Clasificación de enfermedades en hojas de plantas mediante aprendizaje profundo con imágenes reales y de laboratorio

Pablo Rallo Fes

15 de julio de 2026

Resum

Aquest Treball Fi de Grau aborda la classificació de malalties en fulles de plantes mitjançant aprenentatge profund, integrant imatges de laboratori (PlantVillage), conjunts públics de camp i imatges capturades en condicions reals. Per a això s'ha desenvolupat Foliarium, una plataforma que unifica fonts heterogènies, facilita l'etiquetatge col·laboratiu, l'entrenament de models basats en CNN i el desplegament del sistema en producció. El treball se centra en un marc de treball reproducible i extensible que permet seleccionar dinàmicament fonts i classes en experiments configurables. Aquesta memòria documenta el plantejament del problema, el disseny i implementació de la solució, l'experimentació realitzada i l'avaluació dels resultats.

Paraules clau: classificació de plantes, malalties de les plantes, aprenentatge profund, xarxes convolucionals, visió per computador, Foliarium

Resumen

Este Trabajo Fin de Grado aborda la clasificación de enfermedades en hojas de plantas mediante aprendizaje profundo, integrando imágenes de laboratorio (PlantVillage), conjuntos públicos de campo e imágenes capturadas en condiciones reales. Para ello se ha desarrollado Foliarium, una plataforma que unifica fuentes heterogéneas, facilita el etiquetado colaborativo, el entrenamiento de modelos basados en CNN y el despliegue del sistema en producción. El trabajo se centra en un marco de trabajo reproducible y extensible que permite seleccionar dinámicamente fuentes y clases en experimentos configurables. Esta memoria documenta el planteamiento del problema, el diseño e implementación de la solución, la experimentación realizada y la evaluación de los resultados.

Palabras clave: clasificación de plantas, enfermedades de plantas, aprendizaje profundo, redes convolucionales, visión por computadora, Foliarium

Abstract

This Bachelor's Thesis addresses the classification of plant leaf diseases using deep learning, integrating laboratory images (PlantVillage), public field datasets and images captured under real-world conditions. To this end, Foliarium has been developed, a platform that unifies heterogeneous sources, supports collaborative labeling, trains CNN-based models and deploys the system in production. The work focuses on a reproducible and extensible framework that allows dynamic selection of sources and classes in configurable experiments. This thesis documents the problem statement, the design and implementation of the solution, the experiments carried out and the evaluation of the results.

Key words: plant classification, plant diseases, deep learning, convolutional neural networks, computer vision, Foliarium

Índice general

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación y contexto del problema

La detección temprana de enfermedades en cultivos es un factor clave para reducir pérdidas de producción, limitar el uso innecesario de fitosanitarios y mejorar la sostenibilidad de la actividad agrícola. En los últimos años, las técnicas de visión por computador y aprendizaje profundo han demostrado un alto rendimiento en la clasificación de imágenes de hojas bajo condiciones controladas, especialmente con conjuntos de datos de gran escala como PlantVillage. No obstante, la transferencia de esos modelos a escenarios reales sigue siendo un reto abierto, como se analiza en el capítulo de estado del arte.

Este Trabajo de Fin de Grado surge en un contexto de colaboración entre el Grupo de Tecnología Informática e Inteligencia Artificial (GTIIA) del VRAIN de la Universitat Politècnica de València y la Facultad de Ciencias Agrarias y del Medio Natural (agrónomos). La idea fue planteada por el tutor del trabajo, Carlos Carrascosa Casamayor, a partir de la necesidad detectada en ese ámbito agronómico: disponer de herramientas que permitan recopilar, unificar y explotar imágenes de hojas tomadas *in situ*, no solo en laboratorio. Esa necesidad no implica partir de cero: existen conjuntos públicos con imágenes de campo o mixtas, así como aportaciones del propio entorno agronómico; lo que suele faltar es un **marco común** que las integre bajo la misma taxonomía, con trazabilidad y posibilidad de experimentar de forma reproducible.

Al revisar el estado del arte (capítulo 2) se confirma y precisa este punto: recursos como PlantDoc, PlantCLEF o competiciones en Kaggle **sí aportan** imágenes reales o parcialmente realistas, pero están **fragmentados, son heterogéneos** entre sí y no existe un **estándar único** que combine escala, calidad de anotación y representatividad de campo de forma comparable a PlantVillage. Esta situación dificulta la comparación reproducible de métodos y el entrenamiento sobre subconjuntos coherentes de datos procedentes de varias fuentes.

En lugar de abordar únicamente el entrenamiento de un clasificador sobre un dataset fijo, el proyecto se orienta a construir **Foliarium** como plataforma de integración y gestión de múltiples fuentes heterogéneas: importación y normalización de datasets externos, captura colaborativa mediante la aplicación, experimentación configurable y despliegue centralizado del sistema.

1.1.1. Marco COSASS

El trabajo se enmarca parcialmente en el proyecto de investigación **COSASS** (*COordinated intelligent Services for Adaptive Smart areaS*), financiado en el marco de la convocato-

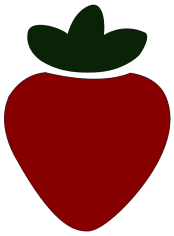


Figura 1.1: Logotipo o icono de la aplicación Foliarium (cliente multiplataforma).

ria estatal de proyectos de I+D+i. Su objetivo general es abordar la coordinación distribuida y la toma de decisiones en entornos adversos mediante dispositivos IoT en el *edge*, con especial atención a **áreas inteligentes adaptativas** en zonas rurales: entornos con conectividad limitada, restricciones energéticas y necesidad de autonomía ante contingencias habituales en el campo [9].

En la UPV, COSASS se desarrolla en el GTIIA-VRAIN con Carlos Carrascosa como investigador principal. Entre sus líneas de trabajo figuran gemelos digitales, simulación y aplicación de inteligencia artificial en entornos rurales; el propio proyecto contempla casos de uso relacionados con el ámbito agrícola (por ejemplo, viticultura). Foliarium no implementa por sí misma toda la arquitectura cloud-edge de COSASS, pero contribuye con un componente software —recopilación de datos, etiquetado, experimentación y despliegue de modelos— alineado con la necesidad de disponer de servicios inteligentes en contextos agrícolas reales.

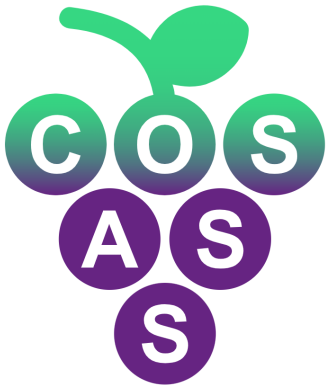


Figura 1.2: Logotipo del proyecto COSASS (contexto de investigación).

1.1.2. Marco de la Cátedra Planeta (UPV)

El desarrollo del sistema también se ha vinculado a la **Cátedra de Cooperación y Desarrollo Sostenible-Eje Planeta** de la UPV, creada en el marco de un convenio entre la universidad y la Generalitat Valenciana e impulsada en el contexto de la Agenda 2030 [10]. La cátedra, adscrita al Instituto de Ingeniería del Agua y Medio Ambiente (IIAMA), promueve actividades de investigación, formación y divulgación en torno a los ODS del *eje Planeta*: agua limpia y saneamiento (ODS 6), producción y consumo responsables (ODS 12), acción por el clima (ODS 13), vida submarina (ODS 14) y vida de ecosistemas terrestres (ODS 15).

Un sistema de apoyo al diagnóstico fitosanitario basado en imágenes y aprendizaje profundo puede relacionarse de forma directa con el **ODS 15** (salud de ecosistemas terrestres y uso sostenible de recursos agrícolas) y, de manera indirecta, con el **ODS 12** (optimización del uso de insumos mediante una detección más temprana y localizada de problemas en cultivos). El anexo de ODS de esta memoria desarrolla esa relación con mayor detalle.

Figura 1.3: Logotipo de la Cátedra de Cooperación y Desarrollo Sostenible–Eje Planeta (UPV).

1.2 Objetivos generales y específicos

El objetivo principal de este Trabajo de Fin de Grado es mejorar la capacidad de generalización de un modelo de clasificación de enfermedades en hojas de plantas mediante el uso combinado de imágenes obtenidas en laboratorio (PlantVillage) e imágenes tomadas en condiciones reales. Para obtener esta segunda clase de imágenes se ha desarrollado una aplicación a través de la que pueden ejecutarse las funcionalidades principales del proyecto.

Para lograr este objetivo general, se plantean los siguientes objetivos específicos:

- Implementar y evaluar un modelo de clasificación basado en **CNN multitarea** (cabezas de planta y enfermedad); en los experimentos documentados se emplea MobileNetV2 como *backbone* (capítulo 7).
- Incorporar imágenes reales y fuentes externas (p. ej. PlantDoc) al proceso de entrenamiento y validación del modelo.
- Evaluar el impacto de la inclusión de imágenes de distintas fuentes en el rendimiento del modelo, especialmente en cuanto a su capacidad de generalización.
- Aplicar *transfer learning* desde pesos ImageNet (*fine-tuning*) en el entrenamiento; la *data augmentation* queda como línea de mejora futura del pipeline (capítulos 7 y 10).
- Desarrollar un *pipeline* reproducible que integre múltiples datasets heterogéneos bajo un esquema común de clases y fuentes, compatible con la plataforma Foliarium.

1.3 Enfoque adoptado

Aunque el objetivo general de este Trabajo de Fin de Grado es abordar la clasificación automática de enfermedades en hojas de plantas, es importante subrayar que el enfoque seguido no se centra únicamente en maximizar la precisión del modelo sobre imágenes reales, al menos no en esta fase inicial.

Actualmente se dispone de un volumen muy limitado de imágenes bien clasificadas y tomadas en condiciones reales (luz natural, fondo no neutro, variedad de dispositivos,

etc.), lo cual impide extraer conclusiones sólidas mediante técnicas de aprendizaje profundo. Ante esta limitación, el enfoque adoptado en este TFG se orienta a construir un marco de trabajo robusto, extensible y reproducible que permita:

- Integrar y gestionar imágenes de distintas fuentes con facilidad.
- Estandarizar el preprocesamiento de las imágenes (conversión a escala de grises, segmentación de hojas, etc.).
- Almacenar y anotar las imágenes con metadatos estructurados en una base de datos consultable.
- Preparar el sistema para futuras ampliaciones del conjunto de datos, ya sea mediante aplicaciones móviles, datasets externos o colaboración con terceros.

Este marco está diseñado con visión a largo plazo, priorizando la escalabilidad del sistema, la heterogeneidad de fuentes y la validación futura de modelos con conjuntos de datos más representativos.

Por otro lado, se reconoce explícitamente que las imágenes reales disponibles en esta etapa pueden presentar nuevas clases, desbalance severo o condiciones significativamente distintas a las del dataset base (PlantVillage). Por tanto, las pruebas realizadas con estas imágenes deben interpretarse como exploratorias y no como una evaluación definitiva del rendimiento del modelo en escenarios reales.

En resumen, este TFG busca aportar valor no solo mediante experimentos concretos, sino también a través de la construcción de una base técnica que facilite el desarrollo futuro de soluciones más generalizables y útiles en contextos agrícolas reales.

1.4 Alcance del trabajo y limitaciones

1.4.1. Alcance

Este TFG abarca el diseño, implementación y despliegue de Foliarium como plataforma integrada, incluyendo:

- API REST (Flask), base de datos MongoDB y aplicación cliente multiplataforma (Flet).
- Flujos de subida, validación, administración de usuarios y taxonomía de clases.
- Pipeline de importación y preprocesado de imágenes (color, escala de grises, segmentación).
- Sistema de experimentos configurables (`config.yaml`) y entrenamiento de modelos CNN multitarea (implementación actual: MobileNetV2).
- Predicción desde la aplicación mediante la API del servidor de la UPV (inferencia centralizada, no en el dispositivo móvil).

Queda dentro del alcance la experimentación con PlantVillage, la integración de al menos un dataset externo de campo (PlantDoc) y, de forma exploratoria, imágenes capturadas mediante la aplicación.

1.4.2. Limitaciones

- **Volumen de datos reales**: el número de imágenes de campo validadas es aún insuficiente para extraer conclusiones definitivas sobre generalización; las pruebas con datos reales deben interpretarse como exploratorias.
- **Horizonte temporal del corpus**: construir un dataset suficientemente representativo de imágenes *in situ* —con validación agronómica, equilibrio entre clases y cobertura de cultivos— es un proceso **largo** que exige la colaboración de usuarios, agrónomos, etiquetadores e ingeniería de datos. En el plazo del TFG, el aporte más sólido a corto plazo es el **marco operativo** (Foliarium desplegado y utilizable); alcanzar el objetivo final de un modelo fiable en campo requiere que esa plataforma permanezca **activa y en uso** durante un tiempo suficiente para acumular y validar imágenes reales.
- **Alcance del modelo**: el sistema se centra en clasificación por imagen de hoja; no aborda detección de objetos, segmentación semántica avanzada ni diagnóstico agronómico integral (síntomas en otras partes de la planta, condiciones edafoclimáticas, etc.).
- **Validación agronómica**: la corrección de las etiquetas en imágenes reales depende del workflow de etiquetadores y de la colaboración con perfiles agrónomos; no se ha realizado un estudio de campo exhaustivo con expertos en el marco de este TFG.
- **Alcance de plataforma**: clientes iOS, publicación en Google Play y versión web completa quedan como líneas futuras; las versiones operativas actuales son Windows, Linux y Android empaquetados, con backend centralizado.

1.5 Impacto esperado

Desde el punto de vista **técnico**, Foliarium aporta un marco reproducible para integrar fuentes heterogéneas de imágenes, entrenar modelos con configuraciones versionadas y desplegar el sistema en producción. Esto reduce la fricción entre recopilación de datos, experimentación y uso operativo del clasificador.

Desde el punto de vista **aplicado**, la plataforma facilita que agrónomos, estudiantes o colaboradores externos contribuyan con imágenes reales y que datasets públicos dispersos se integren bajo un esquema común, reduciendo la fricción entre recopilación, normalización y experimentación. Incluso con datos limitados en algunas fuentes, el sistema ya permite demostrar el flujo completo de trabajo y preparar estudios futuros con mayor volumen de imágenes.

En el marco de **COSASS** y de la **Cátedra Planeta**, el trabajo contribuye a materializar capacidades de IA aplicada a entornos agrícolas y rurales, en coherencia con objetivos de sostenibilidad y de uso eficiente de recursos en el sector primario.

1.6 Metodología general de trabajo

El desarrollo del TFG ha seguido un enfoque iterativo que combina ingeniería de software y experimentación con aprendizaje profundo:

Financiado por la Unión Europea NextGenerationEU

1. **Análisis del problema y revisión bibliográfica**: identificación de la brecha entre datasets de laboratorio y condiciones reales (capítulo 2), y análisis sistemático del problema con la solución elegida (capítulo 3).
2. **Diseño e implementación de la plataforma**: arquitectura cliente-servidor, API, aplicación Flet e integración con MongoDB (capítulos 4 y 5). Parte del código partió de un desarrollo previo, continuado y unificado en un único backend.
3. **Preparación de datos**: importación de PlantVillage y de fuentes externas (p. ej. PlantDoc), preprocesado (grayscale, segmentación) y registro en base de datos; captura progresiva de imágenes reales mediante la app (capítulo 6).
4. **Modelado y experimentación**: definición de experimentos mediante `config.yaml`, entrenamiento CNN multitarea y evaluación de la mezcla de fuentes (capítulos 7 y 8); los formatos Grayscale y Segmentado están disponibles en la base de datos pero no se evaluaron en los entrenamientos documentados.
5. **Validación y despliegue**: pruebas funcionales por rol, containerización con Docker y despliegue persistente en el servidor de la universidad (capítulo 9).

La metodología prioriza la **reproducibilidad** (configuraciones versionadas, scripts de importación y entrenamiento documentados) y la **extensibilidad** (esquema de metadatos flexible, posibilidad de añadir fuentes y clases sin rediseñar la plataforma). Dado el limitado volumen de imágenes reales, las conclusiones sobre generalización se formulan con cautela y se dejan abiertas líneas de trabajo futuro (capítulo 10).

1.7 Convenciones de la memoria y repositorio del proyecto

Foliarium no es únicamente la plataforma desplegada (base de datos, API y clientes), sino también un **repositorio de software** versionado en GitHub (`https://github.com/PabloRalloFes/RepoPlantas`), donde se encuentran el código fuente, los scripts, las plantillas de experimentos, la documentación técnica (`docs/`) y las convenciones de carpetas citadas a lo largo de la memoria. Cualquier persona interesada puede clonar el repositorio, reproducir el entorno de desarrollo, ampliar fuentes de datos o modificar la implementación sobre la misma base.

1.7.1. Convenciones tipográficas

En la redacción de esta memoria se han adoptado las siguientes convenciones:

- **Tipografía monoespaciada** (`\texttt{}`): nombres de ficheros, rutas de carpetas, scripts, endpoints de la API, parámetros de configuración, colecciones de MongoDB y fragmentos literales de código o comandos (p. ej. `scripts/upload_images.py`, `config.yaml`, `/subir_imagen`).
- **Cursiva** (`\textit{}`): términos en inglés aceptados en el contexto técnico cuando no tienen traducción establecida o cuando se citan como concepto (p. ej. *domain shift*, *fine-tuning*, *pipeline*).
- **Texto normal**: nombres de productos, datasets o instituciones (PlantVillage, PlantDoc, MongoDB, Foliarium).

Cuando se menciona «el repositorio», «un script» o «la carpeta `experiments/`», se alude siempre a rutas relativas a la raíz de ese repositorio Git, salvo que se indique explícitamente el servidor de producción.

1.8 Estructura de la memoria

La memoria se organiza en diez capítulos que siguen un hilo discursivo de tipo *problema → diseño → implementación → datos y modelo → evaluación*:

- **Capítulos 1–3** plantean el contexto, los objetivos, el estado del arte y el análisis del problema con la solución adoptada.
- **Capítulos 4–5** describen el diseño de Foliarium y su implementación (API, aplicación cliente y herramientas auxiliares).
- **Capítulos 6–7** abordan el pipeline de datos y la metodología de entrenamiento del modelo.
- **Capítulos 8–9** presentan los experimentos, la validación funcional y el despliegue en producción.
- **Capítulo 10** recoge las conclusiones, limitaciones y líneas de trabajo futuro.

Los apéndices incluyen la guía de usuario (instalación y uso por rol), instrucciones para reproducir experimentos y desplegar el sistema, fragmentos de código de referencia, material gráfico complementario, el listado de requisitos del sistema y la descripción del marco generalizado derivado de Foliarium (apéndice F).

1.9 Colaboradores y agradecimientos

Quiero también expresar mi agradecimiento personal a quienes han hecho posible este Trabajo de Fin de Grado dentro del entorno colaborativo GTIIA–agrónomos de la Universitat Politècnica de València.

A **Carlos Carrascosa Casamayor**, tutor del trabajo, por la orientación, la revisión continua del proyecto y la coordinación con el ámbito agronómico. A **Jaime Carlavilla Lázaro**, por el desarrollo previo sobre el que se construye Foliarium y las bases técnicas heredadas. A **Marta Rallo Fes**, por su colaboración en el proyecto a nivel de desarrollo visual y diseño del logo.

Asimismo, agradezco a las personas que han descargado los clientes de la aplicación, la han utilizado contra el servidor de producción y han aportado comentarios sobre su funcionamiento y su usabilidad. Ese feedback informal, junto con la encuesta publicada en la landing, ha servido para comprobar que el sistema es utilizable en la práctica más allá del entorno de desarrollo.

Por último, reconozco el apoyo institucional del VRAIN, de la Facultad de Ciencias Agrarias y del Medio Natural, y el marco de los proyectos COSASS y la Cátedra Planeta, en los que se inscribe en parte este trabajo.

CAPÍTULO 2
Marco teórico y estado del arte

2.1 Introducción al capítulo

Este capítulo revisa los fundamentos de la clasificación de enfermedades en hojas mediante aprendizaje profundo, el estado del arte en conjuntos de datos y modelos, y las aplicaciones y plataformas existentes. A continuación se formula una crítica de las limitaciones actuales y se sitúa la contribución de Foliarium como plataforma de integración de fuentes heterogéneas, en coherencia con la motivación expuesta en el capítulo 1.

2.2 Fundamentos de aprendizaje profundo aplicados a fitopatología

La detección automática de enfermedades en plantas mediante visión por computador ha crecido de forma notable impulsada por las redes neuronales convolucionales (CNN) y la disponibilidad de conjuntos de imágenes anotadas. En fitopatología, la tarea se formula habitualmente como clasificación de imágenes de hojas: cada fotografía se asigna a una clase que combina especie y estado de salud o tipo de enfermedad.

2.2.1. Referencia base: Mohanty et al. y el *domain shift*

Uno de los trabajos más influyentes en este ámbito es el de Mohanty et al. [1], que emplea AlexNet y GoogLeNet sobre PlantVillage, un dataset con más de 50.000 imágenes capturadas en condiciones controladas (iluminación uniforme, fondo neutro). Los autores alcanzan precisiones superiores al 99 % en laboratorio, pero reportan una caída drástica —hasta aproximadamente el 30 %— al evaluar con imágenes de campo. Este fenómeno, conocido como *domain shift* o cambio de dominio, sigue siendo una referencia obligada al abordar la generalización fuera del laboratorio y constituye el hilo conductor de la revisión bibliográfica de este TFG.

2.2.2. Transferencia de aprendizaje y técnicas de generalización

Diversos trabajos han aplicado *transfer learning*, *data augmentation* y adaptación de dominio para reducir la brecha entre datos de entrenamiento y condiciones reales [7, 8]. Ferentinos [7] demuestra que arquitecturas profundas (VGG y variantes de AlexNet) pueden alcanzar alto rendimiento sobre PlantVillage con entrenamiento desde cero o transferencia. Sladojevic et al. [8] reportan resultados prometedores en un conjunto reducido de clases de campo, aunque con limitaciones de escala y reproducibilidad. Estas técnicas

mitigan el desajuste entre dominios, pero no lo eliminan por completo cuando las fuentes son muy heterogéneas. En Foliarium se ha aplicado *transfer learning* en los experimentos del capítulo 8; la *data augmentation* no se integró en el pipeline de entrenamiento en esta fase (capítulo 7).

2.2.3. Arquitecturas de clasificación

En la literatura agrícola se emplean con frecuencia ResNet, EfficientNet, DenseNet y arquitecturas ligeras como MobileNet [2]. Las redes ligeras se diseñaron para inferencia en dispositivos con recursos limitados; las arquitecturas más profundas suelen orientarse a maximizar precisión cuando el cómputo no es restrictivo. En el contexto de Foliarium, la predicción se ejecuta en el **servidor** mediante la API (`/predict_image`), por lo que la elección de arquitectura debe valorarse en función de precisión, tiempo de inferencia en servidor y facilidad de *fine-tuning*, no únicamente por eficiencia en móvil. La implementación actual utiliza MobileNetV2 multitarea (capítulo 7); otras arquitecturas podrían evaluarse en el mismo pipeline sin modificar la plataforma.

2.2.4. Formulación multiclase y multitarea

PlantVillage y trabajos similares suelen plantear un problema **multiclase** con una etiqueta por imagen (`Planta___Enfermedad`). Foliarium adopta además una formulación **multitarea** con cabezas separadas para planta y enfermedad, lo que facilita restricciones en predicción (planta conocida) y la reutilización parcial del catálogo de clases. Esta decisión de diseño se desarrolla en el capítulo 7.

2.3 Estado del arte: conjuntos de datos y modelos

2.3.1. Panorama de conjuntos de datos públicos

Existen múltiples conjuntos de datos relevantes para la clasificación de enfermedades y plagas en plantas. No obstante, difieren en realismo de captura, escala, balanceo, taxonomía y objetivo específico. La tabla 2.1 resume las características principales de los más citados en la literatura reciente.

Tabla 2.1: Comparativa orientativa de conjuntos de datos públicos en fitopatología por imagen.

Dataset	Realismo	Escala	Balanceo	Notas
PlantVillage	Laboratorio	Muy alta	Desbalanceado	Referencia estándar; fondo controlado [1].
PlantDoc	Web / campo mixto	Media	Desbalanceado	Imágenes web curadas alrededor de clases PlantVillage; heterogeneidad visual [3] (detalle cap. 6).
PlantCLEF	Variable	Muy alta	Heterogéneo	Retos de identificación de especies/plagas; diversidad taxonómica [4].
IP102	Campo / lab	Muy alta	Desbalanceado	102 clases de plagas; objetivo distinto (insectos) [5].
Plant Pathology 2020	Campo	Alta	Competición	Manzano; enfermedades específicas; Kaggle [6].
New Plant Diseases (Kaggle)	Mixto	Alta	Variable	Muy usado en tutoriales; calidad y licencia heterogéneas.

PlantVillage sigue siendo la referencia experimental principal de este TFG por su escala, reproducibilidad y comparabilidad con Mohanty et al. **PlantDoc** [3] aporta imágenes web curadas alrededor de las clases de PlantVillage, con mayor variabilidad de fondo e iluminación que el laboratorio, aunque sin ser un muestreo homogéneo de campo (capítulo 6). Es la fuente externa prioritaria para integración en Foliarium. **PlantCLEF** [4] y **IP102** [5] ilustran la diversidad de objetivos (identificación de especies vs. plagas) y la dificultad de unificar taxonomías. Los conjuntos de **Kaggle** (Plant Pathology 2020 [6], New Plant Diseases Dataset) amplían el panorama pero requieren mapeo cuidadoso de clases y verificación de licencias.

La conclusión de esta revisión es que **sí existen datasets con imágenes reales o parcialmente realistas**, pero están **fragmentados y son heterogéneos**. No hay un único estándar público que cubra de forma completa el problema de clasificación de enfermedades en condiciones reales con la misma coherencia que PlantVillage en laboratorio.

La Figura 2.1 reproduce el contraste visual entre clases de PlantVillage (laboratorio) y PlantDoc (web/campo) tal como lo ilustran Singh et al. [3] en su dataset (figura adaptada del repositorio oficial del proyecto).

Figura 2.1: Comparativa PlantVillage (PVD) frente a PlantDoc para varias clases, adaptada de Singh et al. [3] (repositorio `PlantDoc-Dataset`).

2.3.2. Trabajos de clasificación con aprendizaje profundo

Más allá de Mohanty et al., la literatura reciente confirma tanto el potencial como los límites de las CNN en este dominio:

- **Sladojevic et al.** [8] entrenan una CNN personalizada sobre un conjunto reducido de enfermedades de campo, alcanzando alta precisión en condiciones controladas de experimentación pero con escasa generalización demostrada a otras fuentes.
- **Ferentinos** [7] compara VGG y AlexNet sobre decenas de enfermedades de PlantVillage, estableciendo un referente de rendimiento en condiciones de laboratorio con arquitecturas profundas.
- **Singh et al.** [3] introducen PlantDoc y evalúan modelos entrenados en PlantVillage sobre imágenes de campo, cuantificando de nuevo la caída de rendimiento por *domain shift*.
- Trabajos con **EfficientNet y ResNet** sobre PlantDoc y conjuntos derivados de Kaggle reportan mejoras al combinar *fine-tuning*, aumento de datos y entrenamiento multi-fuente, aunque con resultados difíciles de comparar por la falta de protocolo común entre estudios.

En conjunto, estos trabajos muestran que el problema sigue activo: altas precisiones en dominios controlados y degradación en campo, especialmente cuando no se dispone de un pipeline unificado para mezclar fuentes con taxonomías distintas.

2.4 Estado del arte: sistemas y aplicaciones

2.4.1. Aplicaciones móviles y comerciales

Existen diversas aplicaciones orientadas al diagnóstico fitosanitario asistido por imagen, entre las que destacan:

- **Plantix**: aplicación ampliamente difundida que permite fotografiar cultivos y obtener sugerencias de diagnóstico; utiliza modelos propios entrenados con datos no públicos.
- **PlantVillage Nuru**: herramienta asociada al ecosistema PlantVillage, orientada a agricultores en países en desarrollo; combina captura móvil con modelos de clasificación, con cobertura limitada a ciertos cultivos.
- **Crop Doctor, Leaf Doctor** y aplicaciones similares: ofrecen clasificación de enfermedades en hojas desde el móvil, generalmente con modelos entrenados para un conjunto cerrado de clases y sin posibilidad de reconfigurar experimentos o integrar nuevas fuentes.

Estas soluciones demuestran la viabilidad del diagnóstico asistido en entornos reales, pero comparten limitaciones: **datasets de entrenamiento privados**, dominio fijo, escasa reproducibilidad académica y nula capacidad de combinar fuentes heterogéneas o lanzar experimentos comparativos bajo control del usuario.

2.4.2. Arquitecturas de despliegue: inferencia local frente a cliente-servidor

Las aplicaciones anteriores suelen optar por inferencia en el dispositivo (modelos optimizados para móvil) o por servicios en la nube propietarios. Foliarium adopta una arquitectura **cliente-servidor**: la aplicación Flet (Windows, Linux, Android) actúa como cliente delgado que captura y sube imágenes; el entrenamiento y la predicción se ejecutan en el backend desplegado en la UPV. Esta decisión simplifica la distribución de modelos y centraliza la experimentación, aunque requiere conectividad y relativa la importancia de arquitecturas ultra-ligeras en el dispositivo.

2.4.3. Plataformas académicas y pipelines

Muchos trabajos publican modelos entrenados o notebooks de experimentación, pero **pocos sistemas integran** de forma explícita recopilación colaborativa, etiquetado, importación de datasets externos, configuración de experimentos versionados y despliegue operativo. Foliarium se sitúa en ese espacio intermedio entre un artículo experimental puntual y una aplicación comercial cerrada.

2.5 Crítica al estado del arte

A partir de la revisión anterior, se identifican las siguientes limitaciones estructurales:

- **Falta de generalización**: modelos entrenados en laboratorio degradan su rendimiento en campo (*domain shift*), como demuestra Mohanty et al. [1] y estudios sobre PlantDoc [3].

- **Fragmentación de datasets**: existen conjuntos reales o parcialmente realistas, pero con taxonomías, metadatos, licencias y niveles de balanceo incompatibles entre sí.
- **Ausencia de un estándar unificado**: no hay un dataset único que combine escala, calidad de anotación y realismo de captura de forma comparable a PlantVillage.
- **Sistemas poco flexibles**: aplicaciones comerciales y muchos pipelines académicos están acoplados a un dominio fijo y no permiten seleccionar dinámicamente fuentes o clases para entrenar.
- **Reproducibilidad limitada**: el uso de datos privados en productos comerciales impide validar y comparar resultados de forma abierta.
- **Falta de plataformas de integración**: escasean entornos que unifiquen importación, normalización, experimentación configurable y captura colaborativa bajo un mismo esquema de datos.

Estas limitaciones no invalidan el avance en modelos de clasificación, pero desplazan el cuello de botella hacia la **gestión y unificación de fuentes heterogéneas** y hacia entornos controlados de experimentación reproducible.

2.6 Propuesta y posicionamiento de Foliarium

Frente al estado del arte descrito, este TFG propone Foliarium como **plataforma de integración, estructuración y explotación** de múltiples fuentes de imágenes de hojas, más que como un único modelo optimizado para un dataset fijo. La contribución principal recae en la infraestructura software:

- **Esquema unificado de metadatos**: colecciones `Fuente`, `Clases` y `Docs` en MongoDB, con convención de carpetas `data/{fuente}/color/{clase}/` para importación.
- **Integración de fuentes heterogéneas**: PlantVillage (laboratorio), PlantDoc y otros datasets públicos tras normalización de clases; capturas colaborativas mediante la aplicación (fuente «App»); importaciones ad hoc (p. ej. datos de vid).
- **Experimentación configurable**: selección dinámica de fuentes, formatos (color, grayscale, segmented), plantas y enfermedades mediante `config.yaml` y filtros sobre la base de datos.
- **Despliegue operativo**: API REST, contenedorización Docker y servicio en producción en la UPV; predicción centralizada accesible desde los clientes Flet.
- **Modelo intercambiable**: la implementación actual (MobileNetV2 multitarea) es una primera versión del pipeline de entrenamiento e inferencia; la plataforma está preparada para sustituir o comparar arquitecturas sin rediseñar el flujo de datos (capítulos 7, 8 y 10).

De este modo, Foliarium aborda la brecha laboratorio–campo no solo entrenando un clasificador, sino proporcionando el entorno en el que combinar progresivamente fuentes controladas y reales, disponer de variantes de preprocesado (Color, Grayscale, Segmentado) para evaluaciones futuras y desplegar modelos en un sistema accesible para usuarios, etiquetadores y administradores. El análisis del problema, las alternativas consideradas y la solución elegida se desarrollan en el capítulo 3; el pipeline de datos en el capítulo 6; la metodología de entrenamiento en el capítulo 7.

Financiado por la Unión Europea NextGenerationEU España | digital 2026 Cátedra ENIA UPV VRAIN nunsys GROUP

CAPÍTULO 3
Análisis del problema

3.1 Introducción

Tras la revisión del estado del arte, queda identificada una brecha principal: no es tanto la ausencia total de imágenes de campo como la **falta de un entorno integrado** que permita unificar fuentes heterogéneas, experimentar de forma reproducible y desplegar modelos en un flujo operativo accesible a distintos perfiles de usuario. Este capítulo analiza el problema desde esa perspectiva, evalúa alternativas de solución y justifica la creación de Foliarium.

3.2 Formulación del problema y oportunidades

3.2.1. Problema a resolver

El problema que aborda este TFG puede descomponerse en tres niveles:

1. **Generalización**: los modelos entrenados en condiciones de laboratorio degradan su rendimiento en campo, como demuestra la literatura de referencia [1, 3].
2. **Fragmentación de datos**: existen datasets públicos con imágenes reales o parcialmente realistas (PlantDoc, conjuntos de Kaggle, etc.), pero con taxonomías, metadatos y licencias incompatibles entre sí; no hay un estándar unificado comparable a PlantVillage.
3. **Brecha operativa**: aplicaciones comerciales y muchos trabajos académicos ofrecen clasificadores puntuales, pero no un **ciclo completo** —recopilación colaborativa, validación, importación de fuentes externas, experimentación versionada y despliegue— en un mismo sistema reproducible.

El objetivo del TFG no es únicamente maximizar la precisión de un modelo sobre un dataset fijo, sino **diseñar e implementar la infraestructura** que permita abordar los tres niveles anteriores de forma incremental.

3.2.2. Actores y escenarios de uso

El sistema está pensado para un entorno de colaboración universidad–agrónomos, con los siguientes actores:

- **Usuario colaborador**: captura o sube imágenes de hojas desde la aplicación cliente y las asocia a una clase del catálogo; puede solicitar experimentos y usar modelos publicados para obtener predicciones. Además, si tiene acceso a un conjunto de datos ya etiquetados, puede implementarlas a la base de datos en un solo proceso.
- **Etiquetador**: revisa imágenes pendientes de validación, corrige clases y marca etiquetas de imágenes como validadas antes de usarlas en entrenamientos exigentes.
- **Administrador**: gestiona usuarios y roles, supervisa entrenamientos, decide qué modelos quedan disponibles para predicción y mantiene la taxonomía de clases.
- **Operador técnico** (desarrollador o administrador del servidor): importa datasets externos (PlantVillage, PlantDoc, etc.) mediante scripts o la aplicación, monitoriza el despliegue Docker y el backend en producción.

Los escenarios de uso principales son: (1) recopilación de imágenes individuales o masivas; (2) validación y control de calidad del etiquetado; (3) definición y ejecución de experimentos filtrando por fuente, formato y clases; (4) predicción sobre imágenes nuevas; (5) importación por lotes de fuentes externas. Estos procesos se pueden llevar a cabo en una versión local o contra el servidor de producción (`https://plantas.gti-ia.upv.es`), donde está la base de datos centralizada.

3.2.3. Oportunidades de innovación

A partir del análisis anterior, se identifican las siguientes oportunidades, alineadas con el posicionamiento de Foliarium en el capítulo 2:

- **Plataforma de integración multi-fuente**, con esquema común de metadatos (`Fuente`, `Clases`, `Docs`) y pipeline de normalización (capítulo 6).
- **Experimentación reproducible** mediante `config.yaml` y carpetas versionadas en `experiments/`.
- **Captura colaborativa** de imágenes reales como fuente adicional («App»), complementaria a datasets públicos.
- **Despliegue operativo** en servidor, de modo que el sistema pase de ser un prototipo local a ser utilizable por colaboradores externos.
- **Encaje institucional** con líneas COSASS y Cátedra Planeta (capítulo 1), como componente software de recopilación y explotación de datos agrícolas.

3.3 Identificación y análisis de soluciones posibles

No existe una única forma de abordar el problema formulado. Se han considerado las alternativas siguientes:

3.3.1. Alternativa A: entrenar un clasificador sobre PlantVillage

Consiste en desarrollar un notebook o script que entrene un modelo CNN sobre PlantVillage y evalúe su rendimiento, opcionalmente probando PlantDoc u otro dataset de campo de forma aislada.

- **Ventajas**: mínimo coste de desarrollo software, foco puro en el modelo, resultados comparables con la literatura.
- **Inconvenientes**: no resuelve la integración de fuentes ni la recopilación colaborativa, cada nuevo dataset exige adaptar scripts manualmente, no hay despliegue ni roles de usuario, difícil reutilización por agrónomos o colaboradores, carencia de innovación.

3.3.2. Alternativa B: adoptar una aplicación comercial existente

Herramientas como Plantix o PlantVillage Nuru ya ofrecen captura móvil y diagnóstico asistido.

- **Ventajas**: solución inmediata para el usuario final, modelos entrenados con datos propios a gran escala.
- **Inconvenientes**: datos y modelos no reproducibles académicamente, imposibilidad de configurar experimentos, combinar fuentes públicas o extender la taxonomía, dependencia de un proveedor externo.

3.3.3. Alternativa C: plataforma cliente-servidor de integración (Foliarium)

Desarrollar una plataforma propia con API REST, base de datos, cliente multiplataforma, pipeline de importación, experimentos configurables e inferencia centralizada en el servidor UPV.

- **Ventajas**: cubre el ciclo completo de trabajo, permite unificar PlantVillage, PlantDoc, capturas propias y otras fuentes, experimentación versionada, despliegue bajo control de la universidad, extensible a futuras integraciones.
- **Inconvenientes**: mayor esfuerzo de ingeniería, el rendimiento del modelo depende del volumen y calidad de datos disponibles, mantenimiento del servidor y de la aplicación, menor dedicación a la optimización de la predicción.

3.3.4. Criterios de selección y decisión

La tabla 3.1 resume los criterios aplicados para elegir la alternativa.

Tabla 3.1: Criterios de selección entre alternativas de solución.

Criterio	A: solo modelo	B: app comercial	C: Foliarium
Integración multi-fuente	Baja	Nula	Alta
Reproducibilidad académica	Media	Baja	Alta
Recopilación colaborativa	No	Sí (cerrada)	Sí (propia)
Despliegue institucional	No	No	Sí
Esfuerzo de desarrollo	Bajo	Nulo	Alto
Alineación con objetivos	Parcial	Baja	Alta

Se adopta la **alternativa C (Foliarium)** porque cumple la restricción central del TFG: ir más allá de un experimento puntual de clasificación y aportar una **infraestructura extensible** para unificar fuentes, validar datos y desplegar modelos. El entrenamiento del clasificador pasa a ser un *componente* dentro de la plataforma, no el único entregable.

3.4 Solución propuesta

3.4.1. Descripción general

Foliarium es una plataforma cliente-servidor que integra:

- **Cliente Flet** (Windows, Linux, Android): app que gestiona captura y subida de imágenes, etiquetado, administración, creación de experimentos, consulta de resultados y predicción (vía API).
- **Backend Flask + MongoDB**: autenticación por roles, gestión de metadatos e imágenes, orquestación de entrenamientos e inferencia en el servidor.
- **Pipeline de datos y scripts**: importación de fuentes externas, preprocesado (color, grayscale, segmentación) y registro en base de datos.
- **Módulo de experimentación**: configuración YAML, entrenamiento CNN multitarea y almacenamiento de resultados.

3.4.2. Fases del desarrollo y validación

El TFG se ha organizado en fases coherentes con los capítulos de la memoria:

1. **Análisis y diseño** (capítulos 1–4): contexto, estado del arte, análisis del problema y arquitectura.
2. **Implementación de la plataforma** (capítulo 5): API, cliente Flet, autenticación y herramientas auxiliares.
3. **Datos y modelo** (capítulos 6–7): pipeline de fuentes, preprocesado, entrenamiento e inferencia.
4. **Evaluación** (capítulos 8–9): experimentos, validación funcional por rol y despliegue Docker en producción.
5. **Síntesis** (capítulo 10): conclusiones, limitaciones y trabajo futuro.

La **validación** del sistema combina: (i) pruebas funcionales por rol (subida, etiquetado, administración, experimentos, predicción); (ii) comprobación del despliegue persistente y acceso HTTPS; (iii) experimentos de clasificación sobre subconjuntos configurables de datos; (iv) revisión exploratoria con imágenes de campo cuando estén disponibles. Un listado detallado de capacidades del sistema (requisitos funcionales y no funcionales de referencia) se recoge en el apéndice E.

3.5 Análisis de seguridad

El backend implementa las medidas siguientes:

- **Autenticación JWT**: tras `/iniciar_sesion`, el cliente incluye un token Bearer en las peticiones protegidas. La API valida expiración, firma e integridad del token antes de ejecutar la operación.

- **Control por rol**: operaciones administrativas (gestión de usuarios, asignación de roles, etc.) exigen el rol `admin`. Otros endpoints sensibles pueden restringirse mediante configuración (`AUTH_REQUIRED_ENDPOINTS`, `ROLE_REQUIRED_ENDPOINTS`).
- **Protección de contraseñas**: el registro y el cambio de contraseña almacenan un hash bcrypt en la base de datos `Usuarios` (`utils/auth.py`). El cliente envía la contraseña en texto plano únicamente durante la petición HTTP; en producción el transporte va cifrado por HTTPS y el servidor es el único componente que aplica el hash antes de persistir.
- **Verificación TLS en el cliente**: el módulo `logicav3.py` verifica el certificado del servidor cuando la URL apunta a producción (`https://plantas.gti-ia.upv.es`), pero la desactiva en desarrollo local (`https://localhost`, certificados autofirmados). La decisión se centraliza en `resolve_verify_ssl()`.
- **Rate limiting**: los endpoints de login y registro limitan el número de peticiones por IP (10 por minuto) para mitigar ataques de fuerza bruta.
- **Validación de entradas**: las imágenes subidas deben ser JPEG o PNG y no superar un tamaño máximo configurable (`MAX_IMAGE_SIZE_MB`, por defecto 10 MB).
- **CORS**: orígenes configurables (`CORS_ORIGINS`) para desarrollo y un futuro cliente web; los clientes nativos actuales (Windows, Linux, Android) no lo requieren.
- **Transporte seguro**: HTTPS en producción (`https://plantas.gti-ia.upv.es`); certificados autofirmados en desarrollo local.
- **Token JWT en URL (decisión de diseño)**: además del encabezado `Authorization: Bearer`, la API admite el JWT como parámetro de consulta (`access_token` o `token`) en endpoints que sirven imágenes. Esto permite abrir enlaces autenticados desde componentes del cliente Flet que delegan la visualización al navegador o al visor del sistema (p. ej. ampliación de imagen en Android). El riesgo teórico —registro del token en historial o logs— se mitiga en producción con HTTPS y caducidad del JWT.

Como criterio de calidad funcional, una imagen registrada por la aplicación debe quedar trazable (usuario, fuente, formato, clase) y, si procede, pasar por validación antes de utilizarse en entrenamientos configurados con la opción `solo_validadas`.

3.6 Análisis de eficiencia y coste computacional

3.6.1. Entrenamiento e inferencia

El entrenamiento de modelos y la inferencia (`/predict_image`) se ejecutan en el **servidor** de la UPV, no en el dispositivo del usuario. Implica:

- **Entrenamiento**: coste computacional concentrado en el servidor (CPU/GPU disponible en la máquina de producción o de desarrollo); duración dependiente del tamaño del subconjunto seleccionado, épocas y estrategia de *fine-tuning*. No se ha realizado un perfilado exhaustivo del consumo eléctrico; el criterio práctico ha sido ejecutar experimentos fuera de horas punta cuando el entrenamiento es prolongado.

Financiado por la Unión Europea NextGenerationEU

- **Inferencia**: una petición de predicción implica carga del modelo en memoria, preprocesado de la imagen y paso *forward*; el coste por petición es bajo frente al entrenamiento, pero escala con el número de usuarios concurrentes.
- **Cliente móvil**: la aplicación Android actúa como cliente delgado (captura, subida, visualización); el consumo de batería en móvil se asocia sobre todo a la cámara y la transferencia de datos, no al modelo de aprendizaje profundo. La elección inicial de MobileNetV2 respondía a inferencia local; dado el despliegue servidor-side, arquitecturas más pesadas podrían valorarse en el servidor sin penalizar la batería del dispositivo (capítulo 10).

3.6.2. Adquisición y almacenamiento de datos

La subida de imágenes (base64 en JSON) y su persistencia en MongoDB más disco supone un coste de almacenamiento creciente con el volumen de fuentes integradas. Los scripts de importación masiva permiten procesar lotes en el servidor sin intervención manual repetitiva, optimizando el tiempo del operador frente a subidas individuales.

3.7 Marco legal y ético

3.7.1. Protección de datos

Foliarium registra **datos personales** de usuarios (credenciales, identificador de usuario asociado a cada imagen subida) e **imágenes** que pueden tomarse en entornos agrícolas reales. En un despliegue institucional en la UPV deben considerarse:

- **Minimización y finalidad**: solo se recogen datos necesarios para autenticación, trazabilidad de imágenes y operación del sistema de clasificación con fines académicos e investigadores.
- **Almacenamiento**: metadatos en MongoDB e imágenes en el sistema de archivos del servidor universitario; acceso restringido mediante autenticación y roles.
- **Responsabilidad y normativa**: en un uso extendido más allá del ámbito del TFG, sería necesario formalizar el tratamiento conforme al RGPD y a la política de protección de datos de la UPV (registro de actividades, base legal, información a usuarios, plazos de conservación). En la fase actual del proyecto, el acceso está limitado a entorno controlado y colaboradores identificados.
- **Cesión de imágenes**: las fotografías aportadas por usuarios deben subirse con conocimiento de su uso con fines de investigación y mejora del clasificador.

3.7.2. Propiedad intelectual y licencias de datos

- **Software**: el código del TFG se desarrolla en el marco académico de la UPV; la titularidad y posibles licencias de explotación deben alinearse con la normativa de la universidad y con los acuerdos del proyecto COSASS cuando aplique.
- **Datasets públicos**: PlantVillage, PlantDoc y conjuntos de Kaggle están sujetos a sus propias licencias y condiciones de uso; la importación en Foliarium debe respetar redistribución y atribución. Las imágenes generadas por usuarios de la plataforma son aportaciones originales cuya licencia de uso conviene definir antes de publicar un corpus derivado.

- **Modelos preentrenados**: MobileNetV2 utiliza pesos de PyTorch/ImageNet con las condiciones de uso de la librería correspondiente.

3.7.3. Consideraciones éticas

Un sistema de apoyo al diagnóstico fitosanitario plantea dilemas que deben explicitarse:

- **No sustitución del criterio experto**: las predicciones del modelo son orientativas; una decisión agronómica o fitosanitaria debe validarse por personal cualificado, especialmente en condiciones de campo donde el *domain shift* degrada el rendimiento.
- **Errores de clasificación**: falsos negativos (no detectar una enfermedad) pueden retrasar tratamientos; falsos positivos pueden inducir aplicaciones innecesarias de fitosanitarios. El workflow de validación por etiquetadores mitiga errores en *entrenamiento*, pero no en *inferencia* sobre imágenes nuevas.
- **Sesgos**: desbalanceo entre clases, predominio de cultivos presentes en PlantVillage y escasez de imágenes reales validadas pueden sesgar el modelo hacia clases mayoritarias o condiciones de laboratorio. No se han identificado atributos sensibles (raza, sexo, etc.) en las imágenes de hojas, pero sí **sesgo de dominio** por fuente y dispositivo de captura.
- **Impacto en sostenibilidad**: un uso responsable del sistema puede contribuir a reducir fitosanitarios innecesarios (ODS 12 y 15, capítulo 1); un uso descuidado del diagnóstico automático podría tener el efecto contrario.

3.8 Análisis de riesgos

La tabla 3.2 recoge los riesgos principales identificados durante el análisis.

Tabla 3.2: Análisis de riesgos del proyecto Foliarium.

Riesgo	Impacto	Consecuencia	Medidas
Bajo volumen de imágenes reales validadas	Alto	Conclusiones limitadas sobre generalización	Integrar PlantDoc y otras fuentes; captura colaborativa; validación por etiquetadores
Errores de predicción en campo	Alto	Decisión agronómica incorrecta	Documentar limitaciones; no presentar el sistema como diagnóstico definitivo; revisión por expertos
Rechazo o baja adopción por usuarios	Medio	Pocos datos reales aportados	Interfaz por rol; despliegue accesible; pruebas con colaboradores (cap. 9)
Fallo o indisponibilidad del servidor	Medio	Interrupción del servicio	Docker, health checks, despliegue en UPV
Heterogeneidad al importar datasets	Medio	Clases mal mapeadas, experimentos inconsistentes	Pipeline de normalización (cap. 6); script PlantDoc; validación manual
Desviación de esfuerzo hacia ingeniería vs. modelo	Medio	Menos tiempo para experimentación ML	Priorizar plataforma como objetivo explícito del TFG; experimentos mínimos documentados
Integración futura con COSASS	Bajo	Incompatibilidad de interfaces	Diseño modular API + metadatos; línea de trabajo futuro

3.9 Plan de trabajo y presupuesto orientativo

3.9.1. Plan de trabajo (estimación y desviaciones)

El desarrollo se ha abordado de forma iterativa. La tabla 3.3 resume fases, estimación inicial orientativa y observaciones a posteriori.

Tabla 3.3: Plan de trabajo del TFG (estimación retrospectiva).

Fase	Horas aprox.	Observaciones
Revisión bibliográfica y análisis	60–80	Incluye profundización sobre el estado del arte y los datasets disponibles
Diseño e implementación API + BBDD	100–120	Parte basada en desarrollo previo (Jaime Carlavilla)
Cliente Flet multiplataforma	150–200	Mayor esfuerzo en empaquetado Android/Windows/Linux
Pipeline de datos e importación	60–80	PlantVillage, scripts, preprocesado
Modelo y experimentos	80–100	Entrenamientos en servidor; carpeta `experiments/`
Docker y despliegue UPV	60–80	Dominio HTTPS, distribución de clientes
Memoria y documentación	60–80	Redacción continua en paralelo
Total orientativo	**500–750**	Desviaciones: el empaquetado Flet y el despliegue superaron la estimación inicial

La principal **desviación** respecto al planteamiento inicial fue el esfuerzo en despliegue, empaquetado multiplataforma e infraestructura, frente a un foco más estrecho en el modelo móvil. Esta desviación es coherente con la solución elegida (alternativa C) y con el cambio a inferencia centralizada en servidor.

3.9.2. Presupuesto de recursos

- **Hardware**: PC de desarrollo del alumno; servidor de la UPV para producción (sin coste directo para el TFG); dispositivos de prueba (PC, móvil Android).
- **Software**: stack mayoritariamente abierto (Python, Flask, MongoDB, PyTorch, Flet, Docker); sin licencias comerciales significativas.
- **Datos**: datasets públicos gratuitos (PlantVillage, PlantDoc, etc.); coste de almacenamiento en servidor institucional.
- **Recursos humanos**: esfuerzo principal del alumno; tutoría de Carlos Carrascosa Casamayor; trabajo previo de Jaime Carlavilla Lázaro; colaboraciones puntuales (etiquetado, pruebas).

En conjunto, el coste monetario directo para el alumno es bajo; el principal recurso invertido es **tiempo de ingeniería** para construir la plataforma, coherente con la naturaleza del entregable.

Financiado por la Unión Europea NextGenerationEU

CAPÍTULO 4
Diseño y arquitectura de la solución

4.1 Visión global del sistema

Foliarium se concibe como una plataforma integrada que cubre el ciclo completo de trabajo con imágenes de hojas de plantas: recopilación y etiquetado, almacenamiento estructurado, entrenamiento de modelos de clasificación, evaluación experimental y predicción sobre imágenes nuevas. El diseño separa de forma clara el **cliente** (aplicación de usuario), el **servidor** (API y lógica de negocio) y la **persistencia** (base de datos y sistema de archivos), de modo que distintos clientes —escritorio, móvil o scripts— puedan operar sobre el mismo backend.

A nivel conceptual, el sistema combina dos líneas de actividad que se retroalimentan:

- **Línea de plataforma**: gestión de usuarios, subida y validación de imágenes, administración de clases y metadatos, y operación del sistema en producción.
- **Línea de experimentación**: preparación de conjuntos de datos, configuración y ejecución de entrenamientos, almacenamiento de modelos y análisis de resultados.

Los componentes principales del repositorio son los siguientes:

- `main_app.py`: aplicación gráfica multiplataforma desarrollada con Flet (Windows, Linux y Android).
- `logicav3.py`: capa de cliente que encapsula la comunicación HTTP con la API y el estado de la sesión.
- `main.py`: API REST implementada con Flask; concentra la lógica de negocio, la autenticación y la integración con el modelo de datos.
- `utils/`: módulos compartidos de acceso a base de datos, autenticación, modelo de aprendizaje profundo y entrenamiento.
- `scripts/`: utilidades de mantenimiento, importación masiva de imágenes, configuración inicial de la base de datos y operaciones por lotes.
- Almacenamiento en disco de imágenes (`imagenes/`), modelos entrenados (`models/`) y resultados de experimentos (`experiments/`).

En el entorno de producción de la universidad, el backend está desplegado de forma persistente y accesible mediante `https://plantas.gti-ia.upv.es`, mientras que los clientes se distribuyen como aplicaciones empaquetadas o mediante la página de descargas del propio servidor.

La Figura 4.1 resume la distribución de componentes y el flujo de comunicación entre clientes, servidor y capas de persistencia. Los scripts y notebooks pueden ejecutarse **desde el entorno del servidor** (p. ej. `docker compose exec api`) con acceso directo al repositorio y volúmenes, o **desde una máquina de desarrollo** invocando la misma API REST que usa el cliente Flet.

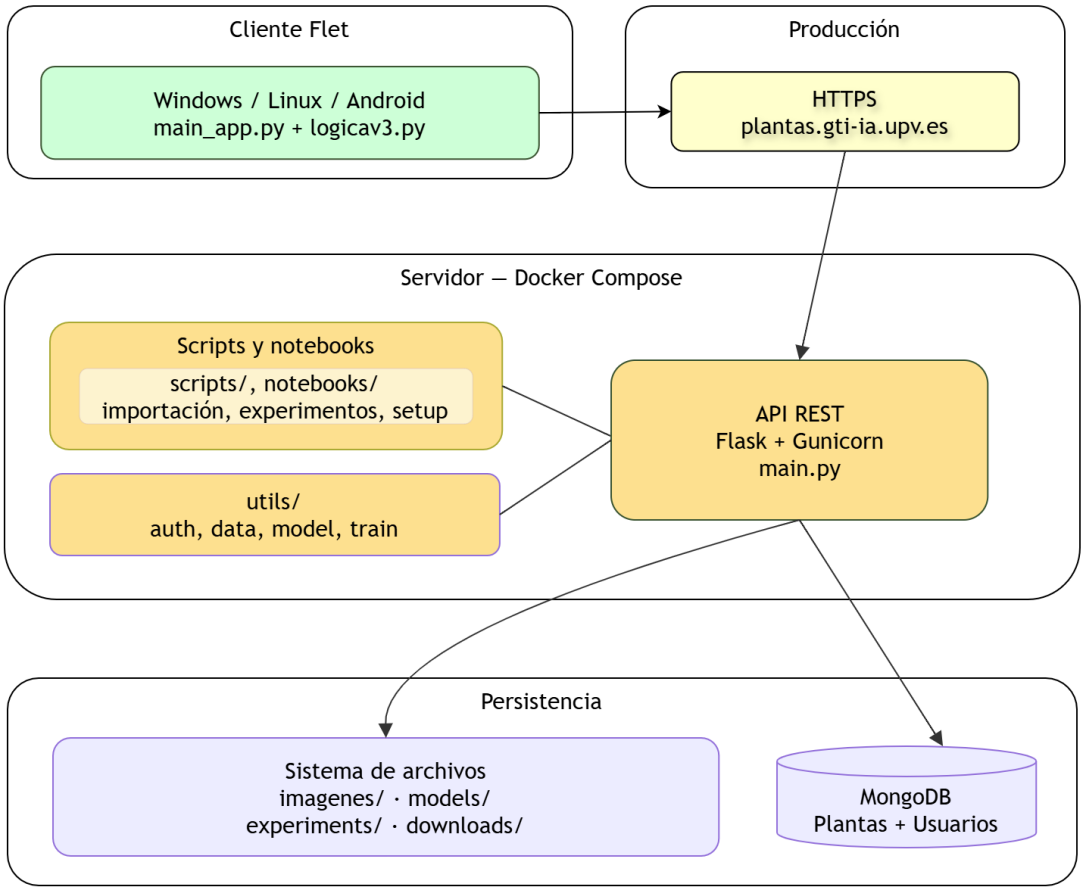


Figura 4.1: Arquitectura global de Foliarium: clientes, API centralizada y persistencia híbrida (MongoDB + disco).

4.2 Arquitectura backend, cliente y servicios auxiliares

4.2.1. API Flask (`main.py`)

El servidor backend expone una API REST monolítica que centraliza todas las operaciones del sistema. Entre sus responsabilidades se incluyen: gestión de usuarios y roles, registro y consulta de imágenes, validación y clasificación manual, administración de taxonomías (plantas, enfermedades, fuentes y formatos), creación y seguimiento de experimentos, entrenamiento de modelos y predicción sobre imágenes individuales.

La API se apoya en los módulos de `utils/` para conectar con MongoDB, cargar configuraciones de experimentos, construir el modelo CNN multitarea (implementación ac-

tual: MobileNetV2) y ejecutar inferencia en el servidor. En producción, el servicio se ejecuta bajo Gunicorn dentro de un contenedor Docker; en desarrollo local puede arrancarse con `run_server.py`, con soporte opcional de HTTPS mediante certificados autofirmados.

4.2.2. Cliente Flet (`main_app.py`)

La interfaz de usuario no es una aplicación web tradicional, sino un cliente Flet empaquetable para distintas plataformas. La aplicación organiza la experiencia en vistas diferenciadas según el rol del usuario (*usuario*, *etiquetador*, *administrador* y variantes con permisos ampliados). Desde el cliente se realizan las operaciones habituales del sistema: inicio de sesión, subida de imágenes, etiquetado, gestión de usuarios, creación de experimentos, consulta de resultados y predicción con modelos entrenados.

La URL del backend es configurable desde la propia aplicación, lo que permite conectar tanto a una instancia local de desarrollo como al servidor de producción de la universidad.

4.2.3. Scripts y utilidades auxiliares

Complementariamente a la API y a la aplicación gráfica, el repositorio incluye scripts en `scripts/` para tareas que no requieren interacción directa del usuario final: inicialización de la base de datos (`setup_bbdd.py`), importación masiva de imágenes, preprocesamiento (conversión a escala de grises, segmentación de hojas), creación de experimentos y comparación de resultados. Estos scripts pueden invocar la **API REST** (como el cliente Flet) o ejecutarse **dentro del contenedor del servidor** con acceso directo a MongoDB y al sistema de archivos montado; comparten las mismas convenciones de datos en ambos casos.

4.3 Diseño de la comunicación entre aplicación y API

La comunicación entre el cliente y el servidor sigue un esquema cliente-servidor basado en HTTP con intercambio de mensajes JSON. El módulo `logicav3.py` implementa la clase `LogicaApp`, que actúa como fachada de acceso a la API: construye las URLs de los endpoints, serializa las peticiones, gestiona los tokens de autenticación y mantiene en memoria el estado de la sesión activa (usuario, rol seleccionado, imágenes en curso, etc.).

El flujo típico de una operación es el siguiente:

1. El usuario interactúa con la interfaz Flet (`main_app.py`).
2. La interfaz delega en `LogicaApp` la preparación de la petición (por ejemplo, codificación de una imagen en base64 antes de subirla).
3. `LogicaApp` envía la petición HTTP al endpoint correspondiente de la API.
4. La API valida la autenticación y los permisos del rol, ejecuta la operación sobre MongoDB o el sistema de archivos, y devuelve una respuesta JSON.
5. El cliente interpreta la respuesta y actualiza la interfaz.

La autenticación se basa en tokens JWT. Tras un inicio de sesión correcto, el cliente incluye el token en el encabezado `Authorization: Bearer ...` en las peticiones a endpoints protegidos. En operaciones que construyen una URL de imagen para abrirse fuera

del cliente HTTP habitual (descarga o visor externo), el mismo token puede añadirse como parámetro de consulta; la API lo acepta como alternativa al encabezado (véase el análisis de seguridad en el capítulo 3). La API aplica control de acceso por rol mediante un filtro previo a la ejecución de cada endpoint sensible.

4.4 Modelo de datos y persistencia en MongoDB

La persistencia del sistema se apoya en MongoDB y en almacenamiento de ficheros en disco. Se distinguen dos bases de datos principales:

- `Plantas`: almacena las imágenes, su metadatos, las clases, las etiquetas dinámicas y la información asociada a experimentos y entrenamientos.
- `Usuarios`: almacena las credenciales y los roles de los usuarios del sistema.

Dentro de `Plantas`, las colecciones más relevantes son:

- `Docs`: documentos de imagen con metadatos (clase, fuente, formato, usuario, estado de validación, rutas o referencias a la imagen codificada).
- `Clases`: taxonomía de plantas y enfermedades utilizada en la clasificación.
- `Fuente`: catálogo de orígenes de las imágenes (por ejemplo, PlantVillage, aplicación móvil o importaciones externas).
- `Campos`: definición del esquema extensible de metadatos; permite asociar colecciones de etiquetas a campos personalizados.
- `Entrenamientos`: solicitudes y estado de entrenamientos de modelos iniciados desde la aplicación.
- Colecciones de etiquetas auxiliares (por ejemplo, formatos de imagen) referenciadas dinámicamente desde `Campos`.

La Figura 4.2 detalla las colecciones, sus atributos principales y las referencias entre documentos. Las flechas sólidas indican claves foráneas almacenadas en `Docs`; las discontinuas, relaciones de configuración o trazabilidad lógica (no siempre materializadas como `ObjectId`).

4.4.1. Elección de MongoDB y criterios de diseño

Foliarium persiste metadatos de imágenes con estructura **heterogénea y evolutiva**: distintas fuentes (PlantVillage, PlantDoc, App), formatos (color, grayscale, segmented), campos auxiliares configurables y filtros dinámicos en experimentos. Se eligió **MongoDB** como base de datos documental porque:

- El modelo de **documentos JSON** encaja con metadatos semi-estructurados y con la serialización ya empleada en la API Flask.
- La colección `Campos` (inicializada desde `src/campos.json`) permite **extender el esquema** —nuevas etiquetas o colecciones auxiliares— sin migraciones rígidas de tablas relacionales.

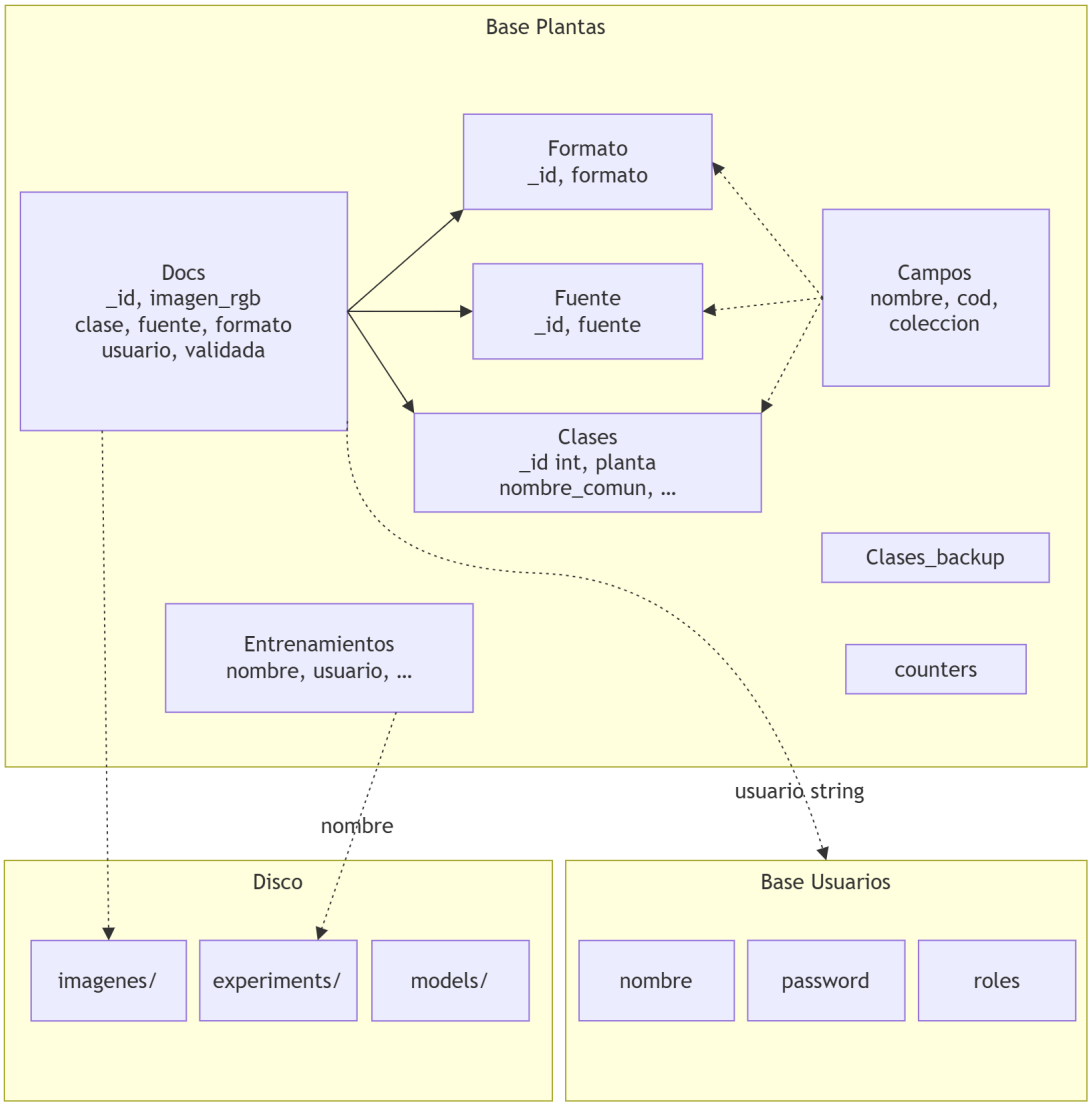


Figura 4.2: Esquema lógico de MongoDB en Foliarium: dos bases (`Plantas` y `Usuarios`), referencias en `Docs` y persistencia híbrida con el sistema de archivos.

- Las consultas de experimentación combinan filtros sobre varios atributos (`fuente`, `formato`, `clase`, `validada`, etc.) de forma natural sobre documentos `Docs`.
- El ecosistema Python (`pymongo`) se integra de forma directa con el resto del stack del proyecto.

Entre las alternativas consideradas, un **SGBD relacional** (p. ej. PostgreSQL) resulta adecuado para usuarios y credenciales, pero más rígido para el catálogo extensible de `Campos` y para evolucionar taxonomías sin rediseñar tablas. Almacenar todas las imágenes en la base de datos (**GridFS** o blobs embebidos) simplificaría el backup lógico unificado, pero penalizaría consultas masivas y el tamaño de la base; por ello se descartó frente al diseño híbrido siguiente.

Las credenciales y roles se aislaron en la base `Usuarios`, separada de `Plantas`, para acotar permisos y copias de seguridad: la información de autenticación no se mezcla con el volumen creciente de documentos de imagen.

4.4.2. Persistencia híbrida: MongoDB y sistema de archivos

Las imágenes en sí se almacenan en el sistema de archivos del servidor (directorio `imagenes/`), mientras que en la base de datos se guardan los metadatos y, en muchos casos, la imagen codificada en base64 o la URL pública de acceso. Los modelos entrenados se guardan en `models/` y los artefactos de cada experimento (configuración, métricas, gráficos) en subcarpetas de `experiments/`. Este diseño híbrido facilita la consulta estructurada de metadatos sin sobrecargar la base de datos con ficheros binarios masivos; la función `prepare_data_splits` (capítulo 6) consulta MongoDB para construir los conjuntos de entrenamiento a partir de esos metadatos, independientemente de la ruta concreta en disco.

4.5 Consideraciones de despliegue

El sistema está pensado para ejecutarse de forma containerizada mediante Docker Compose, con servicios separados para la API y MongoDB. En el entorno de producción, un reverse proxy termina TLS y expone el servicio bajo un dominio HTTPS; los clientes Flet se conectan a esa URL pública. El detalle del despliegue —composición de contenedores, variables de entorno, volúmenes persistentes y validación en el servidor de la universidad— se desarrolla en el capítulo de validación y despliegue.

Financiado por la Unión Europea NextGenerationEU

CAPÍTULO 5
Implementación de Foliarium

5.1 Backend Flask y endpoints principales

La API implementada en `main.py` agrupa la funcionalidad del sistema en dominios coherentes con la solución descrita en el capítulo 3. A continuación se resume la responsabilidad de los bloques principales de endpoints (la lista completa supera cincuenta rutas):

- **Autenticación y usuarios**: `/registro`, `/iniciar_sesion`, `/cambiar_password`, `/cambiar_nombre_usuario`, `/buscar_usuarios`, `/add_rol`, `/eliminar_rol`, `/eliminar_usuario`, `/verificar_rol`.
- **Imágenes**: `/subir_imagen`, `/eliminar_imagen`, `/clasificar`, `/validar_imagen`, `/imagen_base64/<nombre>`, `/servir_n_archivos`, `/servir_n_archivos_sin_validar`.
- **Importación masiva**: `/subida_masiva`, `/subida_masiva_zip`, `/listar_fuentes_importadas`.
- **Metadatos y taxonomía**: `/etiquetas`, `/recuperar_etiquetas/<colección>`, `/add_etiqueta`, `/editar_clases`, `/add_class`, `/opciones_*` (plantas, enfermedades, formatos, fuentes, modelos, filtros).
- **Experimentos y entrenamiento**: `/crear_experimento`, `/obtener_experimentos`, `/solicitar_entrenamiento`, `/entrenar_modelo`, `/obtener_resultados_experimento`, `/comparar_experimentos`.
- **Predicción**: `/predict_image`, `/obtener_modelos`.
- **Utilidades y acceso público**: `/`, `/health`, `/download/<platform>`, `/guia-usuario`.

La lógica de negocio accede a MongoDB a través de `utils/database.py` y delega en `utils/model.py` y `utils/train.py` las operaciones de aprendizaje profundo. Los entrenamientos solicitados desde la API pueden lanzarse como subprocesos que ejecutan el script `run_experiment.py` de la carpeta del experimento correspondiente.

5.2 Interfaz de usuario Flet

La aplicación `main_app.py` implementa la interfaz gráfica con la librería Flet. Tras el inicio de sesión, el usuario accede a una de las vistas principales según el rol seleccionado:

- **Vista de usuario**: subida de fotografías, consulta de experimentos, solicitud de entrenamientos, predicción con modelos disponibles y acceso a resultados.
- **Vista de etiquetador**: listado de imágenes no validadas con filtros, navegación entre documentos, asignación y modificación de etiquetas y acción de validación.
- **Vista de administrador**: gestión de usuarios y roles, supervisión de entrenamientos pendientes (incluida la decisión de publicar un modelo para predicción), edición de clases y operaciones de mantenimiento del catálogo.

La aplicación permite configurar la URL del backend desde el propio cliente, lo que facilita el uso en desarrollo local en caso de que el usuario lo desee. Por defecto, se conecta al servidor `https://plantas.gti-ia.upv.es`. La navegación interna se implementa con rutas (`page.route`) y una barra inferior común con acceso al inicio de cada rol. El estado de la sesión (usuario autenticado, imágenes en curso, resultados de búsqueda) se mantiene en la instancia de `LogicaApp` asociada a la ventana.

5.2.1. Capturas de pantalla representativas

No se incluyen capturas exhaustivas de todas las pantallas —la guía de usuario (apéndice A) ya documenta el uso operativo—, pero sí un **conjunto mínimo** que ilustra los flujos distintivos del sistema (Figuras 5.1–5.2):

- **Inicio de sesión y selección de rol** (Fig. 5.1): muestra la entrada al sistema y la elección entre `usuario`, `etiquetador` y `admin`, clave para entender el modelo de permisos.
- **Subida de imágenes** (Fig. 5.2): captura o selección de foto y envío al servidor; representa la adquisición colaborativa de datos.
- **Etiquetado y validación** (Fig. 5.3): vista del rol `etiquetador` con imagen pendiente, metadatos y acción de validar; representa el control de calidad del corpus.
- **Predicción con modelo publicado** (Fig. 5.4): resultado de `/predict_image` en la interfaz de usuario; cierra el ciclo «subida → entrenamiento → inferencia».

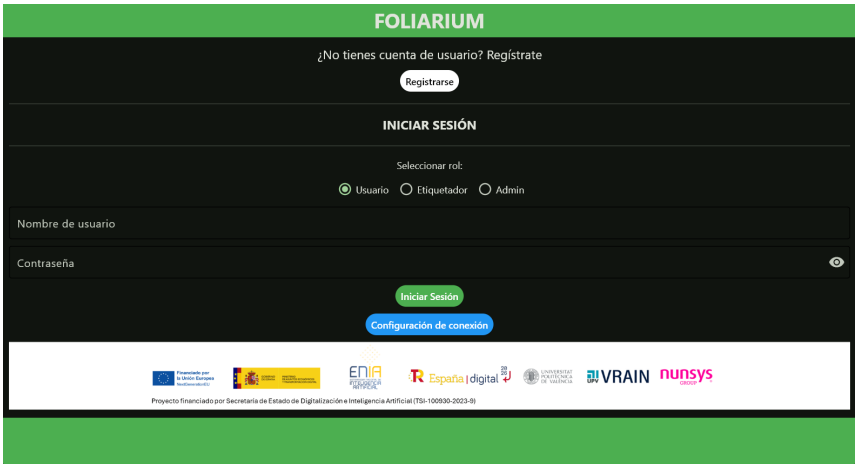


Figura 5.1: Pantalla de inicio de sesión y selección de rol en el cliente Foliarium.

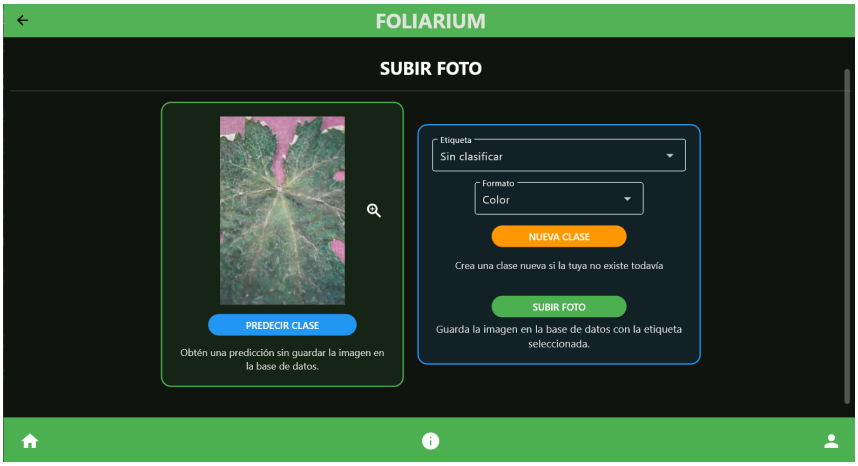


Figura 5.2: Vista de subida de imágenes desde el rol `usuario`.

Figura 5.3: Vista del rol `etiquetador`: imagen pendiente de validación con filtros y acciones de etiquetado.

Figura 5.4: Vista de predicción: planta, enfermedad y confianza devueltas por el modelo en el servidor.

5.2.2. Empaquetado multiplataforma y distribución

Los clientes finales no ejecutan la API ni MongoDB en su dispositivo: son **aplicaciones Flet empaquetadas** que se conectan al backend remoto. La generación de instalables se realiza con `flet build` sobre `main_app.py`, configurado en `pyproject.toml` (metadatos del producto, permisos de red y cámara en Android, exclusión del backend y de ficheros de desarrollo como `.env` del artefacto).

Se han producido builds para **tres plataformas**, con formatos de distribución distintos:

- **Windows**: archivo ZIP (`Foliarium-Windows.zip`) con la aplicación de escritorio generada en `build/windows/` (ejecutable o carpeta portable según el empaquetado de Flet).
- **Linux**: paquete `.deb` (`Foliarium-Linux.deb`) para instalación en distribuciones compatibles.
- **Android**: APK (`Foliarium-Android.apk`) con permisos de Internet y cámara para captura de imágenes *in situ*.

En el repositorio GitHub del proyecto (`RepoPlantas`) se mantiene una rama `main` con el código común del backend, los scripts y la lógica compartida del cliente, y **ramas dedicadas por plataforma** bajo el prefijo `platform/` —`platform/windows`, `platform/linux` y `platform/android`—. Esta organización responde a que el empaquetado con `flet build` exige ajustes distintos en cada sistema (dependencias nativas, artefactos generados, permisos en `pyproject.toml`) y, en Android, a que parte de la interfaz en `main_app.py` se adapta al formato de pantalla móvil (disposición de controles, tamaños y flujos de captura). Los cambios específicos de cada distribución se concentran principalmente en `main_app.py` y en la configuración de build; la rama `main` conserva la versión de escritorio de referencia.

La URL de la API en producción queda fijada en `logicav3.py` antes de compilar; el `.env` de desarrollo no se incluye en el build, de modo que el cliente empaquetado apunta por defecto al servidor institucional salvo que el usuario la modifique desde **Configuración de conexión** en la app.

Distribución: los instalables no se publican en tiendas de aplicaciones en esta fase del TFG. Se sirven desde la **landing pública** en la raíz del dominio, renderizada por `templates/landing.html`. Esa página enlaza la guía de usuario, la encuesta de usabilidad y, para cada plataforma, la descarga directa mediante `/download/windows`, `/download/linux` y `/download/android`. Los ficheros se almacenan en el directorio `downloads/` del servidor (montado en el contenedor `api`); los nombres concretos se configuran con las variables `DOWNLOAD_*` del `.env`. Si un archivo aún no está desplegado, la landing indica que la descarga no está disponible.

5.3 Flujo de autenticación y roles

El flujo de autenticación sigue estos pasos:

1. El usuario introduce nombre, contraseña y rol deseado en la pantalla de inicio de sesión.
2. `LogicaApp` envía una petición `POST` a `/iniciar_sesion` con nombre y contraseña (por HTTPS en producción).

3. Si la autenticación es correcta, la API devuelve un token JWT, la lista de roles del usuario y el rol activo.
4. El cliente almacena el token y lo incluye en el encabezado `Authorization` de las peticiones posteriores.
5. La API, mediante el hook `@app.before_request`, rechaza peticiones no autenticadas a endpoints protegidos y comprueba que el rol del usuario autoriza la operación solicitada.

El registro de nuevos usuarios crea una cuenta con rol `usuario` por defecto; la ampliación a `etiquetador`, `usuario+` o `admin` la realiza un administrador mediante `/add_rol`. Las operaciones sobre credenciales propias o ajenas aplican la regla «el propio usuario o un administrador».

5.3.1. Modificación de credenciales y trazabilidad del nombre

Desde la vista de administración (y, en el caso del propio perfil, con la contraseña actual), un usuario puede **cambiar su contraseña** mediante `/cambiar_password` o **cambiar su nombre de usuario** con `/cambiar_nombre_usuario`. En la interfaz Flet, ambas acciones se ofrecen en diálogos que exigen la contraseña vigente para confirmar la operación; al cambiar el nombre, también se solicita la nueva contraseña.

En MongoDB (base `Usuarios`, colección `usuarios`), las contraseñas se almacenan como **hash bcrypt**, nunca en texto plano. Cuando un usuario modifica su **nombre de acceso**, el documento se actualiza con el nuevo `nombre` y la API añade el identificador anterior al array `nombres_antiguos` mediante `$addToSet`. Este registro conserva un histórico mínimo de alias previos sin duplicar entradas, lo que puede ser útil para auditoría o para relacionar acciones antiguas en `Docs.usuario` (que guarda el nombre como `string`) con la cuenta actual, aunque en la fase del TFG no se ha implementado una migración automática de ese campo en los documentos de imagen ya subidos.

5.4 Herramientas de administración, carga y mantenimiento

Además de las funciones expuestas en la API y accesibles desde Flet, el repositorio incluye scripts en `scripts/` para operaciones de mantenimiento:

- `setup_bbdd.py`: inicialización de la base de datos y creación del usuario administrador (función `initdb` en Docker).
- `upload_images.py` y `subir_imagenes_nueva_fuente.py`: importación masiva desde carpetas bajo `data/<fuente>/`.
- `process_imported_images.py`, `convert_to_grayscale.py`, `segment_leaves.py`: generación de variantes grayscale y segmentadas.
- `make_experiment.py`: creación de la estructura de un nuevo experimento a partir de la plantilla `experiments/BASE/`.
- `compare_experiments.py`: comparación de métricas entre experimentos.
- `backup_mongo.sh`: copia de seguridad de las bases `Plantas` y `Usuarios` (véase capítulo 9).

Desde la propia aplicación, el administrador puede crear copias de seguridad del catálogo de clases y restaurar configuraciones, así como editar y completar metadatos de las clases sin modificar directamente los ficheros JSON del repositorio en tiempo de ejecución.

5.5 Integración con modelos de predicción

La integración entre la plataforma y los modelos entrenados se articula en tres momentos:

1. **Creación del experimento**: desde el cliente o mediante `/crear_experimento`, se define un `config.yaml` con filtros de datos, hiperparámetros, estrategia de *fine-tuning* y opciones de formato de imagen.
2. **Entrenamiento**: la API ejecuta el pipeline del experimento (consulta a MongoDB, generación de CSVs, entrenamiento con PyTorch, evaluación y guardado de resultados en `experiments/<nombre>/results/`). El administrador que acepta el entrenamiento puede aprobar que el modelo resultante se copie a `models/` y quede disponible para predicción.
3. **Inferencia**: el endpoint `/predict_image` recibe la imagen en base64, el identificador del modelo y, opcionalmente, la planta conocida; carga los pesos, aplica las transformaciones de preprocesado y devuelve las predicciones de planta y enfermedad, restringidas a combinaciones válidas según el conjunto de entrenamiento.

Este flujo permite que un usuario final utilice un modelo entrenado sin interactuar con los scripts de línea de comandos, mientras que el desarrollador conserva la posibilidad de lanzar experimentos y análisis desde el propio repositorio.

CAPÍTULO 6
Datos y pipeline de preparación

6.1 Fuentes de datos y organización del repositorio

Foliarium está diseñado para trabajar con **múltiples fuentes heterogéneas** de imágenes, registradas en la colección `Fuente` de MongoDB e identificadas por un nombre de carpeta bajo `data/`. Las fuentes previstas o utilizadas en este TFG son:

- **PlantVillage**: conjunto de referencia en condiciones de laboratorio; base de los experimentos iniciales. Organización en disco: `data/plantvillage/color/`, `grayscale/` y `segmented/`, con subcarpetas por clase (`Planta___Enfermedad`).
- **PlantDoc**: dataset público de referencia para evaluar generalización fuera del laboratorio [3]; en Foliarium se importa la variante *Cropped-PlantDoc* tras normalización con `scripts/prepare_plantdoc_import.py`. Su origen y matices se detallan en el § 6.1.1.
- **App**: imágenes capturadas o subidas mediante Foliarium (fuente «App» en MongoDB); condiciones reales de captura colaborativa.
- **Otras importaciones**: datos aportados externamente. Actualmente, se ha subido un conjunto de imágenes aportado por la facultad de agrónomos **VRAIN Viticultura Enologia** (122 imágenes de vid en campo). Tanto el frontend como el backend están perparados para la integración de conjuntos de otras fuentes, siempre partiendo de una carpeta `data/<fuente>/color/planta___enfermedad`.

En la fase de redacción de esta memoria, los experimentos documentados combinan principalmente **PlantVillage** con integración progresiva de **PlantDoc** y, de forma exploratoria, imágenes de la **App**. Otras fuentes (PlantCLEF, IP102, Plant Pathology) son compatibles con el pipeline pero quedan como línea de trabajo futuro por la complejidad del mapeo taxonómico.

6.1.1. Origen de PlantVillage y PlantDoc

PlantVillage [1] es el conjunto de referencia en la literatura de clasificación fitosanitaria por visión. Sus imágenes se obtuvieron en **condiciones de laboratorio**: hojas individuales sobre fondo uniforme (habitualmente blanco o negro), iluminación controlada y encuadre centrado. El dataset original agrupa decenas de miles de fotografías organizadas por especie y enfermedad; en Foliarium se importa bajo la fuente `PlantVillage` con la convención de carpetas descrita arriba.

PlantDoc [3] fue creado por Singh et al. (IIT Gandhinagar, India) y presentado en CoDS-COMAD 2020 como respuesta a la limitación de PlantVillage en escenarios no controlados. **No es un levantamiento sistemático de campo**: los autores descargaron del orden de **20 900 imágenes** desde buscadores web (Google Images y Ecosia), utilizando los nombres científicos y comunes de las **38 clases** de PlantVillage como consultas. Cuatro anotadores filtraron el material según metadatos de la web y criterios apoyados en la literatura fitopatológica de APSNet (color, extensión y densidad de la lesión, morfología de la hoja, etc.), descartaron duplicados, imágenes fuera de alcance o sin hoja visible, y **eliminaron explícitamente fotografías de apariencia «de laboratorio»** durante la curación. Cada imagen fue revisada por **dos personas**; las clases con menos de 50 ejemplos se excluyeron. El conjunto final publicado contiene **2 598 imágenes** en 13 especies y 27 clases (enfermedad/sano), con un esfuerzo estimado de ≈300 horas-persona de anotación.

Además, los autores delimitaron hojas con **cajas delimitadoras** (herramienta LabelImg) y generaron la variante **Cropped-PlantDoc** (C-PD), recortando cada hoja anotada —9 216 recortes en total— para acercar el formato al de PlantVillage sin perder la variabilidad de fondo e iluminación del material web. Es esta variante recortada la que Foliarium importa tras mapear sus etiquetas al catálogo `Planta___Enfermedad`.

A pesar de la intención de los autores de capturar condiciones «reales», la **procedencia web** introduce una heterogeneidad difícil de controlar: conviven imágenes tomadas *in situ* (fondos naturales, iluminación variable, hojas en la planta) con otras de aspecto más estudiado (fondos neutros, composiciones tipo catálogo) e incluso **marcas de agua** o artefactos de compresión (Figura 6.1). Esta mezcla —observable al inspeccionar visualmente las imágenes importadas— explica que PlantDoc se describa a menudo como «de campo» sin ser un corpus homogéneo de fotografías agrícolas. En la Tabla 6.1 se resume como **campo y web/laboratorio parcial**, no como muestreo puro de parcela.

Figura 6.1: Ejemplos de imágenes PlantDoc con marcas de agua visibles (`Apple_scab`, mancha foliar en `Pepper_bell`).

6.1.2. Fuentes incorporadas en el servidor de producción

Tras la importación masiva realizada en el servidor (`plantas.gti-ia.upv.es`), el repositorio de entrenamiento queda compuesto por las fuentes siguientes. Los nombres deben coincidir **exactamente** con el campo `fuente` registrado en MongoDB al filtrar en `config.yaml` (`fuentes: [PlantVillage, PlantDoc, ...]`).

Tabla 6.1: Características de las fuentes de datos importadas (formato Color).

Fuente (MongoDB)	Entorno	Alcance taxonómico	Vol. aprox.	Validación
PlantVillage	Laboratorio	Catálogo PlantVillage completo (multi-cultivo; clases `Planta___Enfermedad`)	∼54 000	No (carga masiva)
PlantDoc	Web / campo mixto	Subconjunto mapeado al catálogo PlantVillage (C-PD); train+test unificados	∼2 600	No
VRAIN Viticultura Enologia	Campo	Solo *vid* (`Grape`); varias enfermedades del ámbito vitivinícola	122	No
App	Captura colaborativa	Según uso de la plataforma	Variable	Parcial

PlantVillage [1] aporta la escala y la referencia de comparabilidad bibliográfica: imágenes con fondo uniforme, iluminación controlada y alta resolución relativa. Es la base principal para experimentos de línea base.

PlantDoc [3] introduce *domain shift* respecto a PlantVillage: fondos naturales o variables, distintos dispositivos y calidades de imagen, pero también restos de material web (fondos neutros, marcas de agua). Tras `prepare_plantdoc_import.py`, las carpetas originales del dataset se renombran a la convención `Planta___Enfermedad`; solo se importan clases presentes en el mapeo del script o en el catálogo `Clases` (véase § 6.1.1).

VRAIN Viticultura Enologia es un aporte externo reducido pero relevante para el contexto agronómico del proyecto (viticultura, COSASS/VRAIN): todas las imágenes corresponden a vid y cubren distintas patologías. Su volumen limitado impide entrenar modelos solo con esta fuente, pero permite evaluar comportamiento en un cultivo con pocas muestras reales. Dadas las diferencias entre las enfermedades recogidas en este conjunto y los 2 anteriores, se mantiene como fuente adicional sin incluir en los experimentos.

En los experimentos iniciales de este TFG se utiliza únicamente el formato **Color**; las variantes `grayscale` y `segmented` pueden generarse con `process_imported_images.py` y subirse en fases posteriores. La mayoría de imágenes importadas permanecen con `validada=false`; el flag `solo_validadas` del config permite restringir el entrenamiento a imágenes revisadas por etiquetadores cuando exista suficiente volumen validado.

Las imágenes en producción residen en el servidor (`imagenes/` y documentos en MongoDB), no en el repositorio de código, que en desarrollo puede contener solo datos de prueba.

6.1.3. Normalización de fuentes externas

Los datasets públicos no comparten la nomenclatura `Planta___Enfermedad` de PlantVillage. Para la integración en Foliarium se han de seguir estos pasos:

1. **Descarga y revisión** del dataset original (licencia, estructura de carpetas, etiquetas).
2. **Mapeo de clases**: tabla de correspondencia entre etiquetas del dataset externo e identificadores del catálogo `Clases` en MongoDB (p. ej. «Corn_Gray_leaf_spot» en PlantDoc → `Corn___Cercospora_leaf_spot_Gray_leaf_spot`). Las clases no pre-

sentes en el catálogo pueden añadirse con `scripts/add_class.py`, cosa que la subida masiva hace automáticamente.

3. **Reorganización**: copia de imágenes a `data/{fuente}/color/{clase_mapeada}/` mediante `scripts/prepare_plantdoc_import.py` u otro script análogo para una fuente distinta.

4. **Preprocesado común**: generación de `grayscale/` y `segmented/` con `scripts/process_imported_images.py`.

5. **Registro en base de datos**: subida con `scripts/upload_images.py` o `scripts/subir_imagenes_nueva_fuente.py`, creando la entrada en `Fuente` si no existe.

6. **Experimentos**: filtrado por fuente en `config.yaml` (`fuentes: [plantdoc]`) para entrenar y evaluar sobre subconjuntos configurables.

Este flujo garantiza que fuentes con distinto origen compartan el mismo esquema de metadatos (`fuente`, `formato`, `clase`, `validada`) y el mismo pipeline de preprocesado, requisito para comparar formatos de imagen y combinar fuentes en un mismo experimento.

6.2 Taxonomía de plantas y enfermedades

La taxonomía del sistema se modela en la colección `Clases` de MongoDB. Cada documento de clase incluye, entre otros campos, la **planta**, la **clasificación** (por ejemplo «Carencia» o «Virus») y el **nombre común** de la enfermedad o estado. Este esquema permite tanto la visualización en la interfaz (formato «Planta, Clasificación, Enfermedad») como la formulación multitarea del modelo (predicción separada de planta y enfermedad). Los campos `clasificacion` y `nombre_cientifico` están previstos en la colección, pero en la mayoría de casos permanecen vacíos a la espera de una revisión por expertos agrónomos (véase trabajo futuro, capítulo 10).

En la fase de inicialización del proyecto, el catálogo se carga a partir de ficheros JSON en `src/` (por ejemplo `clases.json`, `clases_combinadas.json`), que reflejan la nomenclatura de PlantVillage (`Planta___Enfermedad`). El identificador numérico de clase (`_id`) se utiliza como referencia en los documentos de la colección `Docs`.

Las etiquetas auxiliares —fuentes, formatos y campos personalizados— se gestionan mediante la colección `Campos`, que declara qué colección de MongoDB almacena cada tipo de etiqueta y permite extender el esquema sin modificar el código de la API.

6.3 Limpieza, validación e importación de imágenes

6.3.1. Entrada desde la aplicación

Cuando un usuario sube una imagen desde Flet, el cliente la codifica en base64 y la envía a `/subir_imagen` junto con el identificador de clase y metadatos (`fuente`, `formato`, `usuario`). Por defecto, las imágenes capturadas desde la app se registran en color con la fuente asociada a la aplicación. El documento creado en `Docs` queda inicialmente sin validar hasta que un etiquetador confirme la clase mediante `/validar_imagen`.

6.3.2. Importación masiva desde disco

Para fuentes externas, el pipeline de importación espera la siguiente estructura de carpetas:

```
data/{fuente}/color/{clase}/imagen.jpg
```

donde `{clase}` sigue la convención de nombres de PlantVillage (por ejemplo `Tomato___Early_blight`).

El script `scripts/process_imported_images.py` genera, a partir de las imágenes en color, las carpetas `grayscale/` y `segmented/` invocando `convert_to_grayscale.py` y `segment_leaves.py`. A continuación, `upload_images.py` recorre las imágenes del formato indicado, las codifica y las registra en MongoDB mediante `POST /subir_imagen`, creando la fuente en `Fuente` si no existe. El script `subir_imagenes_nueva_fuente.py` automatiza el procesamiento y la subida en los tres formatos.

Para evitar duplicados en importaciones interrumpidas, `upload_images.py` mantiene registros de imágenes ya subidas por fuente y formato, y los guarda en archivos .txt en la carpeta /logs. La API también expone `/subida_masiva` y `/subida_masiva_zip` para cargas iniciadas desde el cliente con permisos adecuados.

6.3.3. Validación como control de calidad

El campo `validada` en `Docs` distingue imágenes revisadas por un etiquetador. Los experimentos pueden configurarse con `solo_validadas: true` en `config.yaml` para entrenar únicamente con documentos validados, lo que actúa como mecanismo de limpieza lógica del conjunto de entrenamiento.

6.4 Análisis exploratorio de los datos (EDA)

A diferencia de un dataset tabular, cada registro de Foliarium es principalmente una **imagen** con metadatos asociados. Las variables analíticas disponibles antes del entrenamiento son, por tanto, limitadas: **fuente**, **formato** (Color, Grayscale, Segmentado), **clase** (planta y enfermedad), **estado de validación** y atributos auxiliares configurables en `Campos`. No existen variables numéricas continuas por imagen más allá de dimensiones o tamaño de fichero, que no se han explotado en este TFG.

6.4.1. Distribución de clases y fuentes

El análisis exploratorio más útil consiste en cuantificar **cuántas imágenes hay por clase y por fuente**, detectar desbalanceo y comparar la cobertura taxonómica entre PlantVillage, PlantDoc, App y otras importaciones. La Figura 6.2 muestra la frecuencia de imágenes por clase agrupada por fuente (datos del servidor de producción tras la importación documentada). La Figura 6.3 resume el volumen total por fuente.

Estos gráficos se generaron consultando MongoDB (colección `Docs`) o, de forma equivalente, con el notebook exploratorio del repositorio; pueden regenerarse al añadir nuevas fuentes. Confirman el **predominio de PlantVillage** en volumen, la aportación acotada de PlantDoc y App, y el fuerte desbalanceo entre clases mayoritarias (`healthy`, enfermedades frecuentes en PlantVillage) y clases minoritarias.

6.4.2. Formato como variable derivada

El campo **formato** no introduce una nueva muestra independiente: Grayscale y Segmentado se derivan automáticamente de Color mediante `process_imported_images.py`. Por ello, comparten la misma distribución de clases y fuentes que las imágenes originales; la única diferencia es la transformación visual aplicada (capítulo 6, § 6.5). Los experimentos documentados en el capítulo 8 usan solo Color; los formatos alternativos quedan disponibles para estudios comparativos futuros.

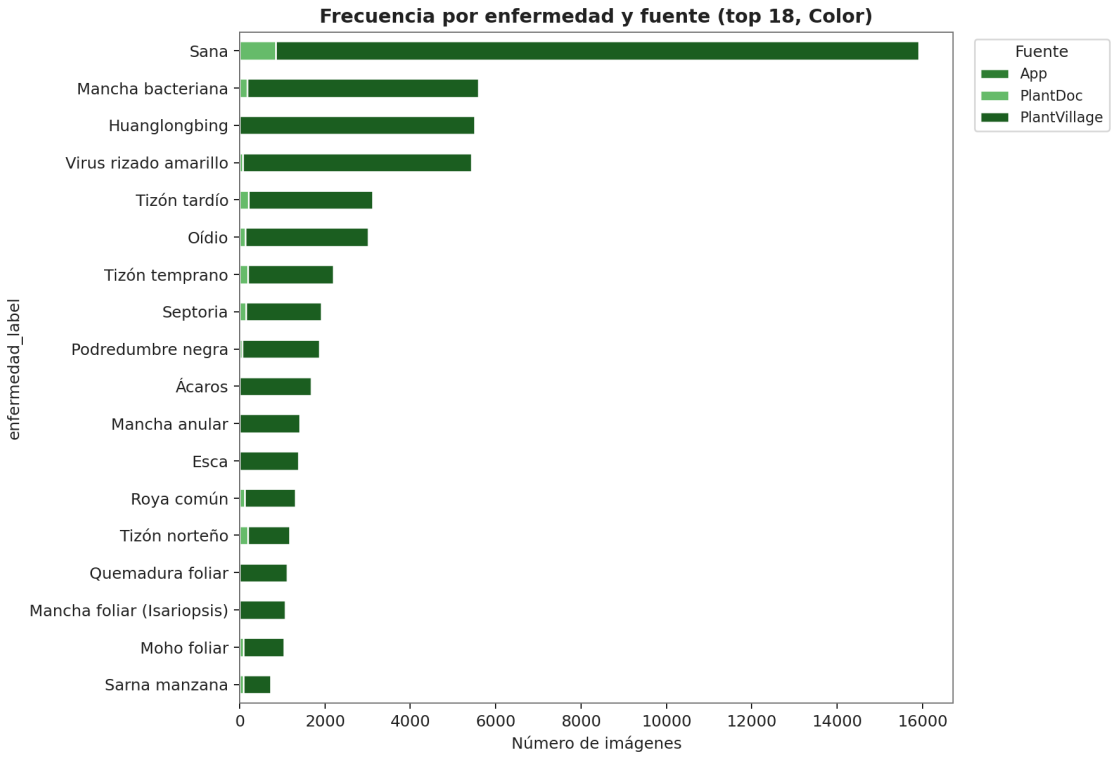


Figura 6.2: Frecuencia de imágenes por clase agrupada por fuente (EDA sobre MongoDB).

6.5 Preprocesamientos: color, grayscale y segmentado

Las imágenes obtenidas mediante la aplicación móvil desarrollada en el proyecto tienen el propósito de representar condiciones de captura realistas: iluminación natural, fondos heterogéneos, presencia de sombras, variaciones en distancia y enfoque, entre otros factores. Esta variabilidad contrasta fuertemente con las condiciones controladas en las que se capturaron las imágenes del dataset PlantVillage, donde se utilizó un fondo blanco uniforme, iluminación constante y encuadres centrados. Esta diferencia introduce un *domain shift* que puede afectar negativamente al rendimiento del modelo si no se tiene en cuenta durante el entrenamiento.

Con el fin de reducir este desajuste entre dominios, se ha desarrollado un sistema de procesamiento que transforma las imágenes reales a formatos alternativos: **escala de grises** y **segmentación de la hoja**. Esta decisión tiene dos objetivos principales:

- **Reducir el sesgo visual asociado a la fuente de las imágenes**, eliminando color o fondo para evitar que el modelo aprenda a distinguir entre dominios en lugar de entre clases de enfermedad.

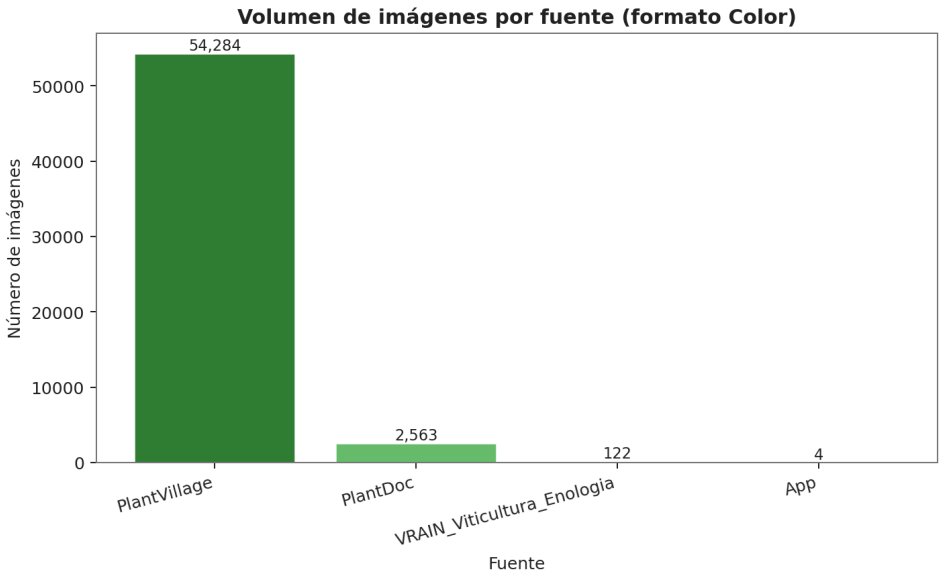


Figura 6.3: Volumen total de imágenes registradas por fuente en el servidor de producción.

- **Permitir una comparación coherente con PlantVillage**, que también incluye versiones en gris y segmentadas en los experimentos del artículo original. (Aunque se utilizará el mismo proceso propio de transformación de imágenes a las procedentes de PlantVillage y de otras fuentes)

La Figura 6.4 muestra, para la **misma imagen de entrada**, las tres variantes generadas: Color (original), Grayscale y Segmentado. Se incluyen ejemplos de **laboratorio** (PlantVillage) y de **captura más realista** (PlantDoc o App) para ilustrar el contraste entre fuentes.

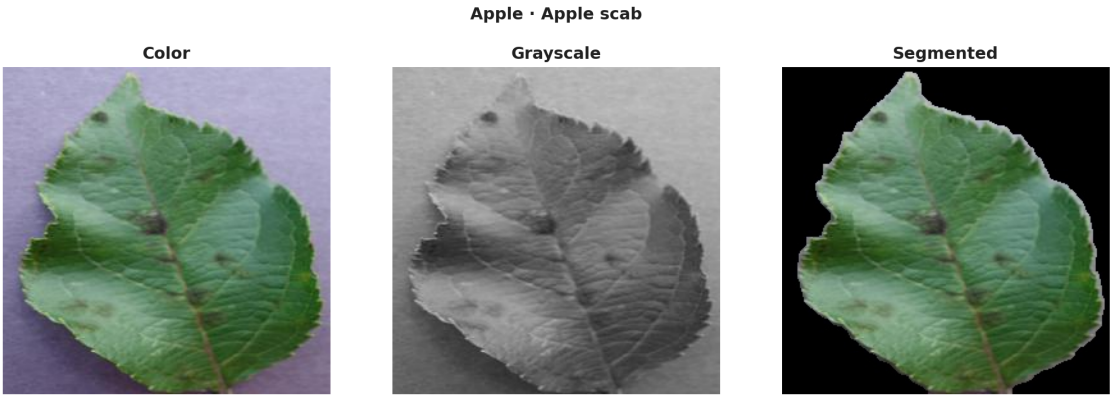


Figura 6.4: Ejemplo de la misma imagen en los tres formatos soportados: Color, Grayscale y Segmentado (filas o paneles: laboratorio vs. fuente más realista).

6.5.1. Conversión a escala de grises

La conversión a escala de grises permite eliminar la componente cromática de las imágenes, manteniendo únicamente la información estructural: bordes, forma y textura de la hoja. Esta transformación se ha implementado utilizando la función `cv2.cvtColor()` de la librería OpenCV, que convierte cada píxel RGB en un valor de intensidad en escala de grises mediante una media ponderada de los tres canales, simulando la percepción visual humana.

Este procesamiento permite estudiar el impacto del color en la clasificación. Si el modelo sigue funcionando bien sobre imágenes grises, puede inferirse que las características estructurales son suficientes para la tarea, y que el modelo generaliza mejor entre dominios.

6.5.2. Segmentación de la hoja

La segmentación busca eliminar el fondo de la imagen y conservar únicamente la región correspondiente a la hoja. Esta técnica, utilizada también en el artículo original de PlantVillage, tiene como objetivo evitar que el modelo aprenda patrones del fondo (color, textura, iluminación) que podrían sesgar el aprendizaje.

Dado que el código original de segmentación no está disponible públicamente, se ha implementado una versión propia inspirada en la metodología descrita por los autores. Tras evaluar visualmente los resultados, se ha decidido aplicar este mismo proceso a las imágenes de distintas fuentes. Esto garantiza una consistencia visual entre fuentes y evita introducir sesgos derivados de diferencias en el preprocesamiento.

Aunque la segmentación obtenida con este método no reproduce con exactitud la precisión de los contornos de las imágenes segmentadas originales de PlantVillage, se considera suficientemente fiable para los objetivos del proyecto. En particular, se ha observado que el algoritmo es efectivo eliminando el fondo, aunque los contornos pueden ser menos definidos.

En imágenes con fondos complejos, oclusiones o iluminación extrema, el resultado automático puede ser **claramente deficiente**. La Figura 6.5 contrasta dos grados de fallo. En **laboratorio** (PlantVillage) el algoritmo suele eliminar el fondo de forma aceptable, aunque el **contorno de la hoja** quede impreciso —como en el caso de *Apple scab* mostrado a la izquierda—. En **campo**, con variabilidad de iluminación, suelo y otras hojas en escena, el mismo método puede ser **inutilizable**: a la derecha, una captura de vid de la fuente *VRAIN Viticultura Enología*, donde la máscara no aísla de forma fiable la región foliar. Estos casos justifican la revisión visual antes de usar el formato Segmentado en entrenamientos críticos y explican por qué los experimentos del capítulo 8 se limitaron a Color.

El algoritmo desarrollado se basa en un enfoque heurístico en el espacio de color *Lab*. El proceso consiste en:

1. **Conversión a espacio Lab:** se transforma la imagen desde RGB a Lab, donde L representa la luminosidad, y a, b codifican la cromaticidad.
2. **Mejora de contraste y suavizado:** el canal L se ecualiza para aumentar el contraste global, y los canales L, a y b se suavizan con un filtro gaussiano para reducir el ruido.
3. **Estimación del color de fondo:** se asume que los bordes de la imagen contienen fondo, por lo que se toma la mediana de los valores de los bordes en a y b como referencia cromática del fondo.
4. **Construcción de máscaras:**
   - **Máscara de color:** marca como fondo aquellos píxeles cuya distancia euclídea en el plano (a, b) sea pequeña respecto al color de fondo.
   - **Máscara de sombras:** identifica regiones oscuras que podrían ser fondo en sombra, basándose en valores bajos de L y color similar al fondo.

Figura 6.5: Segmentación automática imperfecta o errónea. **Izquierda:** PlantVillage, *Apple scab* —contorno impreciso pero fondo en gran parte eliminado—. **Derecha:** fuente VRAIN Viticultura Enología, vid en entorno real —segmentación claramente deficiente (fondo residual y región foliar mal delimitada)—.

5. **Combinación y limpieza de máscaras:** se combinan ambas máscaras eliminando las sombras de la máscara principal. Se aplican operaciones morfológicas (apertura y cierre) para eliminar regiones pequeñas erróneas.
6. **Relleno del fondo y refinamiento:** se utiliza una operación de *flood-fill* desde las esquinas para asegurar que todo el fondo queda cubierto. Además, se conserva únicamente la región conectada más grande de la máscara para evitar errores en imágenes con múltiples objetos o ruido.
7. **Aplicación de la máscara:** la máscara binaria resultante se aplica a la imagen original, conservando la hoja y poniendo el fondo completamente negro.

Este procedimiento permite generar un conjunto coherente de imágenes segmentadas a partir de varias fuentes de datos, con un estilo visual uniforme y controlado. Esto facilita la creación de conjuntos de entrenamiento mixtos y permite realizar comparaciones más justas entre modelos entrenados con distintos formatos de entrada.

6.6 División train/validation/test

La partición de datos para entrenamiento no se realiza de forma estática sobre disco, sino **dinámicamente** a partir de MongoDB en el momento de ejecutar un experimento. La función `prepare_data_splits` (`utils/data.py`) realiza los pasos siguientes:

1. Filtra documentos en `Docs` según la configuración del experimento: plantas, enfermedades, fuentes, formato de imagen, variables adicionales y, opcionalmente, solo imágenes validadas.
2. Agrupa los documentos por identificador de clase y, si procede, limita el número de imágenes por clase (`imagenes_por_clase` en `config.yaml`).
3. Baraja aleatoriamente las imágenes de cada clase y las reparte en tres subconjuntos según las proporciones definidas en `config["split"]` (claves `train`, `val` y `test`).
4. Genera los ficheros `train.csv`, `val.csv` y `test.csv` en la carpeta `data/` del experimento, con columnas de ruta local, planta, enfermedad e identificador de clase.
5. Actualiza en memoria la lista de plantas y enfermedades efectivamente presentes y guarda la configuración resultante como `config_final.yaml` para reproducibilidad.

Las proporciones concretas de partición dependen de cada experimento y se leen del `config.yaml` correspondiente (la plantilla base reside en `experiments/BASE/`).

6.7 Balanceo de clases y control de sesgos

El desbalanceo entre clases es un riesgo conocido en PlantVillage y resulta aún más acusado cuando se mezclan fuentes con distinto número de imágenes. El sistema aborda este problema de varias formas:

- **Limitación por clase**: el parámetro `imagenes_por_clase` acota cuántas muestras por clase entran en el experimento, evitando que clases muy numerosas dominen el conjunto cuando se desea un estudio equilibrado.

- **Filtro de validación**: la variable del config.yaml `solo_validadas` permite excluir imágenes no revisadas, reduciendo ruido de etiquetado en fuentes reales.
- **Ponderación en la función de pérdida**: si `use_class_weights` está activo en la configuración, `utils/train.py` calcula pesos inversos a la frecuencia de cada clase de planta y de enfermedad y los aplica en las `CrossEntropyLoss` multitarea.
- **Pesos manuales por tarea**: los parámetros `peso_planta` y `peso_enfermedad` permiten ponderar la contribución de cada cabeza del modelo en la pérdida total.
- **Avisos de clases minoritarias**: durante la preparación del entrenamiento se emiten advertencias cuando una clase tiene menos muestras que `min_samples_per_class`.
- **Sin *data augmentation* en entrenamiento**: el pipeline actual no aplica transformaciones aleatorias (volteos, rotaciones, *jitter* cromático) al aumentar el conjunto de *train*. Se priorizaron la integración multi-fuente, los pesos por clase y la validación del marco experimental frente a generar variantes sintéticas que, con el volumen limitado de imágenes reales disponibles, no sustituirían de forma satisfactoria la captura de escenas auténticas de campo. Integrar *data augmentation* en `train_model` queda como mejora prevista (capítulos 7 y 10).

La evaluación del impacto de estas medidas sobre métricas concretas se reporta en el capítulo de experimentación.

Financiado por la Unión Europea NextGenerationEU

CAPÍTULO 7
Modelo y metodología de entrenamiento

Este capítulo describe la arquitectura del clasificador, el procedimiento de entrenamiento y evaluación, y el sistema de experimentos reproducibles de Foliarium. La experimentación cuantitativa asociada a estas decisiones se presenta en el capítulo 8.

En la fase actual del TFG, con un volumen de datos aún limitado y heterogéneo, se ha priorizado **validar el marco de adquisición, filtrado y entrenamiento** (capítulos 5 y 6) frente a una búsqueda exhaustiva de hiperparámetros o arquitecturas alternativas. Los entrenamientos documentados son **pruebas iniciales** sobre subconjuntos configurables, no un estudio sistemático de optimización del modelo.

7.1 Arquitectura multitarea basada en CNN (implementación MobileNetV2)

El pipeline de entrenamiento e inferencia utiliza una red **multitarea** con base MobileNetV2 [2]: comparte el extractor de características y dispone de dos cabezas lineales —planta y enfermedad— coherentes con el esquema de `Clases` en MongoDB. La clase `MultiTaskMobileNetV2` (`utils/model.py`) reutiliza `base_model.features`, aplica `AdaptiveAvgPool2d`, *dropout* (0,2) y dos clasificadores `Linear`.

7.1.1. Justificación histórica y limitaciones de la elección

MobileNetV2 es el **único *backbone* implementado y evaluado** en los experimentos documentados en el capítulo 8. La elección responde a tres factores: continuidad con el desarrollo previo del proyecto, planteado en una fase inicial con **inferencia en dispositivo móvil**; disponibilidad de pesos ImageNet en PyTorch como base de *fine-tuning* con el volumen de datos entonces disponible; y suficiencia de una arquitectura estándar para las **pruebas iniciales** de este TFG, orientadas a validar el pipeline de datos y la plataforma más que a comparar redes.

En el despliegue actual la predicción se ejecuta en el **servidor UPV**: el cliente Flet envía la imagen a `/predict_image` y recibe el resultado. La ligereza del modelo deja de ser un requisito crítico en cliente; evaluar ResNet, EfficientNet u otras alternativas sobre los mismos splits queda como trabajo futuro (capítulo 10). En una evolución posterior cabría, además, embeber un modelo en el compilado de la app para inferencia *offline* sin depender del servidor.

7.1.2. Construcción del modelo

Actualmente, la función `build_model(config)` concentra la instanciación de la red: carga MobileNetV2 con pesos ImageNet (`MobileNet_V2_Weights.DEFAULT`), aplica la estrategia de *fine-tuning* indicada en el `config.yaml` y crea las cabezas con cardinalidades `len(plantas)` y `len(enfermedades)` del experimento. Tras `prepare_data_splits`, esas listas reflejan las clases realmente presentes en los filtros seleccionados. Sustituir el *backbone* no exige rediseñar Foliarium: los experimentos se parametrizan en YAML y el contrato de la API (imagen de entrada, planta y enfermedad de salida) es independiente de la arquitectura concreta; probar otras CNN multitarea sobre los mismos splits de MongoDB sería una extensión directa del marco ya desplegado.

7.2 Estrategias de fine-tuning

El campo `fine_tune` admite tres modos en código (`utils/model.py`):

- `none`: congela todo el *backbone*.
- `top`: congela `features` y entrena las capas superiores (modo usado en todos los experimentos documentados).
- Cualquier otro valor distinto de `none` y `top` dejaría entrenable todo el modelo (no utilizado en las pruebas iniciales).

En las pruebas iniciales se fijó `fine_tune: top` con pesos ImageNet: se asume que las capas convolucionales prefabricadas capturan texturas y bordes útiles para hojas, mientras que las cabezas multitarea se adaptan a la taxonomía de Foliarium. No se compararon sistemáticamente `none`, `top` y entrenamiento completo por limitación de tiempo y porque el objetivo prioritario era validar el **pipeline de datos**, no afinar la red.

7.3 Función de pérdida, optimización y métricas

7.3.1. Entrenamiento

La función `train_model` (`utils/train.py`) entrena con PyTorch en el dispositivo detectado por `utils/device.py` (CPU o GPU). Hiperparámetros leídos del `config.yaml`:

- **Optimizador**: Adam (`lr: 0.001`) o SGD con momento 0,9.
- **Pérdida**: dos `CrossEntropyLoss` independientes (planta y enfermedad). La pérdida total es `peso_planta` $\times \mathcal{L}_p +$ `peso_enfermedad` $\times \mathcal{L}_e$ (por defecto ambos pesos valen 1,0).
- **Ponderación por clase**: con `use_class_weights: true`, cada cabeza usa pesos inversos a la frecuencia en `train.csv`, normalizados a media 1. Se avisa en consola si alguna clase tiene menos muestras que `min_samples_per_class` (10 en la plantilla).
- **Datos de entrada**: imágenes redimensionadas a 224×224, normalizadas con las estadísticas ImageNet; `batch_size: 32`. El preprocesado es **determinista** e idéntico en *train*, *val* y *test* (sin *data augmentation*).

No se incorporaron transformaciones aleatorias en la fase documentada: el objetivo prioritario fue cerrar el ciclo entrenamiento–evaluación sobre datos reales ya registrados en MongoDB, y se consideró que ampliar artificialmente muestras de laboratorio aportaría poco frente al *domain shift* principal (mezcla PlantVillage–PlantDoc y, a largo plazo, imágenes *in situ*). Probar *data augmentation* sobre los mismos splits —p. ej. volteos horizontales o variaciones de color— queda como extensión natural de `utils/train.py` (capítulo 10).

En cada época se registran las pérdidas medias de entrenamiento y validación para planta y enfermedad (`history` en `metrics.json`).

7.3.2. Evaluación

La función `evaluate` calcula, para los conjuntos `val` y `test`:

- **Accuracy** por cabeza (`accuracy_planta`, `accuracy_enfermedad`).
- **Accuracy combinada**: fracción de muestras en las que **ambas** etiquetas son correctas simultáneamente. Es la métrica principal de referencia en Foliarium.
- **Precisión, recall y F1 macro** por cabeza.

Tras la predicción independiente de cada cabeza, se aplica una **restricción de plausibilidad**: si el par (planta, enfermedad) predicho no existe en el conjunto de entrenamiento, se sustituye por la combinación válida de máxima probabilidad conjunta. Esto refleja el dominio fitosanitario (no todas las enfermedades aparecen en todos los cultivos) y coincide con la lógica de inferencia en `/predict_image`.

Para el split `test` se generan matrices de confusión ilustradas en `confusion_planta.png`, `confusion_enfermedad.png` y el fichero `misclassified_test.json` con rutas, etiquetas reales y predichas de cada error.

7.4 Gestión de experimentos y configuración reproducible

7.4.1. Estructura de carpetas

Cada experimento vive en `experiments/{nombre}/` con:

- `config.yaml`: configuración elegida al crear el experimento (desde la app o desde `scripts/make_experiment.py`).
- `config_final.yaml`: configuración efectiva tras `prepare_data_splits`, con listas expandidas de plantas y enfermedades.
- `data/`: `train.csv`, `val.csv`, `test.csv`.
- `models/`: pesos `{nombre}.pth`.
- `results/`: `metrics.json`, gráficos y JSON de errores.
- `run_experiment.py`: copia de la plantilla `experiments/BASE/`.

Tabla 7.1: Parámetros comunes de la plantilla `experiments/BASE/config.yaml`.

Campo	Valor por defecto
`formato`	Color
`imagenes_por_clase`	`all` (sin límite práctico)
`split train / val / test`	0,7 / 0,2 / 0,1
`epochs`	10
`fine_tune`	`top`
`optimizer`	`adam`, `lr: 0.001`
`use_class_weights`	`true`
`weights`	`MobileNet_V2_Weights.DEFAULT`

7.4.2. Parámetros relevantes de la plantilla base

La plantilla `experiments/BASE/config.yaml` fija valores por defecto para las pruebas iniciales:

Los filtros `fuentes`, `plantas` y `enfermedades` se definen por experimento. El valor `all` en listas se expande a todos los valores distintos en MongoDB (`utils/data.py`). `imagenes_por_clase: all` incluye todas las imágenes disponibles por clase tras filtrar; un entero acota el muestreo aleatorio por clase.

7.4.3. Flujo de ejecución

1. Consulta a MongoDB y generación de CSVs (`prepare_data_splits`).
2. Construcción del modelo y entrenamiento (`build_model`, `train_model`).
3. Evaluación en validación y test (`evaluate`).
4. Persistencia de métricas, histórico, gráficos y modelo (`utils/io.py`).

Desde la app, el usuario crea el experimento y puede solicitar entrenamiento; en producción el proceso completo se puede lanzar desde la sección de rol `admin` de la app, pero para los primeros experimentos se ha hecho en el servidor con el comando `python experiments/{nombre}/run_experiment.py` (p. ej. vía `docker-compose exec api ...`) para evitar *timeouts* HTTP.

7.5 Scripts de entrenamiento e inferencia

- `experiments/{nombre}/run_experiment.py`: pipeline completo descrito arriba.
- `scripts/make_experiment.py`: crea la carpeta del experimento a partir de `BASE/`.
- `scripts/compare_experiments.py`: compara métricas de varios experimentos y genera gráficos en `experiments/comparison/`.
- `scripts/predict_image.py`: inferencia por línea de comandos (misma lógica que la API).

La inferencia en producción reutiliza los pesos del experimento elegido: la API carga el `.pth`, aplica el mismo preprocesado y la restricción de combinaciones válidas, y devuelve planta, enfermedad y probabilidad al cliente.

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

CAPÍTULO 8
Experimentación y resultados

Este capítulo recoge las **pruebas iniciales de clasificación** ejecutadas sobre el servidor de producción una vez integrados PlantVillage y PlantDoc en MongoDB. El objetivo no fue optimizar al máximo la precisión del modelo, sino **demostrar que el ciclo completo** —filtrado dinámico, entrenamiento, métricas, gráficos y comparación— funciona de extremo a extremo dentro de Foliarium y sentar resultados base sobre los que comparar los conseguidos en el futuro.

8.1 Objetivos y alcance de la experimentación

- Validar el **pipeline reproducible** (capítulos 6 y 7) con datos reales en la base `Plantas`.
- Establecer una **línea base de laboratorio** (solo PlantVillage, condiciones controladas).
- Evaluar el efecto de **mezclar una fuente de campo** (PlantDoc) sobre las mismas clases y plantas.
- Generar artefactos consultables desde la app (tabla de experimentos, gráficos, comparativas).

Fuera de alcance en esta fase: comparativa sistemática de formatos (escala de grises, segmentación), barrido de hiperparámetros, *data augmentation* en entrenamiento, validación agronómica en campo o entrenamientos con datos exclusivamente capturados por la app (aún escasos). Solo se utilizó el formato **Color**.

8.2 Experimentos realizados

8.2.1. Experimento `Inicial` (prueba de humo)

Primer entrenamiento tras corregir el pipeline de experimentos y subir PlantDoc. Configuración resumida:

- **Fuentes**: PlantVillage + PlantDoc.
- **Plantas**: todas (`all`).
- **Imágenes por clase**: 100 (submuestreo).

- **Épocas**: 3.
- Resto de hiperparámetros: plantilla base (MobileNetV2, `fine_tune: top`, Adam, pesos de clase).

Sirvió para verificar que el entrenamiento terminaba sin errores y que las métricas se guardaban correctamente. No se considera referencia principal por el bajo número de épocas y el submuestreo agresivo.

8.2.2. Experimento `E1_PlantVillage_lab` (línea base)

- **Fuentes**: solo PlantVillage.
- **Plantas**: nueve cultivos con al menos dos enfermedades distintas en el catálogo —Apple, Cherry, Corn, Grape, Peach, Pepper, Potato, Strawberry, Tomato— excluyendo especies con una sola enfermedad (p. ej. Blueberry, Orange, Soybean), poco informativas para la cabeza de enfermedad.
- **Imágenes por clase**: `all` (7.831 muestras en test tras el split 70/20/10).
- **Épocas**: 10.
- **Enfermedades**: 20 clases efectivas tras el filtrado (véase `config_final.yaml`).

Representa el escenario más próximo a los trabajos clásicos sobre PlantVillage: fondo uniforme, iluminación controlada, alta coherencia visual.

8.2.3. Experimento `E2_PlantVillage_PlantDoc_mix`

Misma configuración de plantas, épocas e hiperparámetros que E1, pero con **PlantVillage + PlantDoc**. PlantDoc aporta imágenes tomadas en condiciones más cercanas al campo (fondos variables, iluminación irregular), por lo que se esperaba una ligera **degradación** de métricas respecto a E1 si el dominio de campo dificulta la generalización.

8.3 Resultados cuantitativos

La Tabla 8.1 resume las métricas de **test** extraídas de `metrics.json`. La accuracy combinada es la métrica principal; las cabeceras separadas permiten ver si los errores se concentran en planta o en enfermedad.

Tabla 8.1: Métricas en el conjunto de **test** (valores de `metrics.json`).

Experimento	Acc. comb.	Acc. planta	Acc. enf.	F1 planta	F1 enf.	Épocas
`Inicial`	0,805	0,945	0,826	0,937	0,814	3
`E1_PV_lab`	0,936	0,988	0,943	0,985	0,936	10
`E2_PV+PD`	0,919	0,973	0,927	0,968	0,908	10

8.3.1. Comparativa E1 frente a E2

Al añadir PlantDoc (E2) respecto a la línea base (E1):

Tabla 8.2: Métricas en **validación** (misma estructura que test).

Experimento	Acc. comb.	Acc. planta	Acc. enf.
`Inicial`	0,771	0,909	0,814
`E1_PV_lab`	0,942	0,987	0,949
`E2_PV+PD`	0,916	0,971	0,928

- La **accuracy combinada en test** pasa de **0,936** a **0,919** ($\Delta \approx -1,7$ puntos porcentuales).
- La cabeza de **planta** se mantiene alta ($0,988 \rightarrow 0,973$).
- La cabeza de **enfermedad** es la más afectada ($0,943 \rightarrow 0,927$).

Este comportamiento es coherente con la literatura: mezclar dominios de laboratorio y campo aumenta la heterogeneidad visual y suele penalizar sobre todo la clasificación fina de síntomas, no tanto el cultivo.

Los gráficos comparativos generados desde la app o desde scripts se almacenan en `experiments/comparison/E1_PlantVillage_lab_vs_E2_PlantVillage_PlantDoc_mix/`. La Figura 8.1 resume la accuracy combinada en validación y test.

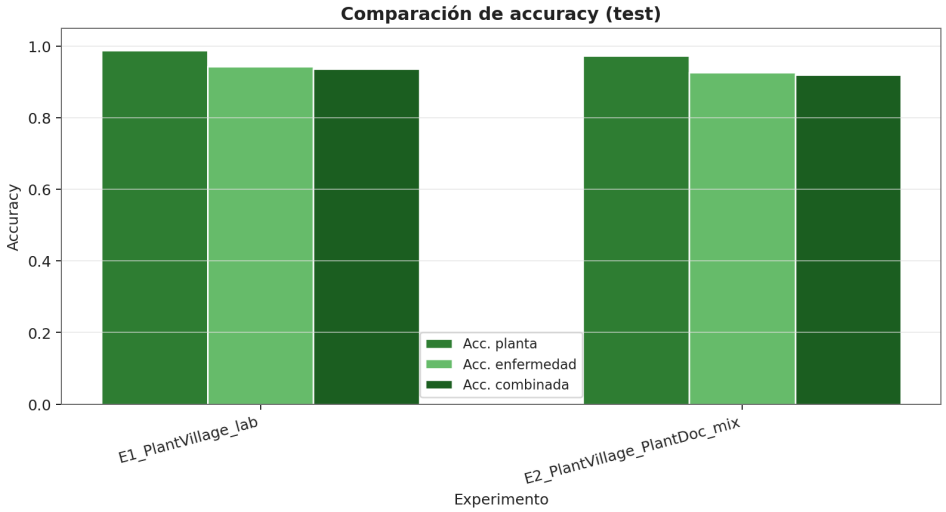


Figura 8.1: Comparativa de accuracy combinada entre E1 (solo PlantVillage) y E2 (PlantVillage + PlantDoc).

8.3.2. Curvas de aprendizaje

En E1 y E2 las pérdidas de entrenamiento son razonablemente similares y validación descienden de forma estable durante las 10 épocas; no se observa un sobreajuste extremo en la cabeza de planta. La cabeza de enfermedad presenta pérdidas superiores y más variabilidad, acorde con el mayor número de clases y la ambigüedad visual entre síntomas.

Los ficheros `history.png` de cada experimento documentan esta evolución; la Figura 8.2 corresponde a E1. El experimento `Inicial`, con solo 3 épocas, no alcanzó el mismo nivel de convergencia.

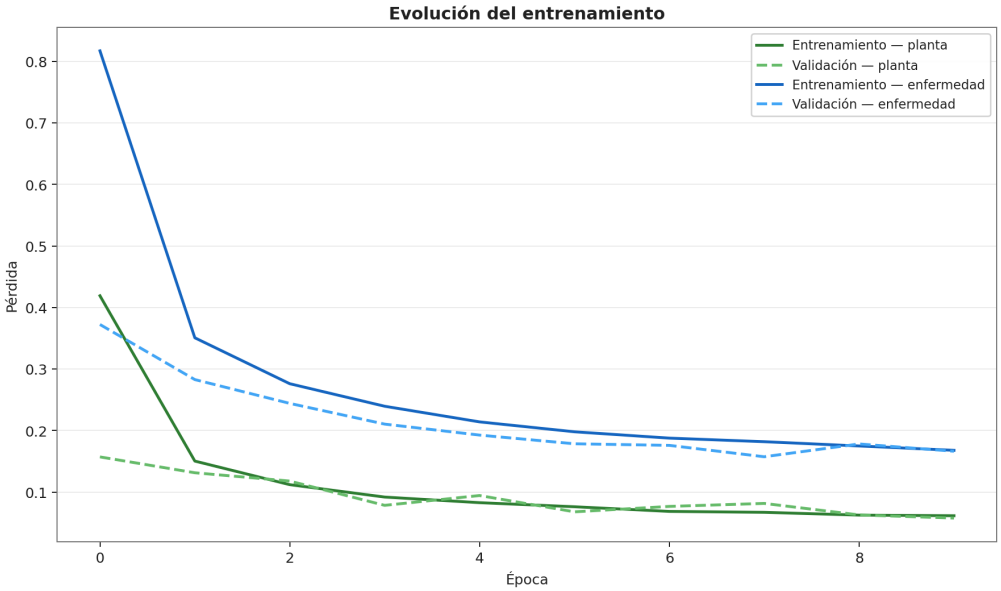


Figura 8.2: Curvas de pérdida de entrenamiento y validación en E1 (cabeza planta y enfermedad).

8.4 Formatos de imagen y otras variantes no evaluadas

Foliarium soporta imágenes en formatos `Grayscale` y `Segmented` (capítulo 6), pero **ningún experimento documentado** los utiliza aún. Una comparativa color vs. segmentación —como en Mohanty et al.— queda pendiente cuando exista tiempo y datos suficientes en esos formatos en MongoDB.

8.5 Análisis cualitativo de errores

8.5.1. Artefactos automáticos

Cada entrenamiento genera en `results/`:

- `misclassified_test.json`: lista de imágenes del test mal clasificadas (ruta, etiqueta real y predicha).
- `misclassified_val.json`: análogo para validación.
- Matrices de confusión por cabeza en forma de imágenes (`confusion_planta.png`, `confusion_enfermedad.png`).

8.5.2. Matriz de confusión de enfermedad (E1)

La Figura 8.3 muestra la matriz de confusión de la cabeza de enfermedad en el conjunto de **test** del experimento E1 (solo PlantVillage). La diagonal concentra la mayor parte de las predicciones correctas; las clases con más muestras —`Sana`, `Mancha bacteriana` y `Virus rizado amarillo`— presentan los recuadros diagonales más oscuros.

Los patrones más destacables fuera de la diagonal son:

- **Confusiones entre tizones y manchas foliares**: `Tizón temprano` y `Tizón tardío` se confunden entre sí y, en menor medida, con `Moho foliar` o `Mancha anular`. `Mancha`

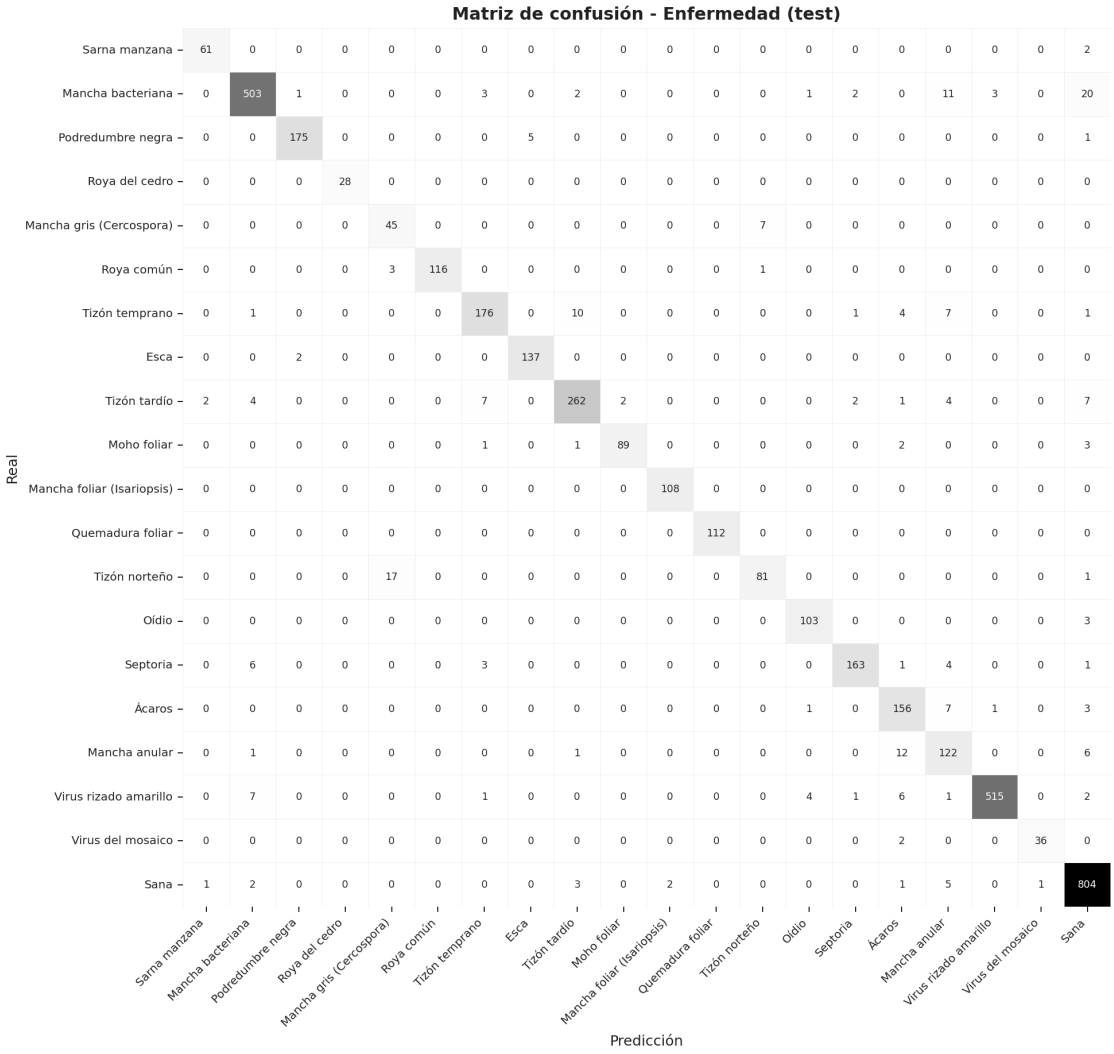


Figura 8.3: Matriz de confusión de la cabeza de enfermedad en test (E1, solo PlantVillage).

`gris (Cercospora)` y `Tizón norteño` también intercambian predicciones, coherente con síntomas necrosados de aspecto similar en hoja.

- **Síntomas incipientes clasificados como** `Sana`: aparecen casos de `Mancha bacteriana`, `Sarna manzana` o `Mancha anular` predichos como `Sana`; es el patrón más frecuente en `misclassified_test.json` y encaja con lesiones poco visibles en imágenes de laboratorio.
- **Clases minoritarias**: `Roya del cedro` y `Virus del mosaico` tienen pocas muestras en test y muestran celdas diagonales más pequeñas; el desbalance dificulta el aprendizaje fino pese a los pesos por clase.
- **Cabeza de planta**: la matriz homóloga (accuracy $\approx$ 0,99) apenas registra confusiones; los errores del modelo multitarea se concentran casi por completo en la taxonomía de enfermedad.

En E2 (PlantVillage + PlantDoc) se observan los **mismos tipos de confusión** con celdas off-diagonal algo más pobladas (p. ej. `Tizón tardío` $\rightarrow$ `Tizón temprano`, `Septoria` $\rightarrow$ `Tizón temprano`), acorde con la caída de $\approx$1,5 puntos en accuracy de enfermedad respecto a E1. No se incluye aquí la figura de E2 para no duplicar un patrón cualitativamente similar. Sin embargo, la imagen `e2_confusion_enfermedad.png` está disponible tanto en el repositorio como mediante la app.

En E1, revisando además el JSON de test, se confirman los mismos hallazgos: confusiones entre enfermedades **visualmente similares** y predicciones `Sana` frente a síntomas leves. El listado completo de casos erróneos queda registrado en `misclassified_test.json` para inspección manual (p. ej. con el notebook auxiliar descrito a continuación).

8.5.3. Notebook `misclassified_and_topk.ipynb`

El archivo `notebooks/misclassified_and_topk.ipynb` se ha utilizado como herramienta auxiliar para **visualizar** imágenes contenidas en el archivo de fallos de predicción `misclassified_test.json` y obtener predicciones *top-k* sobre imágenes concretas. El notebook es auxiliar para inspección manual; no forma parte del pipeline de producción. Puede reutilizarse apuntando `EXPERIMENTS_PATH` al experimento deseado, siempre que las rutas de imagen en el JSON sean accesibles desde el entorno de ejecución.

No se ha realizado un estudio exhaustivo de las primeras k predicciones en producción; dado el alto rendimiento en E1, se reserva para futuras iteraciones con más datos de campo.

8.6 Discusión

8.6.1. Logros

- El **marco experimental** funciona: configuración YAML, splits dinámicos desde MongoDB, entrenamiento en servidor, métricas y comparativas consultables en la app.
- E1 alcanza $\approx$ **93,6 %** de accuracy combinada en test sobre PlantVillage, resultado razonable como **baseline** multitarea sin afinación exhaustiva.
- E2 confirma que integrar PlantDoc es **viable** pero introduce una penalización moderada previsible por **domain shift**. A pesar de ser porcentualmente una diferencia pequeña, tiene relevancia si se considera que el número de imágenes que aporta la segunda fuente no es comparativamente demasiado grande.

Financiado por la Unión Europea NextGenerationEU

8.6.2. Limitaciones

- **Pocos datos propios** de la app y escasa validación manual (se podría considerar validar las imágenes procedentes de PlantVillage y PlantDoc, pero inicialmente se han mantenido como no validadas).
- **Sin búsqueda de hiperparámetros**: mismos `lr`, épocas y `fine_tune` en E1 y E2.
- **Arquitectura fija** (MobileNetV2) pese a que la inferencia se realiza en el servidor.
- **PlantDoc parcialmente alineado** taxonómicamente; no todas las clases de campo tienen equivalente en PlantVillage.
- Métricas de `Inicial` no comparables directamente por configuración distinta (3 épocas, 100 img/clase, todas las plantas).

8.6.3. Relación con el objetivo del TFG

Dado el volumen limitado de imágenes reales propias, resultó más valioso invertir el esfuerzo en la **plataforma de adquisición e integración** (importación PlantDoc, subida masiva, experimentos configurables, despliegue) que en maximizar un punto porcentual de accuracy. Los experimentos E1/E2 **validan el componente de aprendizaje automático** dentro de ese marco y proporcionan una referencia cuantitativa para la memoria y para futuras mejoras (más fuentes, más épocas, otras arquitecturas, formatos segmentados...). No obstante, el **resultado a corto plazo** del TFG es precisamente ese marco desplegado: disponer de un sistema activo al que conectarse, subir imágenes y reentrenar es condición necesaria —pero no suficiente en el plazo académico— para acumular con el tiempo un corpus de campo validado capaz de sostener modelos robustos fuera del laboratorio.

Financiado por la Unión Europea NextGenerationEU

CAPÍTULO 9
Validación y despliegue

Este capítulo aborda la **validación y puesta en producción de la plataforma** Foliarium. Conviene distinguirlo del capítulo 8: allí se evalúan los **experimentos del modelo** (métricas E1/E2, comparativas de fuentes); aquí se documenta que el **sistema desplegado funciona** —conexión cliente–servidor, roles, subida de imágenes, etiquetado y validación, flujos administrativos— y cómo se ha containerizado y operado en el servidor de la UPV.

9.1 Validación funcional del sistema

9.1.1. Alcance de la validación

La validación funcional comprueba que Foliarium cumple su cometido como plataforma integrada una vez desplegada, independientemente del rendimiento numérico del modelo en un experimento concreto. Los aspectos verificables en esta fase incluyen:

- Acceso al backend en producción (`https://plantas.gti-ia.upv.es`) y endpoint de salud (`/health`).
- Registro, inicio de sesión y selección de rol activo.
- Subida de imágenes y registro en MongoDB con metadatos (fuente, formato, clase).
- Flujo de etiquetado y validación por el rol `etiquetador`.
- Operaciones de administración (usuarios, roles, clases).
- Creación de experimentos y consulta de resultados (cuando existan experimentos en el servidor).
- Predicción mediante `/predict_image` con un modelo publicado.
- Descarga de clientes y guía de usuario desde la landing pública.

9.1.2. Metodología de pruebas

La verificación del sistema se ha realizado mediante **pruebas funcionales manuales**, no mediante una suite automatizada (`pytest`) ni un pipeline de integración continua. En un TFG con alcance centrado en la plataforma desplegada y en la experimentación con

datos reales, se priorizó comprobar el comportamiento *end-to-end* en el entorno objetivo —servidor de la UPV y clientes empaquetados— frente a la cobertura automatizada exhaustiva.

Los casos comprobados incluyen, como mínimo: autenticación y selección de rol, subida individual y masiva de imágenes, etiquetado y validación, operaciones de administración (usuarios, roles, clases), consulta de experimentos y predicción cuando hay modelos publicados, y conectividad HTTPS del cliente contra el servidor. Esta aproximación es habitual en proyectos de titulación de esta envergadura y se complementa con la evaluación cuantitativa del modelo (capítulo 8), que constituye un eje distinto de validación.

9.1.3. Pruebas realizadas

El sistema está **desplegado y operativo** en el servidor de la universidad. El autor ha comprobado de forma recurrente los flujos principales —conexión desde clientes empaquetados (Windows, Android), subida de imágenes, navegación por rol y operación contra la API de producción— con resultado satisfactorio.

Además, **varios colaboradores** (compañeros del entorno universitario y externos) han instalado los clientes y probado el acceso al backend remoto, con comentarios positivos sobre conectividad y uso básico de la aplicación. Estas pruebas han sido **informales**; se complementan con una encuesta de usabilidad publicada en la landing (véase §9.3.2), cuyos resultados agregados se incorporarán cuando se disponga de un volumen suficiente de respuestas. Estas pruebas iniciales ya han servido para solucionar ciertos errores, como el hecho de que se inicializaba mal la dirección IP y daba problemas de conectividad, y también para entender mejor la experiencia de usuario.

9.1.4. Limitaciones de esta validación

La validación funcional confirma que la plataforma es utilizable *end-to-end* en producción; **no certifica** por sí sola un rendimiento agronómico concreto del clasificador (evaluado cuantitativamente en el capítulo 8). Tampoco se ha realizado validación por expertos fitosanitarios ni un plan de pruebas automatizado con cobertura medida.

9.2 Despliegue y contenedorización con Docker

9.2.1. Arquitectura Docker Compose

El despliegue del backend se realiza mediante **Docker Compose** (`docker-compose.yml`), con los servicios siguientes:

- `mongo` (imagen `mongo:7`): base de datos MongoDB con volumen persistente `mongo_data` y *healthcheck* mediante `mongosh`.
- `api` (imagen construida desde el `Dockerfile` del repositorio): API Flask servida con **Gunicorn** en el puerto 5001, dependiente de MongoDB en estado saludable.
- `initdb` (perfil `init`, ejecución puntual): ejecuta `scripts/setup_bbdd.py` para inicializar bases de datos, catálogo de clases y usuario administrador por defecto.

El contenedor `api` monta volúmenes para persistir datos fuera de la imagen: `imagenes/`, `models/`, `experiments/`, `data/`, `logs/` y `downloads/` (clientes empaquetados para la landing).

9.2.2. Imagen de la API (`Dockerfile`)

La imagen se basa en `python:3.11-slim`, instala dependencias de sistema necesarias para OpenCV/PIL (`libgl1`, `libglib2.0-0`), copia `requirements.txt` y el código del repositorio, y arranca Gunicorn con parámetros configurables:

```
gunicorn --workers ${GUNICORN_WORKERS:-2} \
         --timeout ${GUNICORN_TIMEOUT:-120} \
         --bind 0.0.0.0:5001 main:app
```

El `Dockerfile` indica que, en despliegue real, **TLS debe terminar en un reverse proxy** (Nginx, Caddy u equivalente) y la API corre en HTTP interno dentro de la red Docker.

9.2.3. Variables de entorno y comprobaciones

Entre las variables relevantes del fichero `.env` (plantilla `.env.docker.example`) figuran:

- `URL_BBDD=mongodb://mongo:27017/` — conexión interna entre contenedores.
- `PUBLIC_API_BASE_URL=https://plantas.gti-ia.upv.es` — URL pública usada al registrar rutas de imágenes en la base de datos.
- `GUNICORN_WORKERS`, `GUNICORN_TIMEOUT`, `MAX_IMAGE_SIZE_MB`.
- `DOWNLOADS_DIR` y nombres de ficheros descargables para Windows, Linux y Android.

Tanto `mongo` como `api` definen *healthchecks* periódicos; el servicio `initdb` solo arranca cuando `api` reporta estado saludable.

9.2.4. Operación habitual

El ciclo de despliegue documentado en `docs/DOCKER.md` comprende lo siguiente: `docker-compose build`, `docker-compose up -d`, verificación con `docker-compose ps`, consulta de logs (`docker-compose logs -f api`) y, en la primera puesta en marcha, `docker-compose -profile init run -rm initdb` para inicializar la base de datos.

9.2.5. Copias de seguridad de MongoDB

Los datos persistentes del sistema se reparten entre el volumen Docker de MongoDB (`mongo_data`), las bases `Plantas` y `Usuarios`, y los directorios montados `imagenes/`, `models/` y `experiments/`. Para la base de datos documental, el repositorio incluye el script `scripts/backup_mongo.sh`, que genera un volcado (`mongodump`) de **ambas bases** en ficheros `.archive` con marca temporal bajo `backups/mongo/`.

Financiado por la Unión Europea NextGenerationEU

- **Copia manual**: se puede crear ejecutando el script desde el host con el stack levantado (`sh scripts/backup_mongo.sh`). Conviene hacerlo antes de importaciones masivas, cambios de esquema o actualizaciones arriesgadas.
- **Copia programada (`cron`)**: Docker Compose no lanza backups por sí solo; en la VM de producción se programa una tarea `cron` del **host** (el usuario debe poder ejecutar `docker exec` sobre el contenedor `plantas-mongo`). Pasos aplicados en el despliegue:
  1. Crear el directorio de destino, p. ej. `/var/backups/foliarium/mongo`.
  2. Probar una copia manual con `BACKUP_ROOT` apuntando a esa ruta.
  3. Editar la crontab del usuario que opera Docker (`crontab -e`) e insertar una línea como la siguiente (semanal, domingos 03:00):

```
0 3 * * 0 cd /ruta/al/repositorio && \
  BACKUP_ROOT=/var/backups/foliarium/mongo \
  BACKUP_RETENTION_DAYS=30 \
  sh scripts/backup_mongo.sh >> /var/log/foliarium-backup.log 2>&1
```

- `BACKUP_RETENTION_DAYS=30` elimina carpetas de copia más antiguas de 30 días. La salida queda registrada en el log indicado para comprobar ejecuciones posteriores.
- **Restauración**: `mongorestore` desde los archivos `.archive` generados, base por base (procedimiento en `docs/DOCKER.md`).

Las imágenes en disco y los modelos entrenados no quedan incluidos en ese volcado; su respaldo depende de la copia de los volúmenes o directorios montados en la VM, según la política del servidor.

9.3 Despliegue en el servidor de producción

9.3.1. Entorno accesible

El sistema en producción está accesible en `https://plantas.gti-ia.upv.es`. Los clientes Flet (Windows, Linux, Android) se configuran para apuntar a esa URL de la API; la misma máquina sirve la landing pública en la raíz del dominio.

9.3.2. Landing, guía y distribución de clientes

La ruta `/` de la API renderiza una **página de presentación** (`templates/landing.html`) organizada en dos bloques: información sobre Foliarium (descripción, guía) y descargas junto con la encuesta de usabilidad. Incluye:

- Descripción breve de Foliarium (subida de imágenes, clases, predicción).
- Enlace a la **guía de usuario** en PDF (`/guia-usuario`, fichero `docs/guia_usuario.pdf`; contenido completo en el apéndice A).
- Enlaces de descarga de clientes en las versiones: `/download/windows`, `/download/linux`, `/download/android`, que sirven los archivos correspondientes almacenados en la carpeta `downloads/` si están disponibles en el servidor.

- **Encuesta de usabilidad** (Google Forms, 2–3 minutos) accesible desde la landing, dirigida a quienes hayan probado la aplicación. Permite recoger impresiones sobre instalación, conexión al servidor y facilidad de uso.

Esta página centraliza la distribución de la aplicación sin depender de tiendas de aplicaciones (Google Play u otras queda como línea futura, capítulo 10).

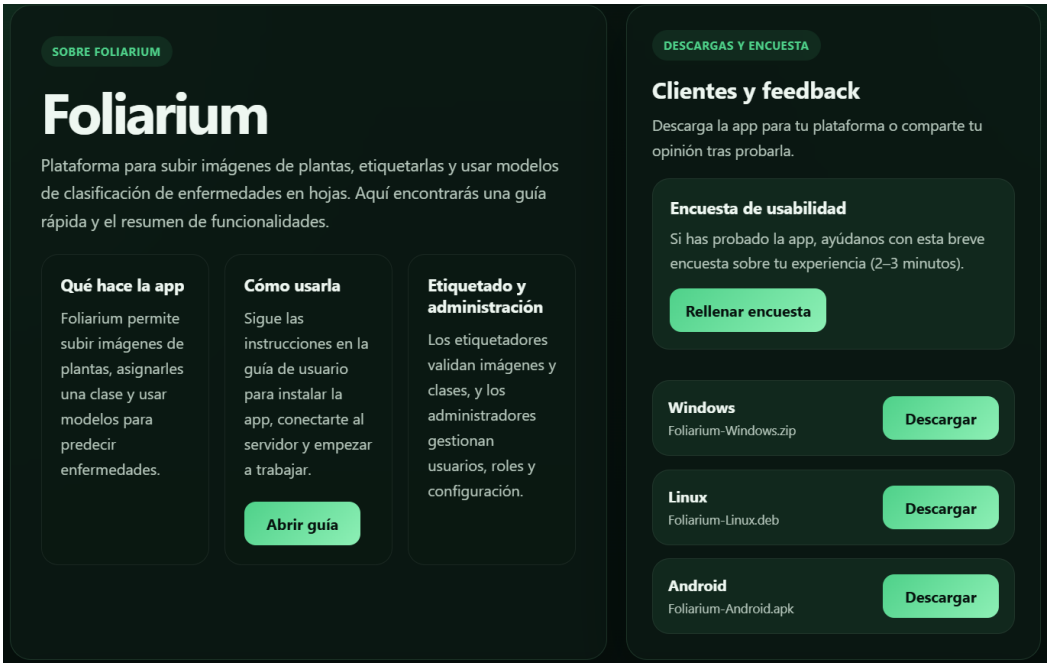


Figura 9.1: Captura de la landing pública (`https://plantas.gti-ia.upv.es/`): descripción, guía, descargas y enlace a la encuesta.

9.3.3. Clientes y conexión

Los usuarios finales instalan el cliente correspondiente a su plataforma, configuran (o aceptan por defecto) la URL del backend de producción e inician sesión con las credenciales registradas. Las operaciones de captura, subida y predicción se realizan contra el backend remoto; no es necesario ejecutar la API en la máquina del usuario.

9.4 Reproducibilidad de los experimentos

A nivel de **infraestructura**, la reproducibilidad queda apoyada en:

- Imagen Docker versionada a partir del repositorio y `requirements.txt`.
- Volúmenes persistentes para `experiments/` y `models/` en el servidor.
- Scripts de importación y entrenamiento compartidos entre entorno local y contenedor.

A nivel de **contenido experimental**, cada carpeta `experiments/<nombre>/` conserva `config.yaml`, `config_final.yaml`, los CSV de split, `metrics.json`, gráficos y el script `run_experiment.py`. Los experimentos E1, E2 e `Inicial` documentados en el capítulo 8 son reproducibles ejecutando `python experiments/<nombre>/run_experiment.py` en el contenedor `api`, siempre que MongoDB contenga las mismas fuentes importadas.

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

9.5 Pruebas con usuarios externos

Varios usuarios del entorno universitario y externos han probado la aplicación instalada contra el servidor de producción, confirmando que la conexión remota y los flujos básicos son utilizables. Para ampliar esa retroalimentación cualitativa, la landing pública incluye desde la redacción de esta memoria un enlace a una **encuesta Google Forms** sobre usabilidad (instalación del cliente, registro, subida de imágenes, claridad de la interfaz).

CAPÍTULO 10
Conclusiones y trabajo futuro

10.1 Conclusiones

Este Trabajo Fin de Grado ha dado lugar a **Foliarium**, una plataforma integrada para la recopilación, etiquetado, experimentación y despliegue de modelos de clasificación fitosanitaria sobre hojas. Las principales aportaciones son las siguientes:

- **Marco unificado de datos heterogéneos**: importación de PlantVillage y PlantDoc en MongoDB bajo un esquema común de fuentes, plantas, enfermedades y formatos (color, escala de grises, segmentación), con flujos de subida colaborativa desde clientes multiplataforma (Flet). Automatización de la importación de nuevas fuentes en el futuro.
- **Pipeline reproducible de experimentos**: configuración declarativa (`config.yaml`), splits dinámicos desde la base de datos, entrenamiento multitarea con MobileNetV2, métricas automáticas, matrices de confusión y comparativas entre experimentos consultables desde la aplicación.
- **Validación inicial del componente de aprendizaje automático**: el experimento E1 (solo PlantVillage) alcanza $\approx$ 93,6 % de accuracy combinada en test; E2 demuestra que mezclar PlantDoc es viable pero introduce una penalización moderada ($\approx$1,7 puntos) coherente con el *domain shift* entre laboratorio y campo.
- **Despliegue en producción en la UPV**: backend containerizado (Docker Compose, Gunicorn, MongoDB), API accesible en `https://plantas.gti-ia.upv.es`, landing con descargas de clientes, guía de usuario y encuesta de usabilidad; inferencia centralizada en servidor.
- **Marco generalizable**: derivado de Foliarium, un repositorio paralelo desacopla el dominio fitosanitario y parametriza las variables predictivas y las cabezas del modelo; se documenta en el apéndice F como extensión del trabajo, sin constituir un segundo caso de estudio evaluado en esta memoria.

En coherencia con el enfoque declarado en la introducción, dado el **volumen limitado de imágenes propias** capturadas en condiciones reales, el esfuerzo principal se invirtió en construir la **infraestructura extensible** —adquisición, integración, experimentación y operación— más que en optimizar al máximo la precisión del clasificador. Los entrenamientos documentados son **pruebas iniciales** que demuestran que el ciclo completo funciona y proporcionan una línea base cuantitativa para iteraciones futuras. El objetivo final —un modelo que generalice bien en imágenes tomadas en entorno real— depende

de un **proceso acumulativo**: hace falta tiempo con Foliarium desplegado, en uso por distintos colectivos (usuarios, agrónomos, etiquetadores), para ir construyendo y validando un corpus representativo; en el horizonte del TFG, la aportación principal es haber dejado operativo ese canal de recopilación y experimentación.

Desde el punto de vista formativo, el proyecto ha permitido integrar visión por computador, ingeniería de software (API REST, contenedores, clientes multiplataforma), gestión de datos y despliegue en un entorno real de la universidad, en línea con los objetivos de colaboración GTIIA–agrónomos y con el marco COSASS/Cátedra Planeta descrito al inicio de la memoria.

10.2 Limitaciones

Las restricciones más relevantes del trabajo, ya parcialmente discutidas en capítulos anteriores, se sintetizan aquí:

- **Datos reales propios escasos**: la fuente «App» aporta actualmente pocas imágenes validadas; las conclusiones sobre generalización en campo se apoyan principalmente en PlantDoc como proxy de condiciones no controladas, no en un corpus extenso capturado *in situ* por usuarios finales.
- **Construcción del corpus a largo plazo**: un dataset suficiente para entrenar y evaluar con rigor en condiciones reales requiere meses o años de captura colaborativa, validación experta y curación por perfiles distintos (campo, fitopatología, ingeniería de datos). El TFG entrega el **marco** para ese proceso; los resultados de clasificación aquí reportados deben leerse como evidencia de viabilidad técnica, no como cierre del objetivo agronómico final.
- **Experimentación no exhaustiva**: no se ha realizado búsqueda sistemática de hiperparámetros, comparativa de arquitecturas CNN, evaluación de formatos Grayscale/Segmentados ni *data augmentation* en entrenamiento; todos los experimentos comparten la plantilla base (Adam, 10 épocas, `fine_tune: top`, preprocesado determinista).
- **Arquitectura fija**: MobileNetV2 se mantiene por herencia del diseño inicial orientado a móvil, pese a que la inferencia actual es en servidor; no se ha cuantificado el coste/beneficio de modelos más pesados.
- **Alineación taxonómica incompleta**: PlantDoc y otras fuentes públicas no cubren exactamente el mismo espacio de clases que PlantVillage; parte del filtrado y mapeo es manual y puede introducir ruido en experimentos multi-fuente.
- **Validación funcional manual**: la plataforma se ha verificado con pruebas *end-to-end* manuales y feedback informal; no existe suite automatizada ni matriz formal de casos por rol en esta memoria. Los resultados de la encuesta de usabilidad estaban pendientes de cierre al redactar este capítulo.
- **Sin validación agronómica experta**: las métricas son puramente de clasificación automática; no se ha contrastado con diagnóstico de técnicos fitosanitarios ni ensayos en campo.

10.3 Trabajo futuro

Las líneas de continuación más naturales del proyecto, ordenadas por dependencia con el estado actual de Foliarium, son:

10.3.1. Datos y fuentes

- **Ampliar imágenes de la fuente «App»** mediante campañas de captura con agrónomos y usuarios de la encuesta, priorizando clases poco representadas y condiciones de iluminación variadas.
- **Integrar conjuntos adicionales** (PlantCLEF, IP102, Plant Pathology en Kaggle, etc.) completando scripts de importación y mapeo taxonómico hacia el catálogo `Clases` de MongoDB.
- **Activar formatos alternativos** en experimentos: comparativa sistemática Color vs. Grayscale vs. Segmentado sobre los mismos filtros de `config.yaml`.
- **Enriquecer la taxonomía con criterio experto**: los campos `clasificacion` (p. ej. Virus, Carencia, Plaga, Hongo), `nombre_cientifico` y las etiquetas legibles en español/inglés de la colección `Clases` están previstos en el esquema pero en su mayoría vacíos o derivados automáticamente de PlantVillage. Queda pendiente una **revisión agronómica** que fije la traducción definitiva bidireccional (inglés ↔ español), el nombre científico de cada condición y la tipología fitosanitaria, de modo que la interfaz, los informes y las matrices de confusión reflejen nomenclatura validada y no solo convenciones heredadas del dataset de laboratorio.

10.3.2. Modelo y experimentación

- **Comparativa de arquitecturas**: EfficientNet, ResNet u otros *backbones* sobre los splits ya generados, aprovechando la inferencia en servidor.
- **Barrido de hiperparámetros y fine-tuning**: épocas, `lr`, modos `fine_tune` (`none`, `top` y completo), pesos relativos entre cabezas.
- **Data augmentation**: integrar transformaciones aleatorias configurables en `train_model` y comparar su efecto frente a ampliar el corpus con imágenes reales validadas.
- **Análisis de errores en profundidad**: explotar `misclassified_test.json` y el notebook `misclassified_and_topk.ipynb`; estudiar métricas *top-k* y confianza en producción.
- **Validación con** `solo_validadas: true` cuando exista masa crítica de imágenes revisadas por etiquetadores.

10.3.3. Plataforma, despliegue y usuarios

- **Publicación ampliada**: Google Play, cliente iOS, PWA o mejoras de accesibilidad web.
- **Pruebas automatizadas**: `pytest` sobre endpoints críticos de la API e integración continua en el repositorio.

- **Cerrar el ciclo de feedback**: analizar la encuesta de usabilidad, iterar la interfaz Flet y documentar incidencias en un registro formal.
- **Validación agronómica**: pilotos con expertos del entorno colaborador (Facultad de Ciencias Agrarias) sobre predicciones en imágenes de campo no vistas en entrenamiento.

10.3.4. Investigación aplicada

- **Estrategias de adaptación de dominio** (*domain adaptation*, mezcla de fuentes ponderada, entrenamiento por etapas laboratorio → campo) para reducir la brecha observada entre E1 y E2.
- **Integración con líneas COSASS**: gemelos digitales, sensores en *edge* o alertas tempranas apoyadas en el mismo catálogo de clases y en modelos desplegados desde Foliarium.

Bibliografía

[1] S. P. Mohanty, D. P. Hughes y M. Salathé. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7:1419, 2016.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] D. Singh, N. Singh, A. Kaur, L. J. G. Villalba, L. O. P. Pascual, J. Kim y K.-S. Choi. PlantDoc: A Dataset for Visual Plant Disease Detection. *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, 2020.

[4] H. Goëau, P. Bonnet, M. J. M. Y. Joly, W. Joly, C. Selmi, I. Molino, P. Carré, E. Mouysset, J.-C. Melet, C. Lamour, J. Champ, A. Affouard, J.-F. Molino, D. Barthélémy, N. Boujemaa, E. Moulin, D. Story, H. G. Mac Seóighe, A. Hanbury, H. Glotin, F. Champagnat, M. Dufour-Kowalski, S. Joly, A. Baffou, H. de Solan, T. Laga, A. Chouaki, A. Boukhers, Z. Akata, I. Naamani y A. Joly. Overview of PlantCLEF 2022: image-based plant identification in the wild. *CLEF 2022 Working Notes*, 2022.

[5] X. Wu, C. Zhan, L. Luo, Y. Zhang, L. Yang, J. Wu y Y. Wang. IP102: A Large-Scale Benchmark Dataset for Insect Pest Recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[6] A. Too, D. S. Weaver y B. M. Wong. Using Deep Learning for Image-Based Plant Disease Detection. Plant Pathology 2020 — FGVC7 Workshop at CVPR (Kaggle competition), 2020.

[7] K. P. Ferentinos. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145:311–318, 2018.

[8] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk y D. Stefanovic. Deep neural networks based recognition of plant diseases by leaf image classification. *Computational Intelligence and Neuroscience*, 2016.

[9] Universitat Politècnica de València. Investigadores de la UPV desarrollan un gemelo digital para incorporar la IA a zonas rurales de la España vaciada (proyecto COSASS). Noticia UPV, 2023. https://www.upv.es/noticias-upv/noticia-14364-proyecto-cosas-es.html

[10] Universitat Politècnica de València. Cátedra de Cooperación y Desarrollo Sostenible-Eje Planeta. https://www.upv.es/contenidos/CAPLANET/

APÉNDICE A
Guía de usuario

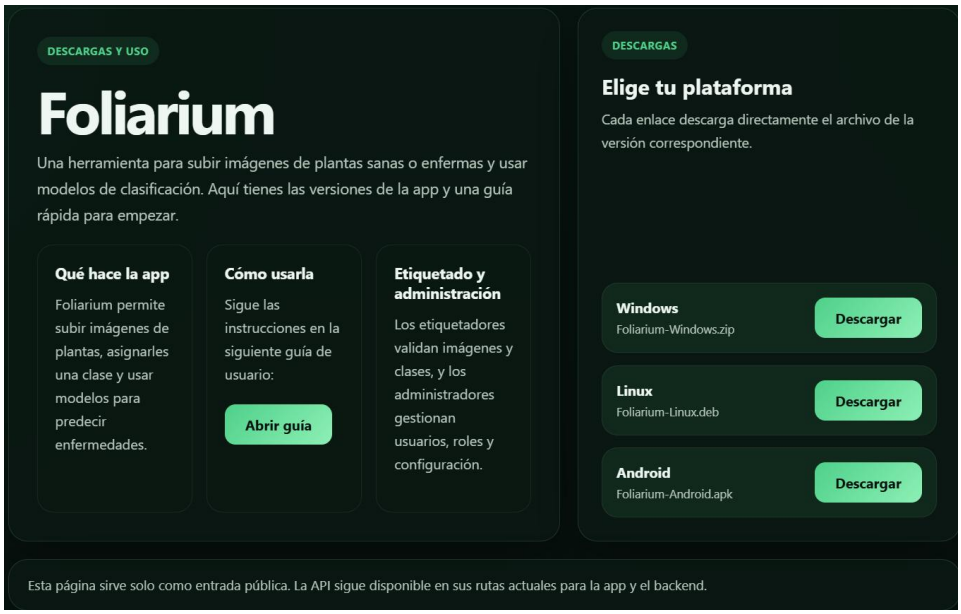
Este apéndice recoge la **guía de usuario** de Foliarium: instalación de los clientes (Windows, Linux y Android), configuración de la conexión al servidor de producción (`https://plantas.gti-ia.upv.es`) y funciones disponibles por rol (`usuario`, `usuario+`, `etiquetador`, `admin`). El contenido operativo se mantiene en un documento PDF independiente —actualizado junto con las versiones de la aplicación— para facilitar su distribución desde la landing (`/guia-usuario`) sin duplicar capturas en el cuerpo de la memoria.

Guía para Usuarios Finales

1. Descargar la aplicación

Descarga la versión de Windows, Android o Linux desde el enlace de distribución de la página web.

En el caso de Windows, el archivo final será un `.exe` que podrás abrir con doble clic.



[Imagen: Pantalla de descarga]

2. Instalar la aplicación

En **Android** se descarga directamente la apk. Busca el archivo descargado, haz click en él y se te instalará la app. Puede que tengas que activar la configuración de instalar desde origen externo.

En **Windows** se descarga un archivo .zip. Tendrás que extraer todo en la carpeta que desees y luego ejecutar el archivo Foliarium.exe en la carpeta extraída.

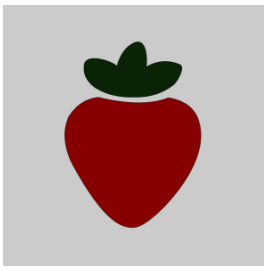
En **Linux** se descarga un archivo .deb. Solo tendrás que hacer click en ese archivo, darle a instalar y abrir Foliarium desde el menú de aplicaciones.

Instrucciones de instalación paso a paso

3. Primer uso

3.1. Iniciar la aplicación

- **Android**: abre `Aplicaciones` y busca `Foliarium`.
- **Windows**: haz doble clic en el icono de Foliarium.exe.
- **Linux**: Busca Foliarium en el menú de aplicaciones y ejecútalo.



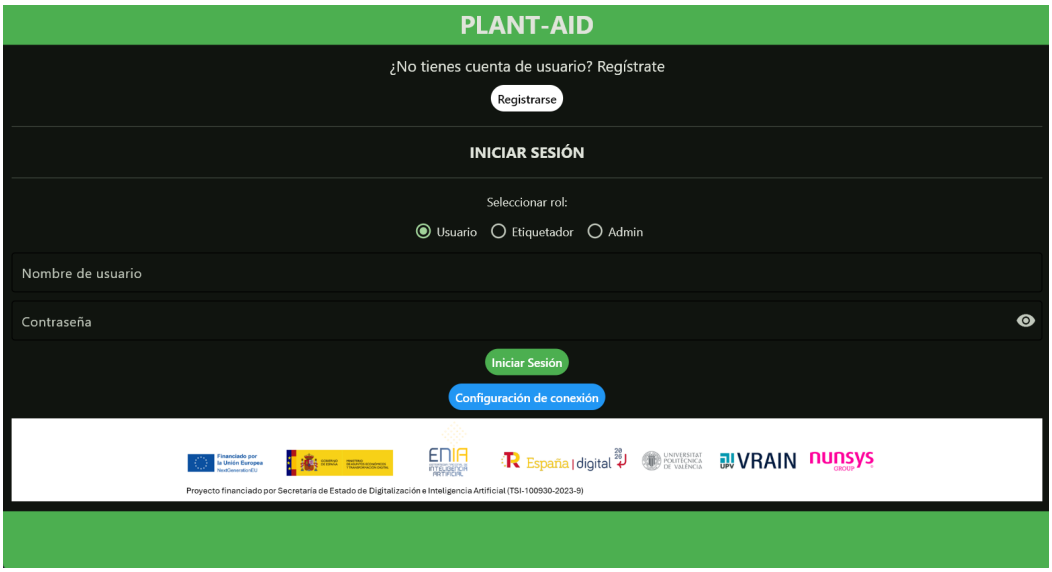
[Imagen: Icono]

3.2. Primera pantalla

Si no tienes una cuenta todavía, regístrate y se te dará el rol de Usuario. Si quieres otro rol, contacta con el desarrollador.

Una vez tengas tu cuenta, selecciona tu rol e inicia sesión con tus credenciales.

Si te da error de conexión, asegúrate de que la dirección a la que te estás conectando es la correcta en **Configuración de conexión**. URL de la API: plantas.gti-ia.upv.es

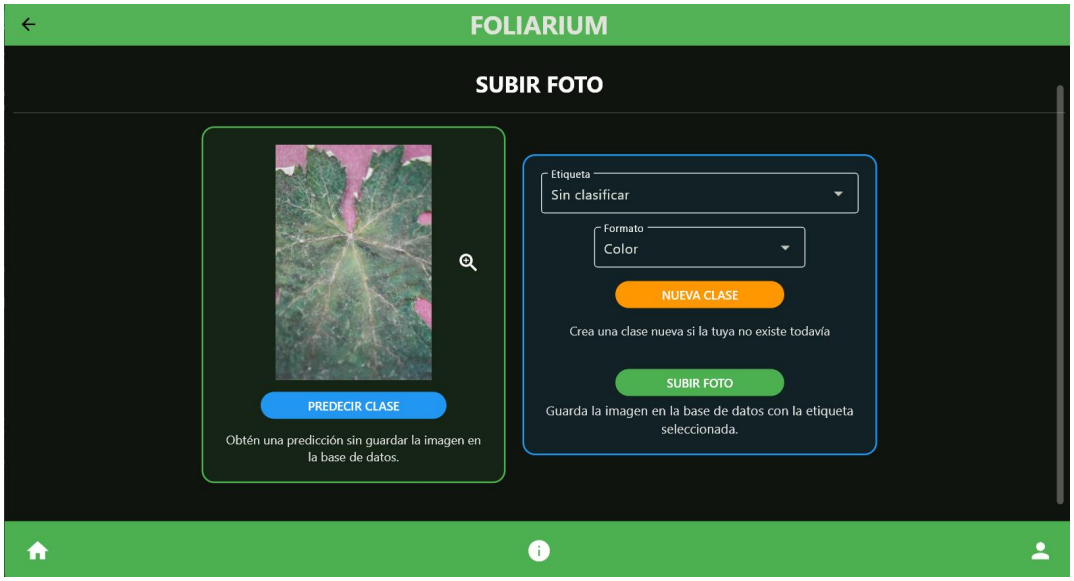


[Imagen: Pantalla de inicio de sesión]

4. Funciones principales para usuarios

4.1. Subir imágenes

- Pulsa el botón **Subir Foto**.
- Selecciona una imagen de una hoja, desde móvil debes tener la imagen ya en la galería.
- Elige la etiqueta de tu imagen:
    1. Primero busca en el desplegable entre las clases ya existentes si hay alguna combinación planta-enfermedad adecuada
    2. Si no existe, crea una clase nueva. Deberás indicar al menos la planta y el nombre común de la enfermedad
    3. Si tienes dudas sobre la enfermedad, simplemente escribe “healthy”
- La imagen se añade a la base de datos.
- Si está disponible, un modelo puede predecir la etiqueta de tu imagen.



[Imagen: Pantalla de subida de imagen]

4.2. Ver experimentos

- Pulsa el botón **Experimentos**.
- Aquí se muestran los modelos ya entrenados y sus características.
- Puedes ver los resultados de cada modelo y compararlos.
- Puedes crear un nuevo experimento y solicitar su entrenamiento.

[Imagen: Lista de experimentos]

4.3. Subir imágenes masivamente

- Pulsa el botón **Subida masiva**.
- Sube un conjunto de imágenes desde una carpeta con una estructura determinada.
- Esta funcionalidad es delicada, contacta con el desarrollador antes de usarla.
- Se requiere rol de **Usuario+**.

[Imagen: Pantalla de subida masiva]

5. Funciones principales para etiquetadores

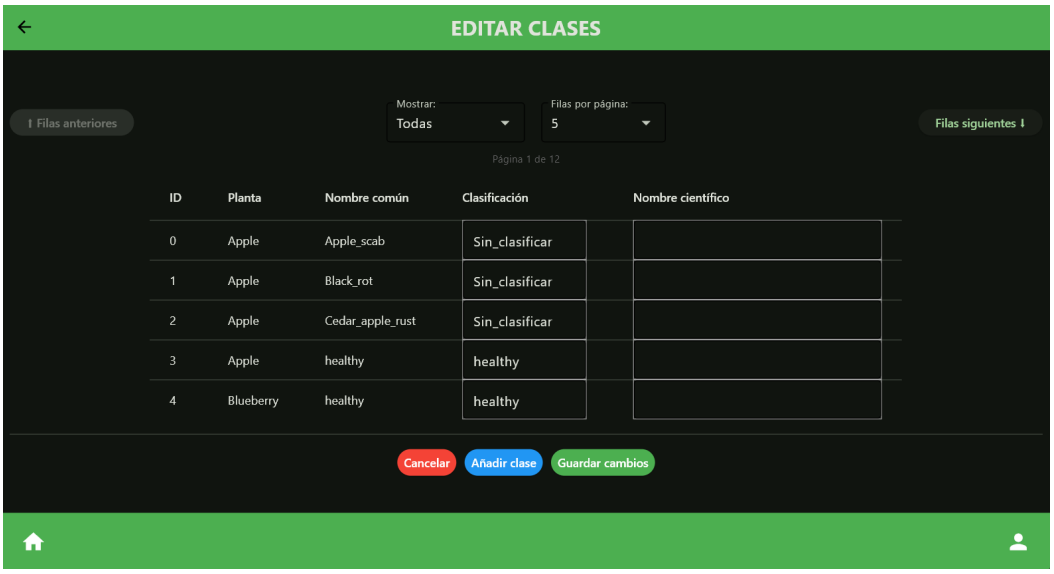
5.1. Etiquetar o validar imágenes

- Pulsa el botón **Etiquetar/Validar**.
- Filtra qué imágenes quieres comprobar.
- Selecciona una imagen de la lista.
- Selecciona la etiqueta.
- Haz clic en **Etiquetar**.
- Valida si estás seguro/a de que la etiqueta es correcta.

[Imagen: Pantalla de etiquetado o validación]

5.2. Editar clases

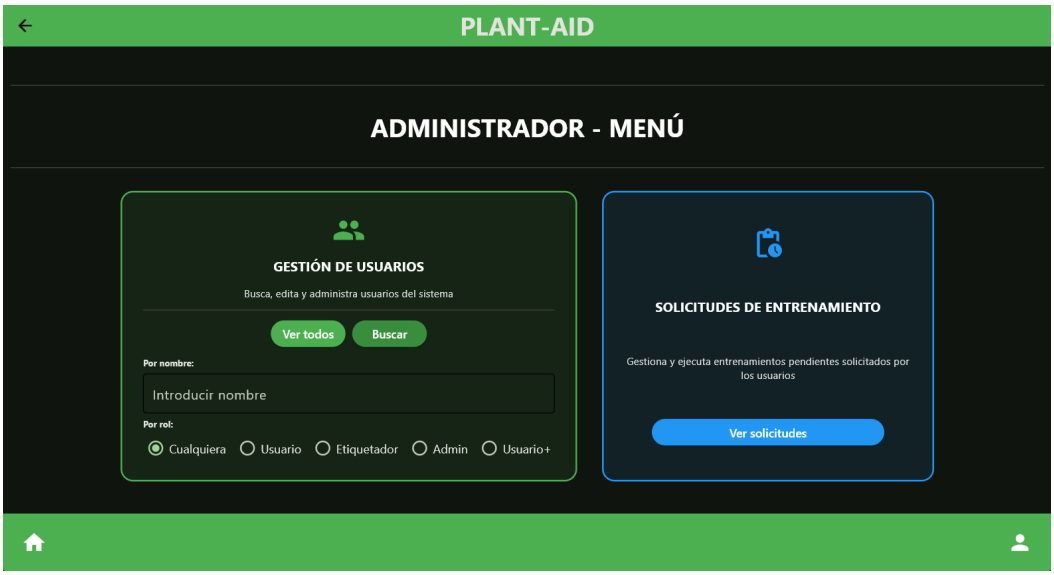
- Pulsa el botón **Editar clases**.
- Navega a través de las clases existentes.
- Modifica la clasificación de la enfermedad o su nombre científico.
- Guarda los cambios.
- Añade clases si es necesario.
- Procede con cuidado y después de consultar.



[Imagen: Pantalla de edición de clases]

6. Funciones principales para administradores

- Gestión de usuarios.
- Gestión de solicitudes de entrenamiento.



[Imagen: Panel de administración]

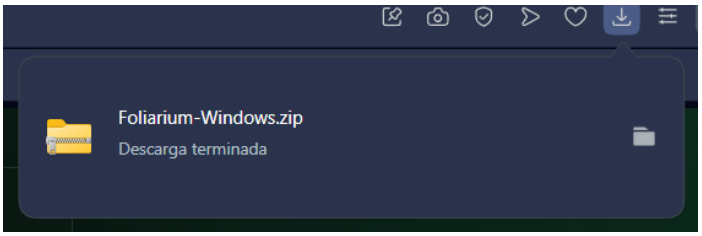
7. Soporte

Si tienes problemas o preguntas, contacta con: pralfes@etsinf.upv.es

INSTALACIÓN PASO A PASO

Windows

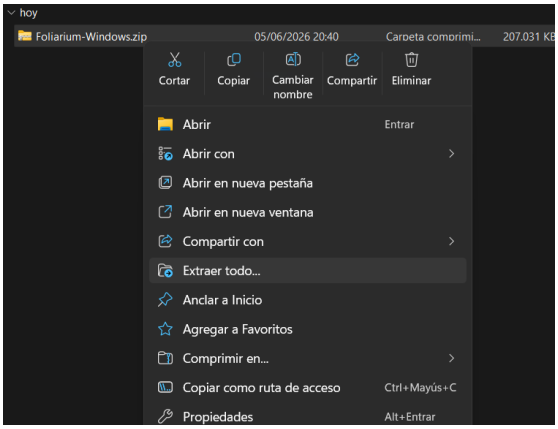
1. Haz click en el botón de descarga de la versión de **Windows** en la página web.



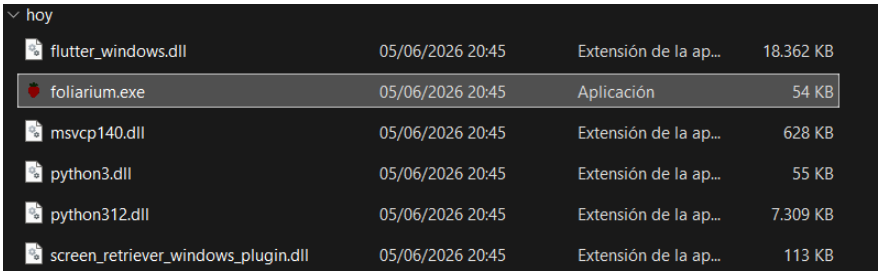
2. Dale a mostrar en carpeta o busca el zip en tu carpeta de **descargas**.

3. Mueve el zip a la carpeta donde quieras tener la aplicación.

4. Haz click derecho en el zip y dale a **extraer todo**.

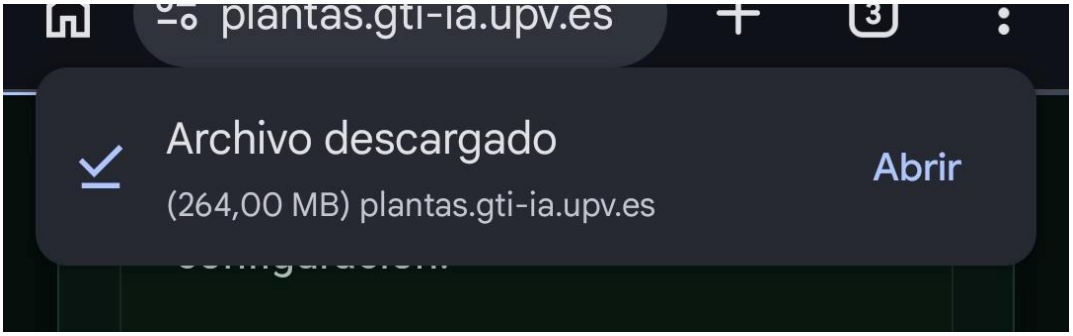


5. Entra en la carpeta extraída y ejecuta **foliarium.exe**

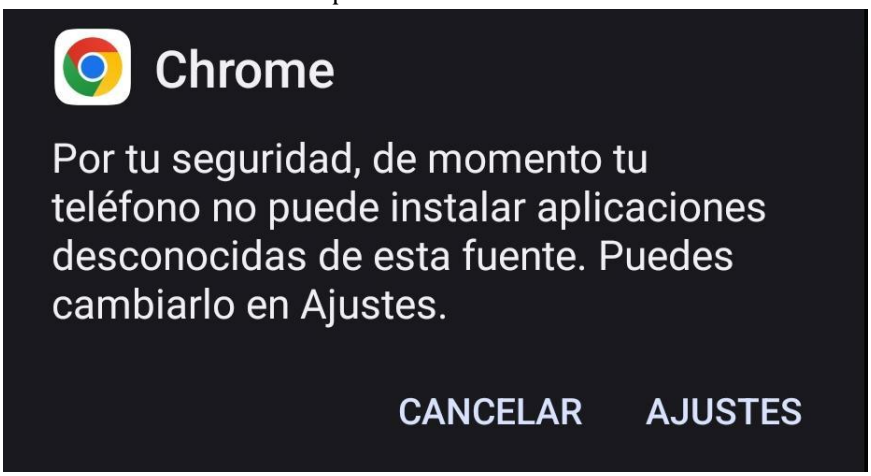


Android

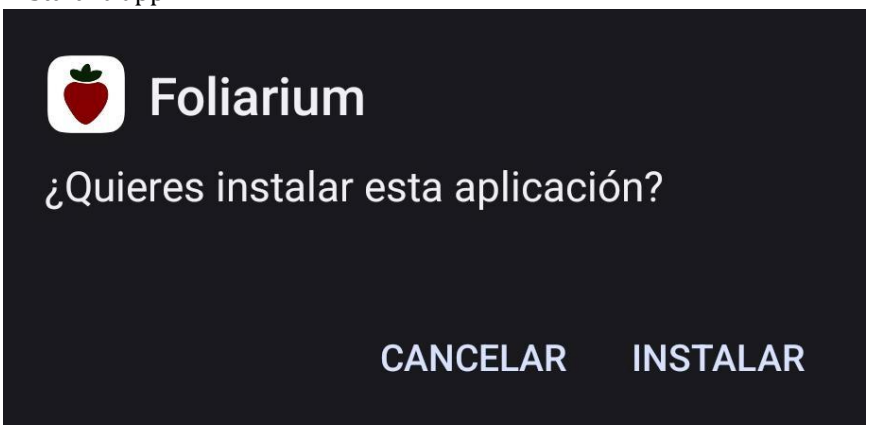
1. Pulsa en el botón de descarga de la versión de **Android** en la página web.



2. **Autoriza** esta fuente si te lo pide



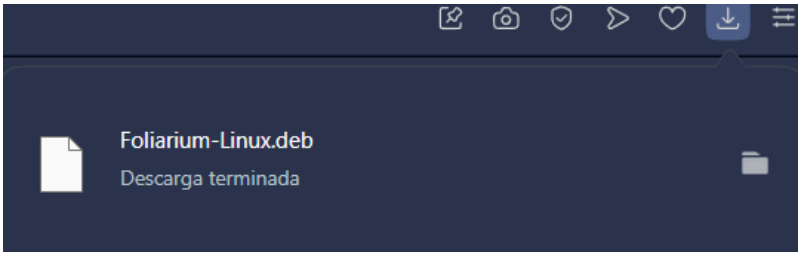
3. **Instala** la app



4. Ya tendrás en tu dispositivo **Foliarium**

Linux

1. Haz click en el botón de descarga de la versión de **Linux** en la página web.



2. Click en el archivo y después en **Instalar**.
3. Abrir **Foliarium** desde el menú de aplicaciones.

APÉNDICE B

Reproducción de experimentos y despliegue técnico

Esta sección está dirigida a quien deba **replicar entrenamientos** o **desplegar el backend** en un entorno similar al del servidor original. La guía de uso para personas no técnicas está en el apéndice A.

B.1 Ejecución local sin containerización

Docker Compose es el camino recomendado para **producción** y para replicar el entorno del servidor UPV (véase § B.2), pero **no es obligatorio** para desarrollo ni para repetir experimentos. El mismo repositorio puede ejecutarse en una máquina de trabajo convencional:

1. Clonar el repositorio e instalar Python (`pip install -r requirements.txt`; entorno `venv` o Conda). Detalle en `docs/DESARROLLO.md`.
2. Tener MongoDB accesible en local (instancia local o contenedor solo de `mongo`) y configurar `.env` con `MONGO_URI` y credenciales.
3. Inicializar bases y usuario administrador: `python scripts/setup_bbdd.py`.
4. Arrancar la API: `python run_server.py` (opcionalmente `-https -dev` en desarrollo).
5. Importar imágenes y lanzar entrenamientos directamente sin por `docker compose exec`, p. ej. `python experiments/E1_PlantVillage_lab/run_experiment.py`.

Los scripts de comparación (`compare_experiments.py`), regeneración de gráficos y utilidades de `scripts/` funcionan igual contra MongoDB y el árbol `experiments/` montado en disco. La containerización aporta aislamiento, healthchecks y paridad con el servidor desplegado; la ejecución nativa reduce fricción en iteraciones locales de modelado.

B.2 Despliegue con Docker Compose

Requisitos: Docker Engine, Docker Compose y Git. Procedimiento resumido (detalle en `docs/DOCKER.md` del repositorio):

1. Clonar el repositorio y copiar la plantilla de entorno: `cp .env.docker.example .env`.
2. Ajustar `PUBLIC_API_BASE_URL` (p. ej. `https://plantas.gti-ia.upv.es` en producción).
3. Construir y levantar: `docker compose build && docker compose up -d`.
4. Primera puesta en marcha: `docker compose -profile init run -rm initdb` (inicializa MongoDB y usuario administrador).
5. Comprobar salud: `docker compose ps` y `curl http://localhost:5001/health`.

Servicios principales: `mongo` (MongoDB 7), `api` (Flask + Gunicorn, puerto 5001) e `initdb` (perfil puntual). Volúmenes montados: `imagenes/`, `models/`, `experiments/`, `data/`, `logs/`, `downloads/`.

B.3 Crear y ejecutar un experimento

1. Crear la carpeta desde la plantilla: `python scripts/make_experiment.py -nombre MI_EXP -config experiments/BASE/config.yaml` (ajustando filtros en el YAML generado).
2. Entrenar dentro del contenedor (evita *timeouts* HTTP):

   ```
   docker compose exec api python experiments/MI_EXP/run_experiment.py
   ```

3. Artefactos generados en `experiments/MI_EXP/`: `data/*.csv`, `models/*.pth`, `results/metrics.json`, gráficos y `misclassified_test.json`.

Para repetir E1 o E2, usar las carpetas `experiments/E1_PlantVillage_lab/` y `experiments/E2_PlantVillage_PlantDoc_mix/` ya versionadas, con PlantVillage y PlantDoc importados en MongoDB.

B.4 Comparar experimentos y regenerar gráficos

```
docker compose exec api python scripts/compare_experiments.py \
  E1_PlantVillage_lab E2_PlantVillage_PlantDoc_mix

docker compose exec api python scripts/regenerate_experiment_plots.py \
  E1_PlantVillage_lab
```

Los gráficos comparativos se guardan en `experiments/comparison/`.

B.5 Copias de seguridad

Volcado manual de las bases `Plantas` y `Usuarios`:

```
sh scripts/backup_mongo.sh
```

Las imágenes en disco y los pesos en `models/` no se incluyen en ese volcado; conviene respaldar también los directorios montados en la VM.

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

APÉNDICE C
Fragmentos de código clave

Extractos simplificados del repositorio que ilustran decisiones descritas en los capítulos 5 y 7.

C.1 Arquitectura multitarea (`utils/model.py`)

```python
def build_model(config):
    base_model = mobilenet_v2(weights=config["weights"])
    if config["fine_tune"] == "top":
        for param in base_model.features.parameters():
            param.requires_grad = False
    return MultiTaskMobileNetV2(
        base_model,
        num_plantas=len(config["plantas"]),
        num_enfermedades=len(config["enfermedades"]),
    )
```

C.2 Restricción de combinaciones válidas en inferencia (`main.py`)

Tras predecir planta y enfermedad de forma independiente, se filtran las enfermedades que existen en entrenamiento para la planta predicha:

```python
combinaciones_validas = set(zip(df_train["planta"],
                                df_train["nombre_comun"]))
enfermedades_validas = [e for (p, e) in combinaciones_validas
                        if p == cultivo_pred]
enfermedad_pred = max(probs_filtradas, key=probs_filtradas.get)
```

C.3 Pipeline de experimento (`run_experiment.py`)

```python
config = prepare_data_splits(db, config, save_dir=DATA_DIR)
model = build_model(config)
model, history = train_model(model, train_loader, val_loader, config, DATA_DIR)
metrics = evaluate(model, val_loader, test_loader, config, RESULTS_DIR)
save_metrics(metrics, METRICS_PATH)
```

APÉNDICE D
Figuras y tablas complementarias

Material gráfico adicional al capítulo 8 (figuras principales en `memoria/figures/cap8/`).

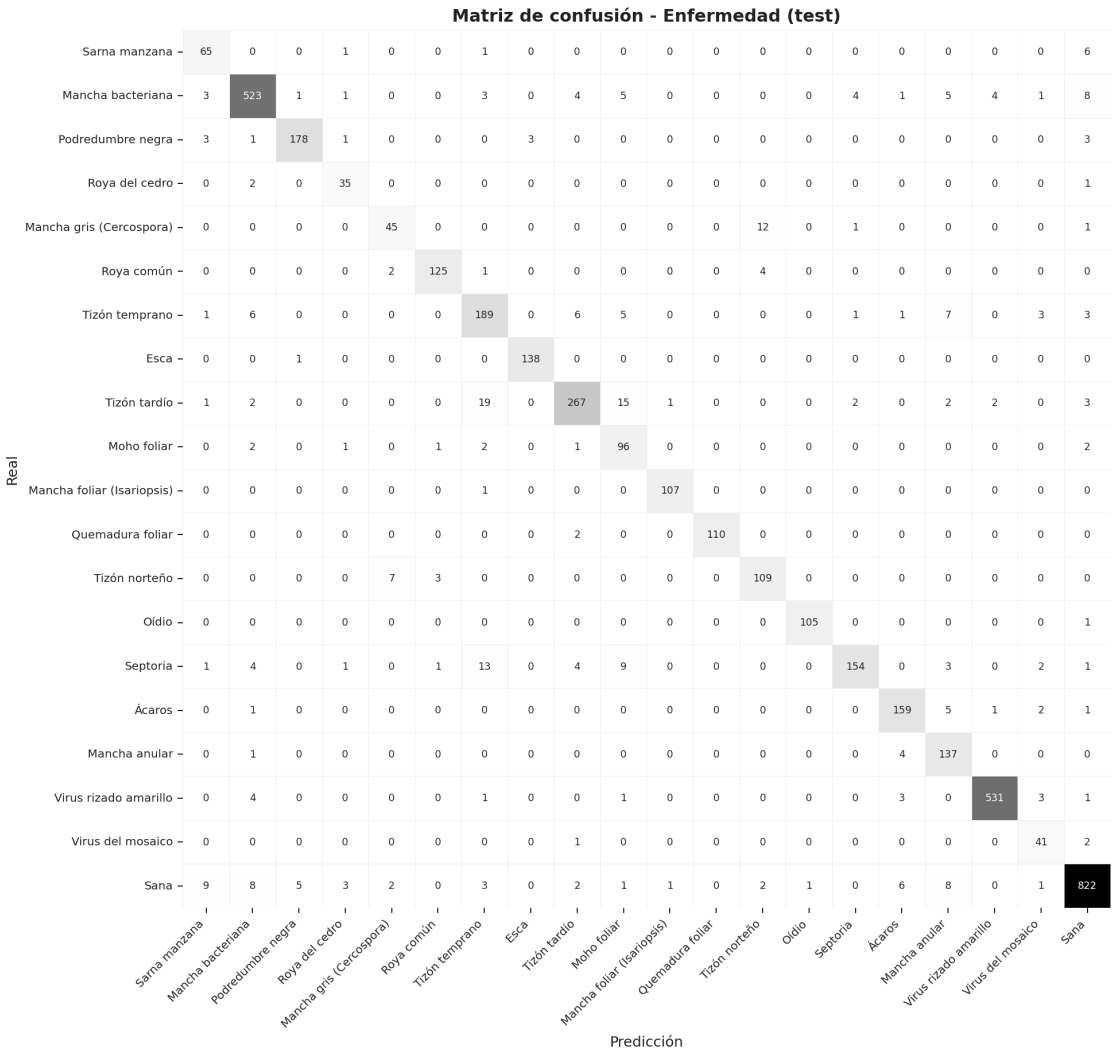


Figura D.1: Matriz de confusión de enfermedad en test (E2, PlantVillage + PlantDoc).

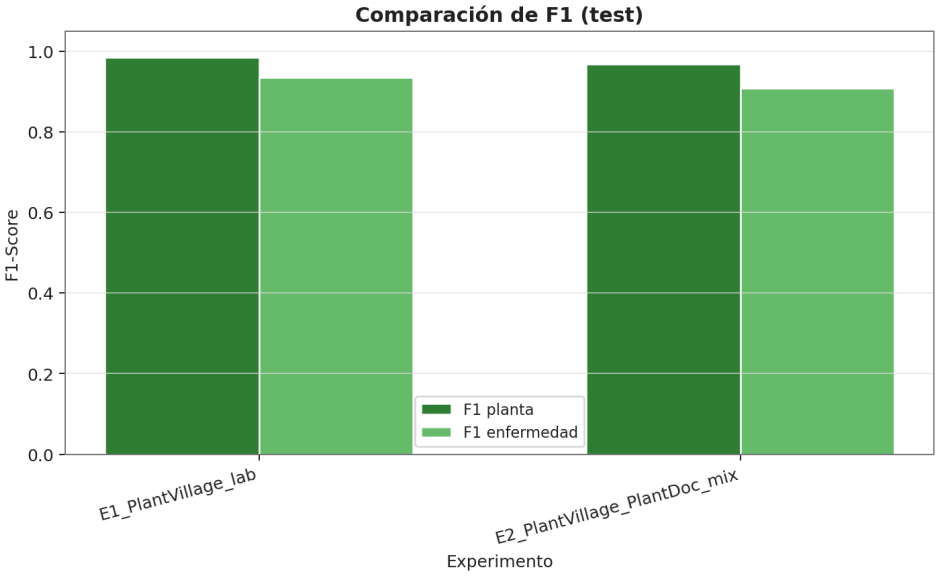


Figura D.2: Comparativa de F1 macro combinado entre E1 y E2.

APÉNDICE E
Requisitos del sistema

Este apéndice recoge el listado detallado de requisitos funcionales y no funcionales derivados de los escenarios de uso de Foliarium. No forma parte del hilo principal del capítulo 3 (análisis del problema y solución elegida), pero sirve como referencia para la implementación (capítulos 4–5) y la validación (capítulo 9).

E.1 Requisitos funcionales

- **RF-01 Autenticación**: registro e inicio de sesión de usuarios con selección de rol activo.
- **RF-02 Subida de imágenes**: registro de imágenes en MongoDB con clase, fuente, formato, usuario y estado de validación.
- **RF-03 Consulta y filtrado**: recuperación paginada de documentos de imagen con filtros por metadatos (clase, fuente, formato, validación, etc.).
- **RF-04 Etiquetado y validación**: clasificación manual de imágenes pendientes y marcado explícito como validadas.
- **RF-05 Gestión de taxonomía**: consulta, edición y ampliación del catálogo de clases (plantas y enfermedades) y de etiquetas auxiliares (fuentes, formatos, campos personalizados).
- **RF-06 Subida masiva**: incorporación de lotes de imágenes desde el cliente o mediante endpoints y scripts de importación.
- **RF-07 Experimentos**: creación de carpetas de experimento con `config.yaml`, generación de particiones train/validation/test desde la base de datos y ejecución de entrenamiento.
- **RF-08 Resultados**: consulta de métricas, gráficos y comparación entre experimentos.
- **RF-09 Predicción**: inferencia sobre una imagen con un modelo seleccionado, devolviendo predicciones de planta y enfermedad coherentes con las clases vistas en entrenamiento.
- **RF-10 Administración de usuarios**: búsqueda, modificación de credenciales y gestión de roles (solo administrador).

- **RF-11 Configuración de conexión**: posibilidad de indicar la URL del backend desde el cliente para operar en local o en producción.
- **RF-12 Landing y distribución**: página pública en la raíz del servidor con enlaces de descarga de clientes y guía de usuario.

E.2 Requisitos no funcionales

- **RNF-01 Seguridad**: autenticación basada en JWT para endpoints sensibles; contraseñas almacenadas con hash bcrypt en el servidor; limitación de intentos en login y registro; validación de tipo y tamaño de ficheros subidos.
- **RNF-02 Disponibilidad**: despliegue persistente del backend en servidor dedicado, con contenedores Docker y comprobaciones de salud (`/health`).
- **RNF-03 Escalabilidad de datos**: separación entre metadatos en MongoDB y ficheros en disco; esquema extensible de campos mediante la colección `Campos`.
- **RNF-04 Multiplataforma**: clientes Flet para Windows, Linux y Android empaquetados e instalables de forma independiente.
- **RNF-05 Reproducibilidad**: experimentos versionados por carpeta con configuración YAML y artefactos de salida en `experiments/`.
- **RNF-06 Mantenibilidad**: scripts auxiliares para inicialización de base de datos, importación masiva y operaciones de mantenimiento sin depender exclusivamente de la interfaz gráfica.
- **RNF-07 Usabilidad**: interfaz por rol con flujos diferenciados para captura, etiquetado y administración; posibilidad de ampliar y descargar imágenes desde el cliente.

E.3 Roles de usuario

- **Usuario (`usuario`)**: subida individual, experimentos, predicción.
- **Usuario ampliado (`usuario+`)**: subida masiva desde el cliente.
- **Etiquetador (`etiquetador`)**: revisión y validación de imágenes pendientes.
- **Administrador (`admin`)**: gestión de usuarios, roles, clases, entrenamientos y modelos publicados.

APÉNDICE F
Generalización del marco

F.1 Motivación y alcance

Durante el desarrollo de este TFG quedó acordado con el tutor, Carlos Carrascosa Casamayor, que la aportación principal no residía en optimizar métricas de clasificación en campo en el plazo disponible, sino en entregar un **marco de trabajo reproducible**: integración multi-fuente, persistencia en MongoDB, API Flask, cliente Flet multiplataforma, experimentos versionados con `config.yaml`, roles de usuario, inferencia centralizada y despliegue Docker. Los experimentos E1 y E2 sobre PlantVillage y PlantDoc (capítulo 8) funcionan como **pruebas de viabilidad** del ciclo completo, no como núcleo evaluable del entregable.

A partir de ese planteamiento, y una vez cerrada la versión fitosanitaria documentada en los capítulos 4–9, se derivó un segundo desarrollo software: el **marco generalizado** alojado en la carpeta local `Esqueleto`, rama `generalize/skeleton` del repositorio GitHub https://github.com/PabloRalloFes/RepoPlantas. Conserva la misma filosofía de plataforma (imágenes + metadatos + fuentes + entrenamiento + predicción + despliegue), pero sustituye el acoplamiento al dominio planta/enfermedad por **variables predictivas configurables**. Este apéndice documenta esa extensión como complemento de la memoria; **no constituye un segundo TFG ni una evaluación en otro dominio dentro del presente trabajo**.

F.2 Relación con Foliarium

Foliarium y el marco generalizado comparten un origen común: la misma arquitectura cliente–servidor, los mismos patrones de autenticación JWT, la misma separación entre ficheros en disco y metadatos en MongoDB, y la misma organización de experimentos bajo `experiments/`*nombre*`/`. La evolución puede describirse como un *fork* conceptual orientado a reutilización:

- **Foliarium** (repositorio principal, rama `main`): sistema concreto para clasificación fitosanitaria; taxonomía centrada en `planta` y `enfermedad`; scripts de importación de PlantVillage y PlantDoc; despliegue documentado en la UPV.
- **Marco generalizado** (carpeta `Esqueleto`, rama `generalize/skeleton`): mismo esqueleto operativo, con metadatos y modelos desacoplados del vocabulario fitosanitario; datos de demostración genéricos y documentación de adaptación en `docs/DESARROLLO.md`.

La Tabla F.1 resume las diferencias estructurales más relevantes. En lo que ambos sistemas comparten (roles, formatos Color/Grayscale/Segmentado, *pipeline* de importación, Docker, inferencia en servidor), se remite al lector a los capítulos 4–9 de la memoria principal.

Tabla F.1: Contraste entre Foliarium y el marco generalizado (`Esqueleto`).

Aspecto	Foliarium	Marco generalizado
Taxonomía de clases	Campos fijos `planta` y `nombre_comun` (enfermedad) en `Clases`	Campos definidos por el adaptador del proyecto; identificador canónico `class_label`
Variables a predecir	Siempre dos cabezas: planta y enfermedad	Lista `target_fields` en `ProjectConfig` (1 o N cabezas)
Construcción del modelo	`MultiTaskMobileNetV2` con `num_plantas` y `num_enfermedades` fijos	`SingleTaskMobileNetV2` o `MultiTaskMobileNetV2` según cardinalidad de `target_fields`
Filtros al crear experimento	Desplegables de plantas, enfermedades y fuentes	Descubrimiento dinámico de campos escalares en `Docs` vía `/opciones_filtros_docs`
Inicialización de BD	`scripts/setup_bbdd.py` con taxonomía fitosanitaria	Mismos JSON semilla en `src/`, más `project_targets.json` y colección `ProjectConfig`
Scripts de dominio	`prepare_plantdoc_import.py`, mapeos PlantVillage, etc.	Eliminados o relegados a `scripts/legacy`; demo con `setup_demo_minimal.py`
Nombre de base de datos	`Plantas` (producción)	Configurable (`DB_NAME`, por defecto `Demo`)

F.3 Qué se mantiene del marco Foliarium

Sin repetir el detalle de implementación ya descrito en la memoria, el marco generalizado hereda los siguientes bloques:

- **Arquitectura en tres capas**: cliente Flet (`main_app.py`), capa HTTP (`logicav3.py`) y API Flask (`main.py`) sobre MongoDB.
- **Esquema de persistencia**: colecciones `Fuente`, `Clases`, `Docs`, `Campos`, `Formato` y base de usuarios; imágenes en disco con rutas relativas en documentos.
- **Formatos de imagen**: color, escala de grises y segmentación, con la misma lógica de generación y almacenamiento.
- **Roles y flujos**: `usuario`, `usuario+`, `etiquetador` y `admin`; subida individual y masiva; validación; gestión de experimentos y modelos publicados para predicción.
- **Experimentos reproducibles**: plantilla `experiments/BASE/`, generación de splits en CSV, métricas en `results/metrics.json`, comparación entre carpetas de experimento.
- **Despliegue**: Docker Compose, Gunicorn, variables de entorno y documentación en `docs/DOCKER.md`.

- **Filosofía de inferencia**: predicción en servidor (no en el dispositivo móvil), coherente con Foliarium.

F.4 Variables predictivas configurables

El cambio central respecto a Foliarium es la **sustitución de la taxonomía fija planta + enfermedad** por un mecanismo declarativo de variables objetivo.

F.4.1. Configuración a nivel de proyecto

Al inicializar la base de datos, `scripts/setup_bbdd.py` puede leer el fichero `src/project_targets.json`, que lista los nombres de campos presentes en los documentos de la colección `Clases` que se desean predecir. El script valida que esos campos existan y persiste el resultado en la colección `ProjectConfig` (documento `_id: "project"`), accesible mediante `utils/database.get_project_config()`.

En el estado actual del repositorio generalizado, el fichero de ejemplo declara dos variables:

```
["planta", "clasificacion"]
```

Con un único campo se entrena un modelo de **una cabeza** (`SingleTaskMobileNetV2`); con varios campos, un modelo **multitarea dinámico** cuyo número de salidas coincide con la longitud de `target_fields`. La función `build_model` en `utils/model.py` resuelve las clases por campo y construye el módulo adecuado:

```
target_fields = config.get("target_fields") or project_config.get("target_fields")
if len(target_fields) == 1:
    model = SingleTaskMobileNetV2(base_model, len(classes))
else:
    head_sizes = []
    for field_name in target_fields:
        head_sizes.append(len(target_classes[field_name]))
    model = MultiTaskMobileNetV2(base_model, head_sizes)
```

Así, el mismo *backbone* MobileNetV2 que en Foliarium puede operar en modo clasificación simple (p. ej. una etiqueta `class_label`) o en modo multitarea con cabezas definidas por el catálogo del proyecto, sin modificar el código fuente del modelo.

F.4.2. Preparación de datos y entrenamiento

El módulo `utils/data.py` sustituye la lógica acoplada a `planta/enfermedad` de Foliarium. La función `prepare_data_splits` lee `target_fields` desde la configuración del experimento o desde `ProjectConfig`, proyecta los campos necesarios de `Clases` y `Docs`, aplica los filtros del `config.yaml` (fuentes, formato, clases, campos dinámicos) y genera los CSV de train/validation/test. `utils/train.py` adapta la función de pérdida: una cabeza con `CrossEntropyLoss`, o varias cabezas con pérdida ponderada por campo, de forma análoga al multitarea fitosanitario pero parametrizada por nombre de variable.

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

F.4.3. Creación de experimentos desde la interfaz

Al crear un experimento, la aplicación Flet consulta el endpoint que agrega metadatos de `Docs` (muestreo equilibrado por fuente) y ofrece **controles dinámicos de filtrado**: por cada campo escalar detectado (`class_label`, `fuente`, `planta`, `clasificacion`, etc.) el usuario puede restringir el subconjunto de datos del experimento. Esos valores se serializan en `config_variables` y se fusionan con la plantilla `experiments/BASE/config.yaml` mediante `scripts/make_experiment.py`.

Es importante distinguir dos niveles: (i) **variables a predecir** (`target_fields`), fijadas al configurar el proyecto y almacenadas en `ProjectConfig`; (ii) **filtros del experimento**, elegidos en la interfaz para acotar fuentes, clases o atributos concretos del corpus. El entrenamiento construye el número y tipo de cabezas en función del primer nivel; el segundo determina qué imágenes entran en los CSV.

F.4.4. Inferencia

El endpoint de predicción en `main.py` carga `config_final.yaml` del experimento, reconcilia `target_fields` con `ProjectConfig` y ejecuta `softmax` sobre una o varias salidas. En modo monotarea devuelve clase y confianza; en modo multitarea, una predicción por cada campo objetivo. No se mantiene la restricción fitosanitaria de coherencia planta–enfermedad tras dos inferencias independientes; cada cabeza se evalúa según las clases vistas en entrenamiento para ese campo.

F.5 Eliminación del acoplamiento fitosanitario

El marco generalizado reduce o generaliza los elementos específicos de Foliarium:

- Sustitución de `plantas/enfermedades` en `config.yaml` por `classes` y filtros genéricos.
- Catálogo `Clases` sembrado desde `src/coleccion_clases.json` y `src/campos.json`, adaptables al dominio destino.
- Scripts de importación PlantVillage/PlantDoc no forman parte del flujo por defecto; permanecen referencias legacy en `scripts/legacy/`.
- Demo autocontenida (`scripts/setup_demo_minimal.py`) con clases `Inofensiva`, `Neutra` y `Peligrosa` e imágenes en `data/Ejemplo/`, útil para comprobar el ciclo sin datos de plantas.
- Documentación de desarrollo (`docs/DESARROLLO.md`) orientada a «clonar, sembrar BD, subir imágenes y lanzar experimentos» en un proyecto arbitrario.

F.6 Escenarios de reutilización ilustrativos

El marco está pensado para problemas con la misma estructura abstracta: imágenes etiquetadas, varias fuentes, necesidad de experimentar y desplegar. A título ilustrativo (sin evaluación documentada en este TFG), encajarían escenarios como:

- Clasificación de tipos de defecto en piezas industriales (una variable `tipo_defecto`).

- Categorización de residuos con metadatos de origen y peligrosidad (dos variables: `material` y `nivel_riesgo`).
- Diagnóstico por imagen con varias etiquetas jerárquicas definidas en `Clases`, siempre que existan scripts de importación y una taxonomía coherente en MongoDB.

En todos los casos, el adaptador deberá definir la semilla de `Clases`, los campos en `Campos`, `project_targets.json` y, si procede, personalizar textos de la interfaz Flet y la landing.

F.7 Limitaciones y esfuerzo de adaptación

El marco generalizado **no es plug-and-play**. Reduce el esfuerzo de reimplementar la plataforma completa, pero exige trabajo de adaptación:

- **Backend y datos**: definir taxonomía, semillas JSON, scripts de importación masiva y convenciones de carpetas bajo `data/`*fuente*`/`.
- **Frontend**: etiquetas visibles, flujos de etiquetado y mensajes de predicción siguen parcialmente orientados al caso de uso original; conviene revisar `main_app.py` por dominio.
- **Variables objetivo**: deben existir como campos en `Clases` y declararse en `project_targets.json`; no hay, en el estado actual del código, un selector gráfico independiente para cambiar `target_fields` por experimento sin reconfigurar el proyecto.
- **Preprocesado**: la segmentación heredada de PlantVillage puede no trasladarse a otros dominios sin sustituir o desactivar esa etapa.
- **Documentación y despliegue**: `README.md` y parte de `docs/` conservan redacción fitosanitaria pendiente de actualizar; el `origin` de Git en la copia local apunta aún al repositorio de Foliarium, con la rama `generalize/skeleton` como vehículo de publicación del marco.

No se han documentado en esta memoria experimentos de clasificación en otro dominio real ni métricas comparativas frente a Foliarium. La demo mínima sirve como prueba de integración del *pipeline*, no como validación científica en un problema externo.

F.8 Estado actual y acceso al código

A fecha de redacción de este apéndice, el marco generalizado está **disponible para adaptación** en la rama `generalize/skeleton` del repositorio https://github.com/PabloRalloFes/RepoPlantas (directorio de trabajo habitual: `Esqueleto/` en el entorno local del autor). Constituye una línea de continuidad natural del TFG —extender la plataforma más allá del caso fitosanitario— sin alterar el alcance evaluado del trabajo principal, centrado en Foliarium y en las pruebas E1/E2 descritas en el capítulo 8.

Para reproducir la demo genérica documentada en `docs/DESARROLLO.md`, basta con sembrar la base (`python scripts/setup_bbdd.py`), opcionalmente ejecutar `setup_demo_minimal.py`, arrancar la API y el cliente, y crear o ejecutar un experimento bajo `experiments/`, siguiendo el mismo esquema reproducible descrito en el apéndice B para Foliarium.

Financiado por la Unión Europea NextGenerationEU · España | digital 2026 · Cátedra ENIA UPV · VRAIN UPV · nunsys GROUP

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**		X		
ODS 3. **Salud y bienestar.**			X	
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**			X	
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**	X			
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFG con los ODS.

Este Trabajo Fin de Grado, titulado *Clasificación de enfermedades en hojas de plantas mediante aprendizaje profundo con imágenes reales y de laboratorio*, ha dado lugar a **Foliarium**: una plataforma que integra fuentes heterogéneas de imágenes foliares (PlantVillage, PlantDoc y capturas propias), facilita el etiquetado colaborativo, entrena modelos de visión por computador y despliega el sistema en un servidor accesible para usuarios finales. Aunque el núcleo técnico es informático, la **finalidad de aplicación** es agronómica y ambiental: apoyar la detección temprana de problemas fitosanitarios para un uso más racional de los cultivos y de los insumos asociados.
La relación con la Agenda 2030 se articula principalmente en torno al **ODS 15 (Vida de ecosistemas terrestres)** y al **ODS 12 (Producción y consumo responsables)**, con vínculos medios al **ODS 2 (Hambre cero)** y al **ODS 9 (Industria, innovación e infraestructuras)**, y contribuciones menores al **ODS 13 (Acción por el clima)**, al **ODS 17 (Alianzas)** y al **ODS 4 (Educación de calidad)**. El resto de objetivos no forman parte directa del alcance del trabajo y se marcan como *no procede* en la tabla anterior.

ODS 15 — Vida de ecosistemas terrestres (relación alta)

El ODS 15 promueve la gestión sostenible de los bosques, la lucha contra la desertificación, la degradación de la tierra y la pérdida de biodiversidad. En el ámbito agrícola, la salud de los cultivos y la detección precoz de plagas y enfermedades contribuyen a reducir pérdidas de producción y a limitar prácticas agresivas que pueden afectar al suelo, al agua y a los ecosistemas circundantes.
Foliarium aborda esta meta de forma **indirecta pero explícita**: proporciona un marco para acumular conocimiento visual sobre el estado de las hojas, entrenar clasificadores y ponerlos a disposición de usuarios que capturan imágenes *in situ*. Los experimentos iniciales (capítulo 8 de la memoria) demuestran que el pipeline funciona sobre datos de laboratorio y sobre mezclas con imágenes de campo (PlantDoc), lo que constituye un paso hacia sistemas de apoyo a la sanidad vegetal más adaptados a la variabilidad real. La plataforma no sustituye el criterio de un técnico agrónomo, pero puede orientar la observación y priorizar revisiones, lo que encaja con una gestión más cuidadosa de los recursos fitosanitarios y, en última instancia, con la salud de los agroecosistemas.

ODS 12 — Producción y consumo responsables (relación alta)

El ODS 12 subraya el uso eficiente de recursos naturales y la reducción de residuos y de impactos ambientales a lo largo del ciclo de producción. En agricultura, un diagnóstico tardío o impreciso suele traducirse en tratamientos fitosanitarios **más amplios o más frecuentes** de lo necesario, con coste económico y ambiental.
Un clasificador de enfermedades foliares, cuando se utiliza de forma responsable, puede favorecer intervenciones **más localizadas y oportunas**. Foliarium incorpora además metadatos de fuente y validación manual, lo que permite distinguir imágenes revisadas y construir conjuntos de entrenamiento más fiables antes de desplegar un modelo en producción. En la memoria se advierte expresamente que un uso descuidado del diagnóstico automático —confiar ciegamente

en la predicción sin validación humana— podría tener el efecto contrario al deseado; por ello la plataforma incluye roles de etiquetador y flujos de validación, alineados con un consumo *responsable* de la tecnología.

ODS 2 — Hambre cero (relación media)

El ODS 2 persigue acabar con el hambre y asegurar la seguridad alimentaria. Las pérdidas por enfermedades de cultivo representan una parte significativa de la producción agrícola mundial. Herramientas que faciliten la detección temprana de síntomas en hojas pueden contribuir, a escala limitada en esta fase del proyecto, a **reducir esas pérdidas** y a mejorar la productividad de explotaciones que dispongan de conectividad y de acceso a la aplicación.
La relación se califica de **media** y no de alta porque el TFG no incluye un despliegue masivo en países en desarrollo ni un estudio de impacto cuantitativo sobre rendimientos agrícolas; el sistema está desplegado en un entorno universitario y las pruebas con usuarios reales son aún reducidas.

ODS 9 — Industria, innovación e infraestructuras (relación media)

Foliarium aporta **innovación tecnológica** en forma de plataforma reproducible: API REST, base de datos documental, clientes multiplataforma, contenedorización Docker y experimentos configurables con aprendizaje profundo. El trabajo se enmarca además parcialmente en el proyecto de investigación COSASS (coordinación de servicios inteligentes en entornos rurales con conectividad limitada), lo que refuerza el vínculo con infraestructuras digitales aplicadas al medio rural.
La relación es media porque el foco del TFG no es la fabricación industrial ni la política pública de I+D+i, sino la **demostración de un prototipo** desplegado y extensible.

ODS 4, ODS 13 y ODS 17 (relación baja)

El **ODS 4** puede vincularse de forma indirecta: el TFG documenta un sistema reproducible, desplegado y con guía de usuario, y se enmarca en colaboración universidad–agrónomos y en la Cátedra Planeta, lo que aporta valor formativo y de divulgación aplicada, aunque no constituya un programa docente ni una intervención educativa directa.
El **ODS 13** puede vincularse de forma indirecta: menos fitosanitarios mal aplicados implican menor huella asociada a la producción y transporte de esos productos, así como menor riesgo de emisiones por quema de restos afectados. No se ha cuantificado este efecto en el TFG.
El **ODS 17** aparece en la colaboración entre informáticos y agrónomos (GTIIA–Facultad de Ciencias Agrarias), la Cátedra Planeta de la UPV y el uso de conjuntos de datos públicos compartidos por la comunidad científica.

Objetivos sin relación relevante

Los ODS 1, 5–8, 10–11, 14 y 16 no se abordan de manera directa: el trabajo no trata la pobreza, la igualdad de género, el agua potable, la energía, el empleo, las ciudades, la vida marina ni la gobernanza institucional. Se han identificado explícitamente como *no procede* para evitar forzar conexiones artificiales.

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es

Limitaciones y uso ético

El impacto real sobre los ODS depende del **uso posterior** de Foliarium: acumulación de datos de campo validados, colaboración agronómica, despliegue en contextos donde la conectividad lo permita y interpretación prudente de las predicciones. En la fase actual, con datos reales propios limitados y sin validación experta sistemática, las contribuciones a los ODS deben entenderse como **potencial** y alineadas con la Cátedra Planeta (eje Planeta de la Agenda 2030), no como resultados ya medidos en producción agrícola a gran escala.

En conjunto, este TFG muestra cómo la ingeniería informática y la ciencia de datos puede servir a metas de desarrollo sostenible en el ámbito agrícola cuando se diseña una plataforma abierta, reproducible y consciente de las limitaciones del aprendizaje automático fuera del laboratorio.