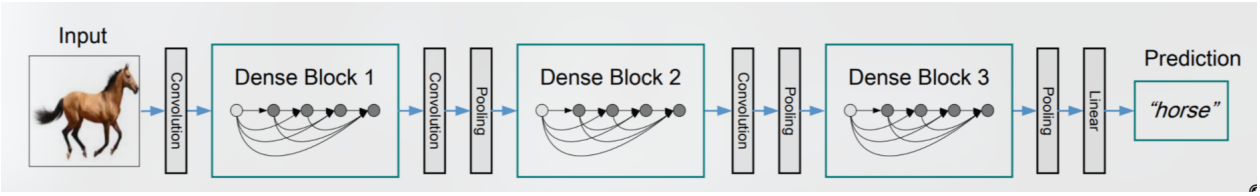

Figura 8: Arquitectura de `DenseNet` [15]

5. Hipótesis

El uso de un modelo basado en mezcla de expertos (`Mixture of Experts, MoE`) con arquitecturas de redes neuronales convolucionales permite mejorar la generalización y el rendimiento en la detección de glaucoma al integrar características aprendidas por diferentes modelos expertos.

5.1. Hipótesis específica 1

El rendimiento del modelo `MoE` aumentará a medida que se incremente el número de expertos seleccionados para cada entrada, dado que la combinación de múltiples perspectivas mejora la capacidad del modelo para generalizar sobre los datos.

6. Objetivo del trabajo

El objetivo de esta investigación es implementar y evaluar una arquitectura de mezcla de expertos para redes neuronales convolucionales con el fin de mejorar la detección de glaucoma mediante la combinación de mapas de características aprendidos por múltiples modelos expertos.

6.1. Objetivo específico 1

Diseñar e implementar una capa personalizada con `Tensorflow` para el lenguaje de programación `Python` para la mezcla de expertos que permita seleccionar y combinar mapas de características de diferentes redes neuronales convolucionales según las características de entrada, utilizando una gate network basada en una red neuronal convolucional.

6.2. Objetivo específico 2

Comparar el desempeño del modelo `MoE` con diferentes configuraciones de selección de expertos (1, 2 y 3 expertos) mediante validación cruzada con 10 folds y analizar el impacto de estas configuraciones en métricas como *Accuracy*, *Recall*, *Precision*, *F1-Score*, *Specificity* y *Error_rate*.

6.3. Objetivo específico 3

Evaluar y comparar el rendimiento de los modelos `MoE` con los modelos individuales utilizados como expertos para determinar si la combinación de mapas de características mejora la capacidad de generalización en la detección de glaucoma.

7. Metodología

El diseño de un modelo basado en mezcla de expertos (`Mixture of Experts, MoE`) implica el uso de múltiples modelos especializados (expertos) que aportan diferentes perspectivas para resolver una tarea específica. En este proyecto, los expertos son redes neuronales convolucionales (CNNs) entrenadas previamente en tareas de clasificación. El componente clave que diferencia esta aproximación es el uso de un router o gate network, que selecciona dinámicamente los expertos más adecuados para cada entrada, permitiendo una combinación más efectiva de mapas de características (feature maps).

Cuando se implementa una arquitectura `MoE`, es importante garantizar que las características extraídas por los expertos sean suficientemente diversas para complementar los resultados de otros modelos. En este proyecto, se importaron modelos base de redes neuronales convolucionales preentrenados en la base de datos `ImageNet`, una base de datos ampliamente utilizada en visión por computadora que contiene millones de imágenes etiquetadas en 1000 categorías. A estos modelos se les añadieron capas densas al final, diseñadas específicamente para realizar la detección de glaucoma. De esta manera, los modelos base conservan las características aprendidas en `ImageNet`, mientras que las capas añadidas permiten ajustar los expertos a la nueva tarea, asegurando una combinación efectiva de mapas de características para la clasificación final.

La capa personalizada (`TopMExpertsFM`) permitirá seleccionar un número variable de expertos, definiendo combinaciones específicas que luego pasan a una red densa encargada de la clasificación final. Esta estructura no solo busca mejorar la capacidad de generalización del modelo, sino también reducir el sobreajuste al aprovechar la diversidad de los expertos.

La gate network es una red neuronal convolucional diseñada específicamente para este proyecto, que analiza las imágenes de entrada y selecciona dinámicamente los expertos más relevantes. A diferencia de las arquitecturas tradicionales, la gate network no utiliza un enfoque estático para la selección de expertos; en cambio, asigna probabilidades a cada

experto según la relevancia de su conocimiento para una entrada específica. Esto asegura que la selección sea adaptativa y eficiente.

El proyecto también incluye la evaluación exhaustiva de la arquitectura `MoE` utilizando validación cruzada con diferentes configuraciones: seleccionando 1, 2 y 3 expertos, así como evaluando el rendimiento de cada experto de forma individual. Estas configuraciones permiten comparar el desempeño del modelo en términos de *Accuracy*, *Recall*, *Precision*, *F1-Score*, *Specificity* y *Error_rate*. Además, se implementa la normalización y preprocesamiento de los datos para garantizar que los resultados sean consistentes.

7.1. Dataset

Para la implementación de este proyecto, se utilizó el dataset **ACRIMA** [16], una base de datos compuesta por 705 imágenes del fondo de ojo, de las cuales 396 corresponden a ojos con glaucoma y 309 a ojos normales. Estas imágenes fueron recolectadas en el centro FISABIO Oftalmología Médica, ubicado en Valencia, España, con el consentimiento previo de los pacientes y en conformidad con los estándares éticos establecidos en la Declaración de Helsinki de 1964.

- **Características del Dataset:**
  - **Etiquetado Experto:** Todas las imágenes fueron anotadas por especialistas en glaucoma con varios años de experiencia, asegurando la precisión en la clasificación de las imágenes como glaucoma o normales.
  - **Preprocesamiento de las imágenes:** Las imágenes del fondo de ojo fueron recortadas alrededor del disco óptico para centrar la atención en la región de interés y renombradas con un formato específico:
    - El nombre de cada imagen comienza con las letras `Im`, seguido de un número de tres dígitos (del `001` al `705`), e incluye un sufijo para indicar la clase:
      - `_g_`: Indica una imagen con glaucoma.
      - `_`: Indica una imagen normal.

- Algunos ejemplos:
  - `Im686_g_ACRIMA`: Imagen glaucomatosa.
  - `Im001_ACRIMA`: Imagen normal.

- **Importancia del Dataset:** El dataset `ACRIMA` se diseñó específicamente para la investigación en el diagnóstico automatizado del glaucoma, un problema crítico en oftalmología debido a las consecuencias irreversibles de la enfermedad si no se detecta a tiempo. La base de datos proporciona una herramienta valiosa para entrenar modelos de aprendizaje profundo, como redes neuronales convolucionales, y evaluar su capacidad de generalización en la detección del glaucoma.

- **Uso en este proyecto:** En este trabajo, las imágenes fueron importadas y sometidas a un preprocesamiento adicional, como la redimensión y normalización, para adaptarlas a las entradas requeridas por los modelos convolucionales. Este dataset permitió explorar la efectividad de arquitecturas basadas en la técnica *Mixture of Experts, (MoE)* y su capacidad para mejorar la detección del glaucoma mediante la combinación de mapas de características (feature maps) de múltiples expertos.

7.2. Separar Dataset

El dataset **ACRIMA** originalmente está estructurado como una sola carpeta que contiene todas las imágenes de fondo de ojo, tanto ojos con glaucoma como ojos normales. Para facilitar el entrenamiento y la validación de los modelos, se realizó una separación de las imágenes en dos carpetas distintas: una para imágenes de ojos con glaucoma y otra para imágenes normales.

El proceso de separación se llevó a cabo utilizando un script en `Python` que emplea las bibliotecas `os` y `shutil`. A continuación, se muestra el código junto al detalle de los pasos realizados:

```python
import os
import shutil

user = os.getlogin()

origin = f"/home/{user}/Datasets/Acrima_Database/Images"
destiny = "./acrimaDataset/"

os.makedirs(name=destiny, exist_ok=True)
os.makedirs(name=os.path.join(destiny, "glaucoma"), exist_ok=True)
os.makedirs(name=os.path.join(destiny, "normal"), exist_ok=True)

img_names = os.listdir(origin)

for im in img_names:
    origin_path = os.path.join(origin, im)
    if ('_g_' in im):
        shutil.copy(origin_path, os.path.join(destiny, "glaucoma", im))
    else:
        shutil.copy(origin_path, os.path.join(destiny, "normal", im))
```

- **Definicion de rutas:** Se especificó la ubicación de las imágenes originales y la ubicación donde se crearán las carpetas organizadas.
- **Creación de carpetas:** Se generaron tres directorios, uno principal para almacenar las imágenes procesadas (`acrimaDataset/`) y dos subdirectorios: `glaucoma/` y `normal/`.
- **Clasificación de imagenes:** El script recorrió cada archivo en la carpeta original y verificó su nombre. Según la convención de nombres del dataset, las imágenes de

ojos con glaucoma contienen el sufijo _g_, mientras que las de ojos normales no lo tienen.

- **Copiado de imágenes:** Las imágenes se copiaron al directorio correspondiente (`glaucoma/` o `normal/`) según su categoría.

7.3. Importar dataset, Redimensionar y normalizar

En esta sección del proyecto, el proceso de importación del dataset se realizó utilizando la función `image_dataset_from_directory` de la API de `Keras` de `TensorFlow`. Esta función simplifica la carga de imágenes desde un directorio, lo que resulta muy útil cuando se trata de datasets con una estructura de carpetas organizada, como en este caso, donde las imágenes de ojos con glaucoma y ojos normales ya se encontraban separadas en carpetas distintas gracias a la separación anteriormente realizada.

```
#parametros por defecto
batch_size = 32
epochs = 30
img_size = (128,128)
print('LOADING TRAIN IMAGES:\n')
train_dataset = keras.preprocessing.image_dataset_from_directory(
    directory = '../input/acrima/acrimaDataset/',
    color_mode = 'rgb',
    batch_size = batch_size,
    image_size = img_size,
    shuffle = True,
    seed = 42,
    validation_split = 0.3,
    subset = 'training',
    interpolation = 'bilinear',
    )
```

El código realiza la importación del conjunto de entrenamiento utilizando la función `image_dataset_from_directory`. Las imágenes son cargadas desde el directorio especificado, redimensionadas a un tamaño definido (128,128,3), y se asigna un valor de semilla para la replicabilidad. El conjunto de datos se divide en un 70 % para entrenamiento y un 30 % para validación.

7.3.1. Enfoque Hold Out

Inicialmente, el conjunto de datos `ACRIMA`, que consta de un total de 705 imágenes (396 ojos con glaucoma y 309 ojos normales), se dividió utilizando un enfoque Hold out. Este enfoque consiste en dividir el conjunto de datos en tres partes: un conjunto de entrenamiento, un conjunto de validación y un conjunto de prueba. La división fue la siguiente:

- **Entrenamiento:** 494 imágenes (70 %)
- **Validación:** 64 imágenes (10 %)
- **Prueba:** 147 imágenes (20 %)

Este enfoque es común cuando se cuenta con una cantidad de datos suficientemente grande. Sin embargo, debido a que el número de imágenes en este conjunto de datos es relativamente pequeño, se decidió más adelante concatenar todos los subconjuntos (entrenamiento, validación y prueba) en un único arreglo de `NumPy`. Esto permitió disponer de un único conjunto de datos para ser utilizado en etapas posteriores.

7.3.2. Normalización

Una vez importadas las imágenes, se les aplica una capa de normalización utilizando la función `keras.layers.Rescaling(1./255)`. Este paso es fundamental en el preprocesamiento de imágenes, ya que ajusta los valores de los píxeles a un rango estándar entre 0 y 1, en lugar de los valores originales que varían entre 0 y 255. Esta normalización ayuda a los modelos a converger más rápido y reduce el riesgo de sesgos al equilibrar la influencia de los píxeles de mayor intensidad, facilitando un aprendizaje más uniforme y estable.

```
normalization_layer=keras.layers.Rescaling(1./255)
#Train_dataset
train_dataset=train_dataset.map(lambda x,y:(normalization_layer(x),y))
#Valid_dataset
valid_dataset= valid_dataset.map(lambda x,y:(normalization_layer(x),y))
#Test_dataset
test_dataset=test_dataset.map(lambda x,y:(normalization_layer(x),y))
```

Esta capa se aplica a las imágenes de los conjuntos de entrenamiento, validación y prueba utilizando el método `map()`, que permite aplicar la normalización a cada imagen de manera eficiente. El paso de normalización mejora la eficiencia y la estabilidad del modelo durante el entrenamiento, facilitando la convergencia más rápida y evitando que los valores de los píxeles de gran magnitud añadan un sesgo y afecten negativamente al aprendizaje del modelo.

7.3.3. Muestra de Batch de entrenamiento

Figura 9: Muestra de un batch de 32 imagenes del conjunto de entrenamiento con sus respectivas etiquetas.

7.3.4. Concatenación de conjuntos de datos para validación cruzada

Como se mencionó anteriormente, debido a la limitada cantidad de imágenes en el conjunto de datos `ACRIMA`, se decidió concatenar los tres subconjuntos (entrenamiento, validación y prueba) en un único arreglo de `NumPy`. Esta unificación permitió disponer de un solo conjunto de datos, el cual será utilizado para realizar la validación cruzada de 10 pliegues (folds). De esta manera, se maximiza el uso de los datos disponibles y se asegura una evaluación más robusta del modelo. A continuación, se muestra el código junto al detalle de los pasos realizados:

```
x_train = []
y_train = []
x_valid = []
y_valid = []
x_test = []
y_test = []

for im_b, lb_b in train_dataset:
    x_train.append(im_b.numpy())
    y_train.append(lb_b.numpy())

for im_b, lb_b in valid_dataset:
    x_valid.append(im_b.numpy())
    y_valid.append(lb_b.numpy())

for im_b, lb_b in test_dataset:
    x_test.append(im_b.numpy())
    y_test.append(lb_b.numpy())

x_train_all = np.concatenate(x_train, axis=0)
y_train_all = np.concatenate(y_train, axis=0)

x_valid_all = np.concatenate(x_valid, axis=0)
y_valid_all = np.concatenate(y_valid, axis=0)

x_test_all = np.concatenate(x_test, axis=0)
y_test_all = np.concatenate(y_test, axis=0)

x_train_cv = np.concatenate((x_train_all, x_valid_all), axis=0)
y_train_cv = np.concatenate((y_train_all, y_valid_all), axis=0)
```

- **Extracción de datos por lotes:** Cada uno de los conjuntos (entrenamiento, validación y prueba) se recorre para extraer las imágenes (`x`) y las etiquetas (`y`) de forma individual y almacenarlas en listas.
- **Concatenación de datos:** Una vez extraídas todas las imágenes y etiquetas, estas se concatenan utilizando la función `np.concatenate`, creando matrices únicas para cada conjunto.
- **Unificación para validación cruzada:** Finalmente, los conjuntos de entrenamiento, validación y prueba se combinan en un único conjunto (`x_train_cv` y `y_train_cv`), listo para ser utilizado en la validación cruzada de 10 pliegues (folds).

7.4. Definir gate network (router)

La gate network (router) es un componente esencial en el diseño de la arquitectura `Mixture of Experts, (MoE)`. Su principal función es recibir una imagen como entrada y decidir, a partir de sus características, qué modelo experto es más adecuado para procesar esa imagen. Este proceso permite asignar dinámicamente las entradas a los expertos, optimizando la utilización de los recursos computacionales y mejorando la capacidad del sistema para adaptarse a distintas entradas.

En esta implementación, la gate network (router) está creada con la siguiente arquitectura:

```
def gateNet(Input_layer, n, name='gate'):
  name = [name+str(i) for i in range(17)]

  # capas convolucionales + capas de pooling

  gate = keras.layers.Conv2D(32, (3,3), padding='same',
   ↪  activation='relu', name=name[0])(Input_layer)
  gate = keras.layers.Conv2D(32, (3,3), padding='same',
   ↪  activation='relu', name=name[1])(gate)
  gate = keras.layers.MaxPooling2D(pool_size=(2,2), name=name[2])(gate)
  gate = keras.layers.Dropout(0.25, name=name[3])(gate)
  gate = keras.layers.Conv2D(64, (3, 3), padding='same',
   ↪  activation='relu', name=name[4])(gate)
  gate = keras.layers.Conv2D(64, (3, 3), padding='same',
   ↪  activation='relu', name=name[5])(gate)
```

```
gate = keras.layers.MaxPooling2D(pool_size=(2,2), name=name[6])(gate)
gate = keras.layers.Dropout(0.25, name=name[7])(gate)
gate = keras.layers.Conv2D(128, (3, 3), padding='same',
↪  activation='relu', name=name[8])(gate)
gate = keras.layers.Conv2D(128, (3, 3), padding='same',
↪  activation='relu', name=name[9])(gate)
gate = keras.layers.Conv2D(128, (3, 3), padding='same',
↪  activation='relu', name=name[10])(gate)
gate = keras.layers.MaxPooling2D(pool_size=(2,2),
↪  name=name[11])(gate)
gate = keras.layers.Dropout(0.25, name=name[12])(gate)

# feature maps + Dense layers
gate = keras.layers.Flatten(name=name[13])(gate)
gate = keras.layers.Dense(512, activation='relu',
↪  name=name[14])(gate)
gate = keras.layers.Dropout(0.5, name=name[15])(gate)
output = keras.layers.Dense(n, activation='softmax',
↪  name=name[16])(gate)

model = keras.Model(inputs=Input_layer, outputs=output, name='gate')

return model
```

- **Capas Convolucionales y de Pooling:**
  - Se utilizan múltiples capas convolucionales (`Conv2D`) con función de activación `ReLU` y kernels de tamaño (3×3), que extraen características relevantes directamente de las imágenes de entrada.
  - Cada bloque convolucional incluye una capa de `Max-Pooling` para reducir la dimensionalidad espacial y acelerar el procesamiento, así como dropout para prevenir el sobreajuste.
- **Capas densas:**
  - Una vez extraídas las características, estas se aplanan mediante una capa `Flatten`.
  - Luego, se procesan en una capa densa con función de activación `ReLU`, que condensa la información en una representación más compacta.

- Finalmente, una capa de salida, con función de activación `Softmax`, produce una distribución de probabilidad sobre los n expertos disponibles. Esta distribución indica qué experto es más adecuado para procesar la imagen.

7.5. Transfer Learning

En el contexto de esta investigación, se utilizó la estrategia de aprendizaje por transferencia (Transfer Learning) aprovechando modelos de redes neuronales convolucionales pre-entrenadas disponibles a través de la biblioteca de `Keras`. Se seleccionaron tres arquitecturas ampliamente validadas en tareas de reconocimiento visual: `VGG19`, `ResNet50` y `DenseNet121`.

La metodología consistió en la descarga de estas redes con sus pesos previamente entrenados sobre el conjunto de datos estándar `ImageNet`. Cada arquitectura fue configurada excluyendo sus capas superiores originales (`include_top=False`), lo que permitió la obtención de sus bloques convolucionales. Adicionalmente, se congelaron los parámetros entrenables de estos modelos pre-entrenados, asegurando que sus pesos permanecieran inalterados durante el entrenamiento en el conjunto de datos `ACRIMA`.

```
# Descargar modelos base pre-entrenados
base_models = [
keras.applications.VGG19(input_shape=(128,128,3),
    weights='imagenet', include_top=False),

keras.applications.ResNet50(input_shape=(128,128,3),
    weights='imagenet', include_top=False),

keras.applications.DenseNet121(input_shape=(128,128,3),
    weights='imagenet', include_top=False)
]

#congelar parametros entrenables
for model in base_models:
  model.trainable = False
```

Esta estrategia permitió aprovechar las representaciones de características visuales aprendidas por estas arquitecturas, reduciendo significativamente el tiempo y los recursos computacionales necesarios para desarrollar un modelo de alto rendimiento desde cero. El entrenamiento se centró exclusivamente en las capas densas añadidas al final de estos modelos, que actuarán como expertos en la clasificación. La combinación de estas capacidades pre-entrenadas con las particularidades del problema de clasificación abordado en este trabajo garantiza un enfoque eficiente y efectivo.

7.6. Creación capa personalizada Mixture Of Experts

En esta sección, se describe la creación de una capa personalizada de `TensorFlow` basada en la arquitectura `Mixture of Experts (MoE)`, adaptada específicamente para redes neuronales convolucionales (CNNs). Esta capa, llamada `TopMExpertsFM`, tiene como objetivo seleccionar dinámicamente los expertos más relevantes para cada entrada de imágenes, utilizando como guía la salida generada por la gate network (router) previamente diseñada.

La gate network (router) asigna a cada experto una puntuación en forma de una distribución de probabilidad, indicando el experto más adecuado para procesar la entrada. A partir de esta distribución, la capa `TopMExpertsFM` selecciona los m mejores expertos, extrae y transforma sus mapas de características (feature maps), y los concatena. Los mapas de características (feature maps) combinados se utilizan posteriormente en capas densas para llevar a cabo la clasificación.

```
from tensorflow import keras
import tensorflow as tf


class TopMExpertsFM(keras.layers.Layer):
    def __init__(self, m, experts, output_dim=512, **kwargs):
        super(TopMExpertsFM, self).__init__(**kwargs)
        self.m = m  # Number of experts to select
        self.experts = experts  # List of CNN models (experts)
```

```
        self.output_dim = output_dim  # Desired output dimension for
        ↪ the experts' feature maps
        self.dense_layers = [keras.layers.Dense(output_dim) for _ in
        ↪ experts]

    def call(self, inputs, gate_output):
        batch_size = tf.shape(inputs)[0]
        num_experts = len(self.experts)

        # Get top M expert indices based on the gate output for each
        ↪ example in the batch
        top_n_indices = tf.argsort(gate_output,
        ↪ direction='DESCENDING', axis=-1)
        top_m_indices = top_n_indices[:, :self.m]  # Select indices
        ↪ of the top M experts

        # Get feature maps from each expert and ensure they have the
        ↪ same dimensions
        feature_maps = []
        for i in range(num_experts):
            expert_feature_map = self.experts[i](inputs)
            pooled_feature_map =
            ↪ GlobalAveragePooling2D()(expert_feature_map)
            # Apply dense layers to unify the dimensions of the
            ↪ feature maps
            reduced_feature_map =
            ↪ self.dense_layers[i](pooled_feature_map)
            feature_maps.append(reduced_feature_map)
```

```
        # Stack feature maps along a new dimension (number of
        ↪ experts)
        feature_maps = tf.stack(feature_maps, axis=1)  # Shape:
        ↪ (batch_size, num_experts, ...)

        # Create indices for batch and expert selection
        batch_indices = tf.range(batch_size)[:, tf.newaxis]  # Shape:
        ↪ (batch_size, 1)
        batch_indices = tf.tile(batch_indices, [1, self.m])  # Shape:
        ↪ (batch_size, m)
        indices = tf.stack([batch_indices, top_m_indices], axis=-1)  #
        ↪ Shape: (batch_size, m, 2)

        # Gather the top M experts' feature maps
        selected_feature_maps = tf.gather_nd(feature_maps, indices)  #
        ↪ Shape: (batch_size, m, ...)

        # Concatenate the selected feature maps along the channel
        ↪ dimension
        concatenated_feature_maps =
        ↪ tf.concat(tf.unstack(selected_feature_maps, axis=1),
        ↪ axis=-1)

        return concatenated_feature_maps

    def get_config(self):
        base_config = super(TopMExpertsFM, self).get_config()
        config = {
            "m": self.m,
            "output_dim": self.output_dim,
```

```
            "experts": [keras.utils.serialize_keras_object(expert) for
            ↪  expert in self.experts],  # Serialize experts
        }
        return {**base_config, **config}

    @classmethod
    def from_config(cls, config):
        experts_config = config.pop("experts")
        experts = [keras.utils.deserialize_keras_object(expert_config)
        ↪  for expert_config in experts_config]  # Deserialize
        ↪  experts
        return cls(experts=experts, **config)
```

7.6.1. Funcionamiento de la capa `TopMExpertsFM`

La implementación de esta capa `TopMExpertsFM` sigue un diseño modular que permite seleccionar dinámicamente los expertos más adecuados para cada entrada. A continuación, se detallará el funcionamiento de cada componente:

- **Inicialización del constructor (`__init__`):**
  El constructor define los parámetros de la capa:
  - `m`: el número de expertos a seleccionar.
  - `experts`: una lista con modelos `CNN` que actuarán como expertos.
  - `output_dim`: la dimensión deseada para los mapas de características finales combinados.

  Luego se inicializan capas densas (`Dense`) asociadas a cada experto para transformar sus mapas de características en una representación de tamaño uniforme.

  Esta etapa asegura que los mapas de características (feature maps) de los expertos seleccionados puedan ser concatenados independientemente de las diferencias entre las arquitecturas individuales de los expertos.

- **Método call:** Este método define el comportamiento de la capa cuando se pasa una entrada durante el entrenamiento o una inferencia. Su funcionalidad se puede dividir en varios pasos:

  - **Selección de expertos:**
    - Utilizando la salida de la gate network (`gate_output`), se determina una lista ordenada de los índices de los expertos basándose en sus puntajes.
    - Los `m` mejores expertos se seleccionan mediante el uso de `tf.argsort` y se extraen los índices correspondientes.
  - **Obtención de mapas de características (feature maps):**
    - Para cada experto en la lista, se pasa la entrada (`inputs`) a través de su modelo CNN correspondiente.
    - Los mapas de características resultantes son procesados con una capa de `GlobalAveragePooling2D` para reducir sus dimensiones espaciales.
  - **Selección y Concatenación:**
    - Los mapas de características de los expertos seleccionados se extraen dinámicamente utilizando los índices calculados previamente. Este paso asegura que solo los `m` mejores mapas de características sean considerados.
    - Finalmente, los mapas seleccionados se concatenan a lo largo de la dimensión de los canales utilizando `tf.concat`. Esto genera un único mapa de características combinado que representa la información más relevante de los expertos seleccionados.

- **Métodos Auxiliares:** `get_config` y `from_config` son métodos que permiten la serialización y deserialización de la capa, asegurando su reutilización para guardar o cargar el modelo.

La capa `TopMExpertsFM` actúa como un componente central en la arquitectura del modelo, integrando las salidas de la gate network con los mapas de características de los expertos. Este diseño permite al modelo:

- Adaptarse dinámicamente a entradas variadas al seleccionar diferentes combinaciones de expertos.
- Capturar información diversa y especializada de los expertos mediante la concatenación de sus mapas de características.

Al final, los mapas combinados se procesan a través de capas densas y una función de activación `Sigmoide` en la capa de salida para realizar la clasificación binaria final entre ojos con glaucoma y ojos normales.

La implementación modular y flexible de esta capa facilitará la comparación experimental, ya que permite variar el número de expertos a seleccionar `(m = 1, 2, 3)` y evaluar el impacto de este parámetro en la capacidad del modelo para generalizar.

7.7. Experimentos

En esta sección se detallan los experimentos realizados para evaluar el desempeño de la capa personalizada `TopMExpertsFM` en combinación con diferentes configuraciones de expertos. Se llevaron a cabo cuatro experimentos, creando modelos con esta capa variando el número de expertos seleccionados y evaluando también los expertos de manera individual. Cada experimento utilizó validación cruzada de 10 pliegues (folds) con 100 épocas de entrenamiento por cada pliegue (fold), garantizando así una evaluación robusta y generalizable de las métricas de desempeño.

- Por cada fold, se generó un reporte que incluye:
    - Una matriz de confusión, para analizar el rendimiento del modelo en términos de sus métricas.

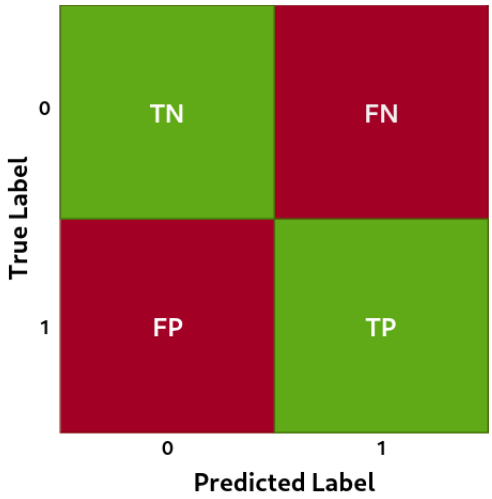

Figura 10: Ejemplo de Matriz de confusión

- Significado de los elementos en la matriz de confusión:
  - **Verdaderos Positivos (TP):** Son los valores que el modelo clasifica como positivos y que realmente son positivos.
  - **Verdaderos Negativos (TN):** Son valores que el modelo clasifica como negativos (0 en este caso) y que realmente son negativos.
  - **Falsos Positivos (FP):** Son valores que el modelo clasifica como positivo cuando realmente son negativos.
  - **Falsos Negativos (FN):** Son valores que el modelo clasifica como negativo cuando realmente son positivos.
- Métricas clave tales como:
  - **Accuracy:** Proporción de predicciones correctas con respecto al total.
  - **Recall:** Capacidad del modelo para identificar correctamente los casos positivos (ojos con glaucoma).
  - **Precision:** Proporción de predicciones positivas que son correctas.
  - **Specificity:** Capacidad del modelo para identificar correctamente los casos negativos (ojos normales).
  - **F1-Score:** Métrica combinada de precisión y sensibilidad, especialmente útil en casos de clases desbalanceadas.
  - **Error_rate:** Proporción de predicciones incorrectas.

- Estas métricas se obtienen de la siguiente forma:
  - **Accuracy:**
$$Accuracy = \frac{(TN + TP)}{(TN + FN + FP + TP)} \tag{1}$$
  - **Recall:**
$$Recall = \frac{(TP)}{(TP + FN)} \tag{2}$$
  - **Precision:**
$$Precision = \frac{(TP)}{(TP + FP)} \tag{3}$$
  - **Specificity:**
$$Specificity = \frac{(TN)}{(TN + FP)} \tag{4}$$
  - **F1-Score:**
$$F1 - Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{5}$$
  - **Error_rate:**
$$Error\_rate = 1 - Accuracy \tag{6}$$

En el contexto de diagnóstico de glaucoma, la evaluación del rendimiento del modelo requiere un enfoque cuidadoso. Dado que el objetivo principal es mejorar la detección de glaucoma, el `Recall` se convierte en una métrica crítica, ya que minimiza el riesgo de pasar por alto casos potencialmente graves. Simultáneamente, `Specificity` juega un papel fundamental para reducir diagnósticos incorrectos en pacientes sanos, evitando innecesarias intervenciones médicas.

7.7.1. Experimento 1: Validación Cruzada 1 Experto

En este experimento, se llevará a cabo una validación cruzada de 10 pliegues (folds) utilizando la capa creada basada en la Mezcla de Expertos `(Mixture of Experts, MoE)` seleccionando a un único experto. Este enfoque permitirá evaluar la capacidad del modelo para clasificar imágenes utilizando únicamente el experto seleccionado por la gate network (router).

Configuración del Experimento 1:

- **Validación Cruzada:** Se utilizó el método `KFold` de la biblioteca `sklearn` con el fin de dividir el conjunto de datos unificado (`x_train_cv` e `y_train_cv`) en 10 subconjuntos. En cada iteración, un subconjunto se utilizó como validación y los restantes como entrenamiento. Este enfoque garantiza que cada muestra sea evaluada una vez, proporcionando una evaluación robusta del rendimiento.

- **Arquitectura del Modelo:** La entrada al modelo fue una imagen de tamaño (128,128,3). Se creó una gate network (router) que, a partir de la entrada, predice cuál de los expertos es más adecuado para procesar cada imagen. El modelo incluyó una capa personalizada (`TopMExpertsFM`) que selecciona el experto con la mayor probabilidad predicha por la gate network (router) y obtiene sus mapas de características. Después de combinar los mapas de características, se añadieron capas densas para realizar la clasificación final.

- **Entrenamiento y Evaluación:** El modelo se entrenó con la función de pérdida *binary_crossentropy* y el optimizador *Adam* con una tasa de aprendizaje de 1e-3. Se registraron las métricas de (*Accuracy*), (*loss*), y otras métricas relevantes por cada pliegue. Durante cada iteración, se almacenaron las predicciones y las etiquetas reales del conjunto de validación para calcular las métricas de evaluación y construir matrices de confusión.

- **Cálculo de Métricas Promedio:** Al finalizar los 10 pliegues (folds), se calcularon los valores promedio de las métricas: *Accuracy*, *Recall*, *Precision* y *F1-Score*. Además, se reportó la desviación estándar de el *Accuracy* para evaluar la estabilidad del modelo.

```
#validacion cruzada y metricas
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, recall_score,
↪  precision_score, f1_score
kfold = KFold(n_splits=10, shuffle=True, random_state=42) #k = 10
↪  folds
fold_n = 1 #numero de fold

# Definir contenedores de score por fold
acc_per_fold = []
loss_per_fold = []

# Contenedores de matrices de confusion para cada fold
y_preds = []
y_trues = []

for train, valid in kfold.split(x_train_cv, y_train_cv):
    #crear modelo

    # crear la entrada para las imagenes
    input_layer = keras.layers.Input(shape=(128,128,3))
    # crear la red Gate
    gate_network = gateNet(input_layer, len(base_models))
    #Obtener salida de la red gate
    gate_output = gate_network(input_layer)
    # Crear la capa TopMExpertsFM
    top_m_layer = TopMExpertsFM(m=1, experts=base_models,
    ↪  output_dim=512)
    concatenated_features = top_m_layer(input_layer, gate_output)
```

```
x = keras.layers.Dense(512,
↪  activation='relu')(concatenated_features)
x = keras.layers.Dropout(0.5)(x)
x = keras.layers.Dense(128, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
x = keras.layers.Dense(64, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
output = keras.layers.Dense(1, activation='sigmoid')(x)

MoeModel = keras.models.Model(inputs = input_layer, outputs =
↪  output, name=f"MoeModel_fold_{fold_n}")

MoeModel.compile(loss='binary_crossentropy',
                 optimizer=keras.optimizers.Adam(),
                 metrics=['Accuracy'])

#print("--"*120)
print(f"Training for fold {fold_n} ...")

# logs for tensorboard
logs = './logs/select1/'
tb = TensorBoard(log_dir=logs+MoeModel.name)
#Entrenar modelo
MoeModel.fit(x_train_cv[train], y_train_cv[train],
          validation_data=(x_train_cv[valid], y_train_cv[valid]),
          epochs = epochs, #100
          batch_size=batch_size, #32
          callbacks=[tb])

#Puntuacion (val_loss y val_accuracy)
```

```
    scores = MoeModel.evaluate(x_train_cv[valid], y_train_cv[valid],
    ↪  verbose=0)
    print(f"Score for fold {fold_n}: {MoeModel.metrics_names[0]} of
    ↪  {scores[0]}; {MoeModel.metrics_names[1]} of {scores[1]*100}%")
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    # Predicciones en set de validación
    print(f"Making predictions for fold {fold_n}...")
    y_pred_k = MoeModel.predict(x_train_cv[valid])
    # Binarizar probabilidades umbral 0.5
    y_pred_k = np.where(y_pred_k >= 0.5, 1, 0)
    y_true_k = y_train_cv[valid]

    # almacenar predicciones y labels de cada fold
    y_trues.append(y_true_k)
    y_preds.append(y_pred_k)

    #aumentar contador fold
    fold_n += 1

avg_metrics_list = [] # lista para metricas promedio

# Matrices de confusion
print('-'*72)
print('Confusion matrix & Metrics per fold:')
print('-'*72)
for i in range(0, len(y_preds)):
    print(f'confusion matrix & metrics fold: {i+1}')
    metrics = plot_cm_metrics(y_trues[i], y_preds[i])
```

```
    avg_metrics_list.append(metrics)
    print('-'*72)

# == Provide average scores ==
print('-'*72)
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('-'*72)
  print(f'> Fold {i+1} - val_loss: {loss_per_fold[i]} - val_accuracy:
  ↪ {acc_per_fold[i]}%')
print('-'*72)
print('Average scores for all folds:')
print(f'> val_Accuracy: {np.mean(acc_per_fold)} (+-
↪ {np.std(acc_per_fold)})')
print(f'> val_Loss: {np.mean(loss_per_fold)}')
print('-'*72)

# == provide average metrics ==
print('Average Metrics for all folds:')

# Inicializar un diccionario para acumular los valores
suma_metricas = {key: 0 for key in avg_metrics_list[0]}

# Sumar los valores de cada métrica
for diccionario in avg_metrics_list:
    for clave, valor in diccionario.items():
        suma_metricas[clave] += valor

# Dividir entre el número de diccionarios para obtener el promedio
```

```
promedio_metricas = {clave: suma / len(avg_metrics_list) for clave,
↪  suma in suma_metricas.items()}

print("Promedio de métricas:")
for clave, valor in promedio_metricas.items():
    print(f"{clave}: {valor:.4f}")
print('-'*72)
```

7.7.2. Experimento 2: Validación Cruzada 2 Expertos

Configuración del Experimento 2: El procedimiento seguido en este experimento es idéntico al descrito en el `Experimento 1`, con la única diferencia de que la gate network selecciona los dos expertos más adecuados `(m=2)` para cada imagen. Esto permite evaluar el impacto de considerar múltiples expertos en el rendimiento del modelo `MoE`.

Se repitió la validación cruzada de 10 pliegues (folds), y las métricas de evaluación, tales como *Accuracy*, *Recall*, *Precision*, *F1-Score* y *Error_rate*, fueron calculadas en cada pliegue. Posteriormente, se promediaron los resultados para obtener el rendimiento general del modelo.

```
#validacion cruzada y metricas
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, recall_score,
↪  precision_score, f1_score
kfold = KFold(n_splits=10, shuffle=True, random_state=42) #k = 10
↪  folds
fold_n = 1 #numero de fold

# Definir contenedores de score por fold
acc_per_fold = []
loss_per_fold = []
```

```
# Contenedores de matrices de confusion para cada fold
y_preds = []
y_trues = []

for train, valid in kfold.split(x_train_cv, y_train_cv):
    #crear modelo

    # crear la entrada para las imagenes
    input_layer = keras.layers.Input(shape=(128,128,3))
    # crear la red Gate
    gate_network = gateNet(input_layer, len(base_models))
    #Obtener salida de la red gate
    gate_output = gate_network(input_layer)
    # Crear la capa TopMExpertsFM
    top_m_layer = TopMExpertsFM(m=2, experts=base_models,
    ↪  output_dim=512)
    concatenated_features = top_m_layer(input_layer, gate_output)
    x = keras.layers.Dense(512,
    ↪  activation='relu')(concatenated_features)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    output = keras.layers.Dense(1, activation='sigmoid')(x)

    MoeModel = keras.models.Model(inputs = input_layer, outputs =
    ↪  output, name=f"MoeModel_fold_{fold_n}")
```

```
MoeModel.compile(loss='binary_crossentropy',
                 optimizer=keras.optimizers.Adam(),
                 metrics=['Accuracy'])

#print("--"*120)
print(f"Training for fold {fold_n} ...")

# logs for tensorboard
logs = './logs/select2/'
tb = TensorBoard(log_dir=logs+MoeModel.name)
#Entrenar modelo
MoeModel.fit(x_train_cv[train], y_train_cv[train],
          validation_data=(x_train_cv[valid], y_train_cv[valid]),
          epochs = epochs, #100
          batch_size=batch_size, #32
          callbacks=[tb])

#Puntuacion (val_loss y val_accuracy)
scores = MoeModel.evaluate(x_train_cv[valid], y_train_cv[valid],
↪  verbose=0)
print(f"Score for fold {fold_n}: {MoeModel.metrics_names[0]} of
↪  {scores[0]}; {MoeModel.metrics_names[1]} of {scores[1]*100}%")
acc_per_fold.append(scores[1] * 100)
loss_per_fold.append(scores[0])

# Predicciones en set de validación
print(f"Making predictions for fold {fold_n}...")
y_pred_k = MoeModel.predict(x_train_cv[valid])
# Binarizar probabilidades umbral 0.5
y_pred_k = np.where(y_pred_k >= 0.5, 1, 0)
```

```
    y_true_k = y_train_cv[valid]

    # almacenar predicciones y labels de cada fold
    y_trues.append(y_true_k)
    y_preds.append(y_pred_k)

    #aumentar contador fold
    fold_n += 1

avg_metrics_list = [] # lista para metricas promedio

# Matrices de confusion
print('-'*72)
print('Confusion matrix & Metrics per fold:')
print('-'*72)
for i in range(0, len(y_preds)):
    print(f'confusion matrix & metrics fold: {i+1}')
    metrics = plot_cm_metrics(y_trues[i], y_preds[i])
    avg_metrics_list.append(metrics)
    print('-'*72)

# == Provide average scores ==
print('-'*72)
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('-'*72)
  print(f'> Fold {i+1} - val_loss: {loss_per_fold[i]} - val_accuracy:
  ↪ {acc_per_fold[i]}%')
print('-'*72)
print('Average scores for all folds:')
```

```
print(f'> val_Accuracy: {np.mean(acc_per_fold)} (+-
↪  {np.std(acc_per_fold)})')
print(f'> val_Loss: {np.mean(loss_per_fold)}')
print('-'*72)

# == provide average metrics ==
print('Average Metrics for all folds:')

# Inicializar un diccionario para acumular los valores
suma_metricas = {key: 0 for key in avg_metrics_list[0]}

# Sumar los valores de cada métrica
for diccionario in avg_metrics_list:
    for clave, valor in diccionario.items():
        suma_metricas[clave] += valor

# Dividir entre el número de diccionarios para obtener el promedio
promedio_metricas = {clave: suma / len(avg_metrics_list) for clave,
↪  suma in suma_metricas.items()}

print("Promedio de métricas:")
for clave, valor in promedio_metricas.items():
    print(f"{clave}: {valor:.4f}")
print('-'*72)
```

7.7.3. Experimento 3: Validación Cruzada 3 Expertos

Configuración del Experimento 3: De manera similar al `Experimento 2`, este experimento empleó el mismo procedimiento, pero incrementó el número de expertos seleccionados a tres (`m=3`). Esto permite observar cómo se comporta el modelo al considerar un conjunto aún mayor de expertos para fusionar las características.

Como en los experimentos anteriores, se realizó la validación cruzada de 10 pliegues (folds) y se calcularon las métricas de rendimiento promedio.

```
#validacion cruzada y metricas
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, recall_score,
↪  precision_score, f1_score
kfold = KFold(n_splits=10, shuffle=True, random_state=42) #k = 10
↪  folds
fold_n = 1 #numero de fold

# Definir contenedores de score por fold
acc_per_fold = []
loss_per_fold = []

# Contenedores de matrices de confusion para cada fold
y_preds = []
y_trues = []

for train, valid in kfold.split(x_train_cv, y_train_cv):
    #crear modelo

    # crear la entrada para las imagenes
    input_layer = keras.layers.Input(shape=(128,128,3))
    # crear la red Gate
    gate_network = gateNet(input_layer, len(base_models))
```

```
#Obtener salida de la red gate
gate_output = gate_network(input_layer)
# Crear la capa TopMExpertsFM
top_m_layer = TopMExpertsFM(m=3, experts=base_models,
↪  output_dim=512)
concatenated_features = top_m_layer(input_layer, gate_output)
x = keras.layers.Dense(512,
↪  activation='relu')(concatenated_features)
x = keras.layers.Dropout(0.5)(x)
x = keras.layers.Dense(128, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
x = keras.layers.Dense(64, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
output = keras.layers.Dense(1, activation='sigmoid')(x)

MoeModel = keras.models.Model(inputs = input_layer, outputs =
↪  output, name=f"MoeModel_fold_{fold_n}")

MoeModel.compile(loss='binary_crossentropy',
                 optimizer=keras.optimizers.Adam(),
                 metrics=['Accuracy'])

#print("--"*120)
print(f"Training for fold {fold_n} ...")

# logs for tensorboard
logs = './logs/select3/'
tb = TensorBoard(log_dir=logs+MoeModel.name)
#Entrenar modelo
MoeModel.fit(x_train_cv[train], y_train_cv[train],
```

```
            validation_data=(x_train_cv[valid], y_train_cv[valid]),
            epochs = epochs, #100
            batch_size=batch_size, #32
            callbacks=[tb])

    #Puntuacion (val_loss y val_accuracy)
    scores = MoeModel.evaluate(x_train_cv[valid], y_train_cv[valid],
    ↪  verbose=0)
    print(f"Score for fold {fold_n}: {MoeModel.metrics_names[0]} of
    ↪  {scores[0]}; {MoeModel.metrics_names[1]} of {scores[1]*100}%")
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    # Predicciones en set de validación
    print(f"Making predictions for fold {fold_n}...")
    y_pred_k = MoeModel.predict(x_train_cv[valid])
    # Binarizar probabilidades umbral 0.5
    y_pred_k = np.where(y_pred_k >= 0.5, 1, 0)
    y_true_k = y_train_cv[valid]

    # almacenar predicciones y labels de cada fold
    y_trues.append(y_true_k)
    y_preds.append(y_pred_k)

    #aumentar contador fold
    fold_n += 1

avg_metrics_list = [] # lista para metricas promedio

# Matrices de confusion
```

```
print('-'*72)
print('Confusion matrix & Metrics per fold:')
print('-'*72)
for i in range(0, len(y_preds)):
    print(f'confusion matrix & metrics fold: {i+1}')
    metrics = plot_cm_metrics(y_trues[i], y_preds[i])
    avg_metrics_list.append(metrics)
    print('-'*72)

# == Provide average scores ==
print('-'*72)
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('-'*72)
  print(f'> Fold {i+1} - val_loss: {loss_per_fold[i]} - val_accuracy:
  ↪ {acc_per_fold[i]}%')
print('-'*72)
print('Average scores for all folds:')
print(f'> val_Accuracy: {np.mean(acc_per_fold)} (+-
↪ {np.std(acc_per_fold)})')
print(f'> val_Loss: {np.mean(loss_per_fold)}')
print('-'*72)

# == provide average metrics ==
print('Average Metrics for all folds:')

# Inicializar un diccionario para acumular los valores
suma_metricas = {key: 0 for key in avg_metrics_list[0]}

# Sumar los valores de cada métrica
```

```
for diccionario in avg_metrics_list:
    for clave, valor in diccionario.items():
        suma_metricas[clave] += valor

# Dividir entre el número de diccionarios para obtener el promedio
promedio_metricas = {clave: suma / len(avg_metrics_list) for clave,
↪  suma in suma_metricas.items()}

print("Promedio de métricas:")
for clave, valor in promedio_metricas.items():
    print(f"{clave}: {valor:.4f}")
print('-'*72)
```

7.7.4. Experimento 4: Validación Cruzada Expertos individuales

En este experimento, se evaluarán los expertos seleccionados (`VGG19`, `ResNet50` y `DenseNet121`) de forma individual para observar su desempeño sin combinar las características mediante la capa `TopMexpertsFM`. La arquitectura específica de cada modelo incluye únicamente el modelo base pre-entrenado en `ImageNet` seguido de capas densas adicionales para la clasificación binaria, como se describe a continuación:

Modelo VGG19: Se empleará la arquitectura `VGG19` como extractor de características. Las características extraídas se aplanarán y se pasarán a través de capas densas con dropout para evitar sobreajuste. La última capa es una neurona con función de activación `Sigmoide` para la clasificación binaria.

```
#validacion cruzada y metricas
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, recall_score,
↪  precision_score, f1_score
```

```
kfold = KFold(n_splits=10, shuffle=True, random_state=42) #k = 10
↪  folds
fold_n = 1 #numero de fold

# Definir contenedores de score por fold
acc_per_fold = []
loss_per_fold = []

# Contenedores de matrices de confusion para cada fold
y_preds = []
y_trues = []

for train, valid in kfold.split(x_train_cv, y_train_cv):
    #crear modelo

    # crear la entrada para las imagenes
    input_layer = keras.layers.Input(shape=(128,128,3))
    x = base_models[0](input_layer)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(512, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    output = keras.layers.Dense(1, activation='sigmoid')(x)

    vgg19_model = keras.models.Model(inputs = input_layer, outputs =
    ↪  output, name=f"vgg19_model_fold_{fold_n}")
```

```
vgg19_model.compile(loss='binary_crossentropy',
                optimizer=keras.optimizers.Adam(),
                metrics=['Accuracy'])

#print("--"*120)
print(f"Training for fold {fold_n} ...")

# logs for tensorboard
logs = './logs/vgg19/'
tb = TensorBoard(log_dir=logs+vgg19_model.name)
#Entrenar modelo
vgg19_model.fit(x_train_cv[train], y_train_cv[train],
          validation_data=(x_train_cv[valid], y_train_cv[valid]),
          epochs = epochs, #100
          batch_size=batch_size, #32
          callbacks=[tb])

#Puntuacion (val_loss y val_accuracy)
scores = vgg19_model.evaluate(x_train_cv[valid],
↪  y_train_cv[valid], verbose=0)
print(f"Score for fold {fold_n}: {vgg19_model.metrics_names[0]} of
↪  {scores[0]}; {vgg19_model.metrics_names[1]} of
↪  {scores[1]*100}%")
acc_per_fold.append(scores[1] * 100)
loss_per_fold.append(scores[0])

# Predicciones en set de validación
print(f"Making predictions for fold {fold_n}...")
y_pred_k = vgg19_model.predict(x_train_cv[valid])
# Binarizar probabilidades umbral 0.5
```

```
    y_pred_k = np.where(y_pred_k >= 0.5, 1, 0)
    y_true_k = y_train_cv[valid]

    # almacenar predicciones y labels de cada fold
    y_trues.append(y_true_k)
    y_preds.append(y_pred_k)

    #aumentar contador fold
    fold_n += 1

avg_metrics_list = [] # lista para metricas promedio

# Matrices de confusion
print('-'*72)
print('Confusion matrix & Metrics per fold:')
print('-'*72)
for i in range(0, len(y_preds)):
    print(f'confusion matrix & metrics fold: {i+1}')
    metrics = plot_cm_metrics(y_trues[i], y_preds[i])
    avg_metrics_list.append(metrics)
    print('-'*72)

# == Provide average scores ==
print('-'*72)
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('-'*72)
  print(f'> Fold {i+1} - val_loss: {loss_per_fold[i]} - val_accuracy:
  ↪ {acc_per_fold[i]}%')
print('-'*72)
```

```
print('Average scores for all folds:')
print(f'> val_Accuracy: {np.mean(acc_per_fold)} (+-
↪  {np.std(acc_per_fold)})')
print(f'> val_Loss: {np.mean(loss_per_fold)}')
print('-'*72)

# == provide average metrics ==
print('Average Metrics for all folds:')

# Inicializar un diccionario para acumular los valores
suma_metricas = {key: 0 for key in avg_metrics_list[0]}

# Sumar los valores de cada métrica
for diccionario in avg_metrics_list:
    for clave, valor in diccionario.items():
        suma_metricas[clave] += valor

# Dividir entre el número de diccionarios para obtener el promedio
promedio_metricas = {clave: suma / len(avg_metrics_list) for clave,
↪  suma in suma_metricas.items()}

print("Promedio de métricas:")
for clave, valor in promedio_metricas.items():
    print(f"{clave}: {valor:.4f}")
print('-'*72)
```

Modelo ResNet50: Se utilizará `ResNet50` como extractor de características con configuraciones similares a las mencionadas para `VGG19`.

```
#validacion cruzada y metricas
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, recall_score,
↪  precision_score, f1_score
kfold = KFold(n_splits=10, shuffle=True, random_state=42) #k = 10
↪  folds
fold_n = 1 #numero de fold

# Definir contenedores de score por fold
acc_per_fold = []
loss_per_fold = []

# Contenedores de matrices de confusion para cada fold
y_preds = []
y_trues = []

for train, valid in kfold.split(x_train_cv, y_train_cv):
    #crear modelo

    # crear la entrada para las imagenes
    input_layer = keras.layers.Input(shape=(128,128,3))
    x = base_models[1](input_layer)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(512, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
```

```
output = keras.layers.Dense(1, activation='sigmoid')(x)

resnet50_model = keras.models.Model(inputs = input_layer, outputs
↪  = output, name=f"resnet50_model_fold_{fold_n}")

resnet50_model.compile(loss='binary_crossentropy',
                optimizer=keras.optimizers.Adam(),
                metrics=['Accuracy'])

#print("--"*120)
print(f"Training for fold {fold_n} ...")

# logs for tensorboard
logs = './logs/resnet50/'
tb = TensorBoard(log_dir=logs+resnet50_model.name)
#Entrenar modelo
resnet50_model.fit(x_train_cv[train], y_train_cv[train],
         validation_data=(x_train_cv[valid], y_train_cv[valid]),
         epochs = epochs, #100
         batch_size=batch_size, #32
         callbacks=[tb])

#Puntuacion (val_loss y val_accuracy)
scores = resnet50_model.evaluate(x_train_cv[valid],
↪  y_train_cv[valid], verbose=0)
print(f"Score for fold {fold_n}: {resnet50_model.metrics_names[0]}
↪  of {scores[0]}; {resnet50_model.metrics_names[1]} of
↪  {scores[1]*100}%")
acc_per_fold.append(scores[1] * 100)
loss_per_fold.append(scores[0])
```

```
    # Predicciones en set de validación
    print(f"Making predictions for fold {fold_n}...")
    y_pred_k = resnet50_model.predict(x_train_cv[valid])
    # Binarizar probabilidades umbral 0.5
    y_pred_k = np.where(y_pred_k >= 0.5, 1, 0)
    y_true_k = y_train_cv[valid]

    # almacenar predicciones y labels de cada fold
    y_trues.append(y_true_k)
    y_preds.append(y_pred_k)

    #aumentar contador fold
    fold_n += 1

avg_metrics_list = [] # lista para metricas promedio

# Matrices de confusion
print('-'*72)
print('Confusion matrix & Metrics per fold:')
print('-'*72)
for i in range(0, len(y_preds)):
    print(f'confusion matrix & metrics fold: {i+1}')
    metrics = plot_cm_metrics(y_trues[i], y_preds[i])
    avg_metrics_list.append(metrics)
    print('-'*72)

# == Provide average scores ==
print('-'*72)
print('Score per fold')
```

```
for i in range(0, len(acc_per_fold)):
  print('-'*72)
  print(f'> Fold {i+1} - val_loss: {loss_per_fold[i]} - val_accuracy:
  ↪ {acc_per_fold[i]}%')
print('-'*72)
print('Average scores for all folds:')
print(f'> val_Accuracy: {np.mean(acc_per_fold)} (+-
↪ {np.std(acc_per_fold)})')
print(f'> val_Loss: {np.mean(loss_per_fold)}')
print('-'*72)

# == provide average metrics ==
print('Average Metrics for all folds:')

# Inicializar un diccionario para acumular los valores
suma_metricas = {key: 0 for key in avg_metrics_list[0]}

# Sumar los valores de cada métrica
for diccionario in avg_metrics_list:
    for clave, valor in diccionario.items():
        suma_metricas[clave] += valor

# Dividir entre el número de diccionarios para obtener el promedio
promedio_metricas = {clave: suma / len(avg_metrics_list) for clave,
↪ suma in suma_metricas.items()}

print("Promedio de métricas:")
for clave, valor in promedio_metricas.items():
    print(f"{clave}: {valor:.4f}")
print('-'*72)
```

Modelo DenseNet121: `DenseNet121` será utilizado como extractor de características con configuraciones similares a las mencionadas para `VGG19` y `ResNet50`.

```
#validacion cruzada y metricas
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, recall_score,
↪  precision_score, f1_score
kfold = KFold(n_splits=10, shuffle=True, random_state=42) #k = 10
↪  folds
fold_n = 1 #numero de fold

# Definir contenedores de score por fold
acc_per_fold = []
loss_per_fold = []

# Contenedores de matrices de confusion para cada fold
y_preds = []
y_trues = []

for train, valid in kfold.split(x_train_cv, y_train_cv):
    #crear modelo

    # crear la entrada para las imagenes
    input_layer = keras.layers.Input(shape=(128,128,3))
    x = base_models[2](input_layer)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(512, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dropout(0.5)(x)
    x = keras.layers.Dense(64, activation='relu')(x)
```

```
x = keras.layers.Dropout(0.5)(x)
output = keras.layers.Dense(1, activation='sigmoid')(x)

densenet121_model = keras.models.Model(inputs = input_layer,
↪  outputs = output, name=f"densenet121_model_fold_{fold_n}")

densenet121_model.compile(loss='binary_crossentropy',
                optimizer=keras.optimizers.Adam(),
                metrics=['Accuracy'])

#print("--"*120)
print(f"Training for fold {fold_n} ...")

# logs for tensorboard
logs = './logs/densenet121/'
tb = TensorBoard(log_dir=logs+densenet121_model.name)
#Entrenar modelo
densenet121_model.fit(x_train_cv[train], y_train_cv[train],
          validation_data=(x_train_cv[valid], y_train_cv[valid]),
          epochs = epochs, #100
          batch_size=batch_size, #32
          callbacks=[tb])

#Puntuacion (val_loss y val_accuracy)
scores = densenet121_model.evaluate(x_train_cv[valid],
↪  y_train_cv[valid], verbose=0)
print(f"Score for fold {fold_n}:
↪  {densenet121_model.metrics_names[0]} of {scores[0]};
↪  {densenet121_model.metrics_names[1]} of {scores[1]*100}%")
acc_per_fold.append(scores[1] * 100)
```

```
    loss_per_fold.append(scores[0])

    # Predicciones en set de validación
    print(f"Making predictions for fold {fold_n}...")
    y_pred_k = densenet121_model.predict(x_train_cv[valid])
    # Binarizar probabilidades umbral 0.5
    y_pred_k = np.where(y_pred_k >= 0.5, 1, 0)
    y_true_k = y_train_cv[valid]

    # almacenar predicciones y labels de cada fold
    y_trues.append(y_true_k)
    y_preds.append(y_pred_k)

    #aumentar contador fold
    fold_n += 1

avg_metrics_list = [] # lista para metricas promedio

# Matrices de confusion
print('-'*72)
print('Confusion matrix & Metrics per fold:')
print('-'*72)
for i in range(0, len(y_preds)):
    print(f'confusion matrix & metrics fold: {i+1}')
    metrics = plot_cm_metrics(y_trues[i], y_preds[i])
    avg_metrics_list.append(metrics)
    print('-'*72)

# == Provide average scores ==
print('-'*72)
```

```
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('-'*72)
  print(f'> Fold {i+1} - val_loss: {loss_per_fold[i]} - val_accuracy:
  ↪ {acc_per_fold[i]}%')
print('-'*72)
print('Average scores for all folds:')
print(f'> val_Accuracy: {np.mean(acc_per_fold)} (+-
↪ {np.std(acc_per_fold)})')
print(f'> val_Loss: {np.mean(loss_per_fold)}')
print('-'*72)

# == provide average metrics ==
print('Average Metrics for all folds:')

# Inicializar un diccionario para acumular los valores
suma_metricas = {key: 0 for key in avg_metrics_list[0]}

# Sumar los valores de cada métrica
for diccionario in avg_metrics_list:
    for clave, valor in diccionario.items():
        suma_metricas[clave] += valor

# Dividir entre el número de diccionarios para obtener el promedio
promedio_metricas = {clave: suma / len(avg_metrics_list) for clave,
↪ suma in suma_metricas.items()}

print("Promedio de métricas:")
for clave, valor in promedio_metricas.items():
    print(f"{clave}: {valor:.4f}")
```

```
print('-'*72)
```

El procedimiento de validación cruzada de 10 pliegues (folds) es el mismo que en los experimentos previos. Para cada modelo, se congelarán los pesos del extractor de características y solo se entrenarán las capas densas añadidas. Se calcularán las mismas métricas de evaluación para cada pliegue y se promediarán para obtener los resultados finales.

8. Resultados

8.1. Resultados del Experimento 1

En este experimento, se evaluará el desempeño del modelo `Mixture of Experts, (MoE)` configurado para seleccionar un único experto en cada iteración de validación cruzada de 10 pliegues (folds). Los resultados se analizarán en términos de las matrices de confusión y las métricas de evaluación obtenidas para cada pliegue (fold). A continuación, se presentarán las matrices de confusión y las métricas detalladas para cada pliegue (fold), seguidas de un análisis de los resultados promedio.

8.1.1. Resultados por fold:

- **Fold 1:**

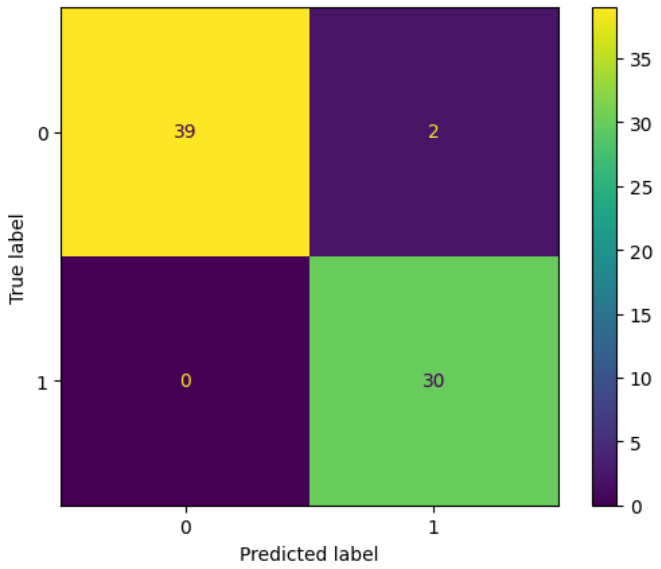

Figura 11: Matriz de confusión fold 1 experimento 1

- Metrics fold 1:
  - **Accuracy:** 0.9718
  - **Recall:** 1.0
  - **Precision:** 0.9375

- **F1-Score:** 0.9677
- **Specificity:** 0.9512
- **Error_rate:** 0.0282

- **Fold 2:**

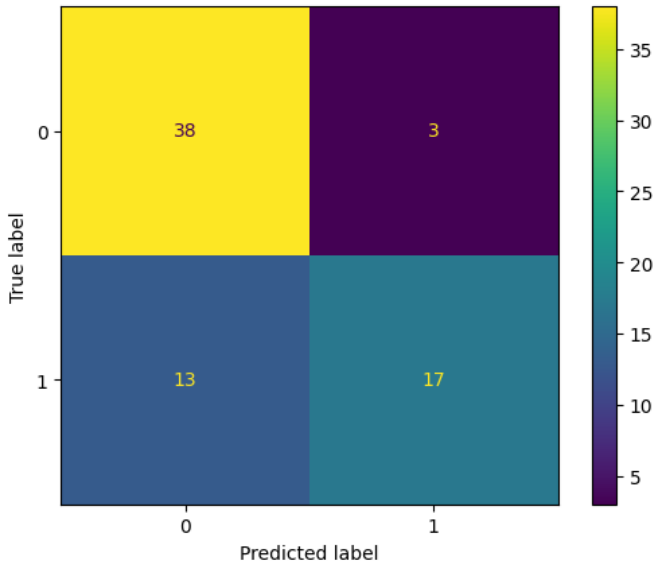

Figura 12: Matriz de confusión fold 2 experimento 1

- Metrics fold 2:

  - **Accuracy:** 0.7746
  - **Recall:** 0.5667
  - **Precision:** 0.8500
  - **F1-Score:** 0.6800
  - **Specificity:** 0.9268
  - **Error_rate:** 0.2253

- **Fold 3:**

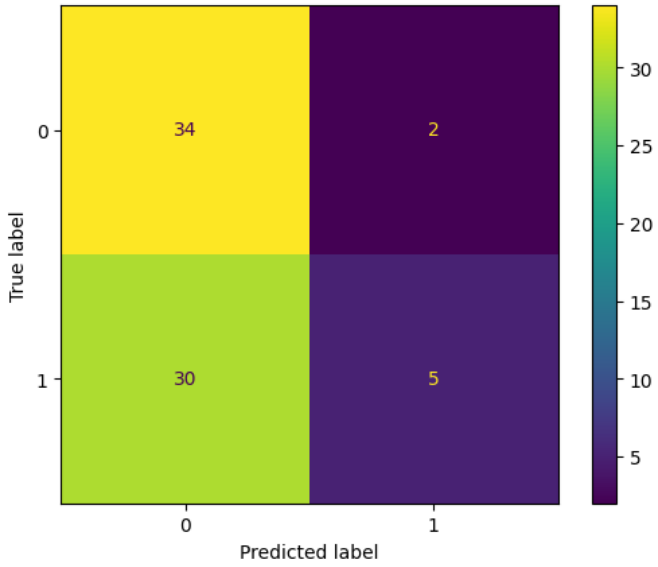

Figura 13: Matriz de confusión fold 3 experimento 1

- Metrics fold 3:
  - **Accuracy:** 0.5493
  - **Recall:** 0.1429
  - **Precision:** 0.7143
  - **F1-Score:** 0.2381
  - **Specificity:** 0.9444
  - **Error_rate:** 0.4507

- **Fold 4:**

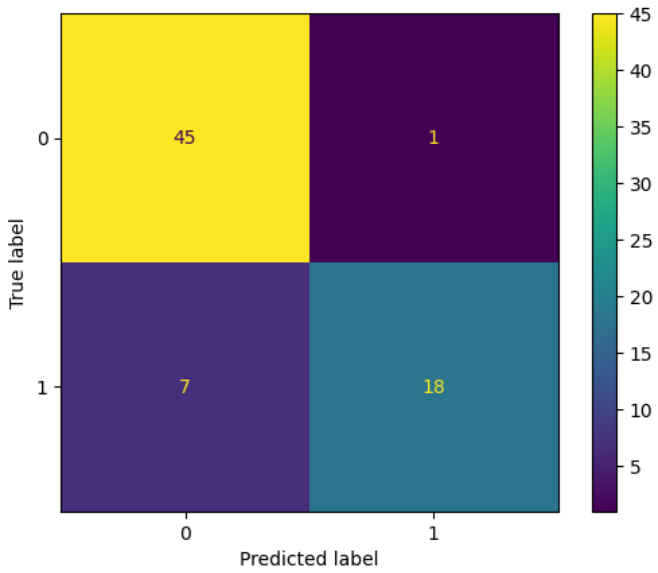

Figura 14: Matriz de confusión fold 4 experimento 1

- Metrics fold 4:
  - **Accuracy:** 0.8873
  - **Recall:** 0.7200
  - **Precision:** 0.9474
  - **F1-Score:** 0.8182
  - **Specificity:** 0.9783
  - **Error_rate:** 0.1127

- **Fold 5:**

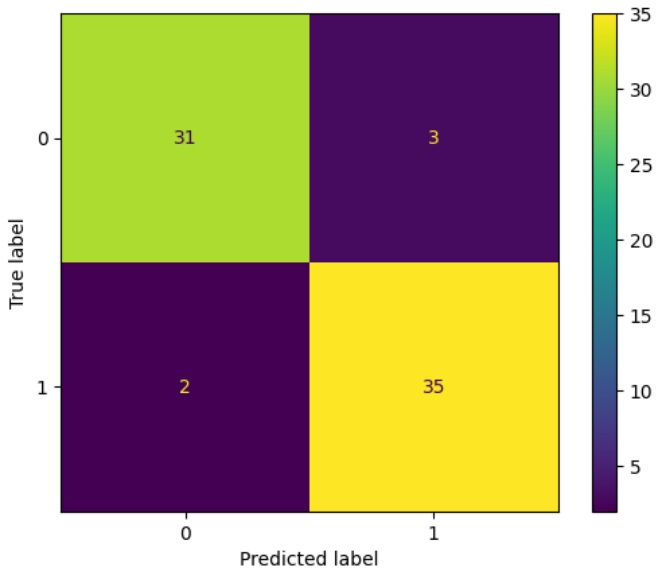

Figura 15: Matriz de confusión fold 5 experimento 1

- Metrics fold 5:
  - **Accuracy:** 0.9296
  - **Recall:** 0.9459
  - **Precision:** 0.9210
  - **F1-Score:** 0.9333
  - **Specificity:** 0.9118
  - **Error_rate:** 0.0704

- **Fold 6:**

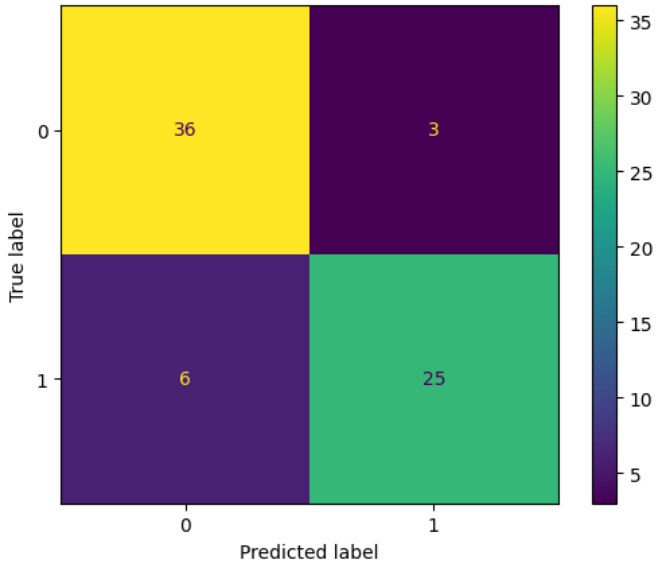

Figura 16: Matriz de confusión fold 6 experimento 1

- Metrics fold 6:
  - **Accuracy:** 0.8714
  - **Recall:** 0.8064
  - **Precision:** 0.8928
  - **F1-Score:** 0.8474
  - **Specificity:** 0.9231
  - **Error_rate:** 0.1286

- **Fold 7:**

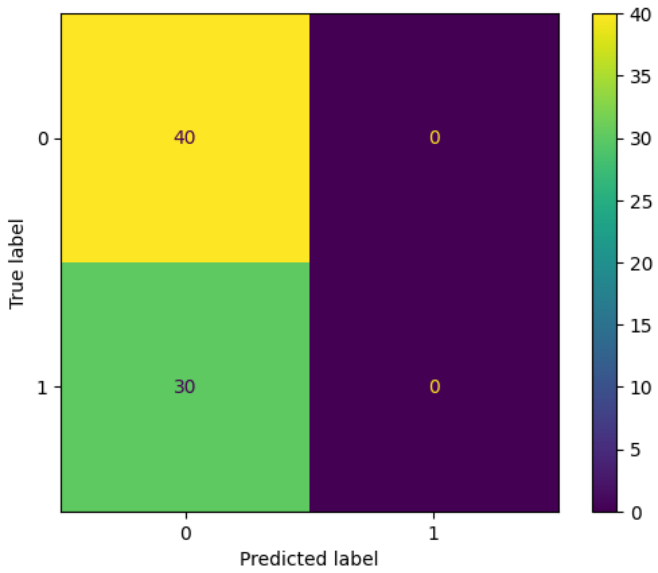

Figura 17: Matriz de confusión fold 7 experimento 1

- Metrics fold 7:
  - **Accuracy:** 0.5714
  - **Recall:** 0.0
  - **Precision:** 0.0
  - **F1-Score:** 0.0
  - **Specificity:** 1.0
  - **Error_rate:** 0.4286

- **Fold 8:**

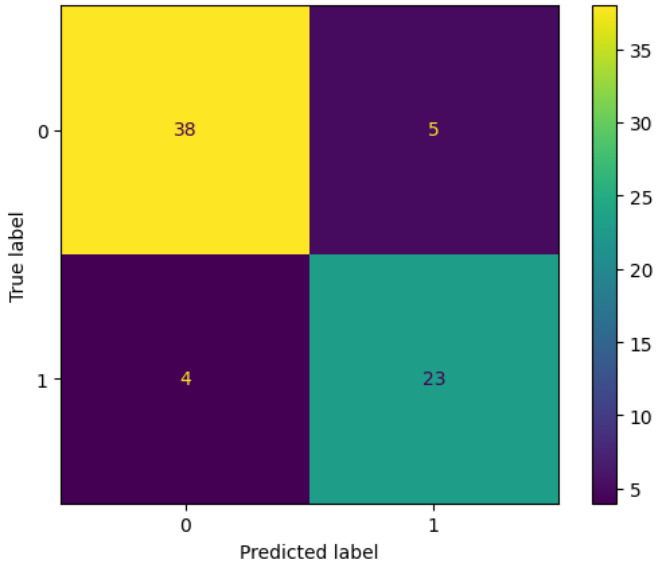

Figura 18: Matriz de confusión fold 8 experimento 1

- Metrics fold 8:
  - **Accuracy:** 0.8714
  - **Recall:** 0.8518
  - **Precision:** 0.8214
  - **F1-Score:** 0.8364
  - **Specificity:** 0.8837
  - **Error_rate:** 0.1286

- **Fold 9:**

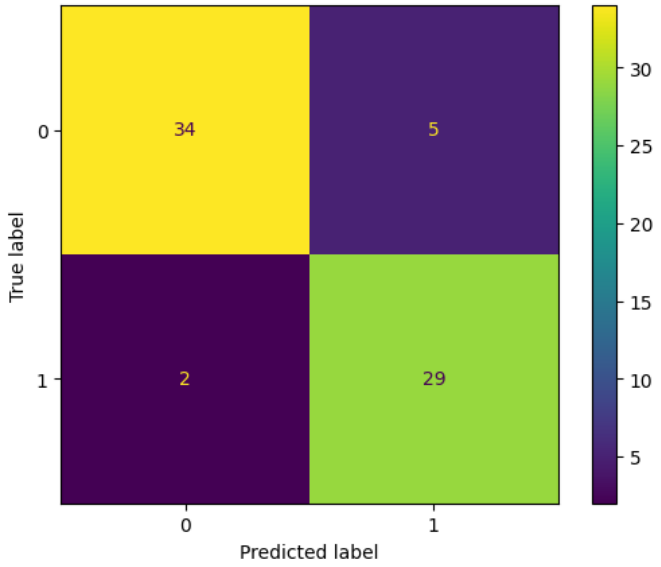

Figura 19: Matriz de confusión fold 9 experimento 1

- Metrics fold 9:
  - **Accuracy:** 0.9000
  - **Recall:** 0.9355
  - **Precision:** 0.8529
  - **F1-Score:** 0.8923
  - **Specificity:** 0.8718
  - **Error_rate:** 0.0999

- **Fold 10:**

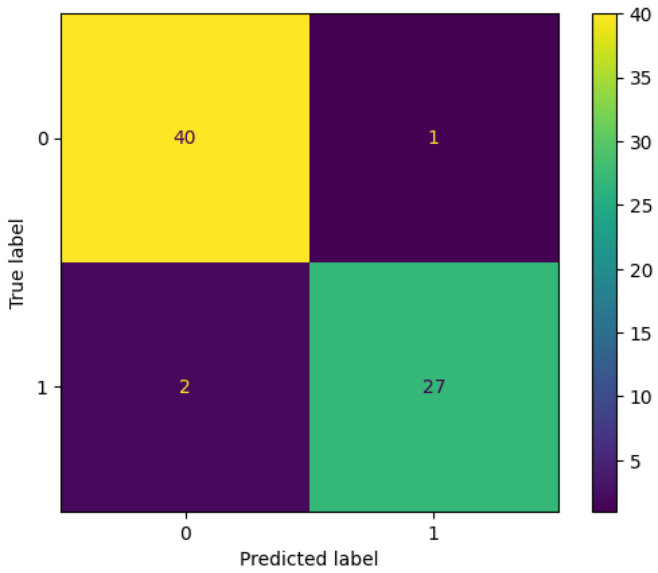

Figura 20: Matriz de confusión fold 10 experimento 1

- Metrics fold 10:
  - **Accuracy:** 0.9571
  - **Recall:** 0.9310
  - **Precision:** 0.9643
  - **F1-Score:** 0.9474
  - **Specificity:** 0.9756
  - **Error_rate:** 0.0428

fold	Pérdida (val_loss)	Accuracy (val_accuracy)
1	0.6064	97.18 %
2	0.5617	77.46 %
3	0.6011	54.93 %
4	0.2679	88.73 %
5	0.2329	92.96 %
6	0.8330	87.14 %
7	0.6796	57.14 %
8	0.4363	87.14 %
9	0.4195	90.00 %
10	0.2600	95.71 %
avg	**0.4898**	**82.84 % ± 14.37 %**

Cuadro 1: Resultados por fold y promedio (avg): pérdida (*val_loss*) y Accuracy (*val_accuracy*).

Métrica	Promedio (avg)
Accuracy	0.8284
Recall	0.6900
Precision	0.7902
F1-Score	0.7161
Specificity	0.9367
Error_rate	0.1716

Cuadro 2: Métricas promedio obtenidas en los folds de validación cruzada.

8.1.2. Análisis preliminar de los resultados Experimento 1

Los resultados obtenidos en el Experimento 1 revelan un desempeño global razonable en el modelo propuesto seleccionando un único experto en la validación cruzada. Como se observa en el Cuadro 1, el *Accuracy* promedio alcanzado fue de 82.84 % ± 14.37 %, con variaciones notables entre los diferentes pliegues, reflejadas en un rango que oscila entre 54.93 % y 97.18 %. La desviación estándar de 14.37 % indica una considerable variabilidad en el desempeño del modelo entre los distintos pliegues. Esto sugiere que el modelo podría ser sensible a la selección de los subconjuntos de datos durante la partición en la validación cruzada, lo que podría afectar su estabilidad y generalización en diferentes conjuntos de datos.

Por otro lado, las métricas promedio presentadas en el Cuadro 2 brindan una visión más completa del desempeño del modelo. Aunque su *Precision* promedio es relativamente

alta, con un valor de 0.7902, *Recall*, que mide la capacidad del modelo para identificar correctamente los ojos con glaucoma, es más modesta, alcanzando 0.6900. Esto podría indicar cierta dificultad para manejar correctamente los casos positivos. Sin embargo, la *Specificity* promedio es notablemente alta, 0.9367, lo que sugiere que el modelo tiene un buen desempeño al identificar los ojos normales.

El *F1-Score* promedio de 0.7161 refleja un balance aceptable entre *Precision* y *Recall*, aunque existen áreas claras para mejorar en futuras iteraciones del modelo. Finalmente, el *Error_rate* promedio es de 0.1716, lo que resalta el porcentaje de predicciones incorrectas en general.

8.1.3. Conclusión preliminar Experimento 1

En estos resultados iniciales, el modelo basado en un único experto demuestra un desempeño razonable, aunque con áreas de mejora, especialmente en términos de *Recall*. La desviación estándar de 14.37 % sugiere una variabilidad considerable en el desempeño del modelo, lo que indica que el modelo podría ser sensible a las particiones de datos utilizadas en la validación cruzada. Este análisis se profundizará una vez que se realicen y comparen los resultados obtenidos en los demás experimentos, donde se explorarán configuraciones con múltiples expertos y arquitecturas individuales adicionales. Esto permitirá evaluar si la combinación de expertos o el ajuste de las arquitecturas individuales pueden ofrecer ventajas significativas sobre la configuración inicial, especialmente en términos de estabilidad y consistencia en el desempeño.

8.2. Resultados del Experimento 2

En el Experimento 2, se evaluará el desempeño del modelo utilizando una configuración con múltiples expertos. A diferencia del Experimento 1, donde se seleccionó un único experto, en este caso se seleccionarán los mejores 2 expertos para permitir una toma de decisiones más robusta y diversificada. A través de la validación cruzada, se obtuvieron los resultados por pliegue (fold) que reflejan el comportamiento del modelo con esta nueva configuración. A continuación, se presentan las métricas obtenidas para cada pliegue (fold), incluyendo sus matrices de confusión. Este análisis proporcionará una visión más

detallada de cómo la combinación de expertos influye en el desempeño global del modelo, en comparación con los resultados del Experimento 1.

8.2.1. Resultados por fold:

- **Fold 1:**

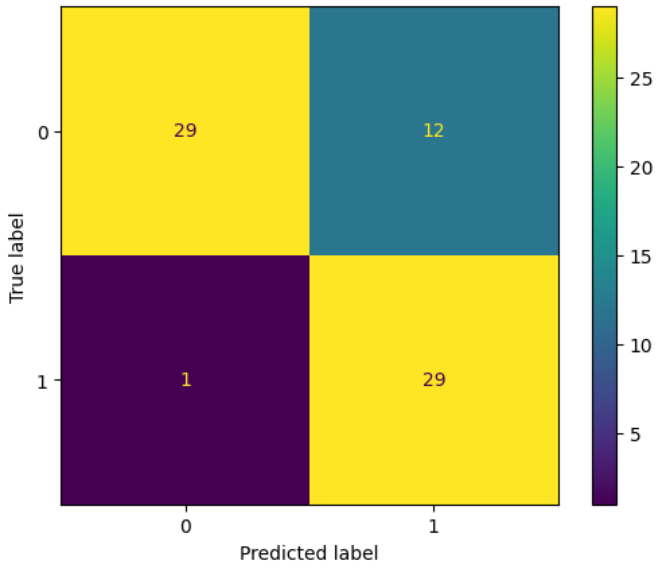

Figura 21: Matriz de confusión fold 1 experimento 2

- Metrics fold 1:
  - **Accuracy:** 0.8169
  - **Recall:** 0.9667
  - **Precision:** 0.7073
  - **F1-Score:** 0.8169
  - **Specificity:** 0.7073
  - **Error_rate:** 0.1831
- **Fold 2:**
- Metrics fold 2:
  - **Accuracy:** 0.9296

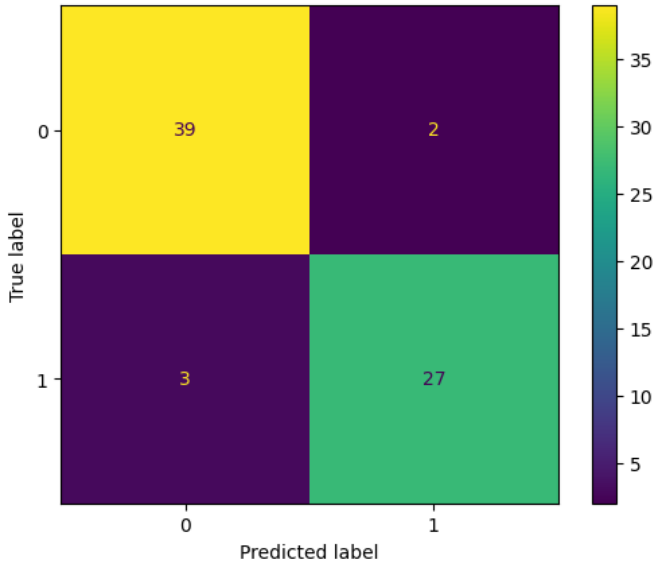

Figura 22: Matriz de confusión fold 2 experimento 2

- **Recall:** 0.9000
- **Precision:** 0.9310
- **F1-Score:** 0.9152
- **Specificity:** 0.9512
- **Error_rate:** 0.0704

- **Fold 3:**

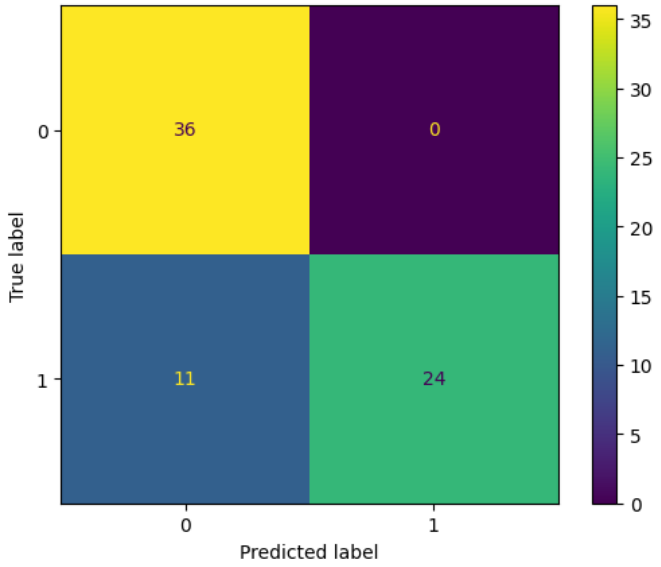

Figura 23: Matriz de confusión fold 3 experimento 2

- Metrics fold 3:
  - **Accuracy:** 0.8451
  - **Recall:** 0.6857
  - **Precision:** 1.0000
  - **F1-Score:** 0.8136
  - **Specificity:** 1.0000
  - **Error_rate:** 0.1549

- **Fold 4:**

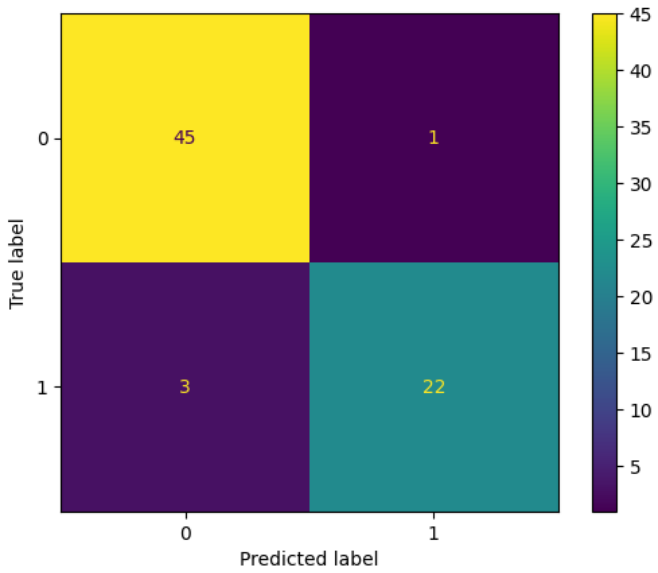

Figura 24: Matriz de confusión fold 4 experimento 2

- Metrics fold 4:
  - **Accuracy:** 0.9437
  - **Recall:** 0.8800
  - **Precision:** 0.9565
  - **F1-Score:** 0.9167
  - **Specificity:** 0.9783
  - **Error_rate:** 0.0563

- **Fold 5:**

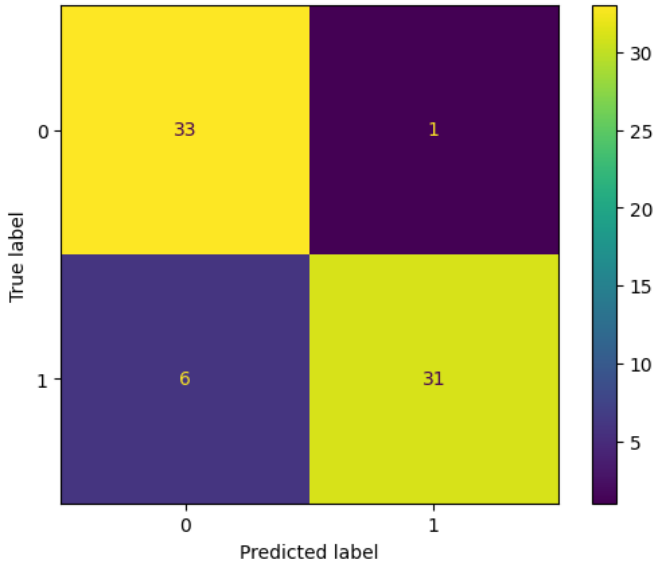

Figura 25: Matriz de confusión fold 5 experimento 2

- Metrics fold 5:
  - **Accuracy:** 0.9014
  - **Recall:** 0.8378
  - **Precision:** 0.9687
  - **F1-Score:** 0.8985
  - **Specificity:** 0.9706
  - **Error_rate:** 0.0986

- **Fold 6:**

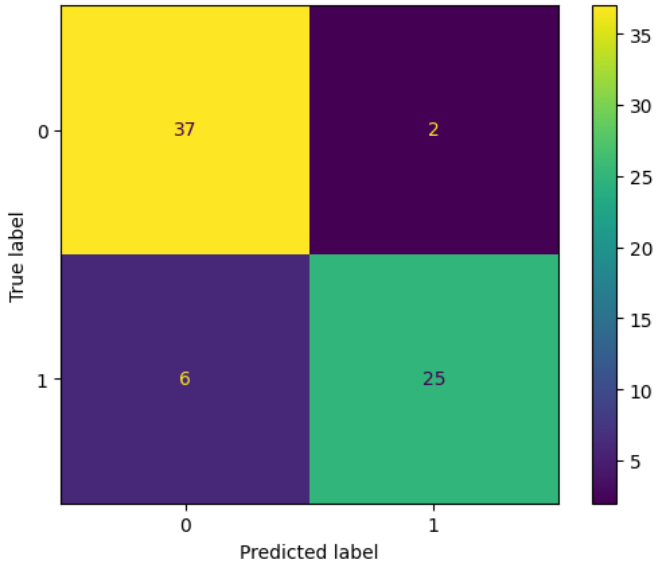

Figura 26: Matriz de confusión fold 6 experimento 2

- Metrics fold 6:
  - **Accuracy:** 0.8857
  - **Recall:** 0.8064
  - **Precision:** 0.9259
  - **F1-Score:** 0.8621
  - **Specificity:** 0.9487
  - **Error_rate:** 0.1143

- **Fold 7:**

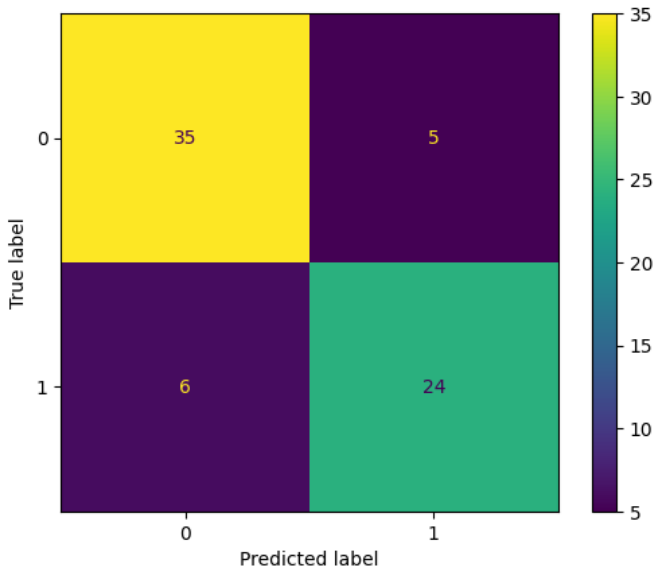

Figura 27: Matriz de confusión fold 7 experimento 2

- Metrics fold 7:
  - **Accuracy:** 0.8429
  - **Recall:** 0.8000
  - **Precision:** 0.8276
  - **F1-Score:** 0.8136
  - **Specificity:** 0.8750
  - **Error_rate:** 0.1571

- **Fold 8:**

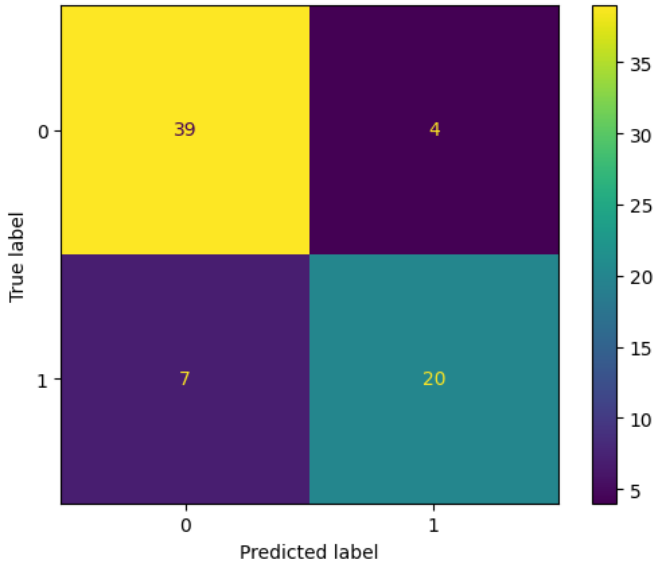

Figura 28: Matriz de confusión fold 8 experimento 2

- Metrics fold 8:
  - **Accuracy:** 0.8429
  - **Recall:** 0.7407
  - **Precision:** 0.8334
  - **F1-Score:** 0.7843
  - **Specificity:** 0.9070
  - **Error_rate:** 0.1571

- **Fold 9:**

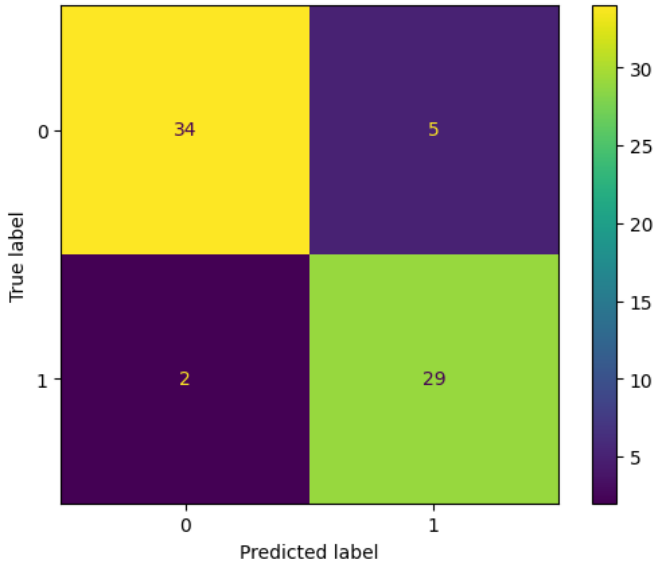

Figura 29: Matriz de confusión fold 9 experimento 2

- Metrics fold 9:
  - **Accuracy:** 0.9000
  - **Recall:** 0.9355
  - **Precision:** 0.8529
  - **F1-Score:** 0.8923
  - **Specificity:** 0.8718
  - **Error_rate:** 0.0999

- **Fold 10:**

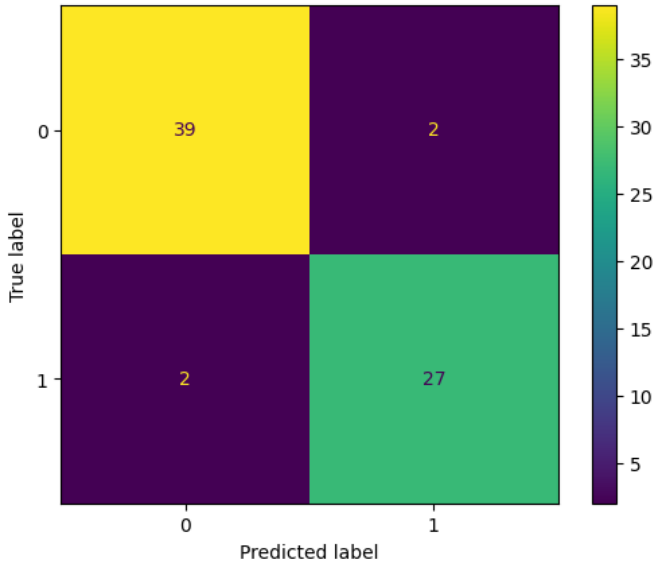

Figura 30: Matriz de confusión fold 10 experimento 1

- Metrics fold 10:
  - **Accuracy:** 0.9428
  - **Recall:** 0.9310
  - **Precision:** 0.9310
  - **F1-Score:** 0.9310
  - **Specificity:** 0.9512
  - **Error_rate:** 0.0571

fold	Pérdida (val_loss)	Accuracy (val_accuracy)
1	0.5157	81.69 %
2	0.2459	92.96 %
3	0.2897	84.51 %
4	0.2582	94.37 %
5	0.3195	90.14 %
6	0.6835	88.57 %
7	0.3209	84.29 %
8	0.3176	84.29 %
9	0.3252	90.00 %
10	0.2493	94.29 %
avg	**0.3525**	**88.51 % ± 4.37 %**

Cuadro 3: Resultados por fold y promedio (avg): pérdida (*val_loss*) y Accuracy (*val_accuracy*).

Métrica	Promedio (avg)
Accuracy	0.8851
Recall	0.8484
Precision	0.8934
F1-Score	0.8644
Specificity	0.9161
Error_rate	0.1149

Cuadro 4: Métricas promedio obtenidas en los folds de validación cruzada.

8.2.2. Análisis preliminar de los resultados Experimento 2

Los resultados obtenidos en el Experimento 2, que utilizó una configuración con 2 expertos seleccionados, muestran un desempeño general sólido. El *Accuracy* promedio de 88.51 % ± 4.37 % indica una mayor consistencia y rendimiento en comparación con el Experimento 1, que presentó un *Accuracy* promedio de 82.84 % ± 14.37 %. En este caso, la desviación estándar de 4.37 % es significativamente menor, lo que sugiere que el modelo seleccionando 2 expertos genera resultados más estables a través de los pliegues (folds).

Aunque el desempeño es positivo, se observa una variabilidad en los resultados entre los pliegues. El rango de *Accuracy* varía entre 81.69 % y 94.37 %, lo que refleja que, aunque el modelo tiene un rendimiento general sólido, existen algunas fluctuaciones en el rendimiento de los folds. La pérdida promedio de 0.3525 también es relativamente baja, lo que sugiere que el modelo está aprendiendo de manera eficiente.

Además del *Accuracy*, las métricas adicionales obtenidas, como el *Recall* (0.8484), *Precision* (0.8934) y *F1-Score* (0.8644), indican que el modelo tiene una capacidad destacada para identificar correctamente las instancias positivas, equilibrando de manera efectiva la capacidad de detección con la precisión en las predicciones. La *Specificity* también es alta (0.9161), lo que sugiere que el modelo presenta una baja tasa de falsos positivos, asegurando una clasificación confiable de los casos con ojos normales.

8.2.3. Conclusión preliminar Experimento 2

Los resultados obtenidos en el Experimento 2 muestran un desempeño mejorado en comparación con el Experimento 1, con un *Accuracy* promedio de $88.51\,\% \pm 4.37\,\%$, lo que refleja una mayor estabilidad en el rendimiento del modelo. Las métricas adicionales, como el *Recall* (0.8484), *Precision* (0.8934) y *F1-Score* (0.8644), sugieren que el modelo tiene una capacidad robusta para identificar correctamente tanto las clases positivas como las negativas, con un buen equilibrio entre la cobertura y la precisión. A pesar de estos resultados positivos, la desviación estándar relativamente baja ($4.37\,\%$) sugiere que el modelo presenta una variabilidad controlada en su desempeño, lo que es indicativo de una mayor estabilidad respecto a las particiones de datos en la validación cruzada. Este análisis se ampliará y validará con los resultados de los próximos experimentos, en los que se evaluarán configuraciones con 3 expertos y arquitecturas individuales adicionales para determinar si estas opciones ofrecen mejoras adicionales.

8.3. Resultados del Experimento 3

En el Experimento 3, se evaluará el desempeño del modelo utilizando una configuración basada en la selección de los 3 mejores expertos. A diferencia de los Experimentos 1 y 2, donde se emplearon 1 experto y los mejores 2 expertos respectivamente, esta nueva configuración busca aprovechar la combinación de 3 expertos para lograr una toma de decisiones aún más robusta y diversificada.

A través de la validación cruzada, se obtuvieron los resultados por pliegue (fold), que reflejan el comportamiento del modelo al incorporar esta selección ampliada de expertos. A continuación, se presentan las métricas obtenidas para cada pliegue (fold), junto con

sus matrices de confusión. Este análisis preliminar permitirá evaluar si la incorporación de un tercer experto mejora significativamente el desempeño y la estabilidad del modelo en comparación con las configuraciones anteriores.

8.3.1. Resultados por fold:

- **Fold 1:**

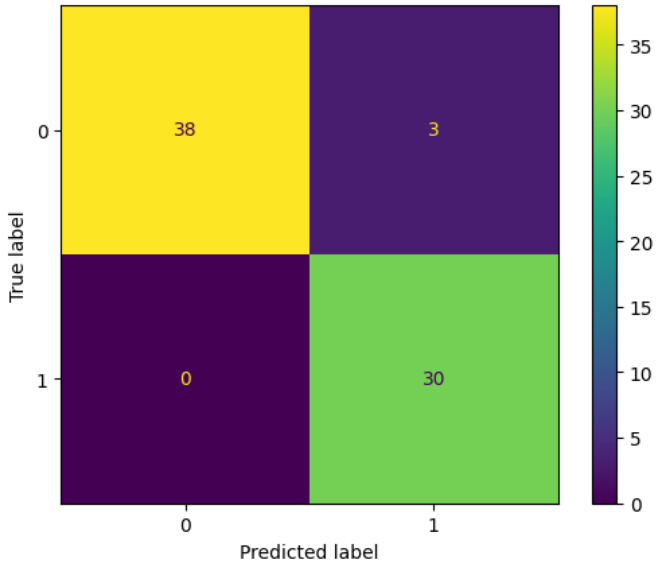

Figura 31: Matriz de confusión fold 1 experimento 3

- Metrics fold 1:
  - **Accuracy:** 0.9577
  - **Recall:** 1.0000
  - **Precision:** 0.9091
  - **F1-Score:** 0.9524
  - **Specificity:** 0.9268
  - **Error_rate:** 0.0422

- **Fold 2:**

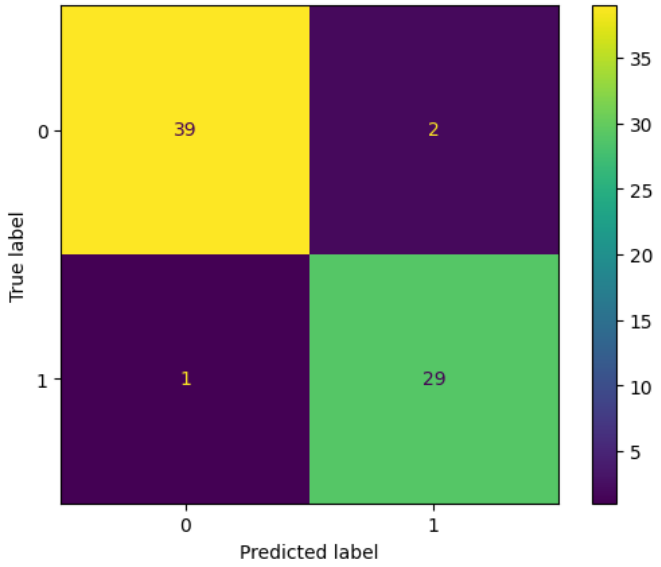

Figura 32: Matriz de confusión fold 2 experimento 3

- Metrics fold 2:
  - **Accuracy:** 0.9577
  - **Recall:** 0.9667
  - **Precision:** 0.9355
  - **F1-Score:** 0.9508
  - **Specificity:** 0.9512
  - **Error_rate:** 0.0422

- **Fold 3:**

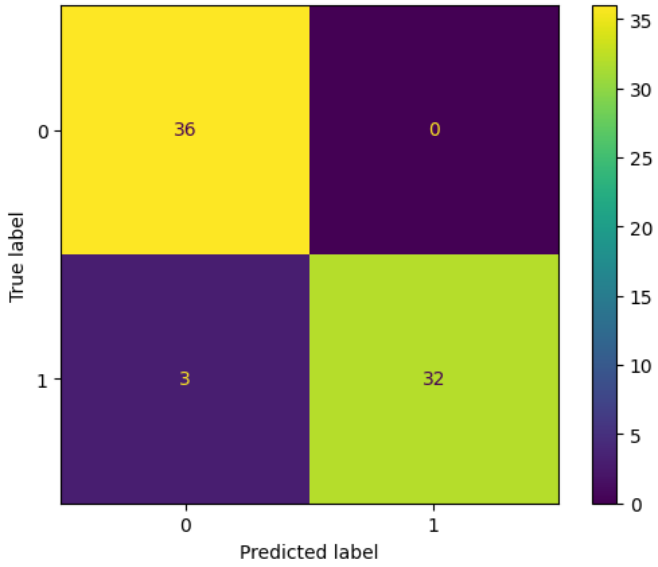

Figura 33: Matriz de confusión fold 3 experimento 3

- Metrics fold 3:
  - **Accuracy:** 0.9577
  - **Recall:** 0.9143
  - **Precision:** 1.0000
  - **F1-Score:** 0.9552
  - **Specificity:** 1.0000
  - **Error_rate:** 0.0422

- **Fold 4:**

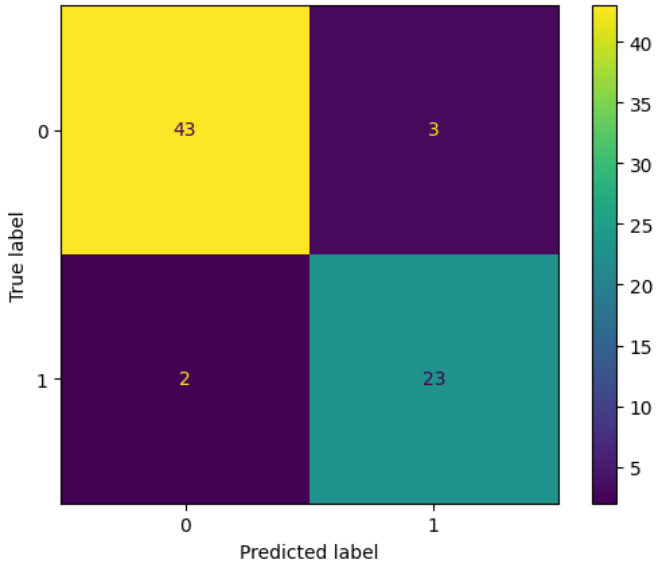

Figura 34: Matriz de confusión fold 4 experimento 1

- Metrics fold 4:
  - **Accuracy:** 0.9296
  - **Recall:** 0.9200
  - **Precision:** 0.8846
  - **F1-Score:** 0.9020
  - **Specificity:** 0.9348
  - **Error_rate:** 0.0704

- **Fold 5:**

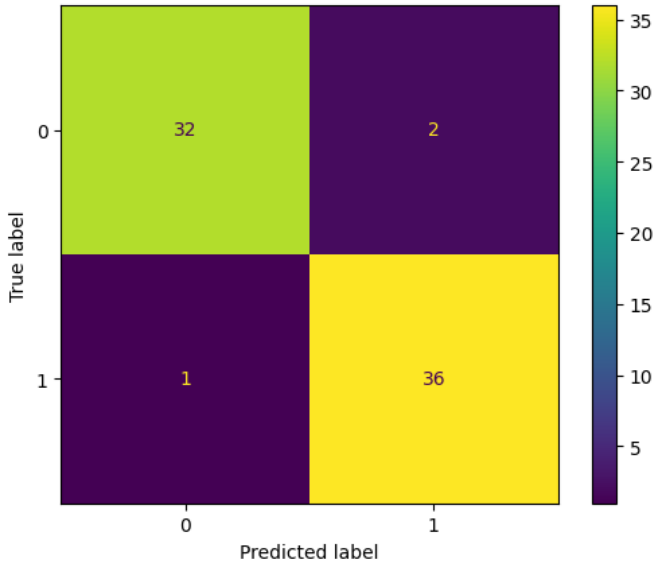

Figura 35: Matriz de confusión fold 5 experimento 3

- Metrics fold 5:
  - **Accuracy:** 0.9577
  - **Recall:** 0.9730
  - **Precision:** 0.9474
  - **F1-Score:** 0.9600
  - **Specificity:** 0.9412
  - **Error_rate:** 0.0422

- **Fold 6:**

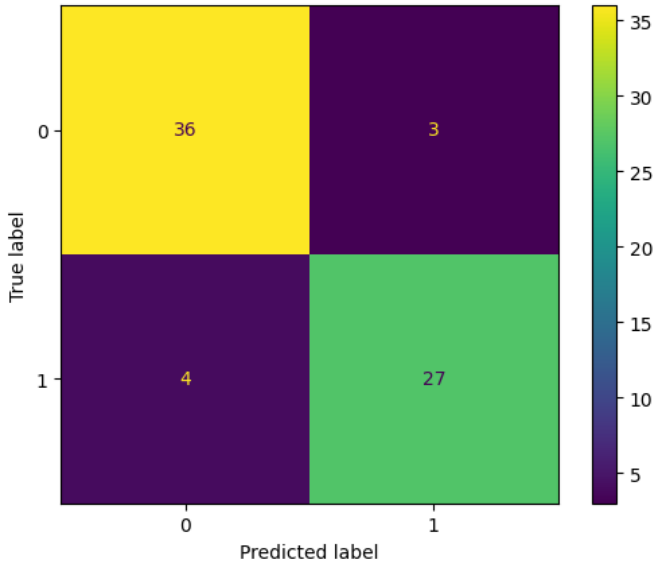

Figura 36: Matriz de confusión fold 6 experimento 3

- Metrics fold 6:
  - **Accuracy:** 0.9000
  - **Recall:** 0.8710
  - **Precision:** 0.9000
  - **F1-Score:** 0.8852
  - **Specificity:** 0.9231
  - **Error_rate:** 0.1000

- **Fold 7:**

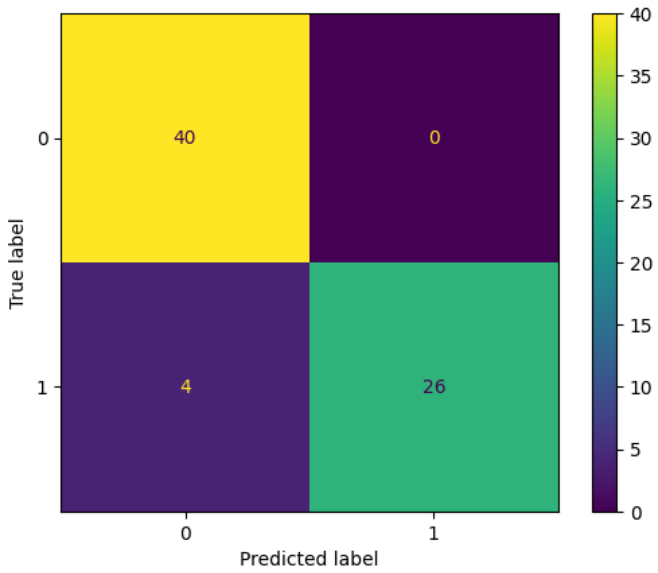

Figura 37: Matriz de confusión fold 7 experimento 3

- Metrics fold 7:
  - **Accuracy:** 0.9429
  - **Recall:** 0.8667
  - **Precision:** 1.0000
  - **F1-Score:** 0.9286
  - **Specificity:** 1.0000
  - **Error_rate:** 0.0571

- **Fold 8:**

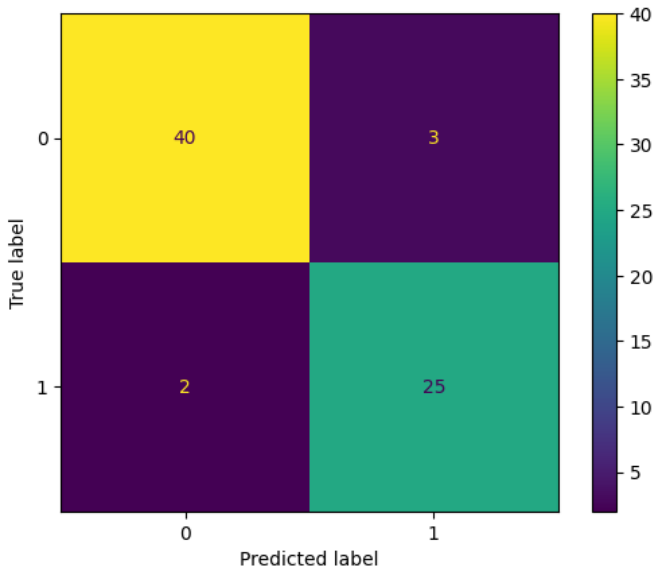

Figura 38: Matriz de confusión fold 8 experimento 3

- Metrics fold 8:
  - **Accuracy:** 0.9286
  - **Recall:** 0.9259
  - **Precision:** 0.8929
  - **F1-Score:** 0.9091
  - **Specificity:** 0.9302
  - **Error_rate:** 0.0714

- **Fold 9:**

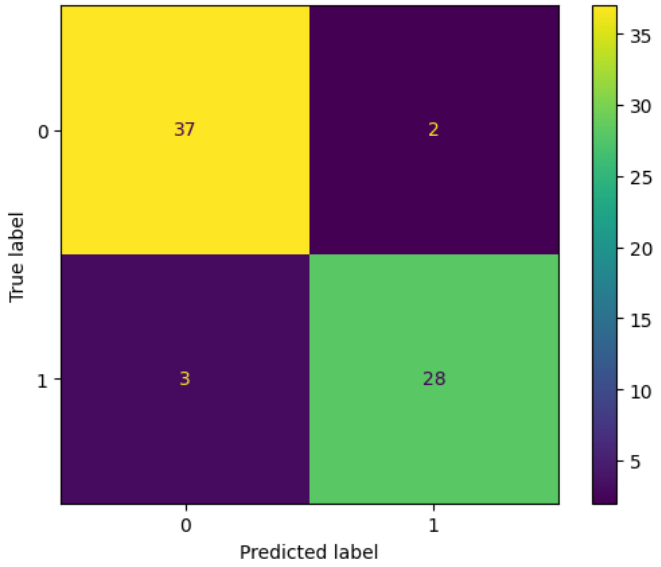

Figura 39: Matriz de confusión fold 9 experimento 3

- Metrics fold 9:
  - **Accuracy:** 0.9286
  - **Recall:** 0.9032
  - **Precision:** 0.9333
  - **F1-Score:** 0.9180
  - **Specificity:** 0.9487
  - **Error_rate:** 0.0714

- **Fold 10:**

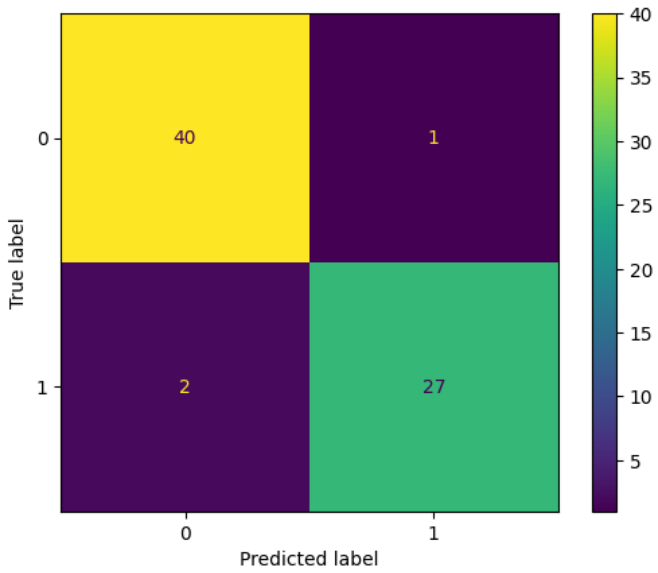

Figura 40: Matriz de confusión fold 10 experimento 3

- Metrics fold 10:
  - **Accuracy:** 0.9571
  - **Recall:** 0.9310
  - **Precision:** 0.9643
  - **F1-Score:** 0.9474
  - **Specificity:** 0.9756
  - **Error_rate:** 0.0428

fold	Pérdida (val_loss)	Accuracy (val_accuracy)
1	0.9744	95.77 %
2	0.3000	95.77 %
3	0.2398	95.77 %
4	0.1149	92.96 %
5	0.1602	97.77 %
6	0.8221	90.00 %
7	0.2946	94.29 %
8	0.8641	92.86 %
9	0.4031	92.86 %
10	0.5039	95.71 %
avg	**0.4677**	**94.18 % ± 1.87 %**

Cuadro 5: Resultados por fold y promedio (avg): pérdida (*val_loss*) y Accuracy (*val_accuracy*).

Métrica	Promedio (avg)
Accuracy	0.9418
Recall	0.9272
Precision	0.9367
F1-Score	0.9309
Specificity	0.9532
Error_rate	0.0582

Cuadro 6: Métricas promedio obtenidas en los folds de validación cruzada.

8.3.2. Análisis preliminar de los resultados Experimento 3

En el Experimento 3, donde se seleccionaron los 3 mejores expertos para la combinación de mapas de características, se observó un notable incremento en el desempeño global del modelo en comparación con los experimentos previos. Según el Cuadro 5, el promedio de *Accuracy* alcanzado fue de 94.18 % ± 1.87 %, mostrando no solo un desempeño superior, sino también una mejora significativa en la estabilidad del modelo, evidenciada por la baja desviación estándar (1.87 %). Esto sugiere que la incorporación de un tercer experto contribuyó a reducir las variaciones en los resultados entre los diferentes pliegues.

En cuanto a las métricas adicionales en el Cuadro 6, el modelo obtuvo un *Recall* de 0.9272, un *Precision* de 0.9367 y un *F1-Score* de 0.9309, lo que refleja un balance muy sólido entre la capacidad de identificar correctamente las clases positivas y la minimización

de falsos positivos. Además, la *Specificity* alcanzada 0.9532 indica que el modelo tiene un excelente desempeño al distinguir correctamente las clases negativas, mientras que el *Error_rate* 5.82 % refuerza esta evaluación positiva.

8.3.3. Conclusión preliminar Experimento 3

Los resultados obtenidos en este experimento indican que la configuración con los 3 mejores expertos es, hasta ahora, la más efectiva y estable. El desempeño promedio, combinado con una desviación estándar baja, sugiere que esta configuración es menos sensible a las particiones de los datos en la validación cruzada y ofrece un balance notable entre *Recall* y *Specificity*. En comparación con los experimentos previos, se observa una mejora significativa en la *Precision* general del modelo, destacando el impacto positivo de incorporar un tercer experto en la combinación.

8.4. Resultados del Experimento 4

En el Experimento 4, se evaluará el desempeño de los expertos de manera individual. El objetivo principal de este experimento es comparar directamente el rendimiento de los expertos entrenados individualmente con los modelos basados en combinaciones de expertos (*Mixture of Experts*, MoE) desarrollados en los experimentos anteriores.

Esta evaluación permitirá determinar si las configuraciones de los modelos `MoE` ofrecen una ventaja significativa frente a los expertos entrenados de manera individual. Al analizar las métricas obtenidas por pliege (fold) y las matrices de confusión, se podrá evaluar de forma detallada cómo se comporta cada experto de manera independiente y si su desempeño individual justifica o no la necesidad de estrategias de combinación. A continuación, se presentan los resultados de este análisis.

8.4.1. Resultados por fold VGG19:

- **Fold 1:**

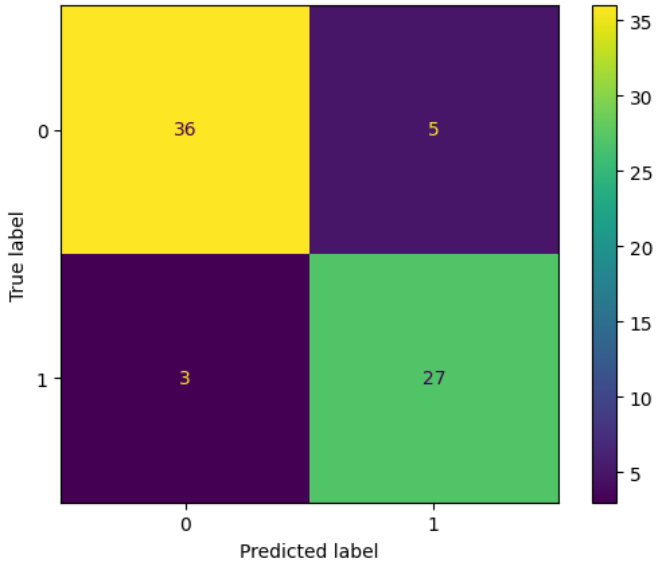

Figura 41: Matriz de confusión fold 1 experimento 4 VGG19

- Metrics fold 1:
  - **Accuracy:** 0.8873
  - **Recall:** 0.9000
  - **Precision:** 0.8437
  - **F1-Score:** 0.8710
  - **Specificity:** 0.8780
  - **Error rate:** 0.1127

- **Fold 2:**

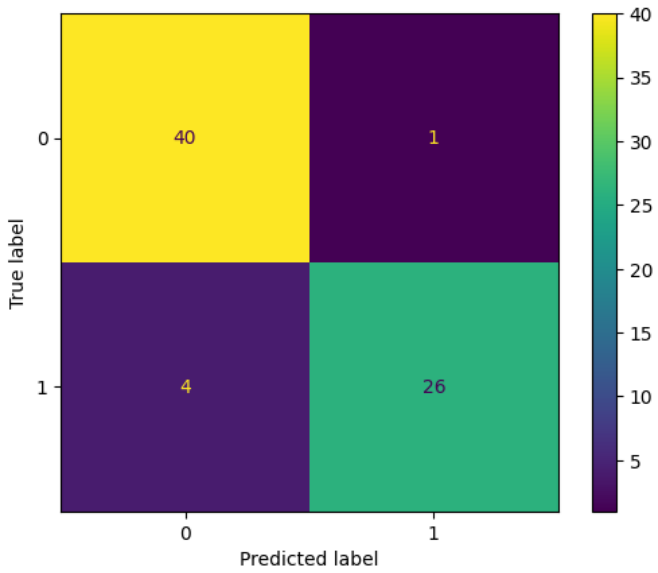

Figura 42: Matriz de confusión fold 2 experimento 4 VGG19

- Metrics fold 2:
  - **Accuracy:** 0.9296
  - **Recall:** 0.8667
  - **Precision:** 0.9630
  - **F1-Score:** 0.9123
  - **Specificity:** 0.9756
  - **Error rate:** 0.0704

- **Fold 3:**

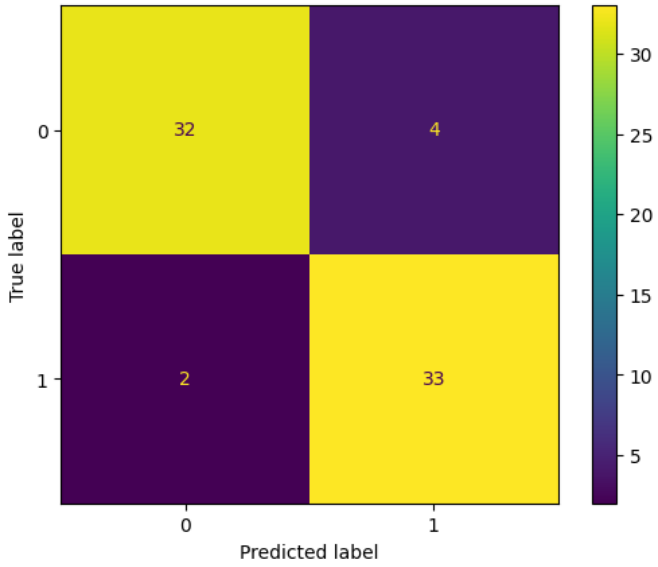

Figura 43: Matriz de confusión fold 3 experimento 4 VGG19

- Metrics fold 3:
  - **Accuracy:** 0.9155
  - **Recall:** 0.9428
  - **Precision:** 0.8919
  - **F1-Score:** 0.9166
  - **Specificity:** 0.8889
  - **Error rate:** 0.0845

- **Fold 4:**

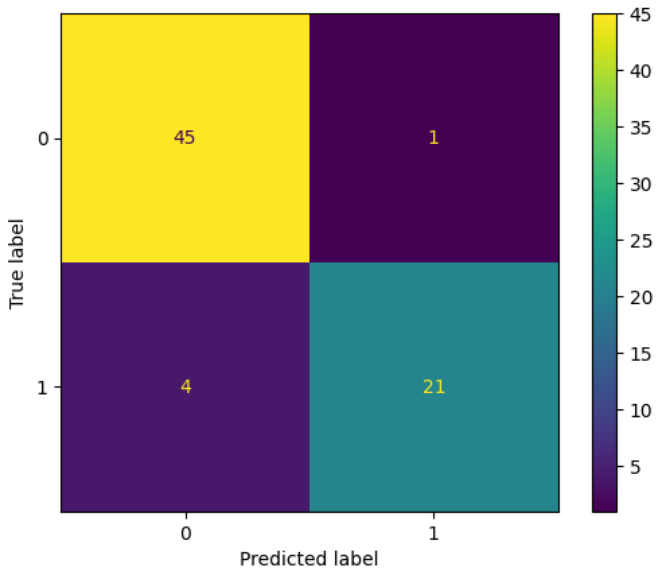

Figura 44: Matriz de confusión fold 4 experimento 4 VGG19

- Metrics fold 4:
  - **Accuracy:** 0.9296
  - **Recall:** 0.8400
  - **Precision:** 0.9545
  - **F1-Score:** 0.8936
  - **Specificity:** 0.9783
  - **Error rate:** 0.0704

- **Fold 5:**

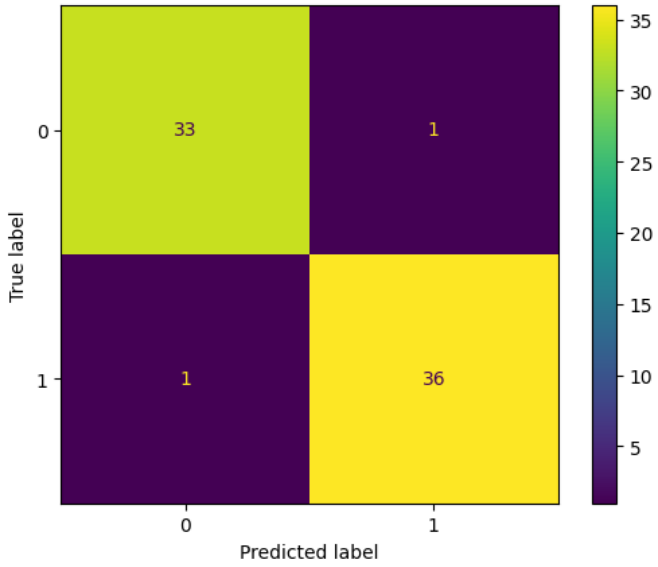

Figura 45: Matriz de confusión fold 5 experimento 4 VGG19

- Metrics fold 5:
  - **Accuracy:** 0.9718
  - **Recall:** 0.9730
  - **Precision:** 0.9730
  - **F1-Score:** 0.9730
  - **Specificity:** 0.9706
  - **Error rate:** 0.0282

- **Fold 6:**

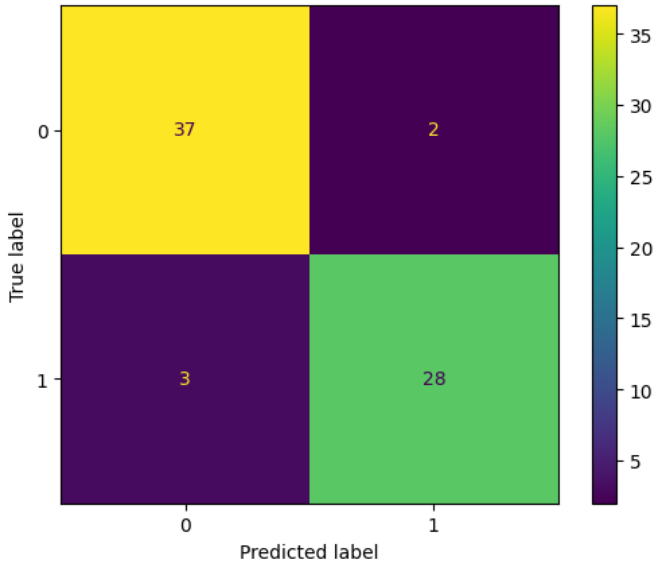

Figura 46: Matriz de confusión fold 6 experimento 4 VGG19

- Metrics fold 6:
  - **Accuracy:** 0.9286
  - **Recall:** 0.9032
  - **Precision:** 0.9333
  - **F1-Score:** 0.9180
  - **Specificity:** 0.9487
  - **Error rate:** 0.0714

- **Fold 7:**

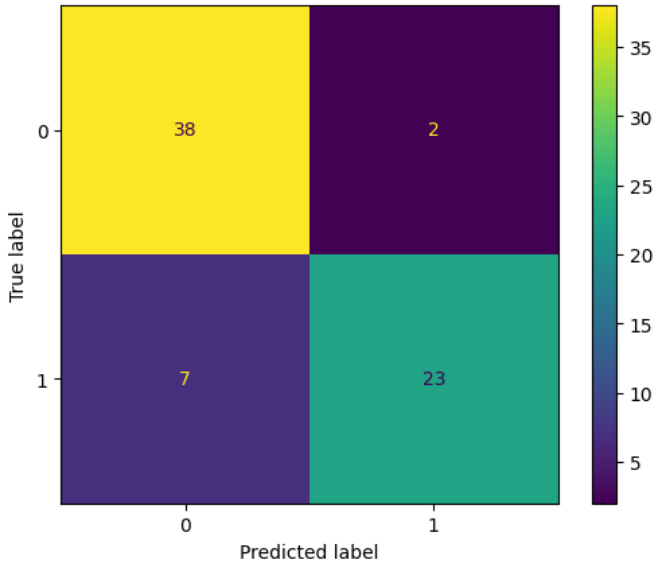

Figura 47: Matriz de confusión fold 7 experimento 4 VGG19

- Metrics fold 7:
  - **Accuracy:** 0.8714
  - **Recall:** 0.7667
  - **Precision:** 0.9200
  - **F1-Score:** 0.8364
  - **Specificity:** 0.9500
  - **Error rate:** 0.1286

- **Fold 8:**

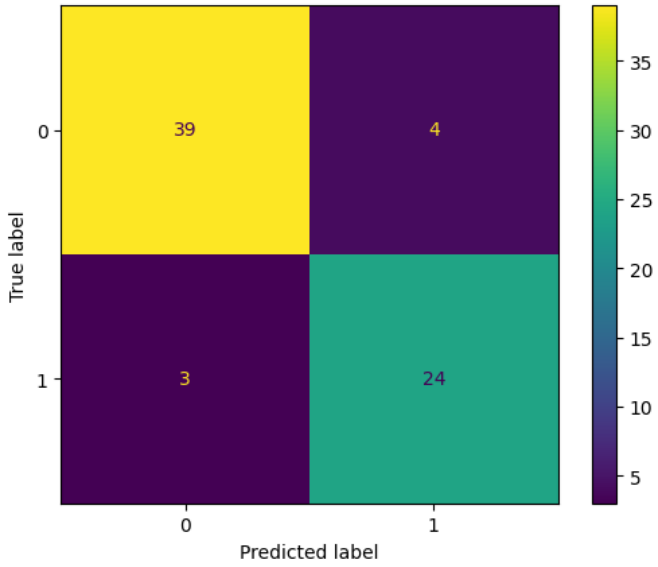

Figura 48: Matriz de confusión fold 8 experimento 4 VGG19

- Metrics fold 8:
  - **Accuracy:** 0.9000
  - **Recall:** 0.8889
  - **Precision:** 0.8571
  - **F1-Score:** 0.8727
  - **Specificity:** 0.9070
  - **Error rate:** 0.0999

- **Fold 9:**

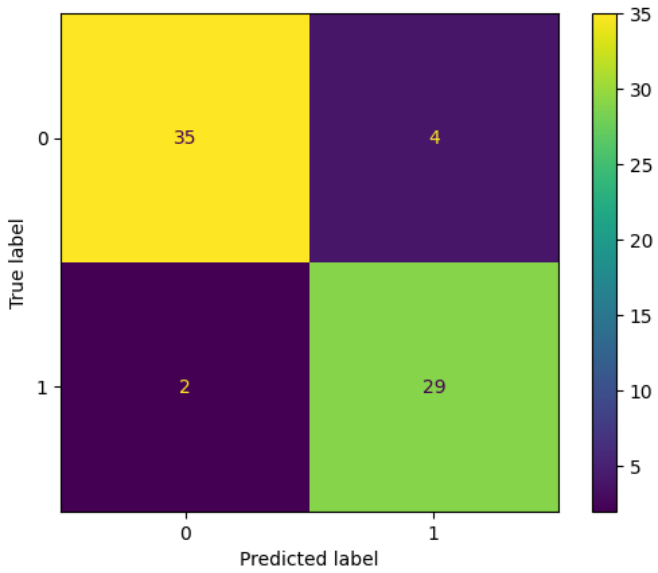

Figura 49: Matriz de confusión fold 9 experimento 4 VGG19

- Metrics fold 9:
  - **Accuracy:** 0.9143
  - **Recall:** 0.9355
  - **Precision:** 0.8787
  - **F1-Score:** 0.9062
  - **Specificity:** 0.8974
  - **Error rate:** 0.0857

- **Fold 10:**

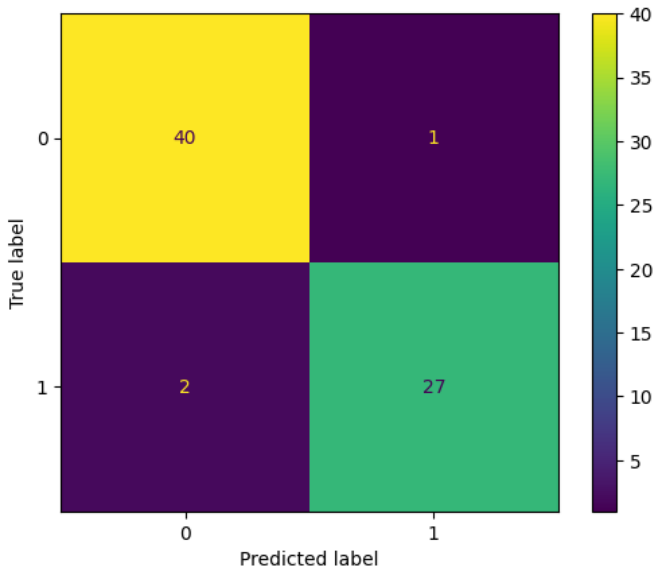

Figura 50: Matriz de confusión fold 10 experimento 4 VGG19

- Metrics fold 10:
  - **Accuracy:** 0.9571
  - **Recall:** 0.9310
  - **Precision:** 0.9643
  - **F1-Score:** 0.9474
  - **Specificity:** 0.9756
  - **Error rate:** 0.0428

fold	Pérdida (val_loss)	Accuracy (val_accuracy)
1	0.3485	88.73 %
2	0.3616	92.96 %
3	0.3864	91.55 %
4	0.2716	92.96 %
5	0.3261	97.18 %
6	0.3798	92.86 %
7	0.4064	87.14 %
8	0.2752	90.00 %
9	0.1609	91.43 %
10	0.1568	95.71 %
avg	**0.3073**	**92.05 % ± 2.86 %**

Cuadro 7: Resultados por fold y promedio (avg): pérdida (*val_loss*) y Accuracy (*val_accuracy*).

Métrica	Promedio (avg)
Accuracy	0.9205
Recall	0.8948
Precision	0.9180
F1-score	0.9047
Specificity	0.9370
Error Rate	0.0795

Cuadro 8: Métricas promedio obtenidas en los folds de validación cruzada.

8.4.2. Resultados por fold ResNet50:

- **Fold 1:**

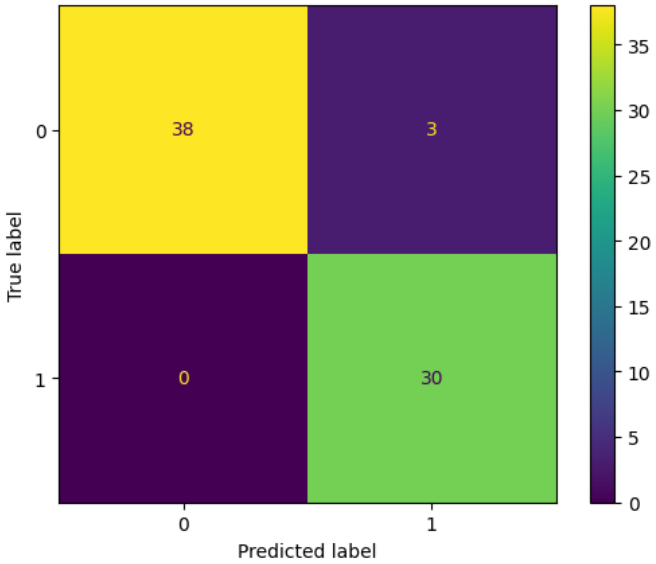

Figura 51: Matriz de confusión fold 1 experimento 4 ResNet50

- Metrics fold 1:
  - **Accuracy:** 0.9577
  - **Recall:** 1.0000
  - **Precision:** 0.9091
  - **F1-Score:** 0.9524
  - **Specificity:** 0.9268
  - **Error rate:** 0.0422

- **Fold 2:**

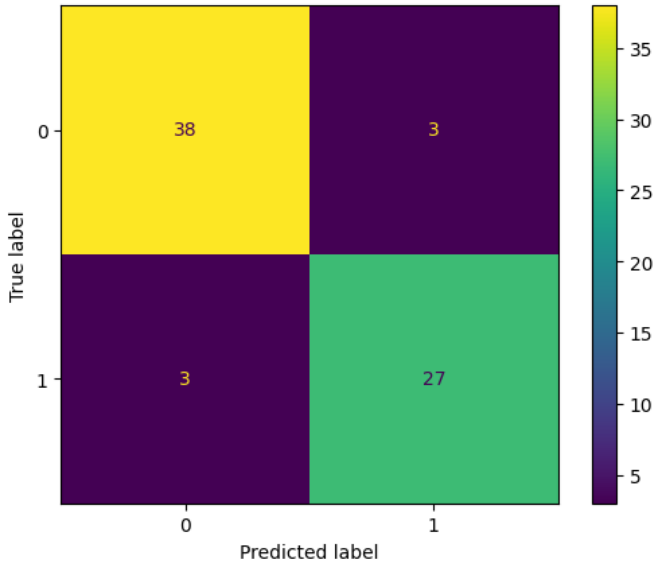

Figura 52: Matriz de confusión fold 2 experimento 4 ResNet50

- Metrics fold 2:
  - **Accuracy:** 0.9155
  - **Recall:** 0.9000
  - **Precision:** 0.9000
  - **F1-Score:** 0.9000
  - **Specificity:** 0.9268
  - **Error rate:** 0.0845

- **Fold 3:**

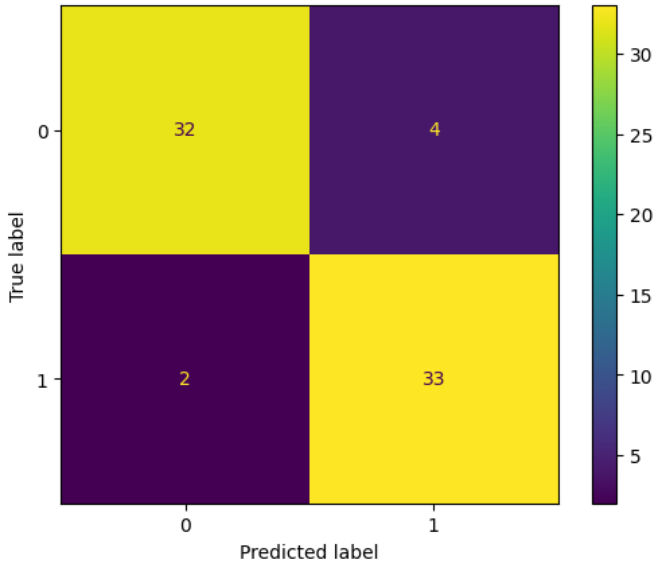

Figura 53: Matriz de confusión fold 3 experimento 4 ResNet50

- Metrics fold 3:
  - **Accuracy:** 0.9155
  - **Recall:** 0.9429
  - **Precision:** 0.8919
  - **F1-Score:** 0.9167
  - **Specificity:** 0.8889
  - **Error rate:** 0.0845

- **Fold 4:**

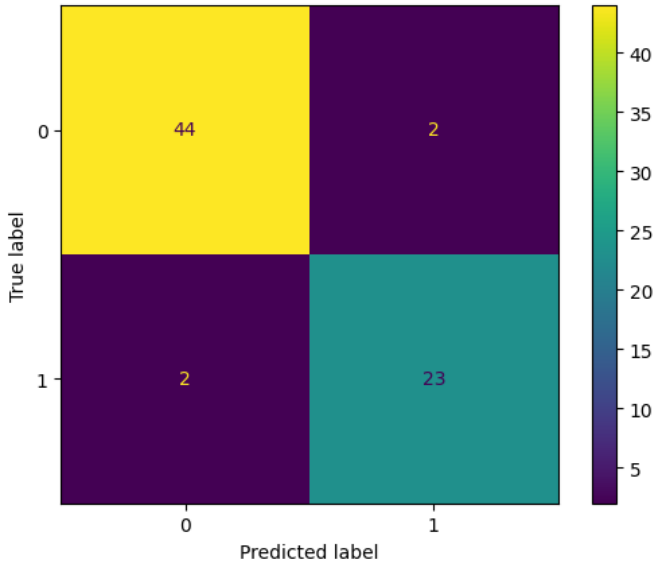

Figura 54: Matriz de confusión fold 4 experimento 4 ResNet50

- Metrics fold 4:
  - **Accuracy:** 0.9437
  - **Recall:** 0.9200
  - **Precision:** 0.9200
  - **F1-Score:** 0.9200
  - **Specificity:** 0.9565
  - **Error rate:** 0.0563

- **Fold 5:**

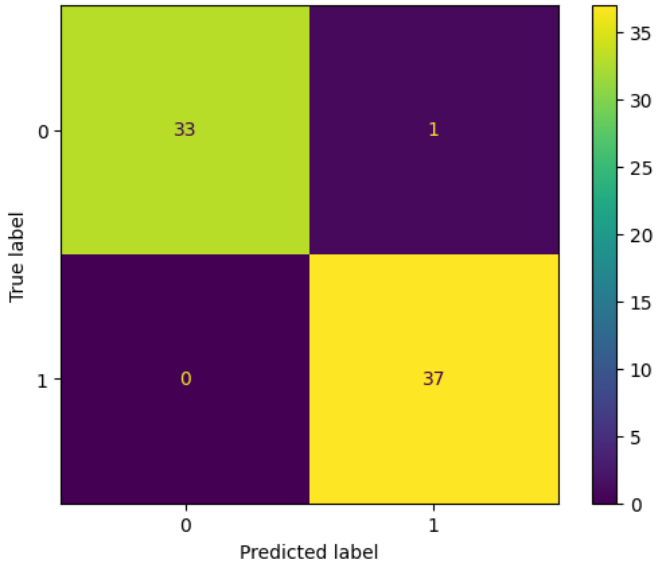

Figura 55: Matriz de confusión fold 5 experimento 4 ResNet50

- Metrics fold 5:
  - **Accuracy:** 0.9859
  - **Recall:** 1.0000
  - **Precision:** 0.9737
  - **F1-Score:** 0.9867
  - **Specificity:** 0.9706
  - **Error rate:** 0.0141

- **Fold 6:**

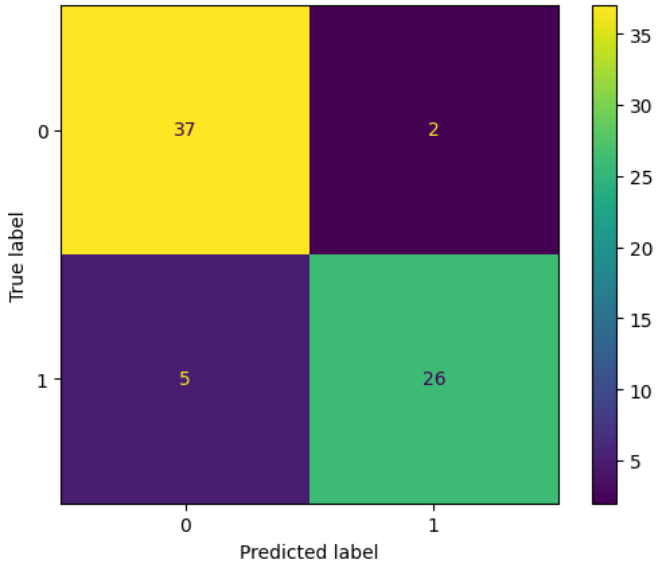

Figura 56: Matriz de confusión fold 6 experimento 4 ResNet50

- Metrics fold 6:
  - **Accuracy:** 0.9000
  - **Recall:** 0.8387
  - **Precision:** 0.9286
  - **F1-Score:** 0.8814
  - **Specificity:** 0.9487
  - **Error rate:** 0.1000

- **Fold 7:**

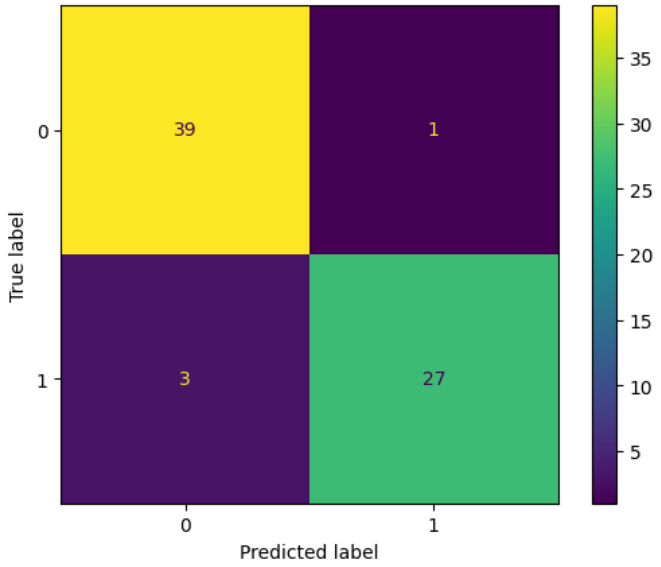

Figura 57: Matriz de confusión fold 7 experimento 4 ResNet50

- Metrics fold 7:
  - **Accuracy:** 0.9428
  - **Recall:** 0.9000
  - **Precision:** 0.9643
  - **F1-Score:** 0.9310
  - **Specificity:** 0.9750
  - **Error rate:** 0.0571

- **Fold 8:**

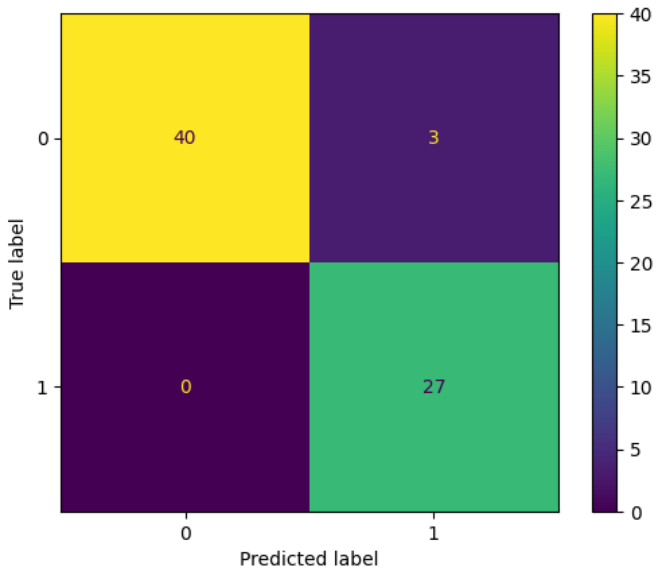

Figura 58: Matriz de confusión fold 8 experimento 4 ResNet50

- Metrics fold 8:
  - **Accuracy:** 0.9571
  - **Recall:** 1.0000
  - **Precision:** 0.9000
  - **F1-Score:** 0.9474
  - **Specificity:** 0.9302
  - **Error rate:** 0.0428

- **Fold 9:**

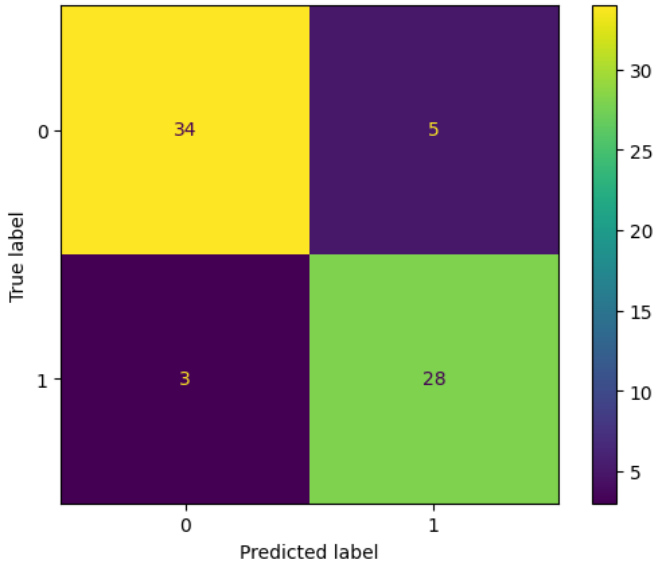

Figura 59: Matriz de confusión fold 9 experimento 4 ResNet50

- Metrics fold 9:
  - **Accuracy:** 0.8857
  - **Recall:** 0.9032
  - **Precision:** 0.8485
  - **F1-Score:** 0.8750
  - **Specificity:** 0.8718
  - **Error rate:** 0.1143

- **Fold 10:**

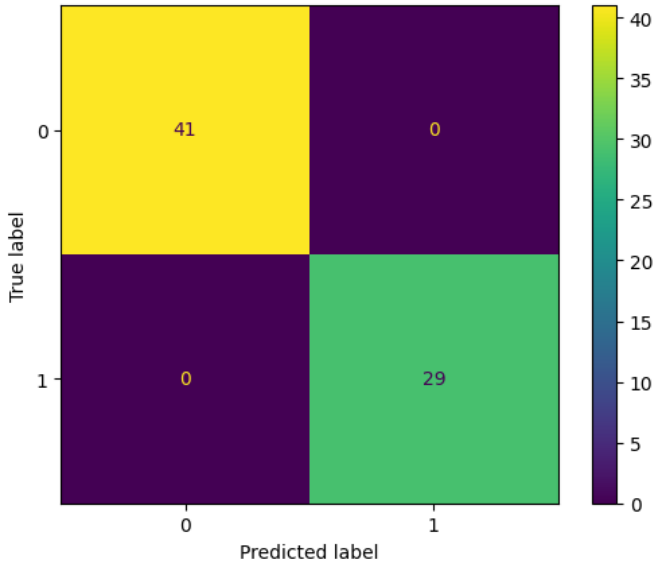

Figura 60: Matriz de confusión fold 10 experimento 4 ResNet50

- Metrics fold 10:
  - **Accuracy:** 1.0000
  - **Recall:** 1.0000
  - **Precision:** 1.0000
  - **F1-Score:** 1.0000
  - **Specificity:** 1.0000
  - **Error rate:** 0.0000

fold	**Pérdida (val_loss)**	**Accuracy (val_accuracy)**
1	0.3246	95.77 %
2	1.3533	91.55 %
3	0.2978	91.55 %
4	0.6454	94.37 %
5	0.0487	98.59 %
6	0.8634	90.00 %
7	0.3641	94.29 %
8	0.1755	95.71 %
9	0.9844	88.57 %
10	0.0148	100.00 %
avg	**0.5072**	**94.04 % ± 3.48 %**

Cuadro 9: Resultados por fold y promedio (avg): pérdida (*val_loss*) y Accuracy (*val_accuracy*).

Métrica	**Promedio (avg)**
Accuracy	0.9404
Recall	0.9405
Precision	0.9236
F1-score	0.9310
Specificity	0.9395
Error Rate	0.0596

Cuadro 10: Métricas promedio obtenidas en los folds de validación cruzada.

8.4.3. Resultados por fold DenseNet121:

- **Fold 1:**

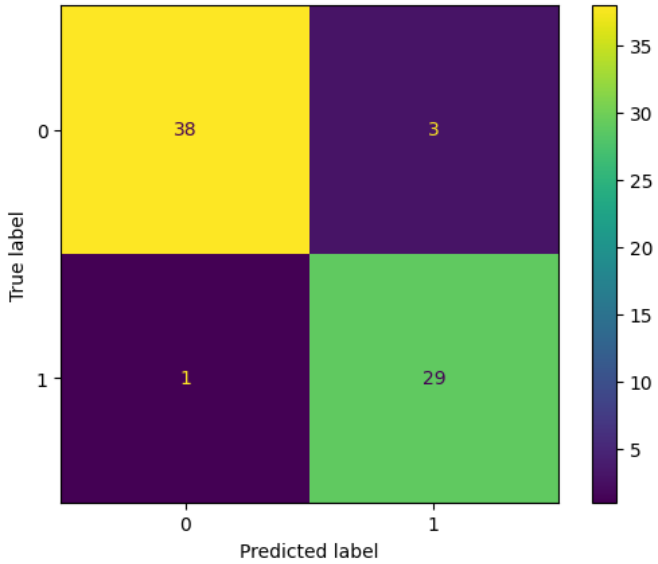

Figura 61: Matriz de confusión fold 1 experimento 4 DenseNet121

- Metrics fold 1:
  - **Accuracy:** 0.9437
  - **Recall:** 0.9667
  - **Precision:** 0.9063
  - **F1-Score:** 0.9355
  - **Specificity:** 0.9268
  - **Error rate:** 0.0563

- **Fold 2:**

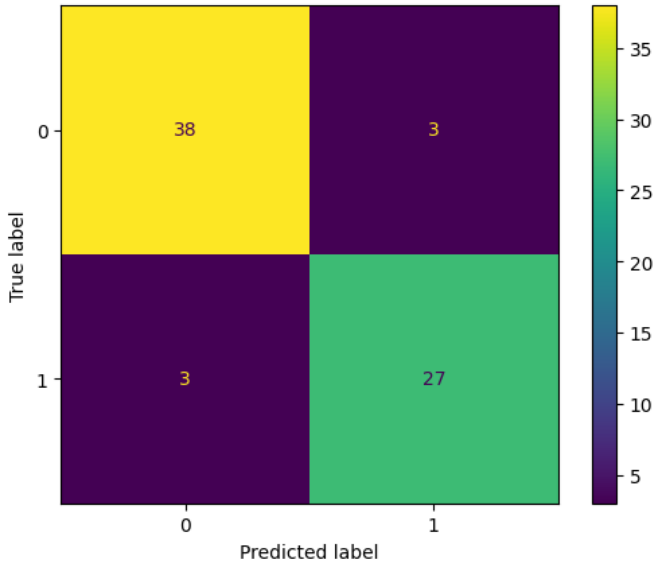

Figura 62: Matriz de confusión fold 2 experimento 4 DenseNet121

- Metrics fold 2:
  - **Accuracy:** 0.9155
  - **Recall:** 0.9000
  - **Precision:** 0.9000
  - **F1-Score:** 0.9000
  - **Specificity:** 0.9268
  - **Error rate:** 0.0845

- **Fold 3:**

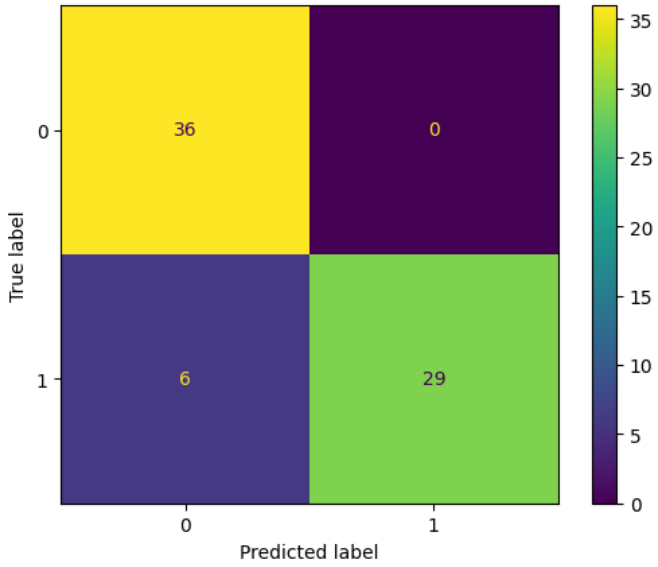

Figura 63: Matriz de confusión fold 3 experimento 4 DenseNet121

- Metrics fold 3:
  - **Accuracy:** 0.9155
  - **Recall:** 0.8286
  - **Precision:** 1.0000
  - **F1-Score:** 0.9063
  - **Specificity:** 1.0000
  - **Error rate:** 0.0845

- **Fold 4:**

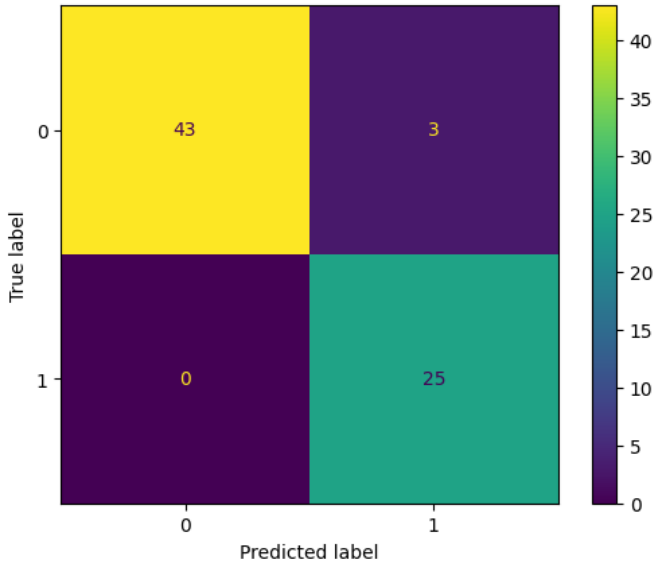

Figura 64: Matriz de confusión fold 4 experimento 4 DenseNet121

- Metrics fold 4:
  - **Accuracy:** 0.9577
  - **Recall:** 1.0000
  - **Precision:** 0.8928
  - **F1-Score:** 0.9434
  - **Specificity:** 0.9348
  - **Error rate:** 0.0422

- **Fold 5:**

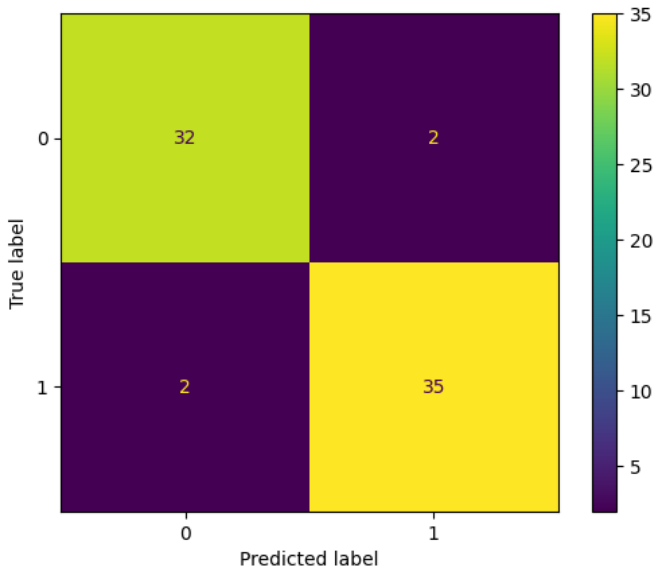

Figura 65: Matriz de confusión fold 5 experimento 4 DenseNet121

- Metrics fold 5:
  - **Accuracy:** 0.9437
  - **Recall:** 0.9459
  - **Precision:** 0.9459
  - **F1-Score:** 0.9459
  - **Specificity:** 0.9411
  - **Error rate:** 0.0563

- **Fold 6:**

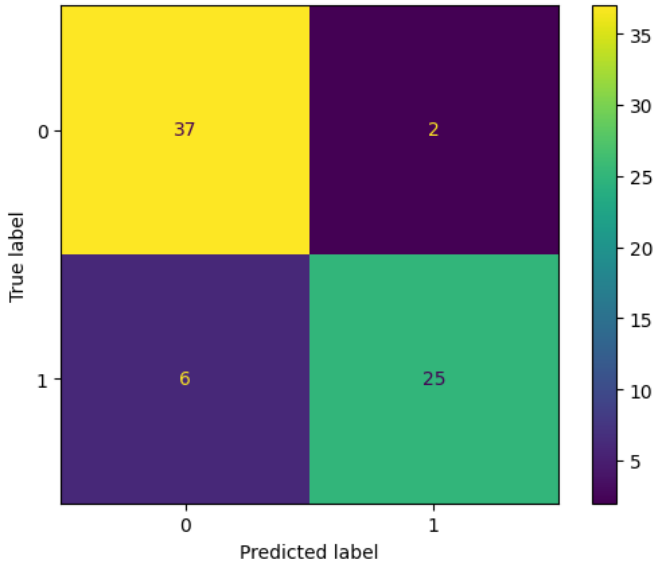

Figura 66: Matriz de confusión fold 6 experimento 4 DenseNet121

- Metrics fold 6:
  - **Accuracy:** 0.8857
  - **Recall:** 0.8065
  - **Precision:** 0.9259
  - **F1-Score:** 0.8621
  - **Specificity:** 0.9487
  - **Error rate:** 0.1143

- **Fold 7:**

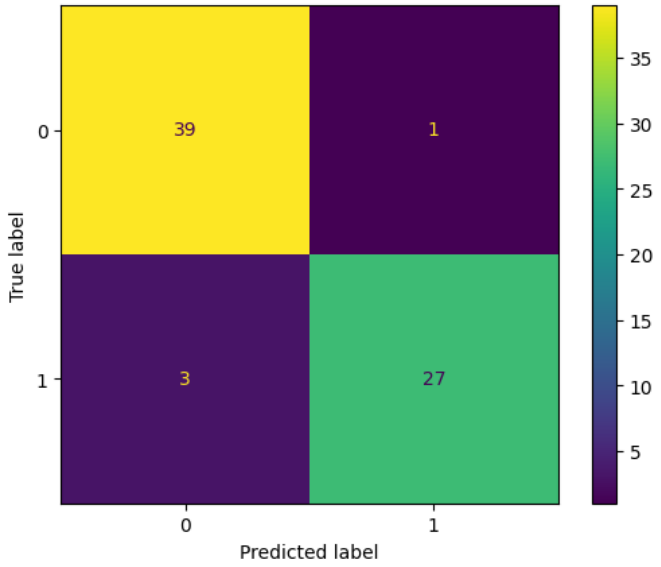

Figura 67: Matriz de confusión fold 7 experimento 4 DenseNet121

- Metrics fold 7:
  - **Accuracy:** 0.9429
  - **Recall:** 0.9000
  - **Precision:** 0.9643
  - **F1-Score:** 0.9310
  - **Specificity:** 0.9750
  - **Error rate:** 0.0571

- **Fold 8:**

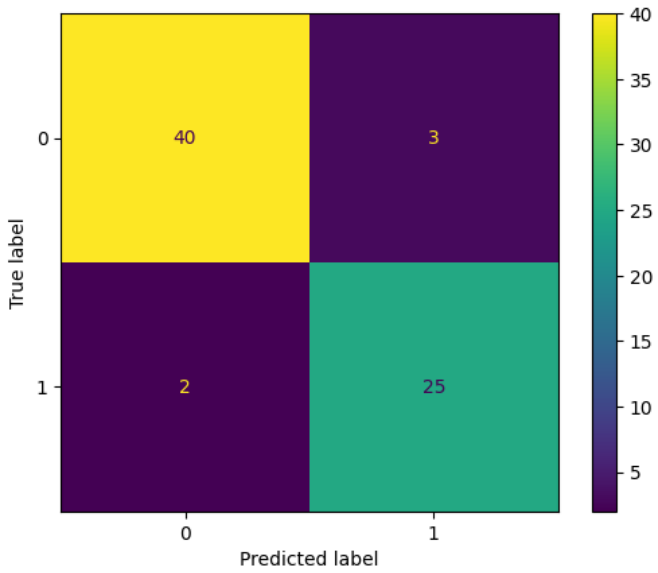

Figura 68: Matriz de confusión fold 8 experimento 4 DenseNet121

- Metrics fold 8:
  - **Accuracy:** 0.9286
  - **Recall:** 0.9259
  - **Precision:** 0.8928
  - **F1-Score:** 0.9091
  - **Specificity:** 0.9302
  - **Error rate:** 0.0714

- **Fold 9:**

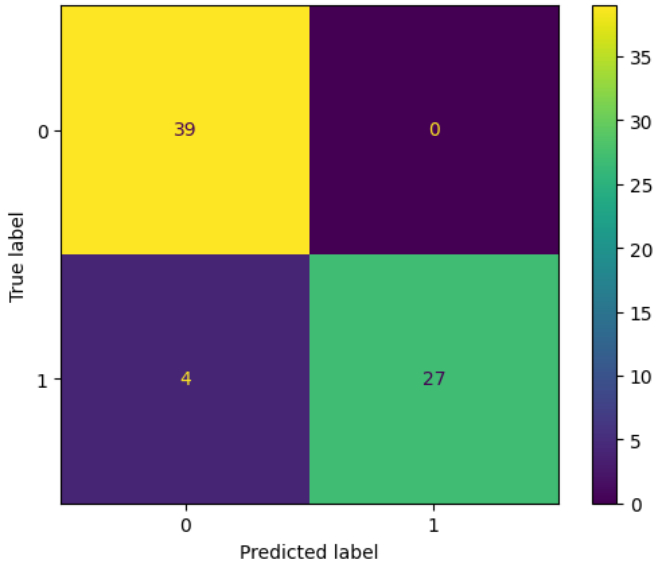

Figura 69: Matriz de confusión fold 9 experimento 4 DenseNet121

- Metrics fold 9:
  - **Accuracy:** 0.9428
  - **Recall:** 0.8710
  - **Precision:** 1.0000
  - **F1-Score:** 0.9310
  - **Specificity:** 1.0000
  - **Error rate:** 0.0571

- **Fold 10:**

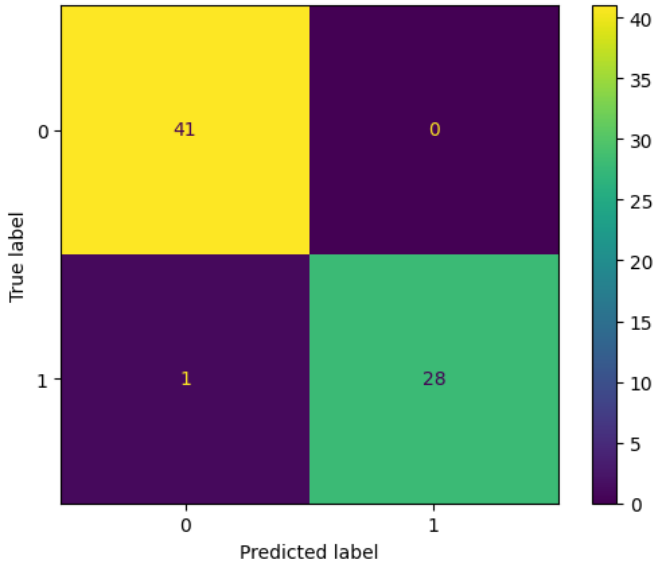

Figura 70: Matriz de confusión fold 10 experimento 4 DenseNet121

- Metrics fold 10:
  - **Accuracy:** 0.9857
  - **Recall:** 0.9655
  - **Precision:** 1.0000
  - **F1-Score:** 0.9824
  - **Specificity:** 1.0000
  - **Error rate:** 0.0143

fold	Pérdida (val_loss)	Accuracy (val_accuracy)
1	0.5860	94.37 %
2	0.4035	91.55 %
3	0.4979	91.55 %
4	0.0697	95.77 %
5	0.1388	94.37 %
6	0.5311	88.57 %
7	0.2536	94.29 %
8	0.3764	92.86 %
9	0.5000	94.29 %
10	0.0464	98.57 %
avg	**0.3403**	**93.62 % ± 2.56 %**

Cuadro 11: Resultados por fold y promedio (avg): pérdida (*val_loss*) y Accuracy (*val_accuracy*).

Métrica	Promedio (avg)
Accuracy	0.9362
Recall	0.9110
Precision	0.9428
F1-score	0.9247
Specificity	0.9584
Error Rate	0.0638

Cuadro 12: Métricas promedio obtenidas en los folds de validación cruzada.

9. Análisis y discusión de los datos

Una vez realizados los 4 experimentos se pudo obtener los siguientes resultados:

Modelos	accuracy	loss	Recall	Precision	F1-score	Specificity	Error_rate
MoE 1 experto	82,8410 % ± 14,3704 %	0.4898	0.6900	0.7902	0.7161	0.9367	0.1716
MoE 2 expertos	88,5090 % ± 4,3651 %	0.3525	0.8644	0.8934	0.8644	0.9161	0.1149
MoE 3 expertos	94,1770 % ± 1,8709 %	0.4677	0.9272	0.9367	0.9309	0.9532	0.0582
VGG19	95,0523 % ± 2,8636	0.3073	0.8948	0.9180	0.9047	0.9370	0.0795
ResNet-50	94,0402 % ± 3,4763	0.5072	0.9405	0.9236	0.9310	0.9395	0.0596
DenseNet121	93,6177 % ± 2,5652	0.3403	0.9110	0.9428	0.9247	0.9584	0.0638

Cuadro 13: Resumen de resultados obtenidos para diferentes modelos evaluados.

Tras la realización de los experimentos, los resultados obtenidos permitieron evaluar el desempeño de los modelos individuales y los modelos con diferentes configuraciones basados en `Mixture of Experts, (MoE)`. A continuación, se presenta una discusión de los hallazgos principales, acompañada del análisis de las métricas clave:

9.1. Desempeño de los modelos individuales

- **VGG19:** El modelo `VGG19` mostró un rendimiento sobresaliente en la detección de glaucoma, con un *Accuracy* promedio de 95.05 % y una variabilidad relativamente baja (± 2.86 %). Su valor *loss* de 0.3073 indica una muy buena capacidad de generalización. Las métricas específicas revelan fortalezas significativas:

  - El *Recall* de 0.8948 sugiere que el modelo identifica correctamente el 89.48 % de los casos positivos de glaucoma.
  - Un *Precision* de 0.9180 indica que cuando el modelo predice un caso positivo, acierta en un 91.80 % de las ocasiones.
  - El *F1-Score* de 0.9047 demuestra un equilibrio robusto entre *Precision* y *Recall*.
  - La *Specificity* de 0.9370 muestra una excelente capacidad para identificar correctamente los casos negativos.
  - Con un *Error_rate* bajo del 7.95 %, `VGG19` se posiciona como un modelo muy confiable para el diagnóstico de glaucoma con áreas de mejora.

- **ResNet50:** Presentó un rendimiento igualmente destacado, con un *Accuracy* promedio de 94.04 % y una variabilidad ligeramente mayor (± 3.48 %). Sus características más notables incluyen:

  - Un *Recall* excepcionalmente alto de 0.9405, indicando que detecta correctamente el 94.05 % de los casos positivos de glaucoma.
  - Un *Precision* de 0.9236, lo que significa que sus predicciones positivas son precisas en un 92.36 % de los casos.
  - *F1-Score* de 0.9310, reflejando un equilibrio casi perfecto entre *Precision* y *Recall*.
  - *Specificity* de 0.9395, demostrando una capacidad superior para identificar casos negativos.
  - Una baja tasa de error del 5.96 % que lo posiciona como un modelo muy preciso y confiable.

- **DenseNet121:** El modelo `DenseNet121` mostró un rendimiento sólido, con un *Accuracy* promedio de 93.62 % y la menor variabilidad entre los modelos (± 2.57 %). Sus características más destacadas son:

  - *Recall* de 0.9110, identificando correctamente el 91.10 % de los casos positivos de glaucoma.
  - *Precision* sobresaliente de 0.9428, indicando que sus predicciones positivas son extremadamente precisas.
  - *F1-Score* de 0.9247, que demuestra un equilibrio robusto entre *Precision* y *Recall*.
  - La *Specificity* más alta de 0.9584, lo que significa una capacidad extraordinaria para identificar correctamente los casos negativos.
  - Un *Error_rate* bajo del 6.38 % que lo convierte en un modelo muy confiable.

9.2. Desempeño de las configuraciones MoE

Como se observa en los resultados finales, las configuraciones MoE demostraron mejoras progresivas en el desempeño conforme se incrementó el número de expertos:

- **MoE seleccionado 1 experto:** Esta configuración obtuvo un *Accuracy* promedio de 82.84 % y un *loss* promedio de 0.4898. Aunque no superó a los mejores modelos individuales (`ResNet50, DenseNet121 y VGG19`), su desempeño es aceptable y representa un punto de partida para la integración de múltiples expertos.

- **MoE seleccionando 2 expertos:** Al seleccionar 2 expertos, el *Accuracy* promedio aumentó a 88.51 %, acompañado de una mejora significativa en *loss* promedio (0.3525) y en métricas como *Recall* (86.44 %) y *Precision* (89.34 %). Esto confirma que la combinación de expertos complementa sus fortalezas individuales, mejorando el rendimiento general.

- **MoE seleccionando 3 expertos:** Esta configuración alcanzó los mejores resultados, con un *Accuracy* promedio de 94.18 % y métricas como *Recall* (92.72 %), *Precision* (93.67 %), y *F1-score* (93.09 %). Estos valores de *Recall* y **Specificity** los cuales son claves para el area médica son superiores junto a las demás métricas.

9.3. Discución General

- **Efectividad de los Modelos Individuales:** `VGG19`, `ResNet50` y `DenseNet121` demostraron ser modelos confiables con alto desempeño. Esto sugiere que estos modelos son opciones viables para problemas similares.

- **Impacto de las Configuraciones de modelos `MoE`:** Los resultados de los modelos `MoE` confirman que las estrategias de combinación de expertos ofrecen ventajas claras en la mejora del desempeño general, aunque no de manera inmediata. Específicamente:

    - El modelo `MoE 1` mostró un *Accuracy* promedio de 82.84 %, con valores limitados en *Recall* (69.00 %) y *Precision* (79.02 %), lo que indica que, en su confi-

guración básica, el modelo no logra superar a los mejores modelos individuales como `DenseNet121`, `VGG19` y `ResNet50`.

- Al incrementar a 2 expertos, el modelo `MoE 2` logró un *Accuracy* promedio de 88.51 %, con una notable mejora en *Recall* (86.44 %) y *Precision* (89.34 %). Sin embargo, sigue por debajo del desempeño alcanzado por `DenseNet121`, `VGG19` y `ResNet50`, lo que sugiere que, aunque el impacto de la combinación de expertos es evidente, aún no es suficiente para competir con los mejores modelos individuales.
- La configuración del modelo `MoE 3`, por otro lado, logró un *Accuracy* de 94.18 %, posicionándose como un modelo competitivo frente a `DenseNet121` (93.62 %), `VGG19` (95.05 %) y `ResNet50` (94.04 %). Además, mostró métricas sólidas en *Recall* (92.72 %), *Precision* (93.67 %) y *F1-Score* (93.09 %). Esto evidencia que al combinar tres expertos, el modelo `MoE` es capaz de capturar patrones más complejos y minimizar errores, superando en algunos casos a los modelos individuales.

- **Relación entre cantidad de selección de expertos y desempeño:** Se observó una tendencia clara de mejora en el desempeño a medida que se incrementó el número de expertos en las configuraciones de los modelos `MoE`. El modelo `MoE` con 1 experto mostró resultados aceptables, pero por debajo de los modelos individuales, mientras que la selección de 2 expertos mejoró considerablemente las métricas clave. Finalmente, con 3 expertos, el modelo logró resultados competitivos, acercándose e incluso superando en algunos casos a los modelos individuales.

  Estos hallazgos validan la hipótesis principal y la hipótesis especifica 1 que se plantearon anteriormente, el aumento en la cantidad de expertos seleccionados permite capturar patrones más complejos y reducir la tasa de error, sugiriendo que con un mayor número de expertos, es posible alcanzar métricas aún más robustas, optimizando el rendimiento general del modelo `MoE`.

10. Conclusiones

En esta investigación, se evaluaron diferentes modelos para la clasificación de ojos con glaucoma y normales, analizando tanto arquitecturas de modelos de redes neuronales convolucionales (CNN) de manera individual como configuraciones de modelos basados en `MoE` con el objetivo de identificar un enfoque que mejore el desempeño global en tareas de clasificación médica. Los resultados muestran que el uso de configuraciones de modelos `MoE` permite combinar las fortalezas individuales de los expertos base, logrando un desempeño superior en métricas clave como *Recall*, *Precision*, *Specificity* y *Error_rate*.

En los modelos individuales, `DenseNet121`, `VGG19` y `ResNet50` demostraron ser altamente efectivos, con métricas robustas y un excelente balance entre *Recall* y *Specificity*.

Por otro lado, las configuraciones de los modelos `MoE` demostraron un progreso iterativo en el desempeño al incrementar la cantidad de selección de expertos. En particular, la configuración con 3 expertos seleccionados no solo superó a la mayoria de modelos individuales en *Accuracy* global y con una baja desviacion estandar en *Accuracy*, sino que también alcanzó un equilibrio sobresaliente entre *Recall*, *Specificity* y un *Error_rate* muy bajo, características esenciales en contextos médicos donde se busca minimizar tanto los falsos negativos como los falsos positivos. Esto valida la efectividad del enfoque `MoE` como un acercamiento para mejorar la capacidad de generalización en modelos complejos. Estos hallazgos confirman la hipótesis inicial de que la combinación de expertos en un modelo `MoE` proporciona un beneficio significativo en problemas que requieren alta *Precision* y *Recall*. Además, se evidencia que el diseño de sistemas de clasificación basados en múltiples modelos puede ser una estrategia prometedora para tareas similares, especialmente en aplicaciones médicas donde los errores tienen un impacto crítico.

Finalmente, estos resultados abren la puerta para futuras investigaciones que busquen perfeccionar este enfoque, explorando el impacto de diferentes combinaciones de modelos base, estrategias de selección de expertos, y ajustes en los hiperparámetros de los modelo `MoE` para lograr un desempeño aún mejor en problemas de clasificación médica.

11. Limitaciones del trabajo y trabajos futuros

A pesar de los resultados positivos obtenidos con los modelos `MoE`, aún existen algunas limitaciones que podrían mejorarse en futuras investigaciones. Algunas de las áreas en las que se podría continuar trabajando para mejorar el desempeño y la aplicabilidad de estos modelos son:

- **Ampliación de la base de datos:** Aunque los resultados fueron satisfactorios, la cantidad de datos utilizados en este estudio podría no ser suficiente para generalizar aún más los modelos. La inclusión de una base de datos más amplia, con una mayor diversidad de imágenes, podría permitir al modelo aprender representaciones más robustas y mejorar la capacidad de generalización.

- **Aumento en la cantidad de expertos en MoE:** Si bien la configuración con tres expertos ha mostrado buenos resultados, se podría explorar el uso de más expertos dentro del modelo `MoE`. La adición de más expertos podría mejorar aún más el rendimiento al permitir que el modelo capture una mayor diversidad de patrones en los datos, lo cual podría ser crucial en tareas complejas.

- **Ajuste de hiperparámetros:** El ajuste adecuado de los hiperparámetros, como la tasa de aprendizaje, el número de capas, y las funciones de activación, podría ser clave para mejorar el rendimiento de los modelos `MoE`. Experimentos adicionales con diferentes configuraciones de hiperparámetros podrían ayudar a encontrar el equilibrio óptimo para cada modelo.

- **Fine-tuning a los expertos:** Antes de integrar los modelos base en la configuración `MoE`, se podría realizar un proceso de fine-tuning a los expertos por separado. Esto permitiría optimizar aún más los expertos y asegurar que cada uno esté lo mejor preparado posible antes de la implementación en `MoE`.

- **Evaluación en diferentes dominios:** Actualmente, los modelos han sido evaluados en un solo tipo de datos (por ejemplo, imágenes médicas). Un enfoque futuro podría ser la evaluación de estas configuraciones `MoE` en otras áreas, como la cla-

sificación de imágenes en otras aplicaciones, como reconocimiento de objetos en entornos industriales, o incluso en imágenes de satélite.

- **Implementación de técnicas de interpretación de modelos:** Otra área de mejora es la capacidad de interpretar las decisiones de los modelos. La integración de técnicas que permitan entender cómo y por qué un modelo `MoE` toma una decisión puede ser crucial, especialmente en aplicaciones críticas como la medicina. Una de estas herramientas que ayudarían a la iterpretacion de las decisiones del modelo seria la implementacion de el algoritmo de Grad-Cam. Esto podría facilitar la confianza en los modelos y su adopción en el mundo real.

Bibliografía

[1] Jost B Jonas, Tin Aung, Rupert R Bourne, Alain M Bron, Robert Ritch, and Songhomitra Panda-Jonas. Glaucoma. *The Lancet*, 390(10108):2183–2193, November 2017.

[2] World Health Organization et al. World report on vision. 2019.

[3] Adnan Haider, Muhammad Arsalan, Min Beom Lee, Muhammad Owais, Tahir Mahmood, Haseeb Sultan, and Kang Ryoung Park. Artificial intelligence-based computer-aided diagnosis of glaucoma using retinal fundus images. *Expert Systems with Applications*, 207:117968, November 2022.

[4] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2023.

[5] D. Kumar. Knowledge based morphological deep transparent neural networks for remote sensing image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 15:1–1, 01 2022.

[6] Zhen Liu, Shiyan Hu, Ye Sun, and Behnam Azmoon. An exploratory investigation into image-data-driven deep learning for stability analysis of geosystems. *Geotechnical and Geological Engineering*, 40, 08 2021.

[7] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts, 2021.

[8] Najeeba Afreen and Rajanikanth Aluvalu. Glaucoma detection using explainable ai and deep learning. *EAI Endorsed Transactions on Pervasive Health and Technology*, 10, April 2024.

[9] S. Puangarom, A. Twinvitoo, S. Sangchocanonta, A. Munthuli, P. Phienphanich, R. Itthipanichpong, K. Ratanawongphaibul, S. Chansangpetch, A. Manassakorn, V. Tantisevi, P. Rojanapongpun, and C. Tantibundhit. 3-lbnets: Tri-labeling deep convolutional neural network for the automated screening of glaucoma, glaucoma suspect, and no glaucoma in fundus images. In *2023 45th Annual International Conference of the IEEE Engineering in Medicine amp; Biology Society (EMBC)*, page 1–5. IEEE, July 2023.

[10] Hyeonsung Cho, Young Hoon Hwang, Jae Keun Chung, Kwan Bok Lee, Ji Sang Park, Hong-Gee Kim, and Jae Hoon Jeong. Deep learning ensemble method for classifying glaucoma stages using fundus photographs and convolutional neural networks. *Current Eye Research*, 46(10):1516–1524, April 2021.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[12] Anuradha Khattar and S. M. K. Quadri. "generalization of convolutional network to domain adaptation network for classification of disaster images on twitter". *Multimedia Tools and Applications*, 81(21):30437–30464, April 2022.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[14] Rahul Gomes, Connor Kamrowski, Jordan Langlois, Papia Rozario, Ian Dircks, Keegan Grottodden, Matthew Martinez, Wei Zhong Tee, Kyle Sargeant, Corbin LaFleur, and Mitchell Haley. A comprehensive review of machine learning used to combat covid-19. *Diagnostics*, 12(8):1853, July 2022.

[15] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[16] Andres Diaz-Pinto, Sandra Morales, Valery Naranjo, Thomas Köhler, Jose M. Mossi, and Amparo Navea. Cnns for automatic glaucoma assessment using fundus images: An extensive validation, 2019.