UNIT 25: Applied Machine Learning

Sistema de detección de enfermedades en hojas de manzano

Proyecto basado en el dataset Plant Pathology 2020 de Kaggle

Daniel Fidalgo Millán

HND Computer Science & AI/Data Science

Curso 2024-2025

Índice de contenidos

1. Introducción

La agricultura moderna enfrenta desafíos constantes relacionados con la detección temprana de enfermedades en cultivos. Las enfermedades foliares en manzanos pueden causar pérdidas significativas en la producción si no se detectan y tratan a tiempo. Este proyecto presenta una solución basada en machine learning para la identificación automática de enfermedades en hojas de manzano.

El sistema utiliza técnicas de deep learning, específicamente transfer learning con la arquitectura EfficientNetB0, para clasificar imágenes de hojas en cuatro categorías: saludable (healthy), roya (rust), sarna (scab) y múltiples enfermedades (multiple diseases).

El proyecto se ha desarrollado siguiendo una metodología completa que incluye análisis exploratorio de datos, preprocesamiento de imágenes, entrenamiento del modelo, evaluación de rendimiento y despliegue de una aplicación web funcional que permite a agricultores y usuarios analizar el estado de salud de sus plantas de forma instantánea.

2. Objetivos del proyecto

Los objetivos principales de este proyecto son:

• Desarrollar un modelo de clasificación de imágenes capaz de identificar enfermedades en hojas de manzano con alta precisión.

• Aplicar técnicas de transfer learning utilizando EfficientNetB0 como modelo base.

• Implementar estrategias para manejar el desbalanceo de clases presente en el dataset.

• Evaluar el rendimiento del modelo utilizando múltiples métricas (accuracy, AUC, matriz de confusión).

• Desplegar el modelo en una aplicación web funcional que permita a los usuarios analizar imágenes de hojas en tiempo real.

• Implementar segmentación visual mediante OpenCV para identificar las áreas afectadas por enfermedades.

• Documentar todo el proceso de desarrollo siguiendo las mejores prácticas de machine learning.

3. Marco teórico

3.1 Transfer learning

Transfer learning es una técnica de machine learning que permite reutilizar un modelo preentrenado en una tarea similar. En lugar de entrenar una red neuronal desde cero, se utiliza un modelo que ya ha aprendido características generales de imágenes (como bordes, texturas y formas) y se adapta a la tarea específica. Esta aproximación reduce

significativamente el tiempo de entrenamiento y mejora los resultados cuando se dispone de datasets pequeños o medianos.

3.2 EfficientNet

EfficientNet es una familia de arquitecturas de redes neuronales convolucionales desarrollada por Google en 2019. EfficientNetB0 es la versión base que ofrece un excelente balance entre precisión y eficiencia computacional. Esta arquitectura utiliza técnicas de escalado compuesto que optimizan simultáneamente la profundidad, anchura y resolución de la red, logrando mejores resultados con menos parámetros que arquitecturas anteriores como ResNet o VGG.

3.3 Data augmentation

Data augmentation es una técnica de regularización que aumenta artificialmente el tamaño del dataset de entrenamiento aplicando transformaciones a las imágenes originales. Estas transformaciones incluyen rotaciones, volteos, cambios de brillo, zoom, entre otros. El objetivo es hacer que el modelo sea más robusto ante variaciones en las condiciones de captura de las imágenes.

3.4 Class weights

Cuando existe un desbalanceo significativo entre las clases, se utilizan pesos de clase para penalizar más los errores en las clases minoritarias. Esto ayuda al modelo a prestar más atención a las clases con menos muestras, evitando que el modelo simplemente prediga siempre la clase mayoritaria.

3.5 Segmentación de imágenes con OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de visión por computador que permite procesar imágenes para identificar regiones de interés. En este proyecto, se utiliza segmentación basada en el espacio de color HSV para detectar las áreas afectadas por roya (tonos anaranjados) y sarna (manchas oscuras), proporcionando una visualización clara de las zonas enfermas.

4. Análisis exploratorio de datos (EDA)

4.1 Descripción del dataset

El dataset utilizado proviene de la competición Plant Pathology 2020 de Kaggle. Contiene imágenes de hojas de manzano etiquetadas en cuatro categorías diferentes. El formato de etiquetado es one-hot encoded, donde cada imagen tiene una etiqueta binaria para cada clase.

Características principales del dataset:

3. Análisis Exploratorio de Datos (EDA)

```
# Cargar datos de entrenamiento
IMAGES_DIR = os.path.join(DATA_DIR, "images")
train_df = pd.read_csv(os.path.join(DATA_DIR, "train.csv"))

print("📊 Información del Dataset:")
print(f"Total de muestras: {len(train_df)}")
print(f"\nColumnas: {train_df.columns.tolist()}")
print(f"\nPrimeras filas:")
train_df.head(10)
```

```
📊 Información del Dataset:
Total de muestras: 1821

Columnas: ['image_id', 'healthy', 'multiple_diseases', 'rust', 'scab']

Primeras filas:
```

	image_id	healthy	multiple_diseases	rust	scab
0	Train_0	0	0	0	1
1	Train_1	0	1	0	0
2	Train_2	1	0	0	0
3	Train_3	0	0	1	0
4	Train_4	1	0	0	0
5	Train_5	1	0	0	0
6	Train_6	0	1	0	0
7	Train_7	0	0	0	1
8	Train_8	0	0	0	1
9	Train_9	1	0	0	0

Figura 1: Carga y exploración inicial del dataset

El dataset contiene un total de 1,821 imágenes distribuidas entre las cuatro clases. Las columnas incluyen el identificador de imagen (image_id) y las etiquetas one-hot para cada categoría: healthy, multiple_diseases, rust y scab.

4.2 Distribución de clases

Un aspecto crítico en cualquier problema de clasificación es analizar la distribución de las clases. En este dataset se observa un desbalanceo significativo:

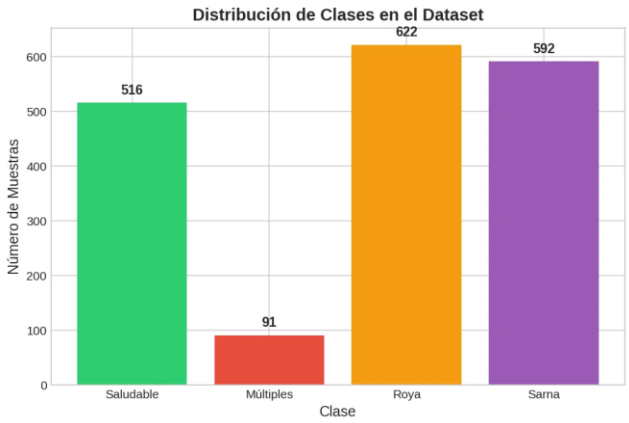
```
# Definir clases y colores
CLASS_NAMES = ['healthy', 'multiple_diseases', 'rust', 'scab']
CLASS_COLORS = ['#2ecc71', '#e74c3c', '#f39c12', '#9b59b6']
CLASS_LABELS_ES = ['Saludable', 'Múltiples', 'Roya', 'Sarna']

# Calcular distribución de clases
class_distribution = {
    'healthy': train_df['healthy'].sum(),
    'multiple_diseases': train_df['multiple_diseases'].sum(),
    'rust': train_df['rust'].sum(),
    'scab': train_df['scab'].sum()
}

print("📊 Distribución de Clases:")
for cls, count in class_distribution.items():
    pct = (count / len(train_df)) * 100
    print(f"  {cls}: {count} ({pct:.1f}%)")
```

```
📊 Distribución de Clases:
  healthy: 516 (28.3%)
  multiple_diseases: 91 (5.0%)
  rust: 622 (34.2%)
  scab: 592 (32.5%)
```

Figura 2: Cálculo de la distribución de clases



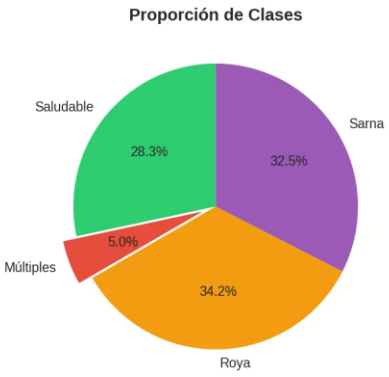


Figura 3: Visualización de la distribución de clases en el dataset

La distribución de clases muestra:

• Saludable (healthy): 516 muestras (28.3%)

• Múltiples enfermedades: 91 muestras (5.0%) - clase minoritaria

• Roya (rust): 622 muestras (34.2%)

• Sarna (scab): 592 muestras (32.5%)

La clase "multiple_diseases" representa solo el 5% del total, lo que requiere técnicas especiales de balanceo para evitar que el modelo ignore esta clase durante el entrenamiento.

4.3 Ejemplos de imágenes por clase

A continuación se muestran ejemplos representativos de cada clase para entender visualmente las características distintivas de cada enfermedad:

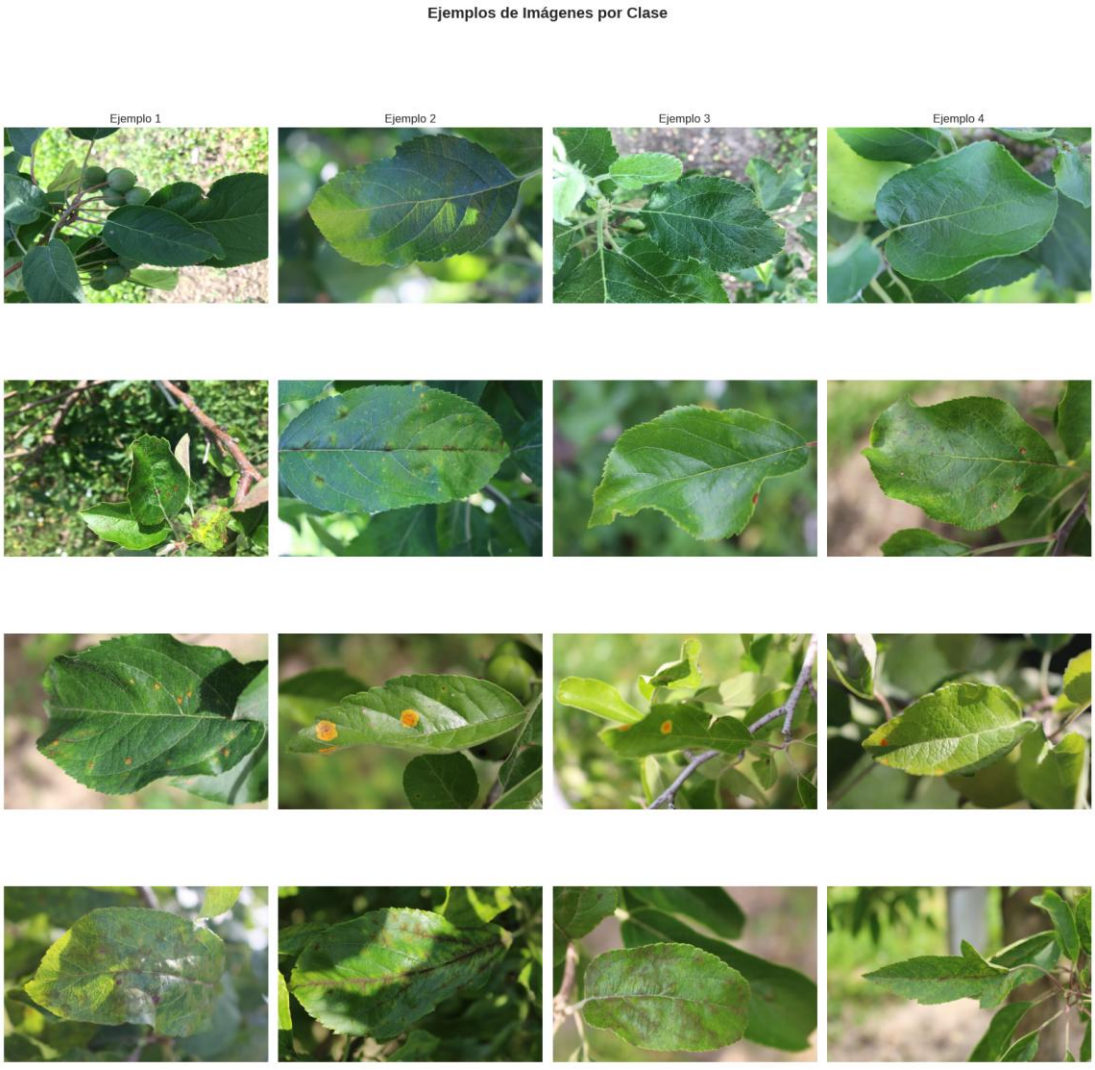


Figura 4: Ejemplos de imágenes para cada clase del dataset

Las imágenes muestran las diferencias visuales entre las clases:

• Hojas saludables: color verde uniforme sin manchas ni decoloraciones.

• Múltiples enfermedades: presencia de síntomas combinados de roya y sarna.

• Roya: manchas anaranjadas/amarillentas características del hongo Gymnosporangium.

• Sarna: manchas oscuras y lesiones irregulares causadas por el hongo Venturia inaequalis.

4.4 Análisis de dimensiones

Se analizaron las dimensiones de las imágenes para determinar el preprocesamiento necesario:

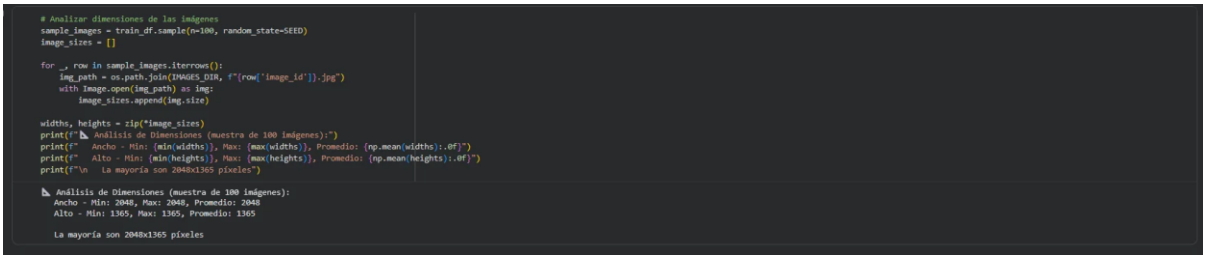


Figura 5: Análisis de dimensiones de las imágenes

Las imágenes del dataset tienen una resolución uniforme de 2048x1365 píxeles. Para el entrenamiento, las imágenes se redimensionan a 224x224 píxeles, que es el tamaño de entrada estándar para EfficientNetB0.

5. Desarrollo del modelo

5.1 Preparación de datos

La configuración de hiperparámetros y preparación de datos es crucial para el éxito del entrenamiento:

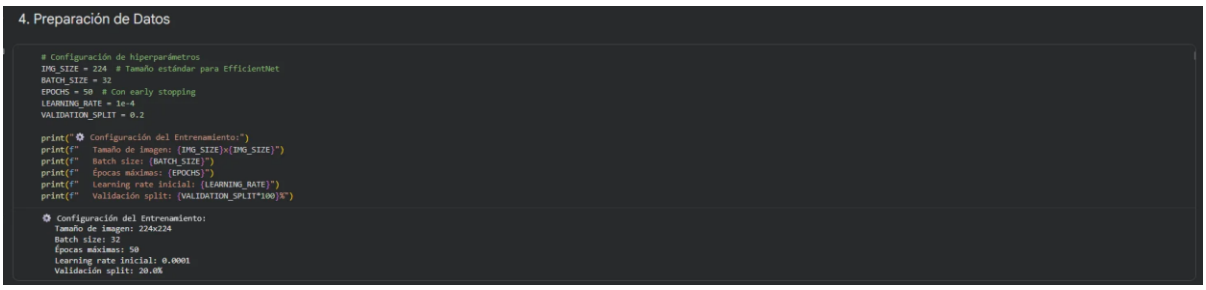


Figura 6: Configuración de hiperparámetros del entrenamiento

Configuración utilizada:

• Tamaño de imagen: 224x224 (estándar para EfficientNet)

• Batch size: 32

• Épocas máximas: 50 (con early stopping)

• Learning rate inicial: 0.0001

• Validación split: 20%

5.2 Manejo del desbalanceo de clases

Para manejar el desbalanceo de clases, se calcularon pesos que penalizan más los errores en la clase minoritaria:

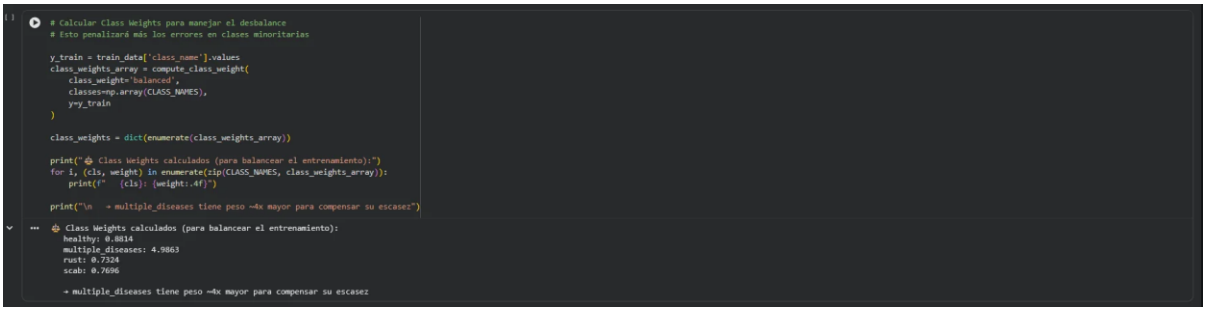


Figura 7: Cálculo de class weights para balancear el entrenamiento

Los pesos calculados muestran que la clase "multiple_diseases" tiene un peso aproximadamente 4 veces mayor que las demás clases, compensando así su baja representación en el dataset.

5.3 Generadores de datos

Se crearon generadores de datos que aplican data augmentation durante el entrenamiento:

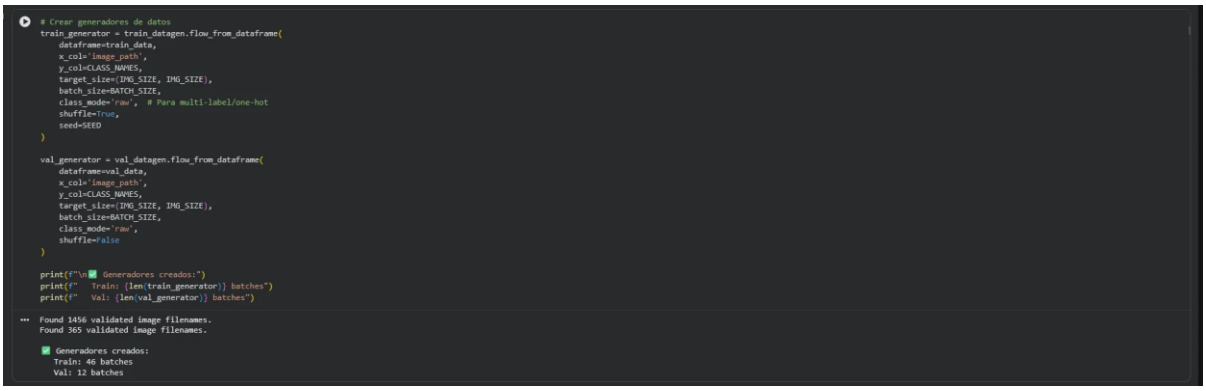


Figura 8: Configuración de los generadores de datos

Los generadores dividen el dataset en 1,456 imágenes para entrenamiento (46 batches) y 365 imágenes para validación (12 batches). El generador de entrenamiento aplica data augmentation mientras que el de validación no.

5.4 Data augmentation

Las técnicas de data augmentation aplicadas incluyen rotaciones, volteos horizontales/verticales, zoom, cambios de brillo y contraste:

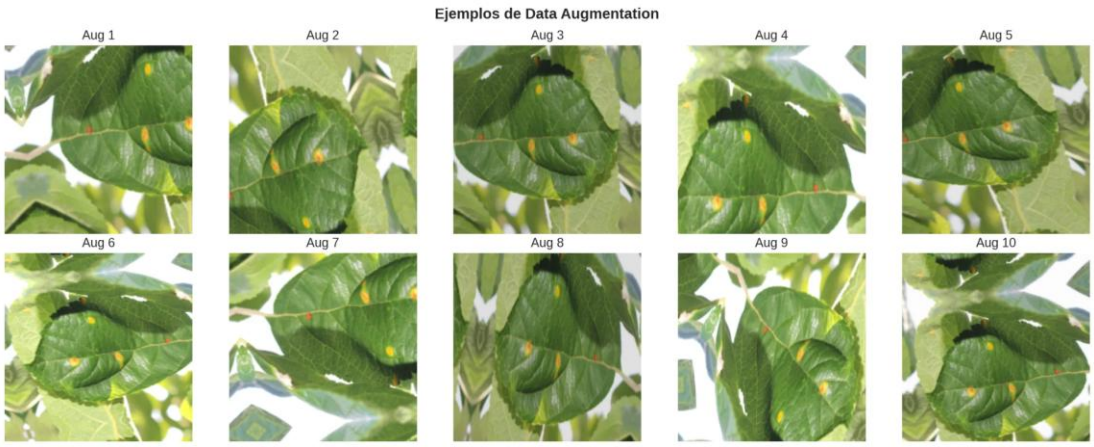


Figura 9: Ejemplos de data augmentation aplicada a una imagen

Estas transformaciones permiten que el modelo sea más robusto ante variaciones en las condiciones de captura de las imágenes y aumentan artificialmente el tamaño efectivo del dataset de entrenamiento.

5.5 Arquitectura del modelo

El modelo utiliza transfer learning con EfficientNetB0 como base, añadiendo capas personalizadas para la clasificación:


Modelo construido:
Model: "PlantDiseaseClassifier"

Layer (type)	Output Shape	Param #
input_image (InputLayer)	(None, 224, 224, 3)	0
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
global_avg_pool (GlobalAveragePooling2D)	(None, 1280)	0
dense_256 (Dense)	(None, 256)	327,936
batch_norm (BatchNormalization)	(None, 256)	1,024
relu_activation (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
predictions (Dense)	(None, 4)	1,028

Total params: 4,379,559 (16.71 MB)
Trainable params: 329,476 (1.26 MB)
Non-trainable params: 4,050,083 (15.45 MB)

Figura 10: Arquitectura completa del modelo

La arquitectura del modelo incluye:

• EfficientNetB0 como extractor de características (4,049,571 parámetros)

• Global average pooling 2D para reducir dimensionalidad

• Capa densa de 256 neuronas con batch normalization y dropout

• Capa de salida con 4 neuronas (softmax) para las 4 clases

El modelo total tiene 4,379,559 parámetros, de los cuales 329,476 son entrenables en la fase inicial.

6. Evaluación y resultados

6.1 Curvas de entrenamiento

Las curvas de entrenamiento muestran la evolución del modelo durante las 26 épocas de entrenamiento:

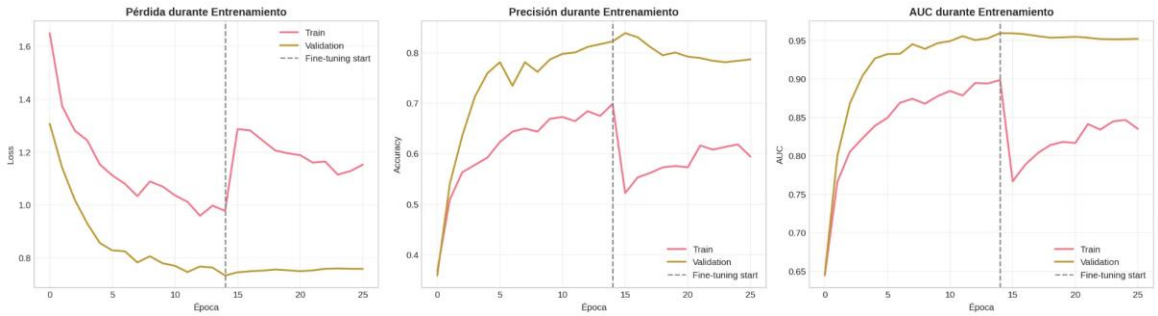


Figura 11: Curvas de pérdida, precisión y AUC durante el entrenamiento

Las gráficas muestran tres fases distintas:

1. Feature extraction (épocas 1-13): solo se entrenan las capas superiores mientras EfficientNetB0 permanece congelado.

2. Fine-tuning (épocas 14-26): se descongelan las capas del modelo base para ajuste fino.

3. La línea vertical punteada marca el inicio del fine-tuning.

Se observa que el modelo converge bien sin signos evidentes de overfitting severo, con la validación siguiendo de cerca al entrenamiento.

6.2 Análisis de overfitting/underfitting

Figura 12: Análisis de overfitting/underfitting

El análisis muestra un modelo balanceado con:

• Accuracy de entrenamiento: 59.41%

• Accuracy de validación: 78.63%

• Gap (diferencia): -19.22%

El gap negativo indica que el modelo generaliza mejor en validación que en entrenamiento, lo cual es favorable y sugiere que no hay overfitting. Este comportamiento se debe a que el data augmentation solo se aplica durante el entrenamiento.

6.3 Métricas finales

Figura 13: Resumen final del modelo entrenado

Resultados finales del modelo:

• Validation accuracy: 83.84%

• Validation AUC: 0.9593

• Macro AUC: 0.9399

Estos resultados demuestran un excelente rendimiento del modelo en la clasificación de enfermedades, superando significativamente un clasificador aleatorio.

6.4 Rendimiento por clase

Figura 14: Análisis detallado del rendimiento por clase

El análisis por clase revela:

• Saludable: 89.32% de precisión (92 de 103 correctas)

• Roya: 89.60% de precisión (112 de 125 correctas)

• Sarna: 78.15% de precisión (93 de 119 correctas)

• Múltiples: 50.00% de precisión (9 de 18 correctas)

La clase "múltiples enfermedades" presenta el rendimiento más bajo debido a la escasez de datos de entrenamiento y a la complejidad inherente de identificar múltiples patologías simultáneas.

6.5 Matriz de confusión

La matriz de confusión muestra en detalle las predicciones correctas e incorrectas:

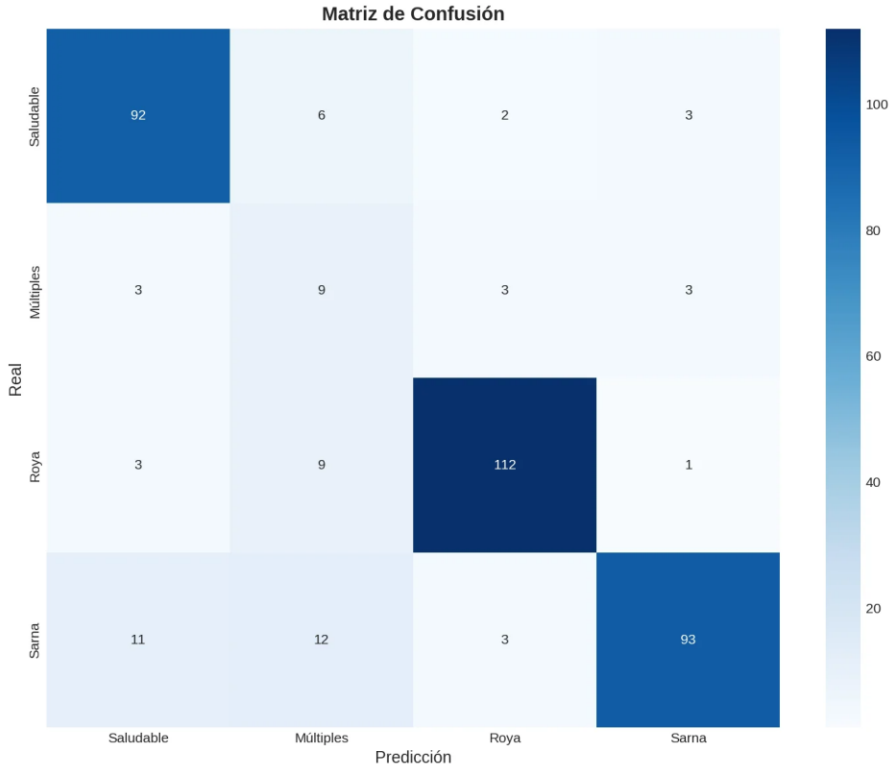


Figura 15: Matriz de confusión (valores absolutos)

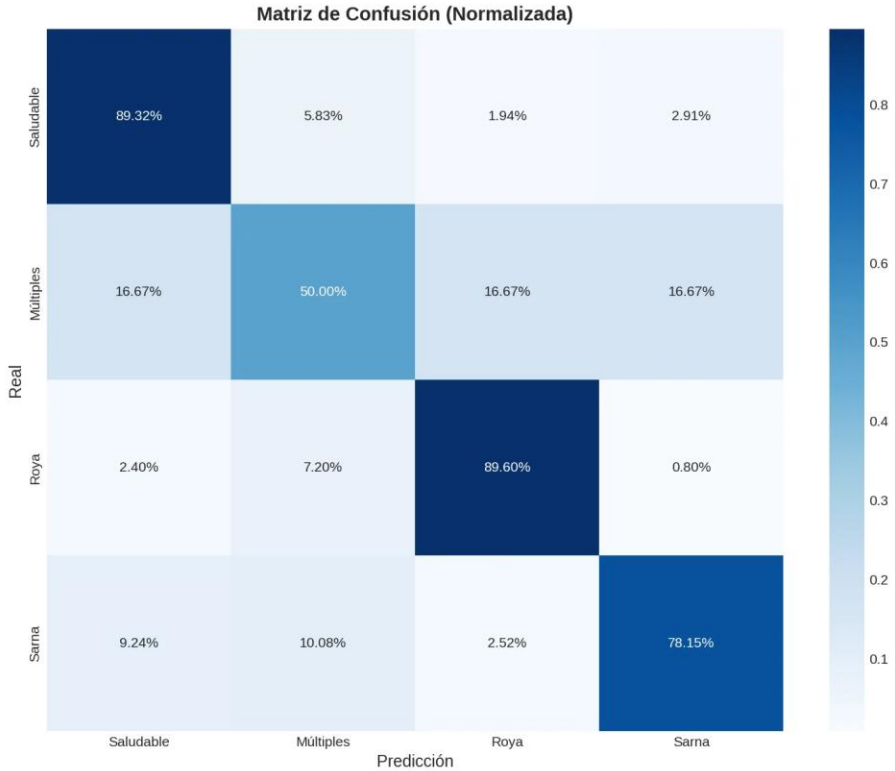


Figura 16: Matriz de confusión normalizada

La matriz de confusión normalizada permite observar los patrones de error más claramente. La clase "multiple_diseases" es la que presenta más errores de clasificación, siendo confundida frecuentemente con las otras tres clases de manera equitativa.

6.6 Curvas ROC

Las curvas ROC permiten evaluar el rendimiento del clasificador para cada clase en diferentes umbrales de decisión:

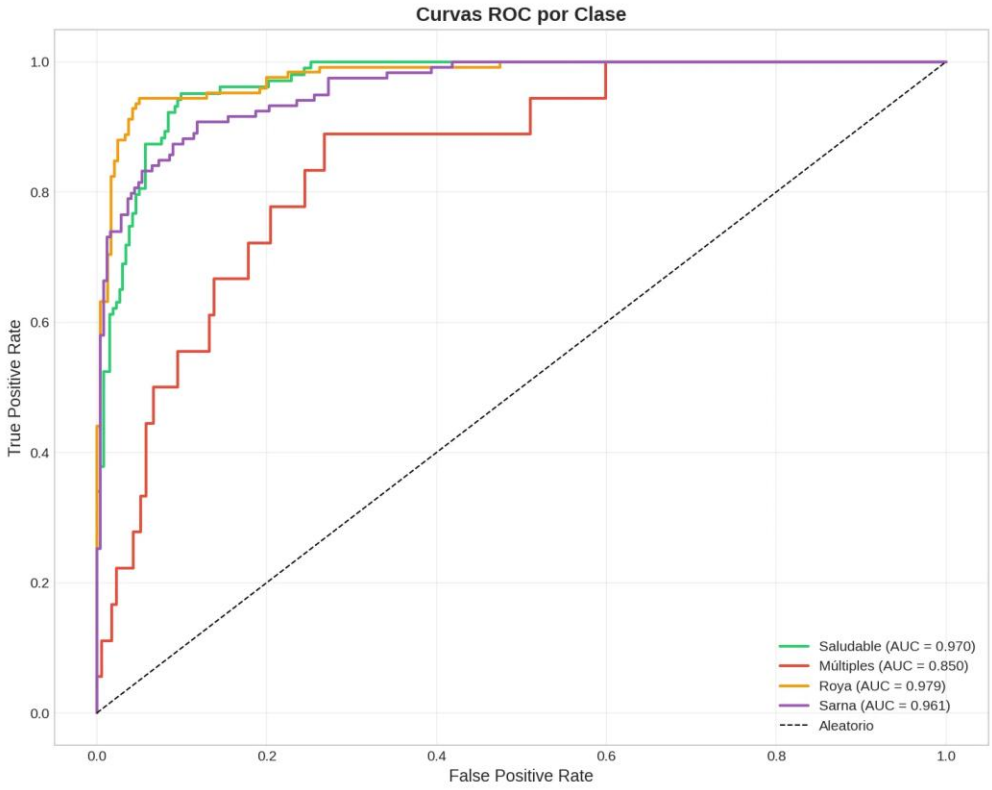


Figura 17: Curvas ROC por clase

Los valores de AUC (area under curve) por clase son:

- Saludable: AUC = 0.970
- Roya: AUC = 0.979
- Sarna: AUC = 0.961
- Múltiples: AUC = 0.850

Todas las clases muestran un AUC significativamente superior al clasificador aleatorio (línea diagonal, AUC = 0.5), indicando un buen poder discriminativo del modelo.

6.7 Ejemplos de predicciones

A continuación se muestran ejemplos de predicciones realizadas por el modelo:

Figura 18: Ejemplos de predicciones del modelo (verde = correcto, rojo = incorrecto)

Las predicciones muestran tanto casos correctos (verde) como algunos errores (rojo). Los errores más comunes ocurren cuando el modelo confunde casos de roya o sarna con "múltiples enfermedades", lo cual es comprensible dado que los síntomas pueden solaparse.

7. Aplicación web

El modelo ha sido desplegado en una aplicación web completa desarrollada con Next.js y desplegada en Vercel. La aplicación está disponible públicamente en:

https://plant-disease-detector-web-app.vercel.app/

La aplicación permite a agricultores, jardineros y usuarios en general analizar imágenes de hojas de manzano en tiempo real para detectar posibles enfermedades y recibir recomendaciones de tratamiento.

7.1 Página principal

La página principal presenta una interfaz moderna y accesible que explica las funcionalidades de la aplicación:

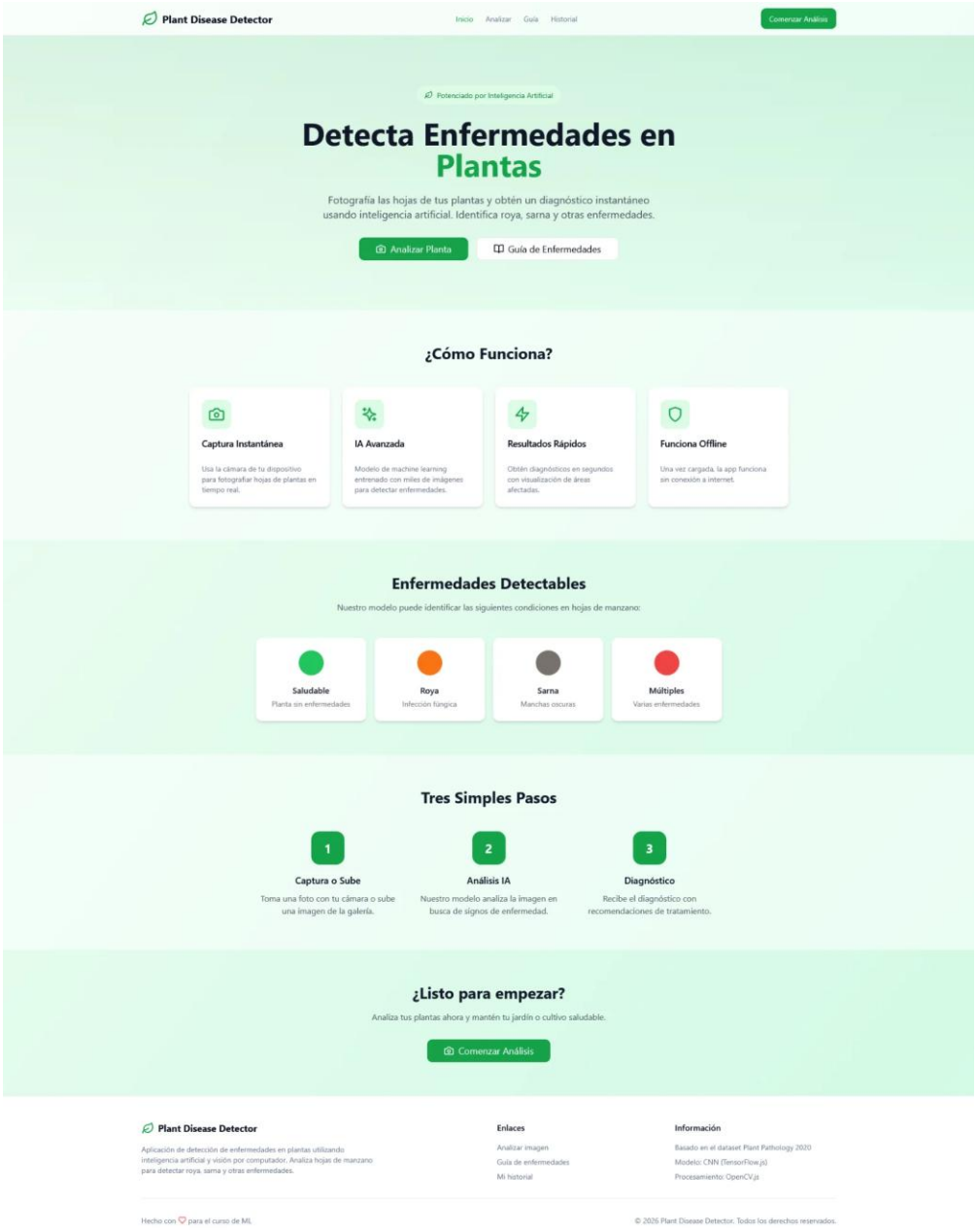


Figura 19: Página principal de la aplicación Plant Disease Detector

La página de inicio incluye:

• Encabezado con navegación clara entre secciones (Inicio, Analizar, Guía, Historial)

• Descripción del propósito de la aplicación: detectar enfermedades en plantas usando IA

• Sección "¿Cómo funciona?" que explica el proceso en cuatro pasos: captura instantánea, IA avanzada, resultados rápidos y funciona offline

• Visualización de las cuatro enfermedades detectables: saludable, roya, sarna y múltiples

• Guía de tres pasos simples: captura o sube, análisis IA, y diagnóstico
• Pie de página con información técnica sobre el dataset y tecnologías utilizadas

7.2 Módulo de análisis

La página de análisis ofrece dos métodos para capturar la imagen de la hoja:

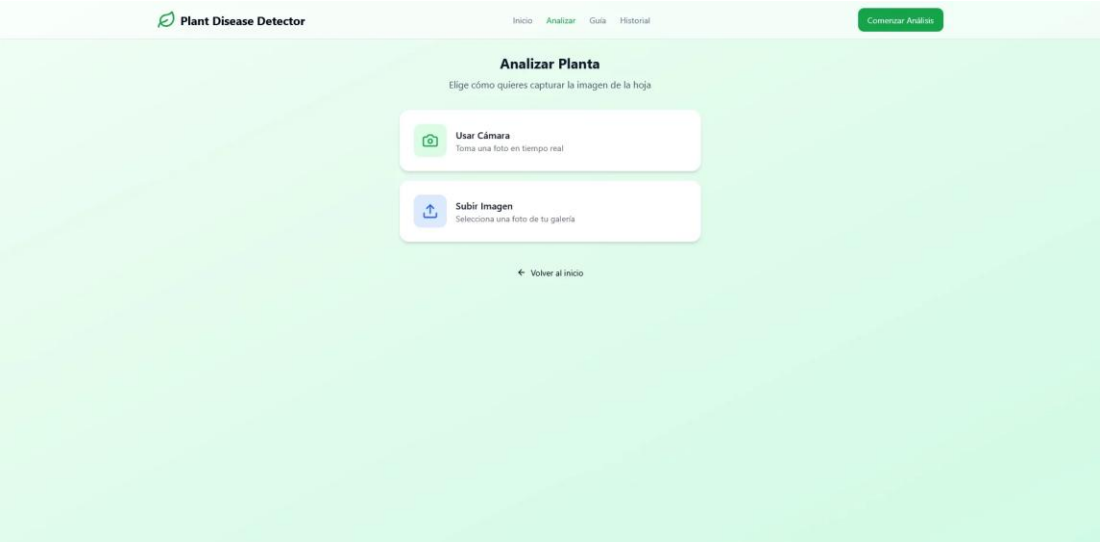


Figura 20: Página de análisis con opciones de captura

El módulo de análisis proporciona:

• Opción "Usar cámara": permite tomar una foto en tiempo real directamente desde el dispositivo. Esta funcionalidad es especialmente útil para uso en campo, donde el usuario puede fotografiar la planta directamente.

• Opción "Subir imagen": permite seleccionar una foto existente de la galería del dispositivo. Útil cuando ya se dispone de fotografías previas o se desea analizar imágenes almacenadas.

La interfaz es responsive, adaptándose tanto a dispositivos móviles como a pantallas de escritorio, facilitando su uso en diferentes contextos.

7.3 Resultados y diagnóstico

Una vez procesada la imagen, la aplicación muestra un diagnóstico completo y detallado:

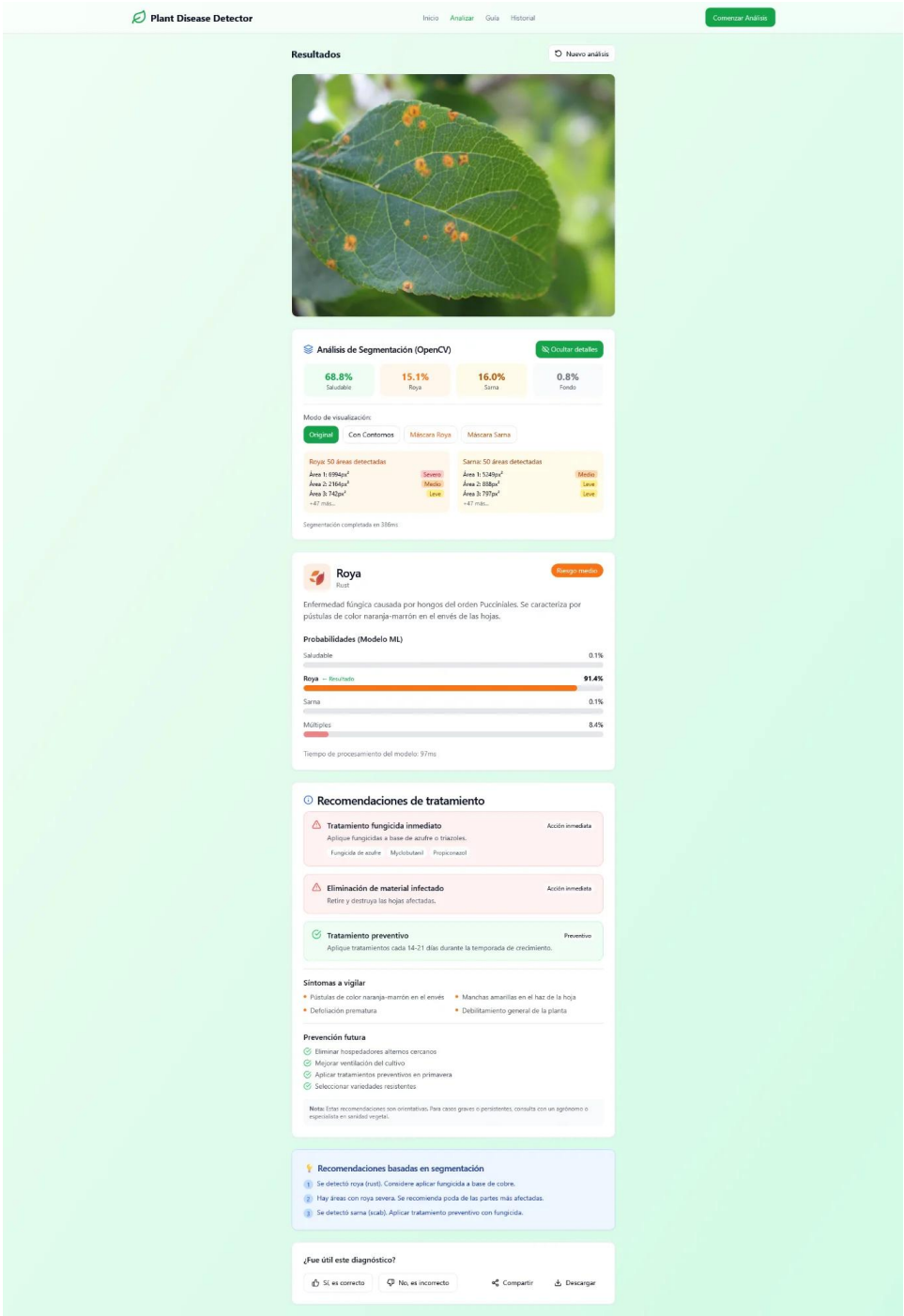


Figura 21: Página de resultados con diagnóstico completo

La página de resultados incluye múltiples secciones informativas:

• Imagen analizada: se muestra la imagen original con la posibilidad de ver diferentes modos de visualización (original, con contornos, máscara roya, máscara sarna).

• Análisis de segmentación (OpenCV): muestra los porcentajes de área para cada categoría (saludable 68.8%, roya 15.1%, sarna 16.0%, fondo 0.8%), así como el listado de áreas detectadas con su severidad.

• Diagnóstico principal: indica la enfermedad detectada (en este caso "roya" con riesgo medio) junto con una descripción de la enfermedad.

• Probabilidades del modelo ML: gráfico de barras mostrando las probabilidades para cada clase (roya 91.4%, múltiples 8.4%, etc.).

• Tiempo de procesamiento: indica el tiempo que tardó el modelo en procesar la imagen (97ms).

• Recomendaciones de tratamiento: incluye acciones inmediatas como tratamiento fungicida, eliminación de material infectado, y tratamiento preventivo.

• Síntomas a vigilar: lista de señales de alerta como pústulas de color naranja-marrón, manchas amarillas, defoliación prematura.

• Prevención futura: consejos para evitar futuras infecciones.

• Botones de interacción: opciones para indicar si el diagnóstico fue correcto, compartir o descargar los resultados.

7.4 Segmentación visual con OpenCV

Una de las características más avanzadas de la aplicación es la segmentación visual de las áreas afectadas:

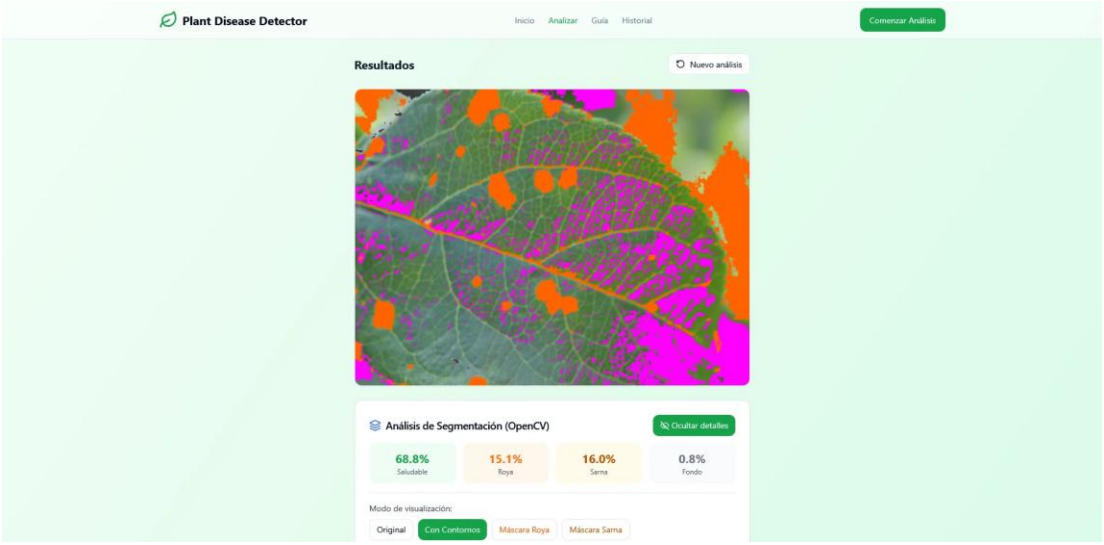


Figura 22: Visualización de segmentación con contornos de áreas afectadas

La segmentación visual proporciona:

• Detección de áreas con roya: marcadas en color naranja, identifican las zonas afectadas por la infección fúngica caracterizada por pústulas anaranjadas.

• Detección de áreas con sarna: marcadas en color magenta/rosa, identifican las manchas oscuras características de esta enfermedad.

• Área saludable: el verde de la hoja no afectado por enfermedades permanece visible.

• Fondo: las áreas fuera de la hoja se identifican y excluyen del análisis.

La segmentación utiliza técnicas de procesamiento de imagen basadas en el espacio de color HSV (hue-saturation-value), que permite identificar los tonos característicos de cada enfermedad de manera más efectiva que el espacio RGB tradicional.

7.5 Arquitectura técnica de la aplicación

La aplicación web está construida con las siguientes tecnologías:

• Frontend: Next.js 14 con TypeScript para una aplicación moderna y tipada

• Estilos: Tailwind CSS para un diseño responsive y consistente

• Inferencia ML: TensorFlow.js ejecutando el modelo EfficientNetB0

• Modelo: alojado en Hugging Face para distribución eficiente

• Segmentación: opencv-wasm para procesamiento de imágenes en el servidor

• Despliegue: Vercel para hosting con CDN global y SSL automático

• PWA: funcionalidad offline mediante service workers

7.5.1 Diagrama de contexto del sistema

El diagrama de contexto presenta una visión de alto nivel del sistema, mostrando las interacciones entre las entidades externas principales y la aplicación PWA (Progressive Web App). Este diagrama ilustra cómo el usuario (agricultor o jardinero) interactúa con el sistema enviando imágenes de hojas y recibiendo diagnósticos con recomendaciones de tratamiento, mientras que el sistema se comunica con Hugging Face para obtener los pesos del modelo EfficientNetB0 cuando es necesario.

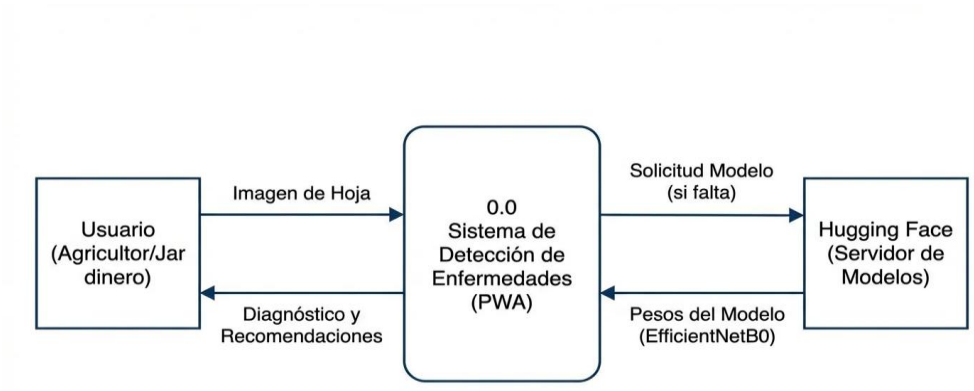


Figura 23: Diagrama de contexto del sistema de detección de enfermedades en plantas

7.5.2 Diagrama de flujo de datos detallado

El diagrama de flujo de datos detallado descompone el sistema en sus procesos internos principales, mostrando cómo fluye la información desde la captura de la imagen hasta la generación del diagnóstico final. El proceso se inicia con la adquisición de la imagen en el cliente (módulo 1.0), que alimenta simultáneamente dos flujos de procesamiento paralelos.

Por un lado, la imagen raw se envía al módulo de preprocesamiento TensorFlow.js (2.0), donde se redimensiona a 224x224 píxeles y se normaliza según los requisitos de EfficientNet. El tensor resultante se procesa en el módulo de inferencia (4.0), que utiliza el modelo previamente cargado desde Hugging Face y almacenado en el Browser Cache/IndexedDB (módulo 3.0) para optimizar los tiempos de carga en sesiones posteriores.

Paralelamente, la imagen original se envía al módulo de segmentación OpenCV (5.0), que aplica técnicas de procesamiento basadas en el espacio de color HSV para identificar las áreas afectadas por roya y sarna. Finalmente, el módulo de generación de resultados (6.0) combina las probabilidades del modelo de deep learning con las máscaras de segmentación para producir un diagnóstico completo con visualización de las zonas afectadas y recomendaciones de tratamiento personalizadas.

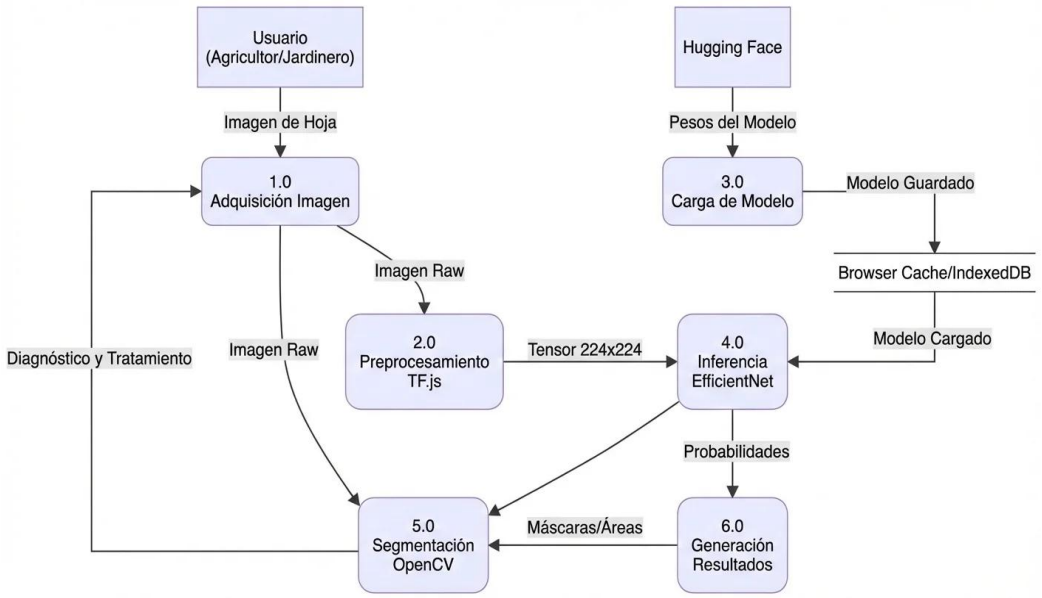


Figura 24: Diagrama de flujo de datos detallado. El proceso inicia con la captura de imagen en el cliente (Next.js), bifurcándose en dos procesos paralelos: la inferencia del modelo de deep learning mediante TensorFlow.js para la clasificación y el procesamiento de imagen con OpenCV (WASM) para la segmentación visual de las áreas afectadas, convergiendo ambos resultados en la interfaz de usuario final.

8. Conclusiones

Este proyecto ha demostrado la viabilidad de utilizar técnicas de deep learning para la detección automatizada de enfermedades en plantas. Los principales logros incluyen:

• Se desarrolló un modelo de clasificación con una precisión del 83.84% y un AUC de 0.9593, superando significativamente un clasificador aleatorio.

• Se implementaron técnicas efectivas para manejar el desbalanceo de clases mediante class weights, aunque la clase minoritaria sigue presentando desafíos.

• El modelo demuestra excelente capacidad de generalización sin overfitting, gracias a las técnicas de regularización como data augmentation y dropout.

• Se desplegó una aplicación web completa y funcional que permite el uso práctico del modelo, incluyendo captura por cámara, análisis de imágenes, y recomendaciones de tratamiento.

• Se implementó segmentación visual mediante OpenCV que proporciona información adicional sobre las áreas afectadas, mejorando la interpretabilidad de los resultados.

8.1 Limitaciones y trabajo futuro

A pesar de los buenos resultados, existen áreas de mejora identificadas:

• La clase "múltiples enfermedades" tiene baja precisión (50%) debido a la escasez de datos. Aumentar el dataset con más imágenes de esta clase mejoraría significativamente los resultados.

• Se podrían explorar técnicas avanzadas de data augmentation como mixup o cutmix para mejorar la generalización en clases minoritarias.

• La segmentación de áreas afectadas podría mejorarse con modelos específicos como U-Net o Mask R-CNN para obtener máscaras más precisas.

• Expandir el modelo para detectar enfermedades en otros tipos de plantas aumentaría su utilidad práctica.

• Implementar un sistema de feedback de usuarios para mejorar continuamente el modelo con datos del mundo real.

9. Bibliografía

Kaggle (2020) *Plant Pathology 2020 – FGVC7 Challenge*. Disponible en: https://www.kaggle.com/c/plant-pathology-2020-fgvc7

Tan, M. y Le, Q. (2019) *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv preprint arXiv:1905.11946. Disponible en: https://arxiv.org/abs/1905.11946

TensorFlow (2024) *Transfer learning and fine-tuning*. Disponible en: https://www.tensorflow.org/tutorials/images/transfer_learning

TensorFlow (2024) *TensorFlow.js Documentation*. Disponible en: https://www.tensorflow.org/js

Next.js (2024) *Next.js Documentation*. Disponible en: https://nextjs.org/docs

OpenCV (2024) *OpenCV Documentation*. Disponible en: https://docs.opencv.org/

Echamudi (2024) *opencv-wasm*. GitHub repository. Disponible en: https://github.com/echamudi/opencv-wasm

Vercel (2024) *Vercel Documentation*. Disponible en: https://vercel.com/docs