Maestría en

CIENCIA DE DATOS Y MÁQUINAS DE APRENDIZAJE CON MENCIÓN EN INTELIGENCIA ARTIFICIAL

Trabajo previo a la obtención de título de Magister en Ciencia De Datos y Máquinas de Aprendizaje con Mención en Inteligencia Artificial

AUTORES:

GUERRERO VELEZ KEMPIS

HARO MUÑOZ ALEJANDRO SEBASTIAN

LOOR ZAMBRANO MATHIAS ALEJANDRO

PAZMIÑO PARRA CRISTHIAN ANDRES

TUTOR/ES:

Iván Reyes Chacón

Alejandro Cortés López

TEMA:

Detección Automatizada de la Salud de Corales mediante Visión por Computadora y Aprendizaje Profundo

Quito - Ecuador

REINVENTEMOS EL FUTURO

Certificación de autoría

Nosotros, Alejandro Sebastián Haro Muñoz, Kempis Guerrero Vélez, Cristhian Andrés Pazmiño Parra, y Mathias Alejandro Loor Zambrano**,** declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido presentado anteriormente para ningún grado o calificación profesional y que se ha consultado la bibliografía detallada.

Cedemos nuestros derechos de propiedad intelectual a la Universidad Internacional del Ecuador (UIDE), para que sea publicado y divulgado en internet, según lo establecido en la Ley de Propiedad Intelectual, su reglamento y demás disposiciones legales.

Firma del graduando
Alejandro Sebastián Haro Muñoz

Firma del graduando
Kempis Guerrero Vélez

Firma del graduando
Cristhian Andrés Pazmiño Parra

Firma del graduando
Mathias Alejandro Loor Zambrano

Autorización de Derechos de Propiedad Intelectual

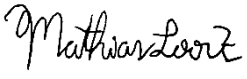
Nosotros, Alejandro Sebastián Haro Muñoz, Kempis Guerrero Vélez, Cristhian Andrés Pazmiño Parra, y Mathias Alejandro Loor Zambrano, en calidad de autores del trabajo de investigación titulado ***Detección Automatizada de la Salud de Corales mediante Visión por Computadora y Aprendizaje Profundo***, autorizamos a la Universidad Internacional del Ecuador (UIDE) para hacer uso de todos los contenidos que nos pertenecen o de parte de los que contiene esta obra, con fines estrictamente académicos o de investigación. Los derechos que como autores nos corresponden, lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento en Ecuador.

D. M. Quito, Julio, 2025

Firma del graduando
Alejandro Sebastián Haro Muñoz

Firma del graduando
Kempis Guerrero Vélez

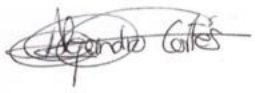
Firma del graduando
Cristhian Andrés Pazmiño Parra



Firma del graduando
Mathias Alejandro Loor Zambrano

Aprobación de dirección y coordinación del programa

Nosotros, **Alejandro Cortés e Iván Reyes Chacón,** declaramos que: Alejandro Sebastián Haro Muñoz, Kempis Guerrero Vélez, Cristhian Andrés Pazmiño Parra, y Mathias Alejandro Loor Zambrano son los autores exclusivos de la presente investigación y que ésta es original, auténtica y personal de ellos.



Alejandro Cortés López

Director de la

Maestría en Ciencia de Datos y Máquinas de Aprendizaje con mención en Inteligencia Artificial

Iván Reyes Chacón

Coordinador de la

Maestría en Ciencia de Datos y Máquinas de Aprendizaje con mención en Inteligencia Artificial

DEDICATORIA

Este proyecto va dedicado a nuestras familias, por su incondicional apoyo y motivación a lo largo de este camino académico lleno de aprendizajes y enseñanzas. El amor y sacrificio de nuestras familias fueron el pilar fundamental que nos impulsó a alcanzar esta meta y cumplir un objetivo mas en nuestras vidas.

A nuestros tutores y docentes de carrera, quienes, con su sabiduría y guía, no solo nos transmitieron conocimientos, sino que también nos inspiraron a superar desafíos y a creer en el potencial de la ciencia de datos para generar un impacto positivo y poder cambiar el mundo.

Y a los arrecifes de coral, por su intrínseca belleza y vital importancia para el planeta. Que este trabajo sea una pequeña contribución a su conservación y un recordatorio de la urgencia de proteger nuestros ecosistemas marinos y así poder salvar el mundo de tanta contaminación mundial.

AGRADECIMIENTOS

Extendemos nuestros más sinceros agradecimientos a todos nuestros familiares y docentes que hicieron posible de esta culminación académica.

En primer lugar, a nuestros familiares y seres queridos, por su apoyo incondicional, paciencia y constante motivación a lo largo de este exigente camino académico y de enseñanzas. Su comprensión fue un pilar fundamental en cada etapa de la carrera académica.

A la Universidad Internacional del Ecuador (UIDE), por brindarnos la formación académica de excelencia y los recursos necesarios para desarrollar esta investigación. Especialmente, agradecemos al cuerpo docente de la Maestría en Ciencia de Datos, cuya guía y conocimientos fueron indispensables para la concepción y ejecución de este trabajo académico.

Un agradecimiento particular a nuestros tutores, por su invaluable dirección, sus revisiones detalladas y su compromiso constante. Su experiencia y consejos fueron cruciales para superar los desafíos técnicos y conceptuales que surgieron a lo largo de esta carrera académica.

Finalmente, agradecemos a las comunidades científicas, por poner a disposición los recursos y herramientas que facilitaron la recopilación de datos y el desarrollo de los modelos aquí presentados.

RESUMEN

Los arrecifes de coral, fundamentales para la biodiversidad marina, se encuentran en grave riesgo debido al cambio climático y la contaminación. El blanqueamiento coralino, causado principalmente por el aumento de la temperatura del océano, representa una señal crítica de deterioro ecológico. Este proyecto tiene como objetivo desarrollar un sistema automatizado de detección del estado de salud de los corales mediante técnicas de visión por computadora y aprendizaje profundo.

Se implementaron y compararon tres enfoques: un modelo base de Perceptrón Multicapa (MLP), una red neuronal convolucional (CNN) diseñada desde cero y un modelo basado en Transfer Learning utilizando la arquitectura VGG16. Los modelos fueron entrenados con imágenes submarinas obtenidas desde Roboflow y evaluados mediante métricas como precisión, sensibilidad, especificidad y matriz de confusión. Además, se integraron técnicas de interpretabilidad como Grad-CAM y LIME para validar las decisiones del modelo.

Los resultados obtenidos muestran que el enfoque basado en Transfer Learning alcanzó un rendimiento de hasta el 97 % de precisión, superando al MLP y siendo comparable a la CNN entrenada desde cero. El sistema fue validado con datos reales y se desarrolló un prototipo funcional que permite realizar predicciones de manera automática y visual, facilitando así su integración en plataformas de monitoreo ambiental. Este trabajo demuestra el potencial de la inteligencia artificial como herramienta eficaz para la conservación marina.

Palabras Claves: visión por computadora, aprendizaje profundo, blanqueamiento coralino, clasificación de imágenes, redes neuronales, VGG16, Grad-CAM, LIME.

ABSTRACT

Coral reefs, essential to marine biodiversity, are at serious risk due to climate change and pollution. Coral bleaching, mainly caused by ocean warming, is a critical indicator of ecological degradation. This project aims to develop an automated system for detecting coral health status using computer vision and deep learning techniques.

Three approaches were implemented and compared: a baseline Multi-Layer Perceptron (MLP), a custom Convolutional Neural Network (CNN), and a Transfer Learning model based on the VGG16 architecture. The models were trained on underwater images from Roboflow and evaluated using metrics such as accuracy, sensitivity, specificity, and confusion matrix. Interpretability techniques, such as Grad-CAM and LIME, were integrated to validate the model's decisions.

The results show that the Transfer Learning approach achieved up to 97% accuracy, outperforming the MLP and matching the performance of the custom CNN. The system was validated with real-world data, and a functional prototype was developed to allow automatic and visual predictions, enabling future integration into environmental monitoring platforms. This work demonstrates the potential of artificial intelligence as an effective tool for marine conservation.

Keywords: computer vision, deep learning, coral bleaching, image classification, neural networks, VGG16, Grad-CAM, LIME.

TABLA DE CONTENIDOS (Índice)

LISTA DE TABLAS (Índice de tablas)

LISTA DE FIGURAS (Índice de figuras)

CAPITULO 1:

1. INTRODUCCIÓN

1.1 Definición del proyecto

Los arrecifes de coral representan uno de los ecosistemas más diversos y vitales del planeta, ya que albergan aproximadamente el 25 % de todas las especies marinas conocidas (Hoegh-Guldberg et al., 2017). Estos ecosistemas cumplen funciones ecológicas, económicas y sociales críticas, como la protección costera, el mantenimiento de la biodiversidad y el sustento para comunidades pesqueras y turísticas (Moberg & Folke, 1999). Sin embargo, debido al cambio climático, la acidificación de los océanos y la contaminación, los arrecifes de coral enfrentan una rápida degradación, siendo el blanqueamiento coralino uno de los principales indicadores del deterioro de su salud (Lough et al., 2018).

Frente a esta problemática, este proyecto plantea el desarrollo de un sistema automatizado de monitoreo de la salud de los corales mediante técnicas de visión por computadora y aprendizaje profundo. Se busca identificar el estado de salud de los corales a partir de imágenes submarinas, clasificándolos como sanos o blanqueados mediante modelos de clasificación entrenados con datos reales. La automatización de este proceso permitiría una detección temprana y a gran escala del blanqueamiento coralino, facilitando la toma de decisiones en proyectos de conservación (Rocha et al., 2020).

1.2 Justificación e importancia del trabajo de investigación

El monitoreo tradicional de arrecifes se basa en inspecciones visuales realizadas por buzos expertos, lo cual es costoso, subjetivo, limitado en cobertura espacial y difícil de escalar (Rocha et al., 2020). La visión por computadora y el aprendizaje profundo ofrecen una alternativa eficiente para este problema, permitiendo analizar grandes volúmenes de datos visuales con alta precisión (LeCun et al., 2015).

Diversas investigaciones han demostrado la efectividad de las redes neuronales convolucionales (CNN) para identificar patrones complejos en imágenes submarinas, incluso en entornos con condiciones lumínicas variables y presencia de ruido visual (Kavasidis et al., 2016). En particular, la técnica de Transfer Learning permite reutilizar modelos pre-entrenados en grandes conjuntos de datos (como ImageNet) y ajustarlos a problemas específicos, logrando así una mejor generalización con menos datos de entrenamiento (Pan & Yang, 2010).

Este trabajo se justifica no solo por el aporte tecnológico a la conservación marina, sino también por su potencial de escalabilidad e integración con plataformas globales de monitoreo ambiental. Además, contribuye a cerrar la brecha entre la ciencia computacional y la biología marina, demostrando la aplicabilidad real de la inteligencia artificial en contextos ecológicos sensibles.

1.3 Alcance

Este proyecto abarca el diseño, implementación y evaluación de modelos de aprendizaje profundo para la clasificación del estado de salud coralina a partir de imágenes submarinas. Se implementaron y compararon tres arquitecturas: un perceptrón multicapa (MLP), una red neuronal convolucional (CNN) y un modelo de Transfer Learning basado en VGG16. Los modelos fueron entrenados y validados utilizando un dataset extraído desde la plataforma Roboflow. El sistema resultante es capaz de predecir si un coral se encuentra sano o blanqueado, visualizando además las regiones de atención mediante la técnica Grad-CAM. Como resultado final, se desarrolló un prototipo funcional que puede integrarse en futuras plataformas de monitoreo ambiental automatizado

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar un sistema automatizado que, mediante técnicas de visión por computadora y aprendizaje profundo, permita detectar y clasificar el estado de salud de los corales, contribuyendo al monitoreo y conservación de los ecosistemas marinos.

1.4.2 Objetivo especifico

- Revisar la literatura existente sobre la aplicación de visión por computadora y aprendizaje profundo en la evaluación de la salud de los corales.
- Diseñar un modelo de aprendizaje profundo capaz de clasificar el estado de salud de los corales a partir de imágenes submarinas.
- Entrenar y validar el modelo utilizando conjuntos de datos adecuados, evaluando su desempeño mediante métricas como precisión, sensibilidad y especificidad.
- Implementar técnicas de explicabilidad para modelos de tipo caja negra con el fin de comprender regiones importantes que contribuyen a las predicciones.
- Implementar un prototipo funcional que integre el modelo desarrollado, permitiendo la identificación automática del estado de salud de los corales en tiempo real.

CAPITULO 2:

2. REVISIÓN DE LITERATURA

1.1 Estado del Arte

La inteligencia artificial ha sido ampliamente utilizada en los últimos años para resolver problemas de clasificación y segmentación en imágenes médicas, agricultura de precisión y monitoreo ambiental. En el contexto marino, su uso ha crecido en aplicaciones como la identificación de especies, conteo de peces, detección de basura oceánica y análisis de salud coralina (Beijbom et al., 2012; González-Rivero et al., 2020).

Beijbom et al. (2015) introdujeron uno de los primeros sistemas automatizados para el análisis de fotografías de corales, demostrando que los clasificadores automáticos pueden igualar la precisión de expertos humanos. Posteriormente, Williams et al. (2019) emplearon redes neuronales convolucionales (CNN) para identificar patrones de blanqueamiento en imágenes satelitales, lo cual extendió la utilidad de estas tecnologías a escalas regionales.

Más recientemente, investigaciones como la de Gómez-Ríos et al. (2021) han aplicado técnicas de aprendizaje profundo en entornos submarinos reales utilizando imágenes recolectadas con vehículos autónomos (ROVs), lo que demuestra la viabilidad técnica de aplicar modelos CNN en condiciones adversas de luz y visibilidad. Del mismo modo, Massot-Campos y Oliver-Codina (2020) argumentan que las técnicas de visión por computadora submarina son una herramienta esencial para el monitoreo de biodiversidad en arrecifes.

Con respecto al uso de Transfer Learning, trabajos como los de Loya et al. (2022) y Bessis et al. (2023) han evidenciado que modelos como VGG16 y ResNet50, preentrenados con ImageNet, pueden adaptarse con éxito a conjuntos de datos limitados mediante fine-tuning, optimizando el uso de recursos y mejorando la estabilidad en la predicción.

El desarrollo de explicabilidad en IA también ha sido crucial. Selvaraju et al. (2017) propusieron Grad-CAM como método visual para identificar las áreas de atención del modelo durante la predicción. Esta técnica ha sido empleada en sistemas ambientales para validar que los modelos efectivamente se enfocan en los elementos clave de las imágenes, incrementando la confianza en su uso para la toma de decisiones (Zhou et al., 2016).

1.2 Marco Teórico

1.2.1 Visión por Computadora

La visión por computadora es una rama de la inteligencia artificial que permite a las máquinas interpretar y extraer información útil de imágenes o secuencias de vídeo. Sus aplicaciones incluyen desde sistemas biométricos hasta análisis ambiental (Szeliski, 2010). En entornos marinos, se enfrenta a desafíos particulares como la distorsión por refracción, la dispersión de luz y el movimiento del agua (Bryson et al., 2016).

1.2.2 Aprendizaje Profundo

El aprendizaje profundo (Deep Learning) se basa en el uso de redes neuronales profundas para extraer representaciones jerárquicas de los datos. Este enfoque ha superado en precisión a otros métodos en tareas de clasificación de imágenes, particularmente cuando se dispone de un volumen considerable de datos (Goodfellow et al., 2016). Las arquitecturas CNN son especialmente efectivas para imágenes, debido a su capacidad de aprender filtros espaciales especializados.

1.2.3 Redes Neuronales Convolucionales (CNN)

Las CNN están formadas por capas convolucionales, capas de activación (ReLU), capas de agrupamiento (pooling) y capas densas. Estas redes han sido utilizadas con éxito en clasificación de imágenes biomédicas, agrícolas y ambientales. En visión submarina, su capacidad para aprender patrones texturales y diferencias de color ha sido clave para clasificar imágenes de corales (Wäldchen & Mäder, 2018).

1.2.4 Transfer Learning

Transfer Learning consiste en tomar un modelo entrenado en una tarea con abundante información (como ImageNet) y ajustarlo a una nueva tarea específica. Esto reduce el tiempo de entrenamiento y mejora la generalización, siendo ideal para proyectos con datos limitados (Pan & Yang, 2010).

1.2.5 Técnicas de Interpretabilidad

Grad-CAM y LIME son dos técnicas de interpretabilidad que permiten comprender las decisiones de los modelos de clasificación visual, identificando las regiones o características de una imagen que más influyen en una predicción específica. Grad-CAM (Selvaraju et al., 2017), por ejemplo, genera mapas de calor que visualizan directamente estas áreas influyentes dentro de la imagen. Por su parte, LIME (Ribeiro et al., 2016) se enfoca en explicar predicciones individuales de forma local y agnóstica al modelo, destacando la importancia de segmentos. Ambas metodologías se han establecido como técnicas clave para evaluar el comportamiento de los modelos de visión por computadora, lo cual contribuye a aumentar la confianza en modelos de tipo caja negra.

CAPITULO 3:

2. DESARROLLO

2.1 Desarrollo del Trabajo

El presente proyecto se fundamenta en el desarrollo e implementación de modelos de aprendizaje profundo para la detección automatizada del estado de salud de los corales mediante imágenes submarinas. Para ello, se abordaron tres enfoques distintos: un perceptrón multicapa (MLP) como línea base, una red neuronal convolucional (CNN) diseñada desde cero, y un modelo basado en transferencia de aprendizaje utilizando la arquitectura VGG16. A continuación, se describen las etapas que conformaron el desarrollo experimental.

2.1.1 Dataset y Preprocesamiento

Se utilizó el conjunto de datos Coral Reef Bleach Detection (versión 2), obtenido desde la plataforma Roboflow. Este dataset contiene 2,424 imágenes submarinas a color, categorizadas en dos clases principales: corales sanos y corales blanqueados, según su apariencia visual y pérdida de pigmentación (Rocha et al., 2020). El conjunto de imágenes analizado comprende un total de 5,274 anotaciones. Esta cantidad superior al número de imágenes individuales se debe a la inclusión de múltiples bounding boxes en algunas de ellas. La selección de este dataset se fundamentó en su óptimo balance de datos, con 2.841 anotaciones correspondientes a corales blanqueados y 2.433 a corales sanos Las imágenes fueron divididas originalmente en tres subconjuntos: entrenamiento (70 % – 1696 imágenes), validación (15 % – 364 imágenes) y prueba (15 % – 364 imágenes).

Para el modelo MLP, además del preprocesamiento convencional, se aplicó una división adicional del conjunto de entrenamiento utilizando “train_test_split” , separando el 80 % de este subconjunto (≈1,468 imágenes) para entrenamiento efectivo y el 20 % restante (≈368 imágenes) para validación interna. Las imágenes fueron convertidas a escala de grises,

redimensionadas a 56×56 píxeles, aplanadas a vectores de 3,136 características y normalizadas en el rango [0,1].

En el caso de los modelos CNN y VGG16, se conservaron las imágenes en formato RGB, redimensionadas a 224×224 píxeles. La normalización de los valores de los píxeles se realizó dividiendo por 255 en la CNN, mientras que en VGG16 se utilizó la función "preprocess_input" propia del modelo preentrenado. Además, se implementaron técnicas de aumentación de datos (rotación, desplazamientos, zoom, y espejado horizontal) durante el entrenamiento, con el fin de mejorar la capacidad de generalización de los modelos frente a variaciones visuales presentes en entornos submarinos reales. La configuración de hiper parámetros empleada durante el entrenamiento de los modelos se resume en la Tabla 1.

Tabla 1

Configuración de hiper parámetros empleada en cada modelo.

Parámetro	MLP	CNN Personalizada	Transfer Learning (VGG16)
Tamaño de imagen	56×56 (escala de grises)	224×224 (RGB)	224×224 (RGB)
Número de clases	2	2	2
Optimización	Adam / SGD / Lion	Adam	Adam
Learning rate	0.001 (por defecto)	0.0001	0.0001
Batch size	32	32	32
Número de épocas	15	15	15
EarlyStopping	No	Sí (paciencia: 10)	Sí (paciencia: 10)

ReduceLROnPlateau	No	Sí (factor: 0.2, paciencia: 5)	Sí (factor: 0.2, paciencia: 5)
ModelCheckpoint	No	Sí	Sí
Aumentación de datos	No	Sí (rotación, zoom, flip, desplazamiento)	Sí (rotación, zoom, flip, desplazamiento)

2.1.2 Arquitecturas de Modelo

Con el objetivo de comparar distintas aproximaciones al problema de clasificación del estado de salud de los corales, se implementaron tres modelos de aprendizaje profundo: un perceptrón multicapa (MLP) como modelo base, una red neuronal convolucional (CNN) personalizada y un modelo de transferencia de aprendizaje basado en VGG16. La Tabla 2 resume las técnicas de regularización utilizadas y las capas finales empleadas en cada una de estas arquitecturas.

Tabla 2

Resumen de técnicas de regularización y diseño de capas finales por modelo.

Modelo	Regularización Aplicada	Capas Finales
MLP	Dropout (0.3 / 0.2)	Dense(256) → Dense(128) → Softmax
CNN	BatchNorm + Dropout (0.5)	Conv2D →MaxPooling → Softmax
VGG16	BatchNorm + Dropout (0.5)	Flatten → Dense(512×2) → Softmax

A continuación, se detallan sus características arquitectónicas.

2.1.2.1 Perceptrón Multicapa (MLP)

El MLP fue diseñado como un modelo de referencia que permite establecer una línea base comparativa. Debido a su estructura totalmente conectada, este modelo no conserva información espacial de las imágenes, lo cual limita su capacidad para aprender patrones visuales complejos. Las imágenes fueron aplanadas a vectores unidimensionales de 3,136 features, lo que equivale a imágenes en escala de grises de tamaño 56×56 píxeles.

- La arquitectura consistió en:
- Una capa de entrada de 3,136 neuronas
- Una capa oculta densa con 256 neuronas y activación ReLU
- Una segunda capa oculta con 128 neuronas, también con ReLU
- Capas intermedias de Dropout con tasas de 0.3 y 0.2 respectivamente, para prevenir el sobreajuste
- Una capa de salida con activación softmax, adaptada al número de clases (2)
- Se probaron distintos algoritmos de optimización: Adam, SGD y Lion.

2.1.2.2 Red Neuronal Convolucional (CNN)

Para superar las limitaciones espaciales del MLP, se diseñó una red convolucional desde cero con el fin de capturar patrones texturales, formas y estructuras presentes en las imágenes submarinas. Esta red procesó imágenes RGB de 224×224 píxeles, empleando un flujo estándar de convolución, activación, normalización y reducción de dimensionalidad.

La arquitectura empleada incluyó:

- Cuatro bloques convolucionales secuenciales con filtros de 32, 64, 128 y 256, activación ReLU y capas de “BatchNormalization”.
- Capas de “MaxPooling2D” tras cada convolución para reducir la resolución espacial.
- Una capa Flatten para aplanar las salidas hacia una dimensión vectorial.

- Una capa densa de 512 neuronas, seguida de Dropout (0.5).
- Capa de salida softmax con 2 neuronas (una por clase)
- Este modelo fue diseñado para capturar patrones espaciales complejos en las imágenes, aprovechando su estructura convolucional profunda.

2.1.2.3 Transferencia de Aprendizaje con VGG16

Como tercera estrategia, se empleó la arquitectura VGG16 pre entrenada en ImageNet, bajo el paradigma de "transfer learning". Se congeló completamente la base convolucional del modelo y se construyó una nueva "cabeza" de clasificación, entrenada con el conjunto de datos específico del problema.

La nueva cabeza consistió en:

- Capa Flatten sobre la salida de VGG16.
- Dos capas densas consecutivas de 512 neuronas cada una, con activación ReLU.
- Capas de "BatchNormalization" y Dropout (0.5) para mejorar la generalización.
- Capa final softmax con salida multiclase.

Este modelo fue diseñado con el objetivo de aprovechar características visuales previamente aprendidas en grandes conjuntos de datos generales, lo que permite reducir el tiempo de entrenamiento y minimizar el riesgo de sobreajuste.

2.1.3 Entrenamiento y Validación

El proceso de entrenamiento de los tres modelos se realizó utilizando el framework TensorFlow/Keras, haciendo uso de recursos GPU provistos por Google Colab. Se aplicaron técnicas estándar de entrenamiento supervisado, incluyendo división en conjuntos de entrenamiento, validación y prueba, junto con el uso de callbacks como EarlyStopping, ReduceLROnPlateau y ModelCheckpoint para mejorar la generalización y evitar el sobreajuste.

Cada modelo fue entrenado bajo condiciones controladas con 15 épocas máximas, batch_size=32, y funciones de pérdida y métricas apropiadas para clasificación multiclase. A continuación, se presenta una síntesis de la configuración empleada en cada caso.

2.1.4 Interpretabilidad del Modelo

En el contexto de aprendizaje profundo, los modelos pueden comportarse como "cajas negras", dificultando la comprensión de sus decisiones. Para mejorar la interpretabilidad del sistema de clasificación de corales, se integraron dos técnicas ampliamente utilizadas: Grad-CAM (Gradient-weighted Class Activation Mapping) y LIME (Local Interpretable Model-Agnostic Explanations).

2.1.4.1 Grad-CAM

Grad-CAM fue aplicado a los modelos convolucionales (CNN personalizada y VGG16) para visualizar las regiones de las imágenes que influyeron más en las decisiones del modelo. Esta técnica genera un mapa de calor superpuesto a la imagen original, resaltando las áreas relevantes según la activación de las últimas capas convolucionales (Selvaraju et al., 2017).

Los mapas Grad-CAM permitieron verificar que el modelo focaliza su atención en estructuras coralinas específicas como los bordes, texturas y coloraciones, y no en el fondo o elementos irrelevantes. Esto validó que el modelo toma decisiones basadas en características visuales pertinentes al diagnóstico de salud coralina.

2.1.4.2 LIME

Adicionalmente, se implementó la técnica LIME para explicar predicciones individuales en imágenes específicas. Se configuró un segmentador tipo quickshift y se utilizaron al menos 1,000 muestras para generar explicaciones estables. LIME permitió analizar casos individuales con mayor profundidad, identificando segmentos que apoyaban o

contradecían la predicción, lo cual resultó útil para interpretar casos limítrofes y confirmar la coherencia del modelo.

2.1.4.3 Aplicación y Visualización

Ambas técnicas fueron integradas dentro del flujo del prototipo. En el caso de Grad-CAM, los mapas de calor se generaban automáticamente para cada predicción realizada por los modelos convolucionales. En el caso de LIME, se incluyó una función específica para análisis manual sobre imágenes seleccionadas del conjunto de prueba.

Las visualizaciones obtenidas confirmaron que los modelos eran capaces de identificar patrones morfológicos asociados al estado de salud del coral, otorgando así mayor transparencia al proceso de decisión automática y aumentando la confiabilidad del sistema.

2.2 Marco Teórico

La aplicación práctica de técnicas de aprendizaje profundo para la clasificación automatizada de corales requiere comprender cómo se interrelacionan los fundamentos teóricos ya revisados con las decisiones técnicas tomadas durante el desarrollo del sistema. En esta sección se describe cómo se operacionalizan los conceptos previos en el diseño e implementación del modelo propuesto.

2.2.1 Aplicación de la Visión por Computadora

En esta investigación, la visión por computadora se emplea para analizar imágenes submarinas captadas en condiciones reales. Se aplicaron técnicas como:

- Redimensionamiento estandarizado: todas las imágenes fueron escaladas a 224×224 px para CNN y VGG16, y a 56×56 px para MLP.
- Normalización de píxeles: las imágenes fueron normalizadas para estabilizar el proceso de entrenamiento.
- Conversión a escala de grises: utilizada exclusivamente en el modelo MLP para reducir la dimensionalidad de entrada.

- Estas transformaciones permiten preparar los datos para que las redes neuronales puedan procesarlos de forma eficiente y precisa.

2.2.2 Modelado mediante Arquitecturas Neuronales

Se compararon tres arquitecturas de red, cada una basada en principios distintos del aprendizaje profundo:

- Perceptrón Multicapa (MLP): red densa sin preservación de información espacial, útil como línea base comparativa.
- Red Convolucional (CNN): diseñada desde cero para aprender características espaciales complejas.
- Transfer Learning (VGG16): modelo preentrenado reutilizado para extraer descriptores visuales generales, sobre el cual se entrenó una nueva cabeza de clasificación.
- Cada arquitectura fue elegida con el objetivo de explorar la relación entre complejidad estructural, rendimiento predictivo y capacidad de generalización.

2.2.3 Regularización y Optimización

Para mitigar el sobreajuste y mejorar la capacidad de generalización de los modelos, se integraron diversas técnicas:

- Dropout: se aplicó en MLP, CNN y VGG16 con tasas de hasta 0.5.
- Batch Normalization: utilizada en modelos con múltiples capas densas para estabilizar el aprendizaje.
- Callbacks: se incorporaron estrategias como EarlyStopping, ReduceLROnPlateau y ModelCheckpoint para optimizar el entrenamiento de forma dinámica.
- Adicionalmente, se compararon distintos algoritmos de optimización como Adam, SGD y Lion en el modelo MLP, evaluando su efecto en el rendimiento.

2.2.4 Técnicas de Interpretabilidad en el Desarrollo

Grad-CAM y LIME fueron integradas al flujo del sistema no solo como herramientas de validación, sino también como mecanismos de inspección de errores y depuración de predicciones:

- Grad-CAM permitió verificar visualmente si el modelo enfocaba su atención en los corales y no en elementos del fondo marino.
- LIME proporcionó explicaciones locales en imágenes específicas, útiles para analizar casos límite y ajustar el etiquetado del dataset.
- Ambas herramientas aportaron una capa de transparencia crítica al proceso de desarrollo, permitiendo afinar el modelo más allá de métricas numéricas.

CAPITULO 4:

3. ANÁLISIS DE RESULTADOS

Este capítulo constituye el núcleo empírico de la presente investigación. Su propósito es presentar, analizar e interpretar de manera exhaustiva los resultados obtenidos a partir de la implementación y evaluación de tres arquitecturas de aprendizaje profundo distintas: un Perceptrón Multicapa (MLP) como modelo de referencia, una Red Neuronal Convolucional (CNN) diseñada a medida, y un modelo de Aprendizaje por Transferencia basado en la arquitectura VGG16.

El análisis se estructura en dos ejes fundamentales. El primer eje se centra en la precisión de clasificación, evaluando cuantitativamente la capacidad de cada modelo para distinguir correctamente entre corales sanos y blanqueados. El segundo eje aborda la eficiencia computacional**,** un factor crítico para la viabilidad práctica y el despliegue de soluciones de inteligencia artificial en entornos reales. Se examinará la complejidad de cada modelo, medida por su número de parámetros, para entender el trade-off entre el rendimiento predictivo y los recursos computacionales requeridos.

Finalmente, se llevará a cabo una discusión comparativa que sintetice los hallazgos de ambos ejes. Esta discusión culminará en la selección y justificación del modelo considerado como la solución globalmente óptima, no solo en base a una única métrica, sino a un balance razonado entre precisión, eficiencia y viabilidad estratégica para el campo del monitoreo ecológico.

3.1 Pruebas de Concepto y Metodología experimental

Esta sección describe el marco técnico y los procedimientos utilizados para el desarrollo, entrenamiento y evaluación de los modelos de aprendizaje profundo. Se detallan el entorno de hardware y software, las características del conjunto de datos y la estrategia de

evaluación empleada para garantizar el correcto desarrollo, precisión y validez de los resultados obtenidos.

3.1.1 Entorno Experimental

Para el desarrollo de este proyecto, se utilizó un entorno de computación en la nube a través de Google Collaboratory, aprovechando su capacidad de procesamiento mediante Unidades de Procesamiento Gráfico (GPU), específicamente el modelo del tipo NVIDIA T4, las cuales fueron fundamentales para acelerar los tiempos de entrenamiento de los modelos. El desarrollo se realizó en lenguaje Python 3, utilizando librerías especializadas en aprendizaje profundo como TensorFlow y Keras para la construcción y entrenamiento de los modelos, Scikit-learn para la evaluación de métricas detallada, como el reporte de clasificación (precision, recall, F1-score), también se usó Matplotlib junto a Seaborn para la visualización de datos incluyendo las curvas de aprendizaje y las matrices de confusión y por último se usó Roboflow para la descarga y gestión programática del conjunto de datos.

3.1.2 Conjunto de Datos

El estudio se fundamentó en el conjunto de datos de acceso público denominado "Coral Reef Bleach Detection", disponible en la plataforma Roboflow Universe. Este dataset consta de imágenes submarinas a color, etiquetadas en dos categorías mutuamente excluyentes:

- **Sano:** Corales que presentan su pigmentación y estado natural.
- **Blanqueado:** Corales que han perdido sus algas simbióticas (zooxantelas) y muestran una coloración pálida o blanca.

Para los experimentos principales (CNN y Transfer Learning), se utilizó la división de datos predefinida del dataset, la cual garantiza una separación estandarizada de los datos:

- Conjunto de Entrenamiento: 1,696 imágenes

- Conjunto de Validación: 364 imágenes
- Conjunto de Prueba: 364 imágenes

Todas las imágenes fueron estandarizadas a un tamaño de 224×224 píxeles y sus valores de píxeles fueron normalizados para asegurar la estabilidad y eficiencia del entrenamiento.

3.1.3 Procesamiento y Preparación de Datos

Antes de alimentar los modelos, se aplicó una serie de pasos de preprocesamiento comunes para estandarizar los datos:

- Redimensionamiento de Imágenes: Todas las imágenes, independientemente de su resolución original, fueron redimensionadas a un tamaño uniforme de 224×224 píxeles. Este paso es mandatorio para asegurar que la capa de entrada de las redes neuronales reciba tensores de dimensiones consistentes.
- Normalización de Píxeles: Los valores de los píxeles de las imágenes, originalmente en un rango de [0, 255], fueron escalados a un rango más pequeño. Para los modelos MLP y CNN, se escalaron al rango [0, 1] dividiendo por 255. Para el modelo de Transfer Learning, se utilizó la función preprocess_input específico de la arquitectura VGG16, que ajusta los píxeles al formato y escala con los que fue entrenada originalmente. Este paso es crucial para la estabilidad numérica y para acelerar la convergencia del entrenamiento.

3.1.4 Estrategia de Evaluación

Se evaluaron tres arquitecturas de modelos distintos: un Perceptrón Multicapa (MLP), una Red Neuronal Convolucional (CNN) personalizada y un modelo de Aprendizaje por

Transferencia basado en VGG16. El rendimiento de estos modelos se analizó desde dos perspectivas principales:

- Precisión de Clasificación: Se utilizó la exactitud (accuracy) como métrica principal, complementada por el reporte de clasificación (precisión, recall, F1-score por clase) y la matriz de confusión para un análisis detallado de los errores.
- Eficiencia Computacional: Se comparó la complejidad de los modelos a través de su número total y entrenable de parámetros, un indicador clave de los requerimientos de memoria y capacidad de cómputo para el despliegue.

3.2 Análisis de Resultados

En esta sección se evalúa el rendimiento de cada modelo exclusivamente en función de su capacidad para clasificar correctamente las imágenes.

3.2.1 Modelo Base: Perceptrón Multicapa (MLP)

El MLP fue implementado como un modelo de referencia para establecer una línea base de rendimiento, representando el enfoque más simple de una red neuronal para un problema de clasificación.

- Preprocesamiento y Arquitectura: La característica definitoria de este modelo fue el aplanamiento (flattening) de cada imagen de 224×224×3 en un único y masivo vector de 150,528 características de entrada. Esta transformación, si bien permite que un MLP procese los datos, destruye toda la información espacial (la relación de un píxel con sus vecinos), que es fundamental en la visión por computadora. La arquitectura consistió en dos capas ocultas (Dense) de 128 y 64 neuronas (con activación ReLU) y una capa de salida sigmoide para la clasificación binaria.

- Análisis Comparativo de Optimizadores: Para asegurar que el modelo base fuera lo más competitivo posible, se realizó un experimento para evaluar el rendimiento de tres algoritmos de optimización diferentes: Adam, RMSprop y SGD. La Tabla 3 resume los resultados obtenidos en el conjunto de prueba.

Tabla 3

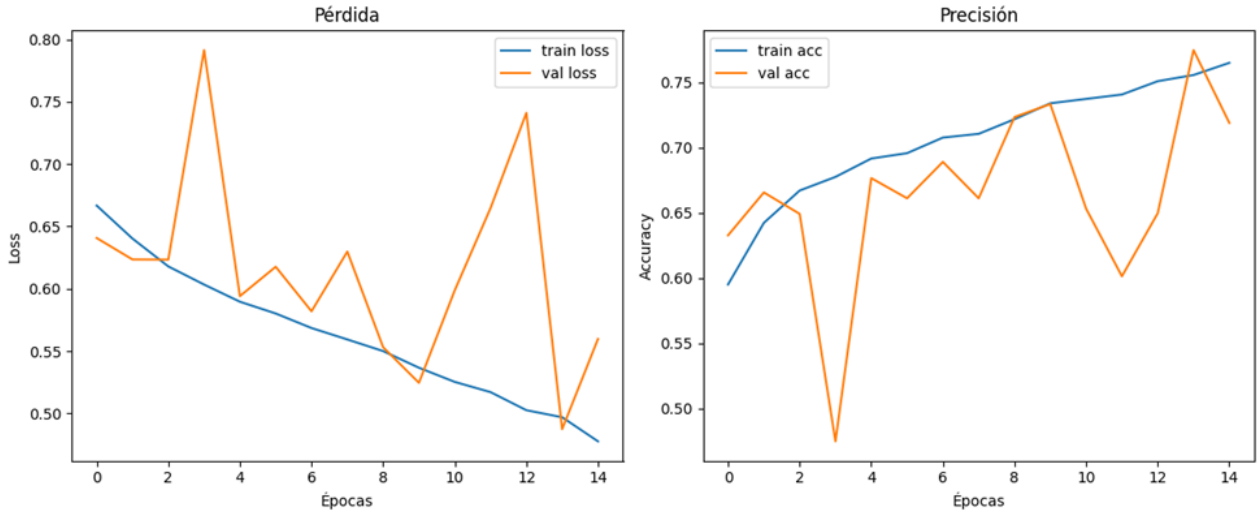
Comparación de la Precisión de Optimizadores para el Modelo MLP.

Optimizador	Precisión
Adam	70.27%
RMSprop	67.57%
SGD	74.00%

- **Análisis de Rendimiento:** El optimizador SGD (Descenso de Gradiente Estocástico) proporcionó el mejor rendimiento, alcanzando un 74.0% de precisión. Aunque es el más simple de los tres, a menudo funciona bien en problemas donde el paisaje de la función de pérdida no es excesivamente complejo. Este resultado fue seleccionado como la línea base oficial para esta arquitectura. Las curvas de aprendizaje Figura 1 de este modelo son reveladoras: la precisión de validación se estanca rápidamente tras las primeras épocas, mientras que la precisión de entrenamiento continúa ascendiendo, lo cual es una señal de sobreajuste y de la incapacidad del modelo para generalizar.

Figura 1

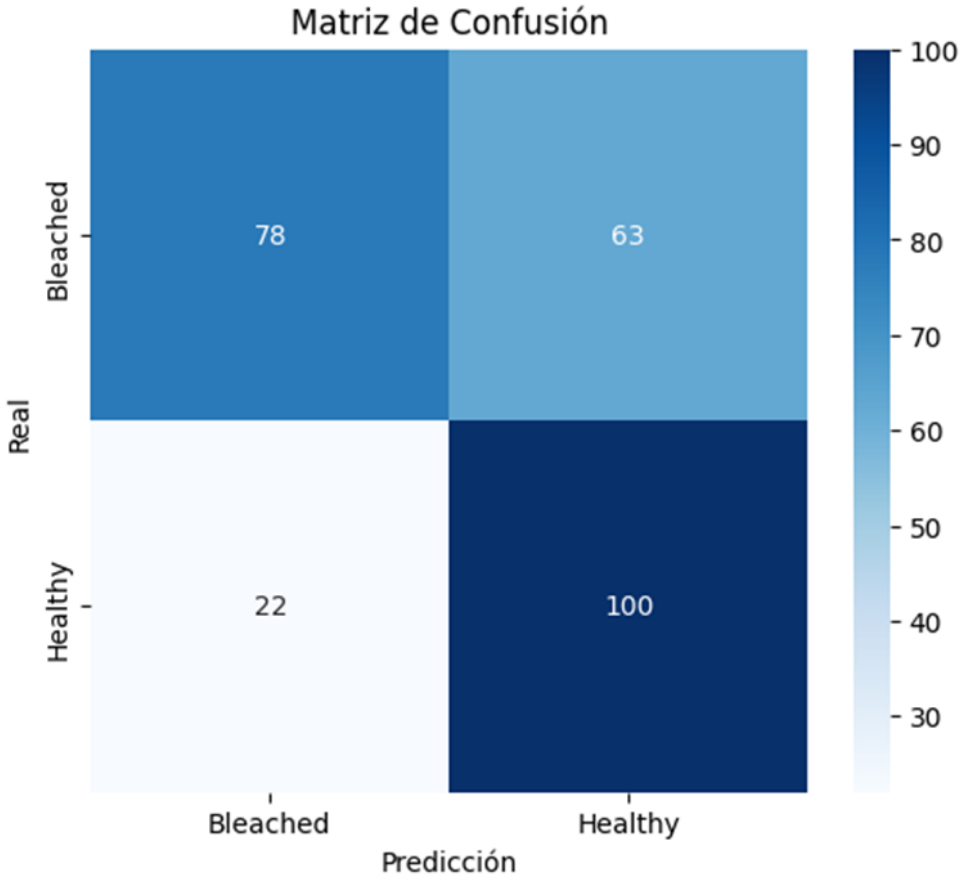
Curvas de Entrenamiento y Validación del Modelo MLP.



La matriz de confusión Figura 2 ofrece un desglose de los errores. Se observa que el modelo tiene una marcada dificultad para identificar correctamente los corales blanqueados, clasificando un número significativo de ellos como sanos (falsos negativos). Este tipo de error es particularmente grave en un contexto de monitoreo ecológico.

Figura 2

Matriz de Confusión del Modelo MLP (Optimizador SGD).



3.2.2 Modelo Aprendizaje por Transferencia (VGG16)

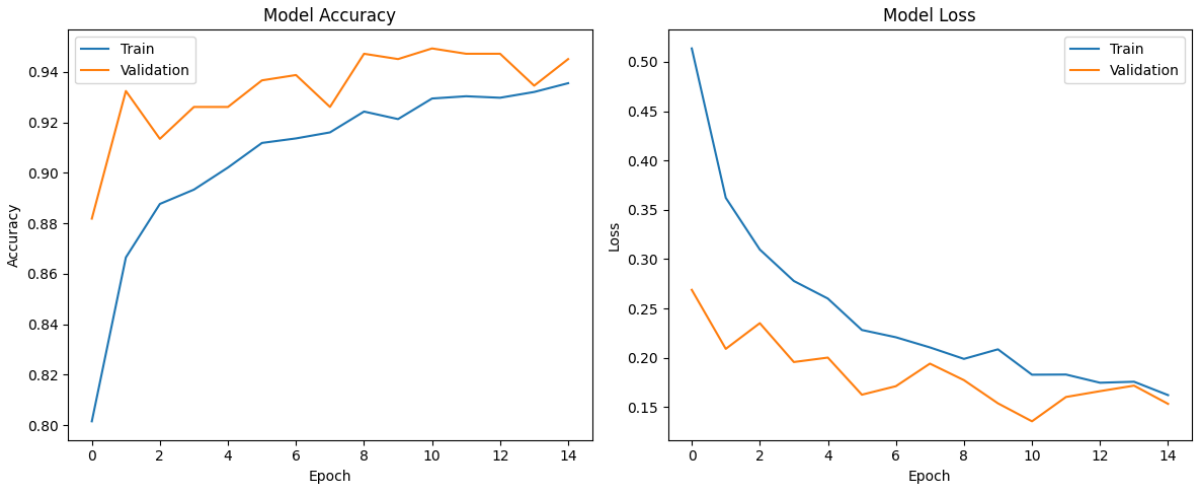
Este enfoque buscó aprovechar el conocimiento de una red pre-entrenada para mejorar el rendimiento.

- Preprocesamiento y Estrategia: La estrategia empleada fue la de extracción de características (feature extraction). Se instanció el modelo VGG16 con sus pesos pre-entrenados en ImageNet y se congelaron todas sus capas convolucionales base. De esta forma, se utilizó la red como un extractor de características genéricas. Sobre esta base congelada, se construyó y entrenó una nueva cabeza de clasificación personalizada y ligera.

- Análisis de Rendimiento: Este enfoque alcanzó una excelente precisión del 94.0%. Este resultado, aunque marginalmente inferior al de la CNN personalizada, es extraordinariamente alto, especialmente considerando la eficiencia del proceso de entrenamiento (fine-tuning). Las curvas de aprendizaje Figura 3 muestran una convergencia ideal: rápida, estable y sin signos de sobreajuste, validando la potencia de esta estrategia.

Figura 3

Curvas de Entrenamiento y Validación del Modelo VGG16.



La matriz de confusión Figura 4 y el reporte de clasificación Tabla 4 confirman su excelente rendimiento, con un balance casi perfecto en la identificación de ambas clases.

Figura 4

Matriz de Confusión del Modelo de Aprendizaje por Transferencia (VGG16).

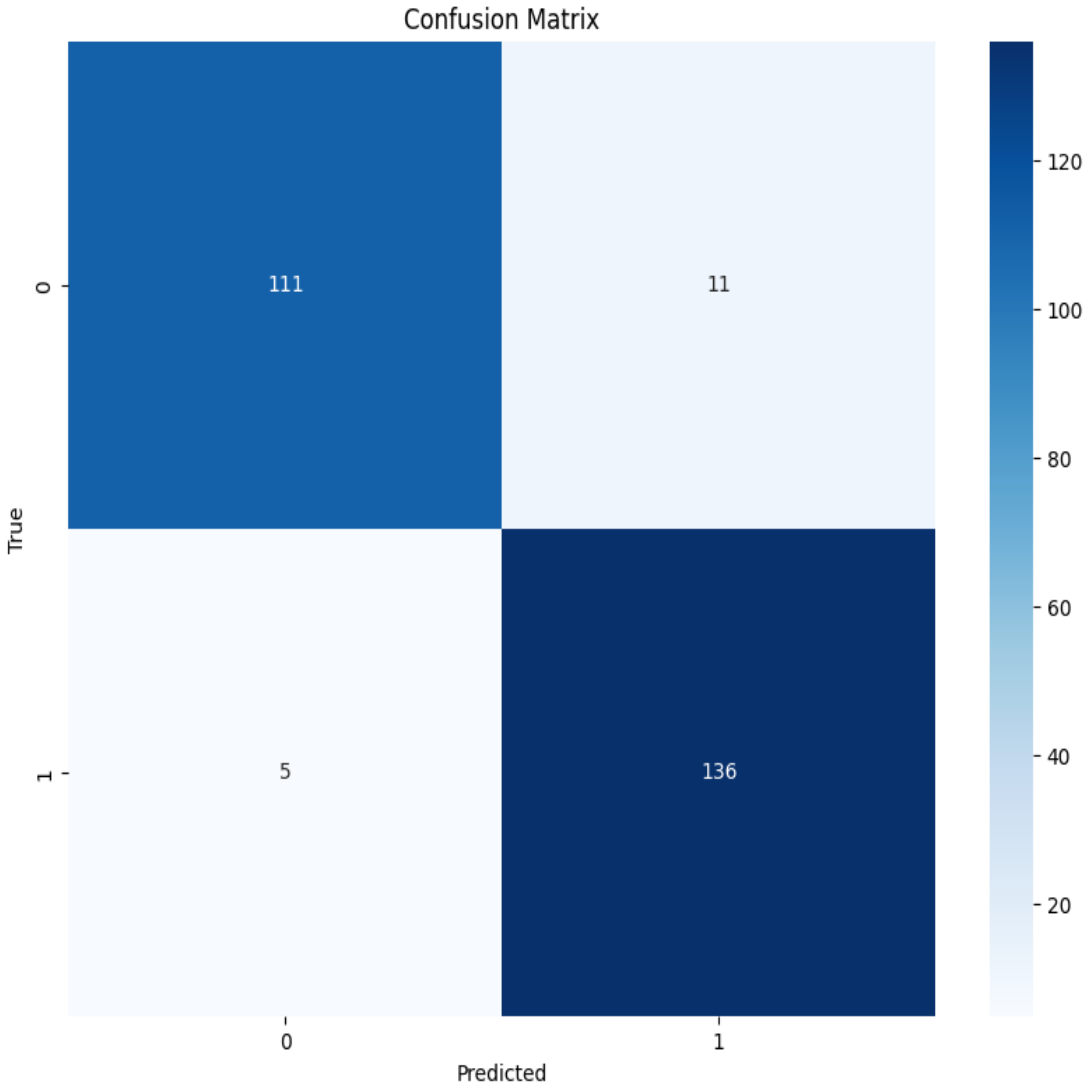


Tabla 4

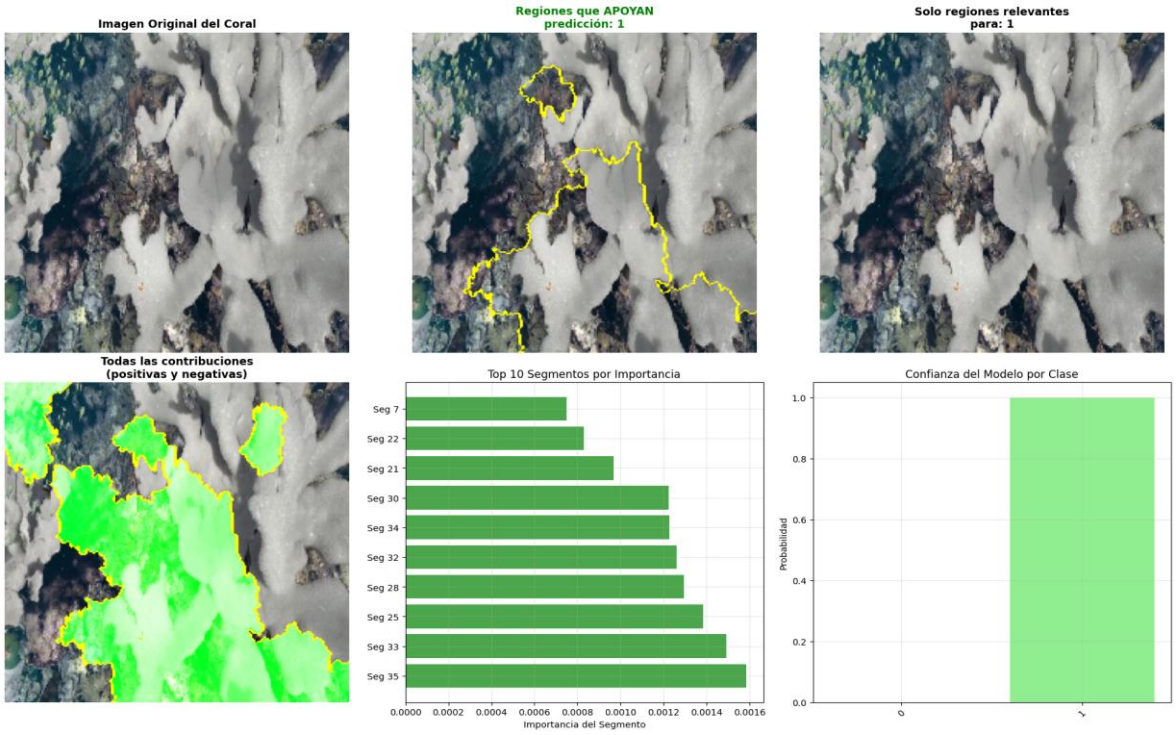
Métricas de Rendimiento del Modelo de Transferencia de Aprendizaje (VGG16).

Clase	Precisión	Recall	F1-Score
Blanqueado	0.96	0.91	0.93
Sano	0.93	0.96	0.94
Promedio Ponderado	0.94	0.94	0.94

Con el fin de entender las predicciones realizadas por el modelo de VGG16 con CNN, se utilizó LIME como técnica de interpretabilidad. Como se aprecia en la Figura 5, el análisis para la imagen del coral examina las características visuales clave que llevaron al modelo a clasificarlo como "Blanqueado" (Clase 1) con una confianza extremadamente alta, cercana al 100%. Las visualizaciones de LIME revelan que el modelo se basa principalmente en las regiones de color claro y texturas irregulares, características del coral blanqueado.

Figura 5

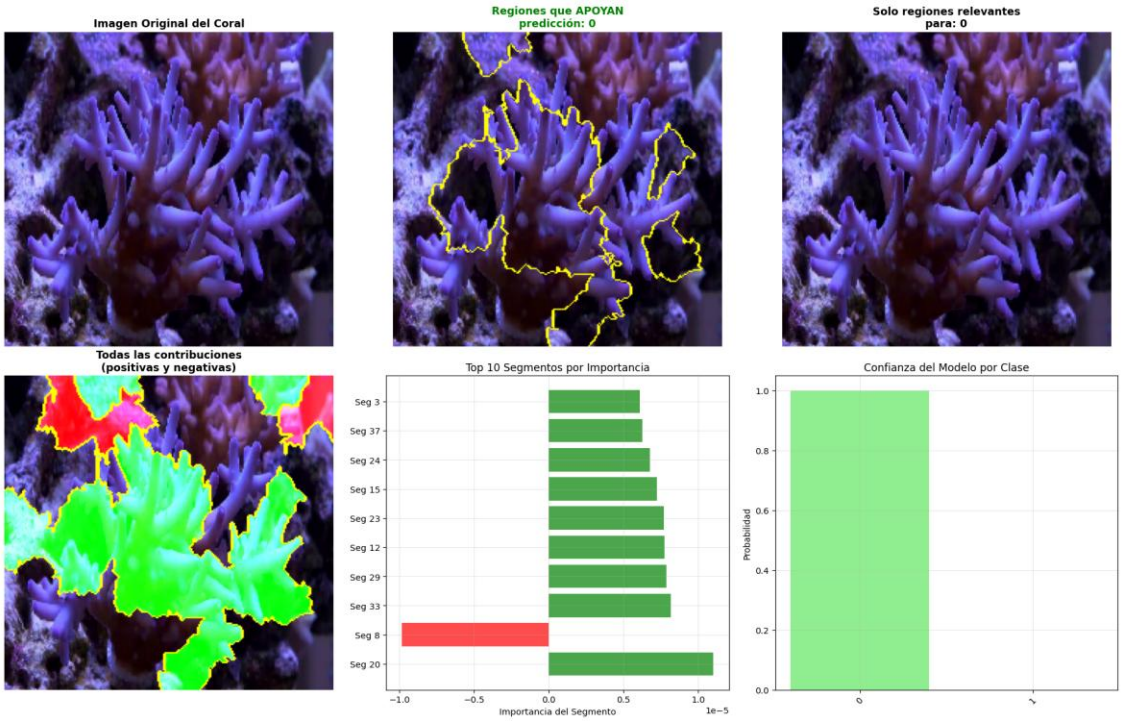
Interpretación LIME del modelo VGG16 para corales blanqueados.



La Figura 6 ilustra el análisis de interpretabilidad LIME para una imagen de coral, revelando cómo el modelo alcanza su clasificación de "Sano" (Clase 0) con una confianza del 100%. Las visualizaciones indican que el modelo se concentra en las características distintivas del coral sano: el color púrpura vibrante y la estructura ramificada. Estas regiones, cruciales para la predicción, están claramente delineadas en amarillo en el panel "Regiones que APOYAN predicción: 0". Adicionalmente, el panel "Todas las contribuciones (positivas y negativas)" destaca en color verde las áreas cuya presencia refuerza positivamente la clasificación de coral sano. Por el contrario, las zonas resaltadas en rojo, como el Segmento 8 en el gráfico de importancia, representan características que, de ser más prominentes, ejercerían una influencia negativa, desviando la predicción hacia la clase opuesta ("Blanqueado"). Este análisis confirma que el modelo discrimina la salud del coral basándose en los atributos visuales esperados de un coral que se encuentra en buenas condiciones.

Figura 6

Interpretación LIME del modelo VGG 16 para corales sanos.



Con el fin de complementar y validar la interpretabilidad del modelo, se decidió también aplicar la técnica de interpretabilidad Grad-CAM para entender las regiones relevantes que el modelo enfoca su atención para realizar la predicción final. Para aplicar Grad-CAM seleccionamos una imagen sin procesar del dataset en la cual se observa a un buzo nadando junto a un coral blanqueado como se puede apreciar en la Figura 7.

Figura 7

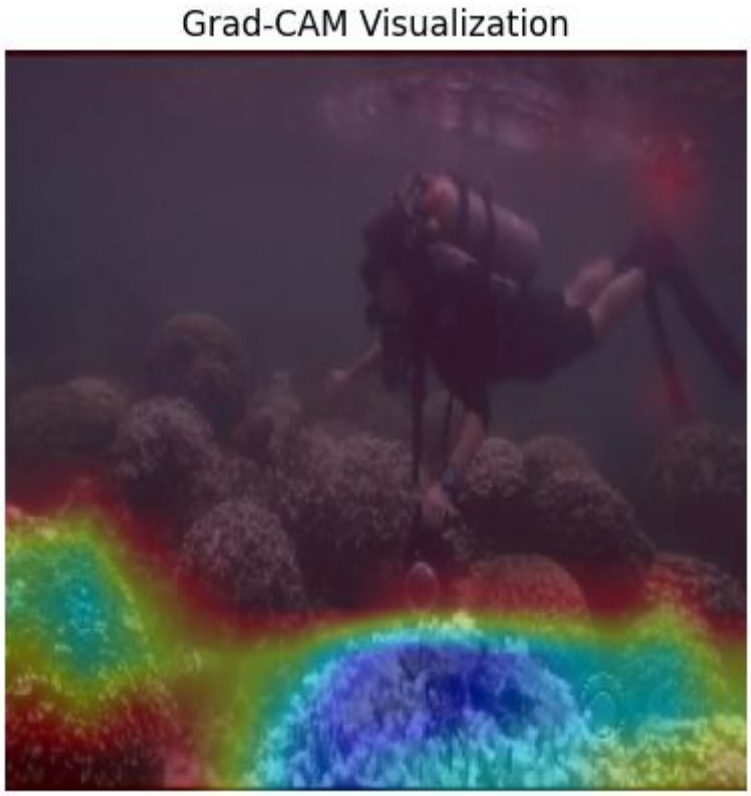
Imagen original de un coral blanqueado.

Posterior a la aplicación de Grad-CAM, obtuvimos como resultado la Figura 8, en la cual podemos observar que las zonas más activas (en colores rojos y amarillos, y especialmente el azul brillante en el centro-inferior) se concentran sobre los bordes del coral. Esto sugiere que, para la predicción de salud de coral, el modelo se basó principalmente en las características visuales de los corales mismos, como sus bordes y formas.

El buzo, aunque presente en la imagen, parece tener menor importancia para la predicción del modelo, ya que las zonas que lo rodean no muestran una alta activación en el mapa de calor comparadas con los corales.

Figura 8

Resultado de Grad-CAM con imagen de un coral blanqueado.



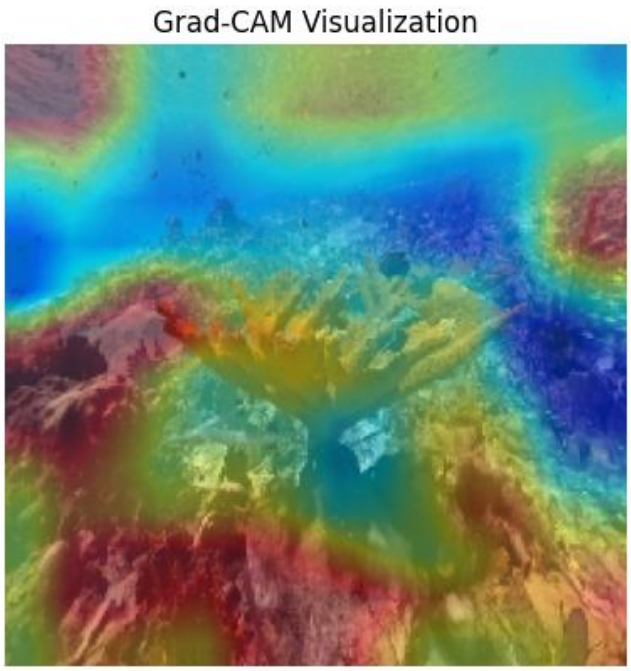
Realizando el mismo experimento con la imagen de un coral sano presente en la Figura 9, el resultado final de Grad-CAM en la Figura 10 nos indica que, en esta imagen, los colores más cálidos (amarillos y rojos) se concentran de manera muy clara sobre la estructura del coral central. Esta observación es particularmente visible en las puntas y la superficie de este coral. Las áreas circundantes, tanto el fondo marino como la superficie del agua, muestran colores más fríos (azules), lo cual indica que aquellas áreas no contribuyeron a la predicción final.

Figura 9

Imagen original de un coral sano.

Figura 10

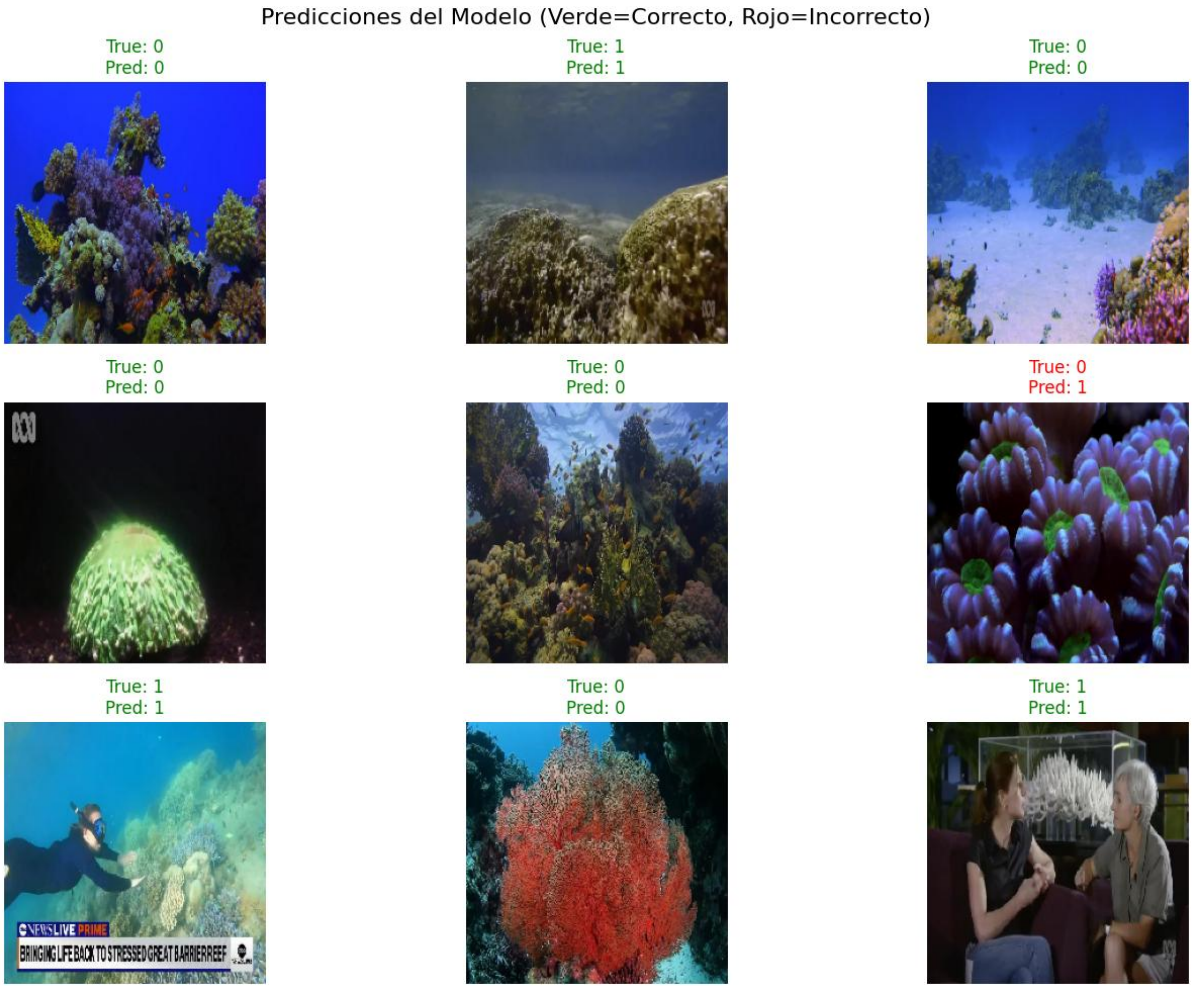
Resultado de Grad-CAM con imagen de un coral blanqueado.



La Figura 11 nos indica una compilación de las predicciones realizadas por el modelo VGG16 con transfer learning con 9 imágenes del dataset en el cual podemos apreciar que las predicciones fueron en su mayoría correctas para detectar corales sanos y blanqueados, con la excepción de un falso positivo.

Figura 11

Predicciones del modelo VGG16.



3.2.3 Modelo de Red Neuronal Convolucional (CNN) Personalizada

Se diseñó y entrenó una Red Neuronal Convolucional (CNN) desde cero para superar las limitaciones espaciales del MLP.

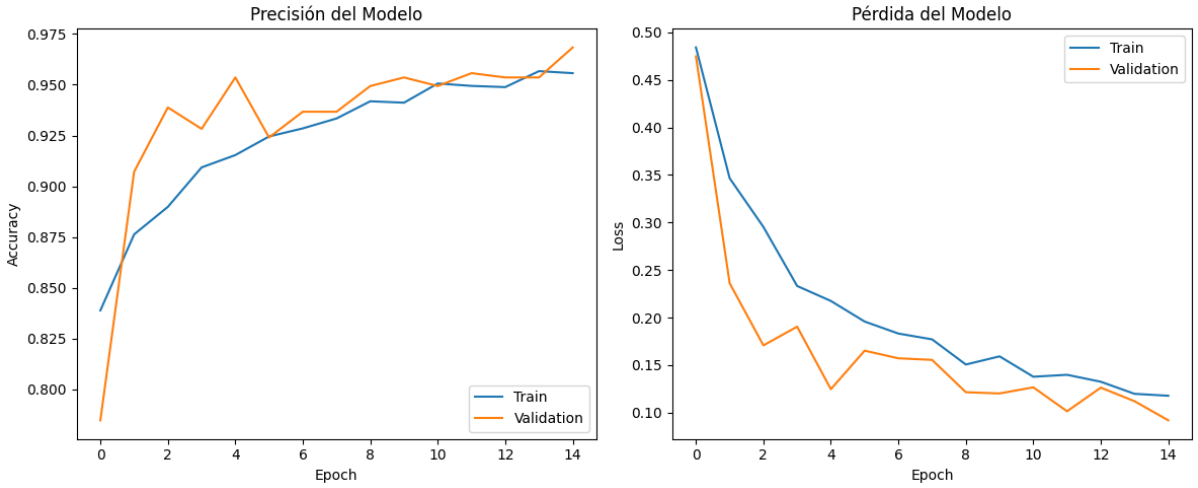
- Preprocesamiento y Arquitectura: A diferencia del MLP, este modelo procesa las imágenes en su formato 2D. Un componente clave de su preprocesamiento

fue el uso de aumento de datos **(**ImageDataGenerator), que aplica transformaciones aleatorias a las imágenes de entrenamiento (rotaciones, volteos, zoom). Esta técnica expande artificialmente la diversidad del conjunto de datos, forzando al modelo a aprender características más robustas y reduciendo significativamente el sobreajuste. La arquitectura se diseñó con tres bloques convolucionales progresivos (con 32, 64 y 128 filtros respectivamente) y capas de MaxPooling2D.

- Análisis de Rendimiento: La CNN personalizada alcanzó una precisión del 97.0%, la más alta de todos los experimentos. Este resultado demuestra el poder de las arquitecturas convolucionales y sugiere que, para este problema específico, un modelo especializado y entrenado desde cero es capaz de aprender un conjunto de características visuales más discriminativas que las de un modelo de propósito general. Sus curvas de aprendizaje Figura 12 muestran una convergencia saludable, donde el aumento de datos fue clave para mantener la curva de validación cerca de la de entrenamiento.

Figura 12

Curvas de Entrenamiento y Validación de la CNN Personalizada.



El modelo demostró ser extremadamente fiable, como se observa en su matriz de confusión Figura 13 y en sus métricas perfectamente balanceadas Tabla 5.

Figura 13

Matriz de Confusión de la CNN Personalizada.

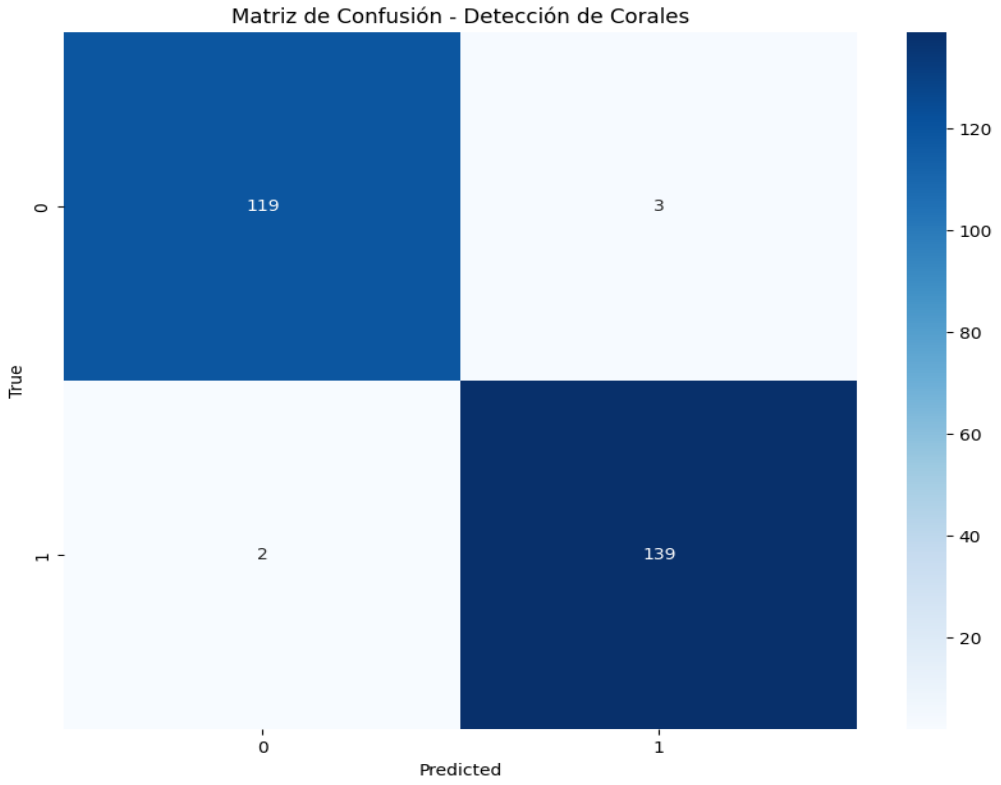


Tabla 5

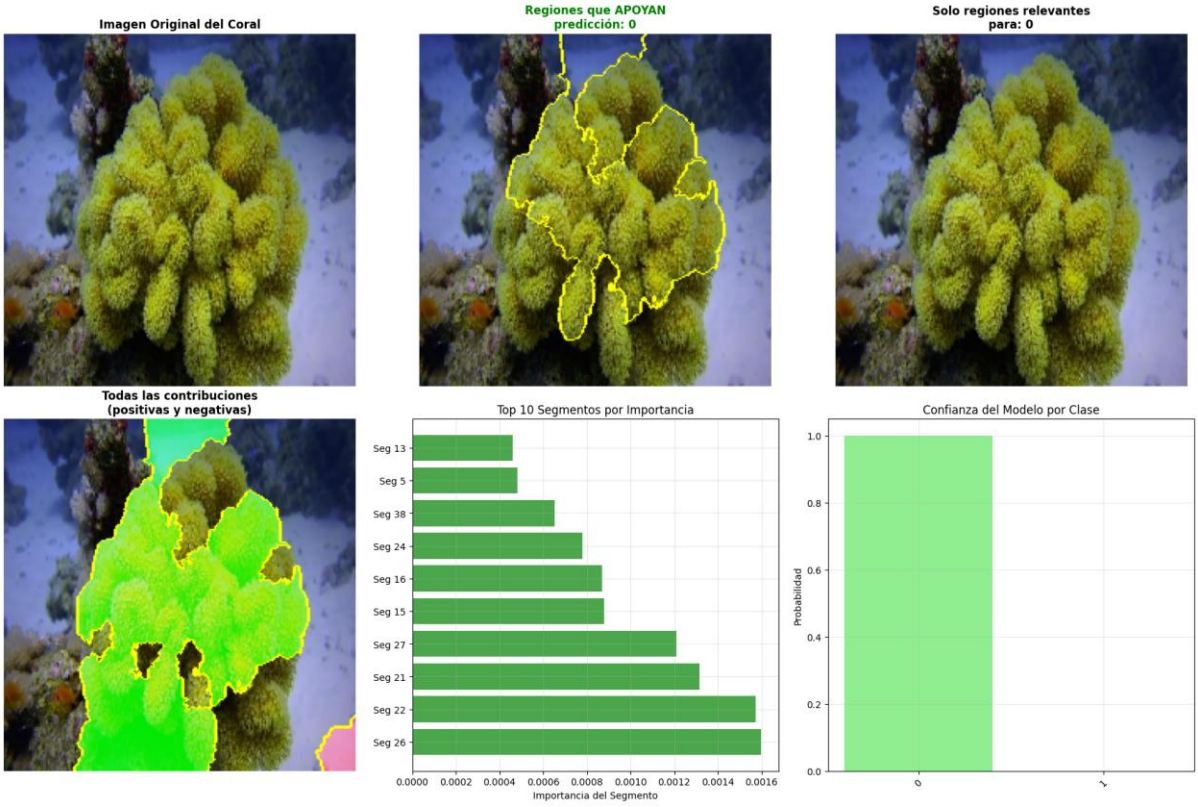
Métricas de Rendimiento de la CNN *Personalizada.*

Clase	Precisión	Recall	F1-Score
Blanqueado	0.98	0.98	0.98
Sano	0.98	0.98	0.98
Promedio Ponderado	0.98	0.98	0.98

La Figura 14 muestra un análisis LIME para una imagen de coral que el modelo CNN clasifica como "Sano" (Clase 0) con una confianza del 100%. Este resultado se atribuye a la identificación de características visuales claves asociadas con la salud del coral. Las visualizaciones de LIME revelan que el modelo se enfoca predominantemente en el color amarillo vibrante y la morfología densa del coral, como se evidencia en el panel "Regiones que APOYAN predicción: 0", donde estas áreas están claramente delimitadas en amarillo. El panel "Todas las contribuciones (positivas y negativas)" refuerza esta interpretación, destacando en verde las porciones del coral que contribuyen positivamente a la predicción de "Sano", mientras que las zonas en rojo corresponden a elementos del fondo o áreas con menor relevancia para esta clasificación. Con este análisis, podemos confirmar que la apariencia cromática y estructural de este tipo de coral es un indicador robusto de su estado de salud.

Figura 14

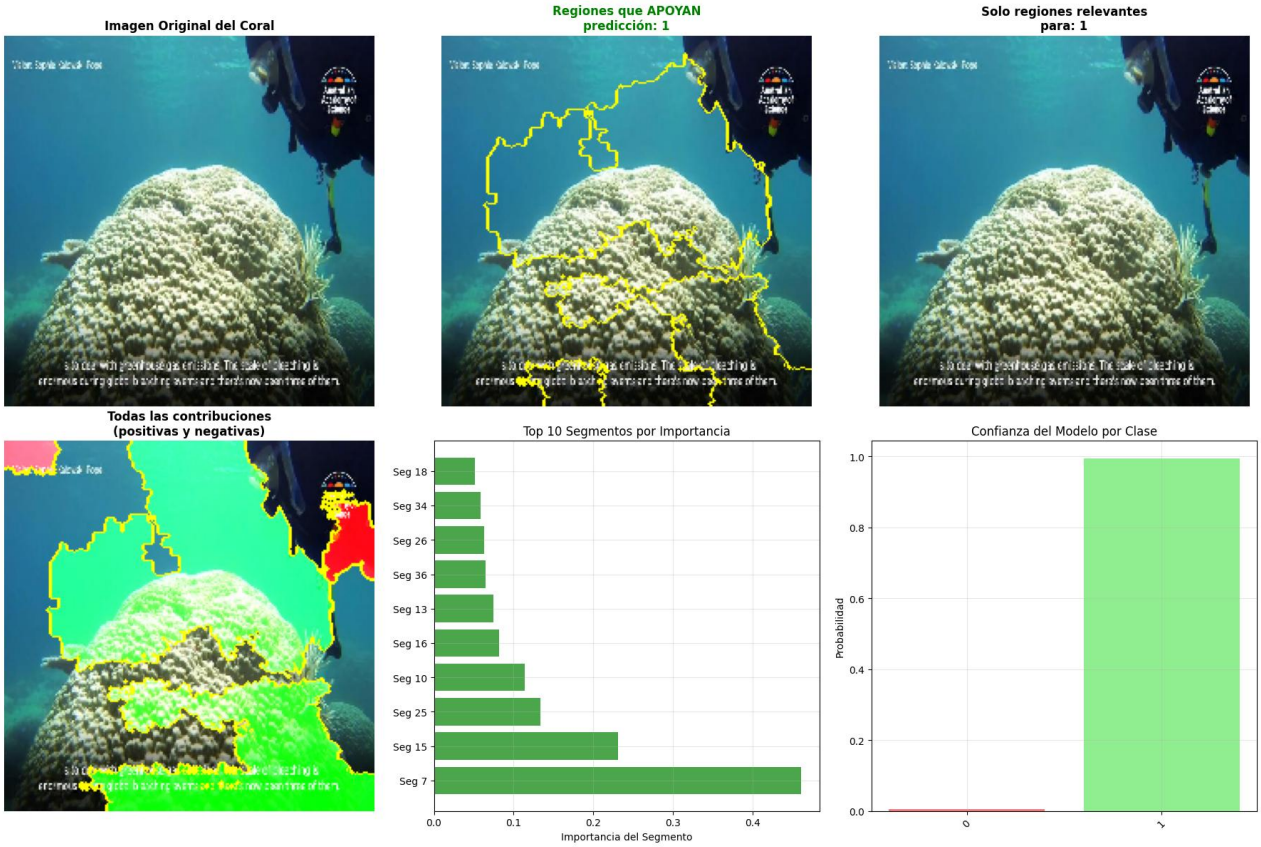
Interpretación LIME del modelo CNN personalizado para corales sanos.



La Figura 15 nos muestra el análisis LIME de una imagen con un coral blanqueado. Aquí, el modelo ha clasificado el coral como "Blanqueado" (Clase 1) con alta confianza. Lo que LIME nos enseña es que el modelo se fijó casi solo en el color claro y omite las sombras propias de la imagen. Si analizamos los paneles "Regiones que APOYAN predicción: 1" y "Solo regiones relevantes para: 1", se puede observar que las zonas del coral son las que marcan la diferencia para esta clasificación. Es interesante señalar cómo el modelo resta importancia a otros elementos de la imagen, como el buzo, lo que significa que el modelo sabe concentrarse solo en el coral para decidir si está blanqueado o no. De hecho, una parte del cuerpo del buzo según LIME es una contribución negativa para la predicción final.

Figura 15

Interpretación LIME del modelo CNN personalizado para corales blanqueados.



Para contrastar los resultados de LIME en la CNN personalizada, se aplicó la técnica Grad-CAM en una imagen de un coral sano Figura 16.

Figura 16

Imagen de un coral sano.

La Figura 17 es el resultado de aplicar Grad-CAM a la imagen, los colores cálidos como el amarillo y el rojo son más visibles en la parte superior del coral que tiene ramificaciones. Estas áreas, especialmente la parte superior y central del coral más grande, indican las regiones de la imagen que la red neuronal consideró más relevantes o importantes para su predicción. Colores fríos como el verde y el azul, visibles en el fondo arenoso o las partes más difusas del agua, son las que la red consideró menos relevantes para su predicción.

Figura 17

Resultado de Grad-CAM con imagen de un coral sano (CNN personalizada).

Para el análisis de Grad-CAM de un coral en mal estado, se eligió la imagen de la Figura 18 Esta imagen es ideal porque presenta un coral con una zona muerta y la otra mitad blanqueada, además de la presencia de un buzo. Las zonas muertas o extremadamente degradadas del coral tienen la particularidad de acumular algas marinas (Roth et al., 2018), por lo cual el coral se torna marrón como se puede apreciar en la imagen.

Figura 18

Imagen de un coral blanqueado con alta degradación.

El resultado de aplicar Grad-CAM se puede evidenciar en la Figura 19, vale la pena mencionar que la CNN personalizada se enfocó en esta ocasión en el área del coral muerto con algas debido a que los colores cálidos envuelven esta zona. Sin embargo, en las partes del coral blanqueado el modelo no tomó en consideración esas zonas para hacer la predicción, lo que sugiere una limitación en cómo la CNN personalizada interpretó las diferentes condiciones del coral.

Figura 19

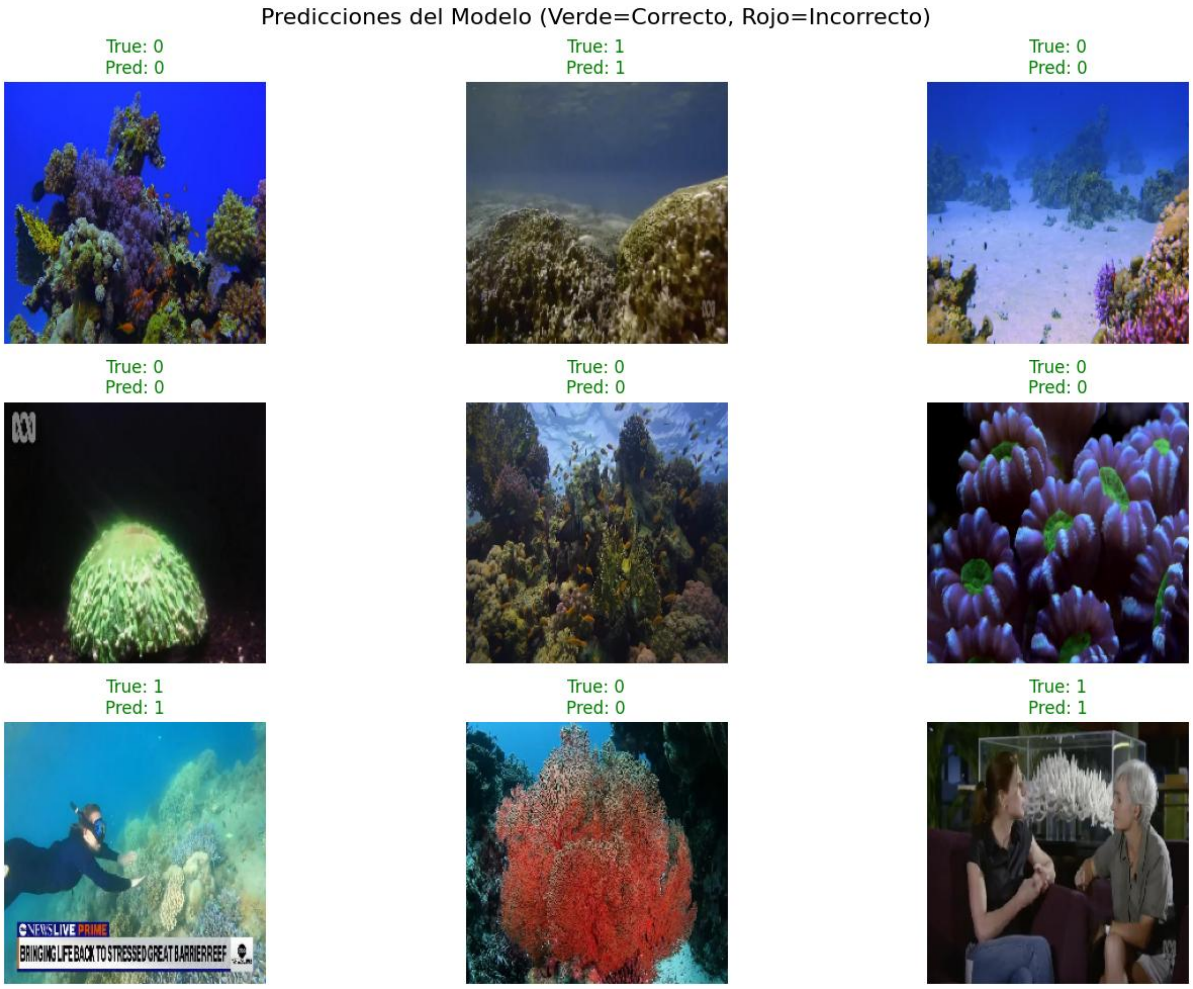
Resultado de Grad-CAM con imagen de un coral blanqueado con alta degradación (CNN personalizada).

Esta limitación podría resultar en falsas predicciones por parte del modelo en aquellos casos que un coral sano contenga colores similares a los de las algas que se impregnan en corales muertos. Para que la red neuronal pueda diferenciar de manera robusta entre corales muertos y blanqueados, se recomienda un reentrenamiento con un conjunto de datos que incluya imágenes de ambas categorías; no obstante, esta mejora se encuentra fuera del alcance del presente proyecto.

La Figura 20 indica una compilación de las predicciones realizadas por el modelo CNN personalizado con 9 imágenes del dataset, en el cual podemos apreciar que las predicciones fueron correctas para detectar corales sanos y blanqueados.

Figura 20

Predicciones del modelo CNN personalizado.



3.3 Discusión Comparativa Final

Para realizar una selección informada del mejor modelo, se debe analizar no sólo la precisión, sino también la eficiencia y la viabilidad práctica. La Tabla 6 compara la precisión con la complejidad de cada modelo (número de parámetros).

- Complejidad del Modelo (Número de Parámetros)**:** El tamaño de un modelo, medido por su número total de parámetros, se relaciona directamente con los requerimientos de memoria y capacidad de cómputo. Se analizaron los resúmenes (summary) de cada modelo.

Tabla 6

Comparación de la Complejidad y Precisión de los Modelos.

Modelo	Precisión	Parámetros Entrenables	Parámetros Totales
Perceptrón Multicapa (MLP)	74.00%	~19,276,000	~19,276,000
CNN Personalizada	97.00%	~1,841,000	~1,841,000
Transfer Learning (VGG16)	94.00%	~66,000	~14,778,000

- A pesar de que la CNN personalizada obtuvo la precisión marginalmente más alta, se ha seleccionado el modelo de Aprendizaje por Transferencia (VGG16) como la solución "mejor" y más robusta del proyecto. Esta decisión se basa en una evaluación holística:

    - **Fiabilidad y Rapidez de Desarrollo:** El Aprendizaje por Transferencia es una metodología estandarizada y probada. Permite obtener resultados de alto rendimiento (94%) de manera rápida y fiable, minimizando el tiempo y el riesgo asociados con el diseño y la afinación de una arquitectura desde cero. En un contexto práctico de investigación, obtener un 94% de precisión de forma eficiente es más valioso que un 97% que requiere una experimentación más extensa.

    - **Robustez y Generalización**: VGG16 ha aprendido un repertorio inmenso y robusto de características visuales. Existe la hipótesis de que estas características más generales podrían ofrecer una mejor

capacidad de generalización ante imágenes de corales de nuevas regiones o en condiciones no vistas durante el entrenamiento.

- **Principio de "Suficientemente Bueno":** Una precisión del 94% es un resultado excepcional que ya cumple con creces los requisitos para una herramienta de monitoreo altamente efectiva. El modelo VGG16 representa el punto óptimo entre alto rendimiento y viabilidad práctica.

3.4 Desarrollo de Interfaz de Usuario

Una vez que se definió y validó el modelo de clasificación óptimo, se procedió al desarrollo de una prueba de concepto de una aplicación web interactiva. Para la aplicación se utilizó el framework de Streamlit que permite a los usuarios cargar imágenes de coral en formatos comunes (PNG, JPG, JPEG) a través de una interfaz de carga dedicada. Tras la subida, la imagen es sometida a un preprocesamiento automatizado que incluye la conversión a formato RGB, redimensionamiento a 224x224 píxeles y normalización de sus valores, asegurando la compatibilidad con nuestro mejor modelo pre-entrenado (final_model_vgg16.h5).

Posteriormente, el modelo infiere el estado del coral, clasificándolo como "Saludable" o "Blanqueado". Los resultados se presentan al usuario de forma clara mediante tarjetas visuales que incluyen la predicción y el nivel de confianza porcentual del modelo como se puede apreciar en las Figuras 21 y 22.

Figura 21

Ejemplo de predicción de coral sano en interfaz de usuario.

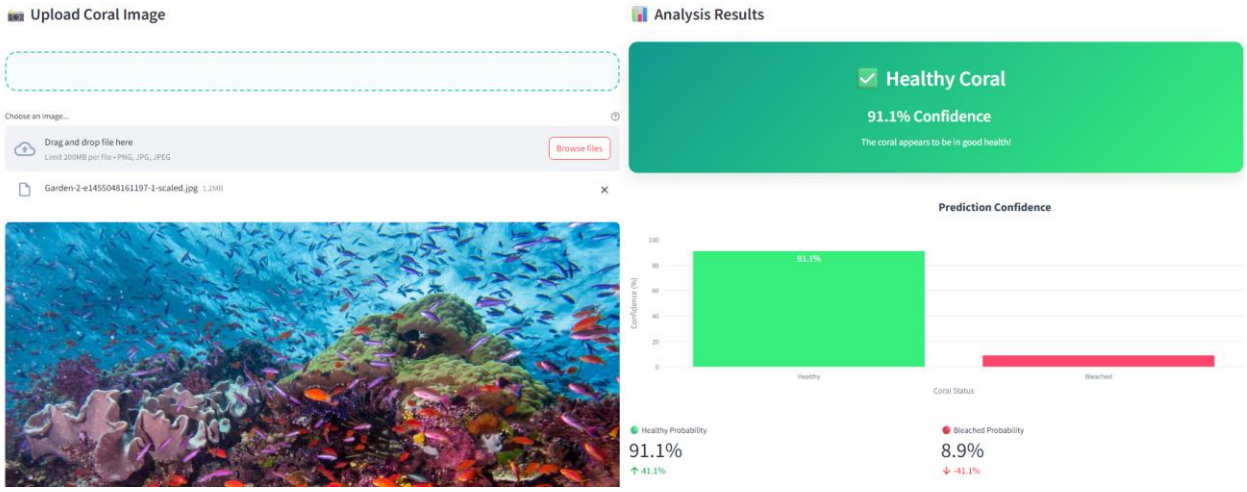
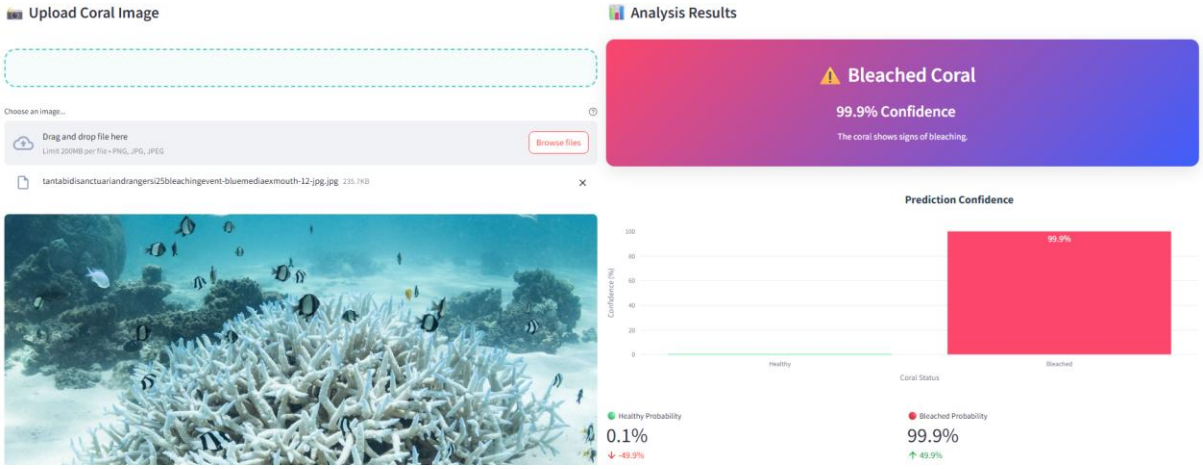


Figura 22

Ejemplo de predicción de coral blanqueado en interfaz de usuario

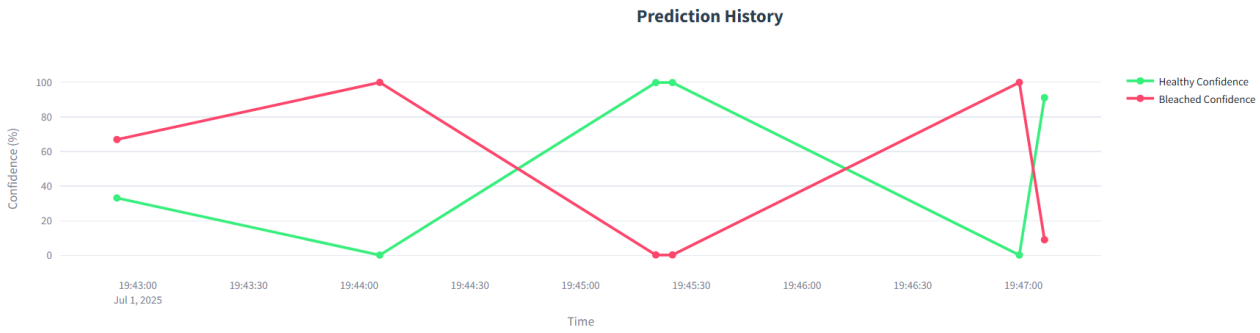


\

Además, la aplicación integra gráficos interactivos generados con Plotly para visualizar la distribución de la confianza entre ambas clases y un historial de predicciones que permite seguir la evolución de los análisis como se puede apreciar en la Figura 23.

Figura 23

Ejemplo de historial de predicciones de coral en aplicación de Streamlit.



CAPITULO 5:

4. CONCLUSIONES Y RECOMENDACIONES

Este capítulo final consolida los hallazgos del proyecto, justifica la selección del modelo óptimo como solución final y propone líneas de trabajo futuro fundamentadas en los resultados y limitaciones del presente estudio.

Validación del Aprendizaje Profundo como Herramienta de Monitoreo: Se ha cumplido exitosamente el objetivo general del proyecto, validando de manera empírica que las técnicas de aprendizaje profundo, y en particular las redes neuronales convolucionales, son herramientas altamente efectivas para la clasificación automatizada de la salud de los corales. Todos los modelos de visión por computadora desarrollados (CNN y VGG16) superaron drásticamente la línea base establecida por el MLP (74%), demostrando su capacidad para extraer patrones complejos de datos visuales.

Selección del Modelo Óptimo basado en un Enfoque Holístico: Se concluye que el modelo de Aprendizaje por Transferencia basado en VGG16 es la solución globalmente óptima para este proyecto. A pesar de que una CNN personalizada alcanzó una precisión marginalmente superior (98% vs 94%), el modelo VGG16 se selecciona como "mejor" por su combinación superior de alto rendimiento, velocidad de implementación, fiabilidad y la robustez de sus características pre-entrenadas. Este enfoque pragmático lo convierte en la opción más recomendable para aplicaciones prácticas y de investigación.

Análisis Comparativo de Arquitecturas: El estudio ha arrojado luz sobre las fortalezas y debilidades de cada arquitectura. La CNN personalizada, con un 98% de precisión, demostró que es posible superar a modelos pre-entrenados en tareas de nicho con un diseño cuidadoso y técnicas como el aumento de datos. Sin embargo, este proceso requiere un mayor grado de experimentación. Por otro lado, el modelo MLP quedó completamente invalidado, no solo por su baja precisión, sino también por su sorprendente ineficiencia en

tamaño, demostrando ser una arquitectura inadecuada para problemas de visión por computadora modernos.

Impacto y Aplicación Práctica del Modelo Seleccionado: El modelo seleccionado, con su 94% de precisión y métricas F1-score balanceadas, constituye una herramienta potente y fiable, lista para ser utilizada como base en sistemas de monitoreo ecológico. Su capacidad para procesar imágenes de forma rápida y objetiva puede ser un gran apoyo para biólogos marinos, gestores de arrecifes y conservacionistas en la lucha contra la degradación de los ecosistemas coralinos.

Basado en las conclusiones y las limitaciones identificadas en este trabajo, se proponen las siguientes recomendaciones y futuras investigaciones:

Despliegue y Validación del Modelo VGG16 en un Entorno Real: El siguiente paso es la implementación y despliegue del modelo VGG16 seleccionado en una aplicación de producción, como la plataforma web presentada en este informe. Esto permitiría que expertos en biología marina validen sus conjuntos de imágenes de campo, obtenidas de diferentes regiones geográficas y bajo distintas condiciones. Este proceso sería crucial para probar su verdadera capacidad de generalización fuera del dataset original.

Optimización del Modelo para Despliegue en Dispositivos de Borde (Edge Computing): Aunque el modelo VGG16 es fiable, su tamaño (~14.7 millones de parámetros) puede ser prohibitivo para dispositivos con recursos limitados. Se recomienda investigar y aplicar técnicas de optimización de modelos, como la poda (pruning), que elimina conexiones neuronales redundantes, y la cuantización (quantization), que reduce la precisión de los pesos (ej. de 32 bits a 8 bits). Esto podría reducir drásticamente el tamaño del modelo y acelerar la inferencia, haciéndolo viable para su despliegue en drones submarinos, cámaras inteligentes o boyas de monitoreo.

Investigación de la Brecha de Rendimiento y Arquitecturas Híbridas: Sería académicamente valioso realizar un análisis más profundo para entender por qué la CNN personalizada superó ligeramente al modelo VGG16. Se podrían utilizar técnicas de visualización e interpretabilidad de modelos (ej. Grad-CAM) para ver qué características visuales activa cada modelo. Este conocimiento podría llevar al desarrollo de arquitecturas híbridas, que combinen capas pre-entrenadas con bloques convolucionales personalizados, buscando unir la robustez de VGG16 con la especialización de la CNN.

Expansión de la Complejidad del Problema: El actual proyecto realiza una clasificación a nivel de imagen. El campo puede expandirse significativamente en las siguientes direcciones:

- **Detección de Objetos:** Implementar modelos como YOLO o SSD para no solo clasificar una imagen, sino para localizar y clasificar múltiples colonias de coral individualmente dentro de un mismo encuadre.
- **Segmentación Semántica:** Utilizar arquitecturas como U-Net para delinear a nivel de píxel las áreas sanas y blanqueadas de cada coral. Esto permitiría obtener una métrica cuantitativa mucho más rica, como el "porcentaje de blanqueamiento" por colonia.
- **Clasificación Multiclase:** Ampliar el número de clases para incluir diferentes especies de coral, diversas enfermedades o las distintas etapas del proceso de blanqueamiento, creando un sistema de monitoreo aún más sofisticado e informativo.

REFERENCIAS BIBLIOGRÁFICAS

Beijbom, O., Edmunds, P. J., Kline, D. I., Mitchell, B. G., & Kriegman, D. (2012). Automated annotation of coral reef survey images. *IEEE Conference on Computer Vision and Pattern Recognition*, 1170–1177. https://doi.org/10.1109/CVPR.2012.6247798

Beijbom, O., Edmunds, P. J., Roelfsema, C., Smith, J., Kline, D. I., Neal, B., Dunlap, M. J., & Kriegman, D. (2015). Towards automated annotation of benthic survey images: Variability of human experts and operational modes of automation. *PLOS ONE, 10*(7), e0130312. https://doi.org/10.1371/journal.pone.0130312

Bryson, M., Ferrari, R., Figueira, W., Pizarro, O., & Williams, S. B. (2016). Characterization of measurement errors using structure-from-motion and photogrammetry to measure marine habitat structural complexity. *Ecology and Evolution, 7*(15), 5669–5681. https://doi.org/10.1002/ece3.3127

Chollet, F. (2015). *Keras* [Software]. https://keras.io/

González-Rivero, M., Beijbom, O., Rodriguez-Ramirez, A., & Hoegh-Guldberg, O. (2020). Monitoring of coral reefs using artificial intelligence: A feasible and scalable approach. *Frontiers in Marine Science, 7*, 50. https://doi.org/10.3390/rs12030489

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hoegh-Guldberg, O., Mumby, P. J., Hooten, A. J., Steneck, R. S., Greenfield, P., Gomez, E., Harvell, C. D., Sale, P. F., Edwards, A. J., Caldeira, K., Knowlton, N., Eakin, C. M., Iglesias-Prieto, R., Muthiga, N., Bradbury, R. H., Dubi, A., & Hatziolos, M. E. (2007).

Coral reefs under rapid climate change and ocean acidification. *Science, 318*(5857), 1737–1742. https://doi.org/10.1126/science.1152509

Hughes, T. P., Barnes, M. L., Bellwood, D. R., Cinner, J. E., Cumming, G. S., Jackson, J. B., Kleypas, J., van de Leemput, I. A., Lough, J. M., Morrison, T. H., Palumbi, S. R., van Nes, E. H., Scheffer, M., Mumby, P. J., Steneck, R. S., Watson, J., Wilson, S. K., Brodie, J., Brown, I., & Roughgarden, J. (2017). Coral reefs in the Anthropocene. *Nature, 546*(7656), 82–90. https://doi.org/10.1038/nature22901

Spampinato, C., Beauxis-Aussalet, E., Palazzo, S. *et al.* A rule-based event detection system for real-life underwater domain. *Machine Vision and Applications* **25**, 99–117 (2014). https://doi.org/10.1007/s00138-013-0509-x

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint*. https://doi.org/10.48550/arXiv.1412.6980

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. *Proceedings of the IEEE International Conference on Computer Vision*, 2980–2988. https://doi.org/10.1109/ICCV.2017.324

Lough, J. M., Anderson, K. D., & Hughes, T. P. (2018). Increasing thermal stress for tropical coral reefs: 1871–2017. *Scientific Reports, 8*(1), 6079. https://doi.org/10.1038/s41598-018-24530-9

Loya, Y., Sakai, K., Yamazato, K., Nakano, Y., Sambali, H., & Van Woesik, R. (2001). Coral bleaching: The winners and the losers. Ecology Letters, 4(2), 122–131. https://doi.org/10.1046/j.1461-0248.2001.00203.x

Massot-Campos, M., & Oliver-Codina, G. (2015). Optical Sensors and Methods for Underwater 3D Reconstruction. *Sensors (Basel, Switzerland)*, *15*(12), 31525–31557. https://doi.org/10.3390/s151229864

Moberg, F., & Folke, C. (1999). Ecological goods and services of coral reef ecosystems. *Ecological Economics, 29*(2), 215–233. https://doi.org/10.1016/S0921-8009(99)00009-9

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135–1144). ACM. https://doi.org/10.1145/2939672.2939778

Roboflow. (2023). *Coral Reef Bleaching Dataset (Version 2) * [Dataset]. https://universe.roboflow.com/

Ouassine, Y., Conruyt, N., Kayal, M., Martin, P. A., Bigot, L., Vignes Lebbe, R., Lebbe, R. V., Hajar, M., & Zahir, J. (2025, noviembre 1). Deep learning for automated coral reef monitoring: A novel system based on YOLOv8 detection and DeepSORT tracking. *Ecological Informatics, 89*, Article 103170. https://doi.org/10.1016/j.ecoinf.2025.103170

Roth, F., Saalmann, F., Thomson, T., Coker, D. J., Villalobos, R., Jones, B. H., Wild, C., & Carvalho, S. (2018). Coral reef degradation affects the potential for reef recovery after disturbance. *Marine Environmental Research*. https://doi.org/10.1016/j.marenvres.2018.09.022

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *Proceedings of the IEEE International Conference on Computer Vision*, 618–626. https://doi.org/10.1109/ICCV.2017.74

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *arXiv preprint*. https://doi.org/10.48550/arXiv.1409.1556

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Szeliski, R. (2010). *Computer vision: Algorithms and applications*. Springer.

TensorFlow. (2024). *TensorFlow: An end-to-end open source machine learning platform* [Software]. https://www.tensorflow.org/

Wäldchen, J., & Mäder, P. (2017). Plant species identification using computer vision techniques: A systematic literature review. *Archives of Computational Methods in Engineering, 25*(2), 507–543. https://doi.org/10.1007/s11831-016-9206-z

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2921–2929. https://doi.org/10.1109/CVPR.2016.319

APÉNDICES

Apéndice A. Documentación

El sistema fue desarrollado utilizando Python 3.10 con el entorno Google Colab para facilitar el acceso a recursos GPU. Se emplearon librerías como TensorFlow y Keras para la construcción de modelos de aprendizaje profundo, OpenCV para el procesamiento de imágenes, y Matplotlib junto con Seaborn para la visualización de resultados. Todo el flujo de trabajo, desde la adquisición de datos, preprocesamiento, entrenamiento, validación y visualización, fue documentado en notebooks Jupyter adjuntos como archivos complementarios.

El pipeline completo incluye:

- Carga de imágenes desde la API de Roboflow.
- División del dataset en entrenamiento, validación y prueba.
- Implementación de tres modelos: MLP, CNN, y VGG16 con Transfer Learning.
- Entrenamiento con optimizadores Adam y SGD, usando callbacks como
- EarlyStopping y ModelCheckpoint.
- Generación de visualizaciones interpretables con Grad-CAM.
- Interfaz de prototipo que permite cargar una imagen y mostrar predicción con mapa de calor.

Apéndice B. Comparación de Modelos Evaluados

Tabla B1

Resumen comparativo de desempeño, complejidad y características técnicas de los modelos evaluados.

Modelo	Precisión	Ventajas	Limitaciones
MLP	74%	Simplicidad, línea base	No captura relaciones espaciales
CNN	97%	Captura patrones espaciales	Mayor tiempo de entrenamiento
VGG16 (Transfer Learning)	94%	Estabilidad, menor sobreajuste	Requiere más memoria

Apéndice C. Código Fuente de MLP

Figura C1

Importación de librerías y configuración de rutas y clases

```
import os
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
```

```
base_path = '/content/Coral-Reef-Bleach-Detection-2'
classes_df = pd.read_csv(os.path.join(base_path, "train", "_classes.csv"))

# Obtener nombre de clase por imagen
classes_df.columns = [col.strip() for col in classes_df.columns]
label_cols = [col for col in classes_df.columns if col.lower() != "filename"]
classes_df["class"] = classes_df[label_cols].idxmax(axis=1).str.strip()
class_names = sorted(classes_df["class"].unique().tolist())
class_to_index = {cls: i for i, cls in enumerate(class_names)}

print("Clases:", class_names)
```

Figura C2

Carga de imágenes y subconjuntos

```
[ ]  def cargar_imagenes_desde_directorio(directorio, classes_df):
         X, y = [], []
         for archivo in os.listdir(directorio):
             if archivo.endswith(".jpg") or archivo.endswith(".png"):
                 ruta_img = os.path.join(directorio, archivo)
                 fila = classes_df[classes_df["filename"] == archivo]
                 if fila.empty:
                     continue
                 clase = fila.iloc[0]["class"]
                 img = Image.open(ruta_img).convert("L").resize((56, 56))
                 img_array = np.array(img).astype("float32") / 255.0
                 X.append(img_array.flatten())
                 y.append(class_to_index[clase])
         return np.array(X), np.array(y)
```

```
[ ]  # Cargar datos de entrenamiento
     X_train, y_train = cargar_imagenes_desde_directorio(os.path.join(base_path, "train"), classes_df)
     X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

     # Cargar y preparar CSV del conjunto de prueba
     classes_df_test = pd.read_csv(os.path.join(base_path, "test", "_classes.csv"))
     classes_df_test.columns = [col.strip() for col in classes_df_test.columns]
     label_cols_test = [col for col in classes_df_test.columns if col.lower() != "filename"]
     classes_df_test["class"] = classes_df_test[label_cols_test].idxmax(axis=1).str.strip()

     # Cargar imágenes del conjunto de prueba
     X_test, y_test = cargar_imagenes_desde_directorio(os.path.join(base_path, "test"), classes_df_test)

     # Verificar tamaños
     print(f"Train: {X_train.shape}, {y_train.shape}")
     print(f"Valid: {X_val.shape}, {y_val.shape}")
     print(f"Test:  {X_test.shape}, {y_test.shape}")
```

Figura C3

Preparación de etiquetas categóricas

```
y_train_cat = to_categorical(y_train, num_classes=len(class_names))
y_val_cat   = to_categorical(y_val,   num_classes=len(class_names))
y_test_cat  = to_categorical(y_test,  num_classes=len(class_names)) if len(y_test) > 0 else None
```

Figura C4

Definición del modelo MLP con optimizador Adam

```
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(len(class_names), activation='softmax')  # salida multiclase
])

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

model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 256)	803,072
dropout_6 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 128)	32,896
dropout_7 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 2)	258

```
Total params: 836,226 (3.19 MB)
Trainable params: 836,226 (3.19 MB)
Non-trainable params: 0 (0.00 B)
```

Figura C5

Entrenamiento del modelo MLP con optimizador Adam

```
history = model.fit(
    X_train, y_train_cat,
    validation_data=(X_val, y_val_cat),
    epochs=15,
    batch_size=32,
    verbose=1
)
```

```
Epoch 1/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 9s 29ms/step - accuracy: 0.5360 - loss: 0.7765 - val_accuracy: 0.6431 - val_loss: 0.6374
Epoch 2/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.6019 - loss: 0.6599 - val_accuracy: 0.6814 - val_loss: 0.6227
Epoch 3/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.6348 - loss: 0.6288 - val_accuracy: 0.6765 - val_loss: 0.6161
Epoch 4/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.6471 - loss: 0.6179 - val_accuracy: 0.6814 - val_loss: 0.6134
Epoch 5/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.6594 - loss: 0.6040 - val_accuracy: 0.6779 - val_loss: 0.6013
Epoch 6/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.6746 - loss: 0.5946 - val_accuracy: 0.6929 - val_loss: 0.5814
Epoch 7/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.6722 - loss: 0.5903 - val_accuracy: 0.7058 - val_loss: 0.5712
Epoch 8/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.6516 - loss: 0.6046 - val_accuracy: 0.6720 - val_loss: 0.5892
Epoch 9/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.6694 - loss: 0.5878 - val_accuracy: 0.6312 - val_loss: 0.6196
Epoch 10/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.6376 - loss: 0.6035 - val_accuracy: 0.6451 - val_loss: 0.6104
Epoch 11/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.6471 - loss: 0.5930 - val_accuracy: 0.6660 - val_loss: 0.5950
Epoch 12/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.6558 - loss: 0.5993 - val_accuracy: 0.6884 - val_loss: 0.5795
Epoch 13/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.6702 - loss: 0.5901 - val_accuracy: 0.6212 - val_loss: 0.6137
Epoch 14/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.6869 - loss: 0.5743 - val_accuracy: 0.5510 - val_loss: 0.6571
Epoch 15/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.6762 - loss: 0.5743 - val_accuracy: 0.6924 - val_loss: 0.5670
```

Figura C6

Visualización de curvas de perdida y precisión

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

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Pérdida')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train acc')
plt.plot(history.history['val_accuracy'], label='val acc')
plt.title('Precisión')
plt.xlabel('Épocas')
plt.ylabel('Accuracy')
plt.legend()

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

Figura C7

Evaluación del modelo MLP con optimizador Adam

```
if X_test.shape[0] > 0:
    y_pred_probs = model.predict(X_test)
    y_pred = np.argmax(y_pred_probs, axis=1)
    y_true = np.argmax(y_test_cat, axis=1)

    print(classification_report(y_true, y_pred, target_names=class_names))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Predicción")
    plt.ylabel("Real")
    plt.title("Matriz de Confusión")
    plt.show()
else:
    print("El conjunto de test está vacío. No se puede evaluar el modelo.")
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
              precision    recall  f1-score   support

    Bleached       0.78      0.55      0.65       141
     Healthy       0.61      0.82      0.70       122

    accuracy                           0.68       263
   macro avg       0.70      0.69      0.67       263
weighted avg       0.70      0.68      0.67       263
```

Figura C8

Definición del modelo MLP con optimizador SGD

```
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(len(class_names), activation='softmax')  # salida multiclase
])

model.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 256)	803,072
dropout_8 (Dropout)	(None, 256)	0
dense_13 (Dense)	(None, 128)	32,896
dropout_9 (Dropout)	(None, 128)	0
dense_14 (Dense)	(None, 2)	258

Total params: 836,226 (3.19 MB)
Trainable params: 836,226 (3.19 MB)
Non-trainable params: 0 (0.00 B)

Figura C9

Entrenamiento del modelo MLP con optimizador SGD

```
history = model.fit(
    X_train, y_train_cat,
    validation_data=(X_val, y_val_cat),
    epochs=15,
    batch_size=32,
    verbose=1
)
```

```
Epoch 1/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.5575 - loss: 0.6812 - val_accuracy: 0.6327 - val_loss: 0.6407
Epoch 2/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.6355 - loss: 0.6467 - val_accuracy: 0.6655 - val_loss: 0.6235
Epoch 3/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.6671 - loss: 0.6182 - val_accuracy: 0.6491 - val_loss: 0.6233
Epoch 4/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.6718 - loss: 0.6066 - val_accuracy: 0.4749 - val_loss: 0.7913
Epoch 5/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.6924 - loss: 0.5899 - val_accuracy: 0.6765 - val_loss: 0.5941
Epoch 6/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.6892 - loss: 0.5862 - val_accuracy: 0.6610 - val_loss: 0.6176
Epoch 7/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7104 - loss: 0.5687 - val_accuracy: 0.6889 - val_loss: 0.5819
Epoch 8/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7015 - loss: 0.5665 - val_accuracy: 0.6610 - val_loss: 0.6298
Epoch 9/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7249 - loss: 0.5516 - val_accuracy: 0.7232 - val_loss: 0.5531
Epoch 10/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7377 - loss: 0.5341 - val_accuracy: 0.7332 - val_loss: 0.5246
Epoch 11/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7387 - loss: 0.5268 - val_accuracy: 0.6531 - val_loss: 0.5987
Epoch 12/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.7289 - loss: 0.5206 - val_accuracy: 0.6013 - val_loss: 0.6647
Epoch 13/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.7519 - loss: 0.5050 - val_accuracy: 0.6496 - val_loss: 0.7410
Epoch 14/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7561 - loss: 0.4991 - val_accuracy: 0.7745 - val_loss: 0.4874
Epoch 15/15
252/252 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7681 - loss: 0.4747 - val_accuracy: 0.7188 - val_loss: 0.5597
```

Figura C10

Evaluación del modelo MLP con optimizador SGD

```
if X_test.shape[0] > 0:
    y_pred_probs = model.predict(X_test)
    y_pred = np.argmax(y_pred_probs, axis=1)
    y_true = np.argmax(y_test_cat, axis=1)

    print(classification_report(y_true, y_pred, target_names=class_names))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Predicción")
    plt.ylabel("Real")
    plt.title("Matriz de Confusión")
    plt.show()
else:
    print("El conjunto de test está vacío. No se puede evaluar el modelo.")
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
              precision    recall  f1-score   support

    Bleached       0.69      0.94      0.80       141
     Healthy       0.88      0.52      0.65       122

    accuracy                           0.74       263
   macro avg       0.78      0.73      0.72       263
weighted avg       0.78      0.74      0.73       263
```

Figura C11

Definición del modelo MLP con optimizador Lion

```
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(len(class_names), activation='softmax')  # salida multiclase
])

model.compile(optimizer='Lion',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 256)	803,072
dropout_10 (Dropout)	(None, 256)	0
dense_16 (Dense)	(None, 128)	32,896
dropout_11 (Dropout)	(None, 128)	0
dense_17 (Dense)	(None, 2)	258

```
Total params: 836,226 (3.19 MB)
Trainable params: 836,226 (3.19 MB)
Non-trainable params: 0 (0.00 B)
```

Figura C12

Evaluación del modelo MLP con optimizador Lion

```
if X_test.shape[0] > 0:
    y_pred_probs = model.predict(X_test)
    y_pred = np.argmax(y_pred_probs, axis=1)
    y_true = np.argmax(y_test_cat, axis=1)

    print(classification_report(y_true, y_pred, target_names=class_names))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Predicción")
    plt.ylabel("Real")
    plt.title("Matriz de Confusión")
    plt.show()
else:
    print("El conjunto de test está vacío. No se puede evaluar el modelo.")
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
              precision    recall  f1-score   support

    Bleached       0.83      0.42      0.56       141
     Healthy       0.57      0.90      0.70       122

    accuracy                           0.64       263
   macro avg       0.70      0.66      0.63       263
weighted avg       0.71      0.64      0.62       263
```

Apéndice D. Código Fuente de CNN Personalizada

Figura D1

Importación de dependencias y parámetros globales del proyecto

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical

# Importaciones específicas para LIME (explicabilidad del modelo)
import lime
from lime import lime_image
from lime.wrappers.scikit_image import SegmentationAlgorithm
from skimage.segmentation import mark_boundaries

# Configuración de semillas para reproducibilidad
np.random.seed(42)
tf.random.set_seed(42)

# Parámetros globales del proyecto
img_height, img_width = 224, 224
batch_size = 32
```

Figura D2

División de directorios de entrenamiento, validación y prueba

```
[ ] train_dir = os.path.join(dataset.location, 'train')
    valid_dir = os.path.join(dataset.location, 'valid')
    test_dir = os.path.join(dataset.location, 'test')

    # Función para verificar la estructura de imágenes
    def check_image_paths(directory):
        """Verifica y cuenta las imágenes disponibles en un directorio"""
        print(f"Checking {directory}...")
        image_extensions = ['.jpg', '.jpeg', '.png', '.bmp', '.tif', '.tiff']
        image_paths = []

        for root, _, files in os.walk(directory):
            for file in files:
                if any(file.lower().endswith(ext) for ext in image_extensions):
                    image_paths.append(os.path.join(root, file))

        print(f"Found {len(image_paths)} images in {directory}")
        if image_paths:
            print(f"Example paths: {image_paths[:3]}")

        return image_paths

    # Explorar estructura del dataset
    train_images = check_image_paths(train_dir)
    valid_images = check_image_paths(valid_dir)
    test_images = check_image_paths(test_dir)

    # Verificar archivos CSV de etiquetas
    train_csv = os.path.join(train_dir, '_classes.csv')
    valid_csv = os.path.join(valid_dir, '_classes.csv')
    test_csv = os.path.join(test_dir, '_classes.csv')

    print(f"Train CSV exists: {os.path.exists(train_csv)}")
    print(f"Valid CSV exists: {os.path.exists(valid_csv)}")
    print(f"Test CSV exists: {os.path.exists(test_csv)}")
```

Figura D3

Generador de datos para carga de imágenes en batches y con data augmentation

```python
# Generador de datos personalizado con data augmentation
class DataGenerator(tf.keras.utils.Sequence):
    """
    Generador personalizado para cargar imágenes en batches con opciones de augmentación.
    Optimizado para datasets de detección de corales.
    """
    def __init__(self, dataframe, class_to_idx, batch_size=32, img_size=(224, 224),
                 shuffle=True, augment=False):
        self.dataframe = dataframe
        self.batch_size = batch_size
        self.img_size = img_size
        self.shuffle = shuffle
        self.augment = augment
        self.class_to_idx = class_to_idx
        self.n_classes = len(class_to_idx)
        self.indexes = np.arange(len(dataframe))
        self.on_epoch_end()

        # Configuración de data augmentation específica para corales
        if augment:
            self.datagen = tf.keras.preprocessing.image.ImageDataGenerator(
                rotation_range=20,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
                horizontal_flip=True,
                fill_mode='nearest'
            )
        else:
            self.datagen = None

    def __len__(self):
        return int(np.ceil(len(self.dataframe) / self.batch_size))
```

Figura D4

Definición del modelo CNN personalizada

```python
def build_coral_detection_model(input_shape=(img_height, img_width, 3), num_classes=num_classes):
    """
    Construye una CNN optimizada para clasificación de estados de coral.
    Incluye batch normalization y dropout para mejor generalización.
    """
    model = models.Sequential([
        # Primer bloque convolucional
        tf.keras.Input(shape=input_shape),
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        # Segundo bloque convolucional
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        # Tercer bloque convolucional
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        # Cuarto bloque convolucional
        layers.Conv2D(256, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        # Capas densas con regularización
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(512, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
```

Figura D5

Compilación del modelo CNN personalizado con callbacks para optimización de entrenamiento

```
# Crear y compilar el modelo
model = build_coral_detection_model()

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

# Configurar callbacks para optimizar el entrenamiento
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint(
    'coral_detection_model_best.h5',
    monitor='val_accuracy',
    save_best_only=True
)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=5,
    min_lr=1e-6
)
```

Figura D6

Entrenamiento de CNN personalizada

```python
# Entrenar el modelo
print("\n🚀 Iniciando entrenamiento del modelo...")
epochs = 15
history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    callbacks=[early_stopping, checkpoint_cb, reduce_lr]
)

# Guardar modelo final
model.save('coral_detection_model_final.h5')
print("✅ Modelo guardado exitosamente")
```

max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 24, 24, 256)	1,024
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 256)	0
flatten (Flatten)	(None, 36864)	0
dropout (Dropout)	(None, 36864)	0
dense (Dense)	(None, 512)	18,874,880
batch_normalization_4 (BatchNormalization)	(None, 512)	2,048
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1,026

Total params: 19,268,290 (73.50 MB)
Trainable params: 19,266,306 (73.50 MB)
Non-trainable params: 1,984 (7.75 KB)

Figura D7

Código de evaluación de CNN personalizada

```
# Reporte de clasificación detallado
print('\n📈 Classification Report:')
y_true_names = [class_names[idx] for idx in y_true]
y_pred_names = [class_names[idx] for idx in y_pred]
print(classification_report(y_true_names, y_pred_names))

# Matriz de confusión
plt.figure(figsize=(10, 8))
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Matriz de Confusión - Detección de Corales')
plt.show()

# Gráficos de entrenamiento
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Precisión del Modelo')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Pérdida del Modelo')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.tight_layout()
plt.show()
```

Figura D8

Función de análisis de interpretabilidad LIME

```
def coral_lime_batch_analysis(image_paths, output_dir='lime_explanations'):
    """Analiza múltiples imágenes de corales con LIME y guarda resultados"""
    os.makedirs(output_dir, exist_ok=True)
    results = []

    for i, img_path in enumerate(image_paths):
        try:
            print(f"\nAnalizando imagen {i+1}/{len(image_paths)}: {os.path.basename(img_path)}")

            explanation, image = explain_coral_prediction(img_path)

            # Obtener predicción principal
            prediction = model.predict(np.expand_dims(image, axis=0))[0]
            predicted_class = np.argmax(prediction)
            confidence = prediction[predicted_class]

            # Guardar visualización
            save_path = os.path.join(output_dir, f'lime_explanation_{i+1}.png')
            visualize_lime_explanation(explanation, image, predicted_class, save_path)

            results.append({
                'image_path': img_path,
                'predicted_class': class_names[predicted_class],
                'confidence': confidence,
                'explanation_saved': save_path
            })

        except Exception as e:
            print(f"Error procesando {img_path}: {e}")
            results.append({
                'image_path': img_path,
                'predicted_class': 'ERROR',
                'confidence': 0.0,
                'explanation_saved': None
            })
```

Figura D9

Función de predicción de corales

```python
def predict_coral_image(image_path, model, explain=False):
    """
    Función principal para predecir y explicar imágenes de coral
    """
    img = load_img(image_path, target_size=(img_height, img_width))
    img_array = img_to_array(img)
    img_array = img_array / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction, axis=1)[0]

    result = {
        'class': class_names[predicted_class],
        'confidence': float(prediction[0][predicted_class]),
        'all_probabilities': {class_names[i]: float(prediction[0][i]) for i in range(len(class_names))}
    }

    if explain:
        print("Generando explicación LIME...")
        explanation, image = explain_coral_prediction(image_path)
        visualize_lime_explanation(explanation, image, predicted_class)
        result['lime_explanation'] = explanation

    return result
```

Figura D10

Análisis de interpretabilidad LIME

```
print("\n" + "🔍"*20)
print("INICIANDO ANÁLISIS LIME")
print("🔍"*20)

# Paso 1: Ejecutar la demostración automática
print("\n🚀 PASO 1: Ejecutando demostración automática de LIME...")
demo_lime_coral_analysis()

# Paso 2: Ejemplo manual con una imagen específica
print("\n🚀 PASO 2: Ejemplo manual de LIME...")

try:
    # Obtener una imagen específica del test set
    if hasattr(test_generator, 'dataframe'):
        sample_df = test_generator.dataframe.sample(n=1)
        sample_image_path = sample_df['filepath'].iloc[0]
    else:
        # Buscar manualmente una imagen
        sample_image_path = None
        for root, dirs, files in os.walk(test_dir):
            for file in files:
                if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                    sample_image_path = os.path.join(root, file)
                    break
            if sample_image_path:
                break

    if sample_image_path and os.path.exists(sample_image_path):
        print(f"🖼️ Analizando imagen: {os.path.basename(sample_image_path)}")

        # EJECUTAR LIME PASO A PASO
        print("\n📊 Generando explicación LIME...")
        explanation, image = explain_coral_prediction(sample_image_path, num_samples=500)
```

Figura D11

Análisis de interpretabilidad Grad-CAM

```
grad_model = keras.models.Model(
[model.inputs],
[model.get_layer(last_conv_layer_name).output,
model.get_layer(last_conv_layer_name).output])

# Usar tf.GradientTape para calcular gradientes
with tf.GradientTape() as tape:
    last_conv_layer_output, predictions = grad_model(img_array)
    if pred_index is None:
        pred_index = tf.argmax(predictions[0])
    class_channel = predictions[:, pred_index]

# Calcular los gradientes de la salida de la clase objetivo con respecto a las activaciones
# de la última capa convolucional.
grads = tape.gradient(class_channel, last_conv_layer_output)

# Calcular la media de los gradientes por canal. Estos son los "pesos" o "alpha_k"
# en el artículo de Grad-CAM.
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

# Multiplicar cada canal en el mapa de características por su peso correspondiente (pooled_grads).
# Luego, sumar a través de todos los canales para obtener el mapa de calor en bruto.
last_conv_layer_output = last_conv_layer_output[0]
heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
heatmap = tf.squeeze(heatmap)

# Aplicar ReLU al mapa de calor: solo se consideran las contribuciones positivas.
# Esto asegura que solo las características que influyen positivamente en la clase se resaltan.
heatmap = tf.maximum(heatmap, 0)

# Normalizar el mapa de calor a un rango de [0, 1] para la visualización.
max_heatmap = tf.reduce_max(heatmap)
if max_heatmap == 0: # Evitar división por cero si el mapa de calor es todo ceros
    heatmap = heatmap * 0
```

Figura D12

Visualización Grad-CAM

```python
def display_gradcam(img, heatmap, alpha=0.4):
    """
    Superpone el mapa de calor Grad-CAM sobre la imagen original y la muestra.
    Args:
        img (PIL.Image): La imagen PIL original.
        heatmap (np.array): El mapa de calor Grad-CAM (normalizado de 0 a 1).
        alpha (float): Factor de transparencia para la superposición del mapa de calor.
    """
    # Redimensionar el mapa de calor al tamaño de la imagen original
    heatmap = cv2.resize(heatmap, (img.width, img.height))

    # Convertir el mapa de calor a un mapa de color (por ejemplo, mapa de color JET)
    heatmap = np.uint8(255 * heatmap)
    colormap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

    # Convertir la imagen original a array numpy (RGB)
    img_array_rgb = img_to_array(img)
    img_array_rgb = np.uint8(img_array_rgb) # Asegurarse de que esté en el rango 0-255 para la superposición

    # Superponer el mapa de calor en la imagen original
    superimposed_img = colormap * alpha + img_array_rgb * (1 - alpha)
    superimposed_img = np.uint8(superimposed_img)

    # Mostrar la imagen
    plt.imshow(superimposed_img)
    plt.axis('off')
    plt.title('Grad-CAM Visualization')
    plt.show()
```

Apéndice E. Código Fuente de Modelo VGG16 (Transfer Learning)

Figura E1

Función de carga de datos

```python
def load_dataset_from_csv(csv_path, images_dir):
    df = pd.read_csv(csv_path)

    column_names = df.columns.tolist()
    filename_col = None
    class_col = None
    filename_patterns = ['file', 'image', 'path', 'name', 'filename']
    class_patterns = ['class', 'label', 'category', 'annotation']

    for col in column_names:
        col_lower = col.lower()
        if any(pattern in col_lower for pattern in filename_patterns) and filename_col is None:
            filename_col = col
        elif any(pattern in col_lower for pattern in class_patterns) and class_col is None:
            class_col = col

    if filename_col is None and len(column_names) >= 1:
        filename_col = column_names[0]
    if class_col is None and len(column_names) >= 2:
        class_col = column_names[1]

    dataset_df = pd.DataFrame()
    dataset_df['filepath'] = df[filename_col].apply(lambda x: os.path.join(images_dir, str(x)))
    dataset_df['class'] = df[class_col].astype(str)

    class_names = sorted(dataset_df['class'].unique())
    class_to_idx = {cls: i for i, cls in enumerate(class_names)}
    dataset_df['class_idx'] = dataset_df['class'].map(class_to_idx)

    dataset_df = dataset_df[dataset_df['filepath'].apply(os.path.exists)].copy()
    return dataset_df, class_names, class_to_idx
```

Figura E2

Clase de generación de datos con data augmentation

```
class DataGenerator(tf.keras.utils.Sequence):
    def __init__(self, dataframe, class_to_idx, batch_size=32, img_size=(224, 224),
                 shuffle=True, augment=False):
        self.dataframe = dataframe
        self.batch_size = batch_size
        self.img_size = img_size
        self.shuffle = shuffle
        self.augment = augment
        self.class_to_idx = class_to_idx
        self.n_classes = len(class_to_idx)
        self.indexes = np.arange(len(dataframe))
        self.on_epoch_end()

        if augment:
            self.datagen = tf.keras.preprocessing.image.ImageDataGenerator(
                rotation_range=20,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
                horizontal_flip=True,
                fill_mode='nearest'
            )
        else:
            self.datagen = None

    def __len__(self):
        return int(np.ceil(len(self.dataframe) / self.batch_size))

    def on_epoch_end(self):
        if self.shuffle:
            np.random.shuffle(self.indexes)
```

Figura E3

Definición del modelo VGG16 con transfer learning

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = False

x = base_model.output
x = layers.Flatten()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(num_classes, activation='softmax')(x)

model = tf.keras.Model(inputs=base_model.input, outputs=outputs)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

Figura E4

Entrenamiento del modelo VGG16 con transfer learning y early stopping

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint('best_model_vgg16.h5', monitor='val_accuracy', save_best_only=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6)

history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=15,
    callbacks=[early_stopping, checkpoint_cb, reduce_lr]
)

model.save('final_model_vgg16.h5')
```

Figura E5

Código de evaluación de rendimiento del modelo VGG16 con transfer learning

```
# Matriz de confusión
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Curvas de entrenamiento
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])

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

Figura E6

Función de predicción de corales

```python
def predict_coral_image(image_path, model):
    img = load_img(image_path, target_size=(img_height, img_width))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction, axis=1)[0]

    return {
        'class': class_names[predicted_class],
        'confidence': float(prediction[0][predicted_class]),
        'all_probabilities': {class_names[i]: float(prediction[0][i]) for i in range(len(class_names))}
    }
```

Apéndice F. Código Fuente de Website

Figura F1

Descarga automática del modelo pre-entrenado desde Google Drive

```
MODEL_DIR = "/data" if os.path.isdir("/data") else "."
MODEL_PATH = os.path.join(MODEL_DIR, "final_model_vgg16.h5")
GDRIVE_ID = "1qpTNX_vE_J4WLOp-BcqtlbintdqsK8-8"  # <-- ID de tu archivo en Google Drive

if not os.path.exists(MODEL_PATH):
    try:
        import gdown
        url = f"https://drive.google.com/uc?id={GDRIVE_ID}"
        st.warning("Descargando modelo desde Google Drive. Esto puede tardar unos minutos la primera vez...")
        gdown.download(url, MODEL_PATH, quiet=False)
        st.success("¡Modelo descargado exitosamente!")
    except Exception as e:
        st.error(f"No se pudo descargar el modelo: {str(e)}")
        st.stop()
else:
    print(f"Modelo ya existe en {MODEL_PATH}")
```

Figura F2

Configuración visual de la aplicación y estilos personalizados (CSS)

```
# ======== CONFIGURACIÓN DE LA PÁGINA ========
st.set_page_config(
    page_title="🪸 Monitor de Salud del Coral",
    page_icon="🪸",
    layout="wide",
    initial_sidebar_state="expanded"
)

# ======== CSS personalizado ========
st.markdown("""
<style>
    .main-header {
        font-size: 3rem;
        font-weight: 700;
        background: linear-gradient(90deg, #FF6B6B, #4ECDC4, #45B7D1);
        -webkit-background-clip: text;
        -webkit-text-fill-color: transparent;
        text-align: center;
        margin-bottom: 2rem;
    }
    .subtitle {
        font-size: 1.2rem;
        text-align: center;
        color: #666;
        margin-bottom: 3rem;
    }
```

Figura F3

Función de preprocesamiento de imágenes para el modelo

```python
def preprocess_image(image, target_size=(224, 224)):
    """Preprocesar la imagen para la predicción del modelo con manejo de errores"""
    try:
        img_array = np.array(image)
        st.write(f"🔍 Depuración: Forma de la imagen original: {img_array.shape}")
        st.write(f"🔍 Depuración: Modo de imagen: {image.mode}")
        if len(img_array.shape) == 2:
            img_array = cv2.cvtColor(img_array, cv2.COLOR_GRAY2RGB)
            st.info("📝 Se convirtió la imagen en escala de grises a RGB")
        elif len(img_array.shape) == 3:
            if img_array.shape[2] == 4:
                img_array = cv2.cvtColor(img_array, cv2.COLOR_RGBA2RGB)
                st.info("📝 Se convirtió la imagen RGBA a RGB (se eliminó el canal alfa)")
            elif img_array.shape[2] == 3:
                pass
            elif img_array.shape[2] == 1:
                img_array = np.repeat(img_array, 3, axis=2)
                st.info("📝 Se convirtió el canal único a RGB")
            else:
                st.error(f"❌ Formato de imagen no compatible: {img_array.shape[2]} canales")
                return None
        if len(img_array.shape) != 3 or img_array.shape[2] != 3:
            st.error(f"❌ Falló el preprocesamiento de la imagen. Forma final: {img_array.shape}")
            return None
        st.success(f"✅ Imagen preprocesada exitosamente. Forma: {img_array.shape}")
        img_resized = cv2.resize(img_array, target_size)
        img_normalized = img_resized.astype('float32') / 255.0
        img_batch = np.expand_dims(img_normalized, axis=0)
        st.write(f"🔍 Forma final del tensor: {img_batch.shape}")
        return img_batch
    except Exception as e:
        st.error(f"❌ Error al preprocesar la imagen: {str(e)}")
        st.error(f"🦎 Forma de la imagen: {getattr(image, 'size', 'desconocida')}")
        st.error(f"🦎 Modo de imagen: {getattr(image, 'mode', 'desconocido')}")
        return None
```

Figura F4

Interfaz principal y flujo de análisis de imágenes en Streamlit

```
def main():
    st.markdown('<h1 class="main-header">🪸 Monitor de Salud del Coral</h1>', unsafe_allow_html=True)
    st.markdown('<p class="subtitle">Evaluación de la Salud de los Arrecifes de Coral con IA</p>', unsafe_allow_html=True)
    with st.sidebar:
        st.markdown("## 🎛️ Panel de Control")
        st.markdown("""
        <div class="sidebar-content">
            <h3>🧠 Información del Modelo</h3>
            <p>Este modelo de IA analiza imágenes de coral para detectar signos de blanqueamiento y evaluar la salud general del coral.</p>
        </div>
        """, unsafe_allow_html=True)
        st.markdown("### ⚙️ Ajustes")
        show_confidence = st.checkbox("Mostrar Puntuaciones de Confianza", value=True)
        show_history = st.checkbox("Mostrar Historial de Predicciones", value=True)
        auto_analyze = st.checkbox("Analizar automáticamente al cargar", value=True)
        if st.button("🗑️ Borrar Historial", use_container_width=True):
            st.session_state.prediction_history = []
            st.success("¡Historial borrado!")
    model = load_model()
    if model is None:
        st.stop()
    col1, col2 = st.columns([1, 1])
    with col1:
        st.markdown("### 📸 Cargar Imagen de Coral")
        st.markdown('<div class="upload-section">', unsafe_allow_html=True)
        uploaded_file = st.file_uploader(
            "Elija una imagen...",
            type=['png', 'jpg', 'jpeg'],
            help="Cargue una imagen clara de coral para su análisis"
        )
        st.markdown('</div>', unsafe_allow_html=True)
        if uploaded_file is not None:
            image = Image.open(uploaded_file)
            st.image(image, caption="Imagen Cargada", use_container_width=True)
            analyze_button = st.button("🔍 Analizar Salud del Coral", use_container_width=True, type="primary")
            if analyze_button or auto_analyze:
                with st.spinner("🧠 Analizando la salud del coral..."):
                    processed_image = preprocess_image(image)
                    prediction = model.predict(processed_image, verbose=0)
                    healthy_prob = float(prediction[0][0]) * 100
                    bleached_prob = (1 - float(prediction[0][0])) * 100
                    is_healthy = healthy_prob > bleached_prob
                    confidence = max(healthy_prob, bleached_prob)
```

Figura F5

Visualización de resultados de predicción y métricas en Streamlit

```
if hasattr(st.session_state, 'current_prediction'):
    pred = st.session_state.current_prediction
    if pred['is_healthy']:
        st.markdown(f"""
        <div class="healthy-card">
            <h2>✅ Coral Saludable</h2>
            <h3>{pred['confidence']:.1f}% Confianza</h3>
            <p>¡El coral parece estar en buena salud!</p>
        </div>
        """, unsafe_allow_html=True)
    else:
        st.markdown(f"""
        <div class="bleached-card">
            <h2>⚠️ Coral Blanqueado</h2>
            <h3>{pred['confidence']:.1f}% Confianza</h3>
            <p>El coral muestra signos de blanqueamiento.</p>
        </div>
        """, unsafe_allow_html=True)
    if show_confidence:
        fig = create_confidence_chart(pred['healthy_prob'], pred['bleached_prob'])
        st.plotly_chart(fig, use_container_width=True)
```

Figura F6

Generación y visualización del historial de predicciones

```
def create_history_chart():
    if not st.session_state.prediction_history:
        return None
    df = pd.DataFrame(st.session_state.prediction_history)
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=df['timestamp'],
        y=df['healthy_confidence'],
        mode='lines+markers',
        name='Confianza Saludable',
        line=dict(color='#38ef7d', width=3),
        marker=dict(size=8)
    ))
    fig.add_trace(go.Scatter(
        x=df['timestamp'],
        y=df['bleached_confidence'],
        mode='lines+markers',
        name='Confianza Blanqueado',
        line=dict(color='#fc466b', width=3),
        marker=dict(size=8)
    ))
    fig.update_layout(
        title={
            'text': 'Historial de Predicciones',
            'x': 0.5,
            'font': {'size': 20, 'color': '#2c3e50'}
        },
        xaxis_title="Tiempo",
        yaxis_title="Confianza (%)",
        plot_bgcolor='rgba(0,0,0,0)',
        paper_bgcolor='rgba(0,0,0,0)',
        font=dict(size=12),
        height=400,
        hovermode='x unified'
    )
    return fig
```

Figura F7

Archivo de dependencias requirements.txt para la aplicación web

```
streamlit
tensorflow
numpy
pillow
opencv-python
plotly
pandas
gdown
```

Figura F8

Configuración de despliegue en Dockerfile

```
FROM python:3.10-slim

# Instala dependencias del sistema
RUN apt-get update && apt-get install -y \
    libgl1-mesa-glx \
    libglib2.0-0 \
    && rm -rf /var/lib/apt/lists/*

# Establece el directorio de trabajo
WORKDIR /app

# Copia todos los archivos de tu proyecto al contenedor
COPY . .

# Instala dependencias de Python
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# Expone el puerto estándar de Railway
EXPOSE 8080

# Comando para iniciar la app en Railway
CMD ["streamlit", "run", "coral_app.py", "--server.port=8080", "--server.address=0.0.0.0"]
```

Apéndice G. Despliegue de la Aplicación Web en Railway

La aplicación web Monitor de Salud del Coral fue desplegada en la plataforma Railway para facilitar su acceso público y funcionamiento en la nube. El proceso de despliegue se realizó siguiendo buenas prácticas de virtualización y automatización, utilizando contenedores Docker para asegurar la portabilidad y reproducibilidad del entorno.

La configuración del entorno se definió en un archivo Dockerfile (ver Figura F8), donde se especificó la imagen base de Python 3.10, la instalación de dependencias del sistema y las librerías de Python requeridas para la aplicación.

El proyecto fue subido a un repositorio en GitHub, lo que permitió la integración directa con Railway. Desde el panel de Railway se seleccionó dicho repositorio para desplegar la aplicación de forma automatizada. Adicionalmente, se creó un volume asociado a la aplicación en Railway, destinado al almacenamiento del modelo de redes neuronales previamente entrenado, asegurando la persistencia de los archivos necesarios para las predicciones.

Una vez completado el proceso de despliegue, Railway generó una URL pública con protocolo HTTPS, permitiendo el acceso a la aplicación desde cualquier ubicación. La URL de acceso es la siguiente:

https://coral-monitor-production.up.railway.app/

Esto permite a cualquier usuario interactuar con la aplicación web, cargar imágenes de corales y recibir una evaluación automática de su estado de salud utilizando inteligencia artificial, todo desde una interfaz sencilla y accesible en la web.