

Visión por computador aplicada a la detección de enfermedades y plagas agrícolas: Fundamentos, aumento de datos y metodología

Nuvia Aracelly Beltrán Robayo

Darwin Roberto Pow Chon Long Vásquez

Mario Manuel Cárdenas Rodríguez

Jhonny Joffre Choez Burgos

Mario Alberto Ibarra Martínez

Luis Andrés Rodríguez Valle



**Visión por computador aplicada a la detección
de enfermedades y plagas agrícolas:
Fundamentos, aumento de datos y metodología**

Nuvia Aracelly Beltrán Robayo
Darwin Roberto Pow Chon Long Vásquez
Mario Manuel Cárdenas Rodríguez
Jhonny Joffre Choez Burgos
Mario Alberto Ibarra Martínez
Luis Andrés Rodríguez Valle

ISBN: 978-9942-53-217-6
Primera edición, 2026



© Nuvia Aracelly Beltrán Robayo

Instituto de Investigación, Milagro, Universidad Agraria del Ecuador
Computación, Milagro, Universidad Agraria del Ecuador
nbeltran@uagraria.edu.ec
<https://orcid.org/0000-0002-3335-576X>

Darwin Roberto Pow Chon Long Vásquez

Computación, Milagro, Universidad Agraria del Ecuador
dpow@uagraria.edu.ec
<https://orcid.org/0009-0000-7860-2106>

Mario Manuel Cárdenas Rodríguez

Computación, Milagro, Universidad Agraria del Ecuador
mcardenas@uagraria.edu.ec
<https://orcid.org/0000-0002-1824-8747>

Jhonny Joffre Choez Burgos

Computación, Guayaquil, Universidad Agraria del Ecuador
jchoez@uagraria.edu.ec
<https://orcid.org/0000-0003-3001-4561>

Mario Alberto Ibarra Martínez

Computación, Milagro, Universidad Agraria del Ecuador
miibarra@uagraria.edu.ec
<https://orcid.org/0009-0004-1721-5953>

Luis Andrés Rodríguez Valle

Computación, Milagro, Universidad Agraria del Ecuador
luis.rodriguez.valle@uagraria.edu.ec
<https://orcid.org/0009-0004-1721-5953>

© Editorial Grupo Compás, 2026

Guayaquil, Ecuador
www.grupocompas.com
<http://repositorio.grupocompas.com>

Esta obra ha sido sometida a un proceso de evaluación bajo el sistema de arbitraje doble ciego (double-blind peer review), garantizando el anonimato tanto de los autores como de los evaluadores externos. El dictamen favorable certifica que el contenido cumple con los más altos estándares de rigor científico, calidad editorial y originalidad exigidos por la comunidad académica internacional para su indexación y reconocimiento científico.

ISBN: 978-9942-53-217-6

DOI: 10.31876/9789942532176

Distribución online

Acceso abierto



Cita

Beltrán, N., Pow Chon Long, D., Cárdenas, M., Choez, J., Ibarra, M., Rodríguez, L. (2026). Visión por computador aplicada a la detección de enfermedades y plagas agrícolas: Fundamentos, aumento de datos y metodología. Editorial Grupo Compás.

Primera edición, 2026

Índice

CAPÍTULO 1: FUNDAMENTOS DE LA VISIÓN POR COMPUTADOR Y EL APRENDIZAJE AUTOMÁTICO EN LA IDENTIFICACIÓN DE PLAGAS Y ENFERMEDADES AGRÍCOLAS	4
Resumen	4
Abstract.....	5
Introducción.....	5
Desarrollo	6
Visión por computador.....	6
Procesamiento de imágenes digitales.....	8
Calidad de imagen	9
Extracción de características.....	9
Análisis de datos.....	10
Técnicas de identificación de enfermedades y plagas.....	10
Aprendizaje automático para la detección de enfermedades y plagas	11
Casos de estudios de implementación de modelos de visión por computador en la detección de plagas.....	13
Beneficios en la agricultura.....	19
Desafíos enfrentados	20
Conclusiones.....	22
 CAPÍTULO 2: TÉCNICAS DE AUMENTO DE DATOS PARA LA IDENTIFICACIÓN DE ENFERMEDADES Y PLAGAS AGRÍCOLAS MEDIANTE VISIÓN POR COMPUTADOR	 24
Resumen	24
Introducción.....	25
Desarrollo	26
El problema de los datos en la detección de enfermedades y plagas agrícolas.....	26
Fundamentos del aumento de datos.....	27
Categorías principales de técnicas de aumento de datos.....	28
Transformaciones geométricas.....	28

Transformaciones fotométricas.....	29
Métodos de mezcla de imágenes.....	30
Métodos basados en parches y oclusión.....	31
Herramientas y bibliotecas para el aumento de datos	31
Impacto del aumento de datos en arquitecturas de detección de enfermedades y plagas.....	32
Efecto sobre modelos YOLO	32
Efecto sobre CNN y modelos de clasificación	33
Casos de estudio: aumento de datos en la identificación de enfermedades y plagas.....	33
Beneficios del aumento de datos en la agricultura de precisión.....	35
Conclusiones.....	36
 CAPÍTULO 3: PASO A PASO PARA LA APLICACIÓN DE VISIÓN POR COMPUTADOR EN LA IDENTIFICACIÓN DE ENFERMEDADES Y PLAGAS AGRÍCOLAS	 37
 Resumen	 37
Abstract.....	38
Introducción.....	38
Desarrollo	39
Visión general del proceso	39
Definición del problema	40
Entregables del Paso 1	42
Errores frecuentes.....	42
Planificación y adquisición de imágenes	43
Consideraciones generales para la adquisición de fotos	43
Uso de datasets públicos existentes.....	44
Errores frecuentes.....	44
Construcción y gestión del conjunto de datos	45
Consideraciones generales.....	45
Etiquetado de imágenes	46
División del dataset	47
Errores frecuentes.....	48
Preprocesamiento y aumento de datos.....	48
Operaciones de preprocesamiento (train + val + test).....	49
Pipeline de aumento de datos (solo train).....	49

Errores Frecuentes	50
Selección y configuración de la arquitectura	50
Términos claves	51
Elección de modelos	53
Entrenamiento y ajuste del modelo.....	53
Configuración del modelo	54
Evaluación y validación del desempeño.....	54
Análisis de errores.....	56
Validación en condiciones reales.....	57
Despliegue e integración en sistemas agrícolas.....	57
Conclusiones.....	59
 CAPÍTULO 4: DE LA TEORÍA AL MODELO: APLICACIÓN PARA LA DETECCIÓN DE ENFERMEDADES Y PLAGAS CON PYTHON	 61
Introducción	62
Desarrollo	63
Definición del problema y alcance del sistema	64
Planificación y adquisición de imágenes	65
Construcción y gestión del conjunto de datos.....	69
Preprocesamiento y aumento de datos.....	71
Selección y configuración de la arquitectura	73
Entrenamiento y ajuste fino del modelo.....	75
SVM.....	76
CNN.....	77
Transformer.....	78
Evaluación y validación del desempeño.....	78
Despliegue e integración en sistemas agrícolas.....	80
Conclusiones.....	81
 Referencias bibliográficas	 82

CAPÍTULO 1: FUNDAMENTOS DE LA VISIÓN POR COMPUTADOR Y EL APRENDIZAJE AUTOMÁTICO EN LA IDENTIFICACIÓN DE PLAGAS Y ENFERMEDADES AGRÍCOLAS

Resumen

Este capítulo se presenta una revisión sobre el uso de la visión por computadora y el procesamiento digital de imágenes en la detección automatizada de plagas y enfermedades agrícolas, destacando su relevancia como herramientas clave para el fortalecimiento de la agricultura de precisión. Se describen los fundamentos de la visión artificial, el proceso de adquisición y mejora de imágenes, así como las técnicas de extracción de características y análisis de datos necesarios para identificar plagas de manera eficiente. Asimismo, se analizan los principales enfoques de aprendizaje automático y aprendizaje profundo, incluyendo modelos como CNN, YOLO y Vision Transformers, que han demostrado altos niveles de precisión en distintos cultivos y escenarios reales. Además, se expone beneficios significativos como la detección temprana, la optimización de recursos y la toma de decisiones basada en datos, junto con los desafíos asociados a la variabilidad ambiental y la calidad de los conjuntos de datos.

Palabras clave: Agrícola, Detección, Identificación, Plagas, Visión por computador

Abstract

This chapter presents a review on the use of computer vision and digital image processing in the automated detection of agricultural pests and diseases. It highlights their relevance as key tools for advancing precision agriculture. The fundamentals of machine vision are described, including the process of image acquisition and enhancement, as well as the feature extraction and data analysis techniques necessary for efficient pest identification. Furthermore, the main approaches in machine learning and deep learning are analyzed. This includes models such as Convolutional Neural Networks (CNNs), YOLO (You Only Look Once), and Vision Transformers, which have demonstrated high levels of accuracy across various crops and real-world scenarios. Significant benefits, such as early detection, resource optimization, and data-driven decision-making, are outlined. This is presented alongside the associated challenges, primarily related to environmental variability and dataset quality.

Keywords: Computer Vision, Detection, Identification, Agricultural Pests, Precision Agriculture

Introducción

La agricultura moderna enfrenta desafíos crecientes relacionados con el control de plagas, detección de enfermedades, la optimización de recursos y la necesidad de garantizar la seguridad alimentaria en un contexto de cambio climático y aumento de la demanda global de alimentos. En este escenario, la visión por computador y las técnicas de procesamiento digital de imágenes, apoyadas por algoritmos de aprendizaje automático y aprendizaje profundo, han surgido como herramientas clave para transformar los métodos tradicionales de monitoreo y manejo de cultivos. Estas tecnologías permiten analizar información visual de forma automatizada, precisa y en tiempo real, facilitando la detección temprana de plagas y enfermedades que afectan el rendimiento agrícola.

Por lo que en este capítulo aborda los fundamentos y aplicaciones de la visión por computadora en la detección de enfermedades y plagas agrícolas. En primer lugar, se presentan los conceptos básicos de la visión artificial y su relevancia en entornos agrícolas, destacando la importancia de la adquisición

de imágenes y la calidad de los datos visuales. Posteriormente, se analizan las principales técnicas de procesamiento digital de imágenes, incluyendo la mejora de la calidad, la extracción de características y el análisis de datos, que constituyen la base para una identificación eficaz de plagas y enfermedades. Asimismo, se examina el papel del aprendizaje automático y del aprendizaje profundo en la automatización de estos procesos, haciendo énfasis en modelos ampliamente utilizados como las redes neuronales convolucionales y los enfoques basados en aprendizaje por transferencia.

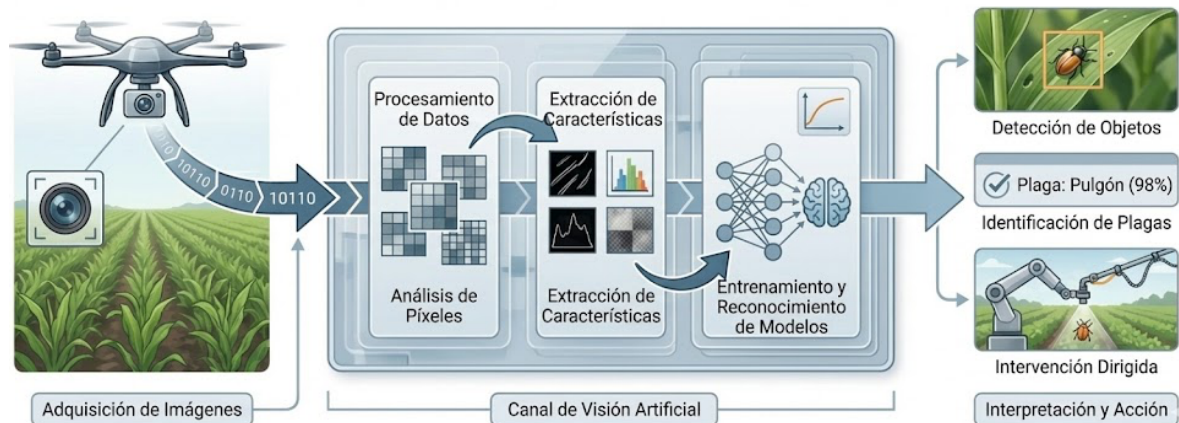
Finalmente, el capítulo presenta una revisión de casos de estudio recientes que evidencian el impacto y la efectividad de los modelos de visión por computador aplicados a la detección de plagas en distintos cultivos, así como los beneficios y desafíos asociados a su implementación en escenarios reales. De este modo, se ofrece al lector una visión clara y actualizada del estado del arte, sentando las bases para futuras investigaciones y desarrollos tecnológicos orientados a una agricultura más eficiente, sostenible y basada en datos.

Desarrollo

Visión por computador

La visión artificial es un campo multidisciplinario que permite a las computadoras interpretar y comprender la información visual del mundo, de forma similar a como los humanos procesamos imágenes (Ding et al., 2024). Esta capacidad aprovecha varios conceptos clave que sirven de base para comprender el funcionamiento de la visión artificial, especialmente en contextos agrícolas (Aldakheel et al., 2024).

Figura 1. Fases del proceso de identificación de plagas mediante Visión por computador



En la agricultura es utilizado para:

Figura 2. Uso de la visión por computador en cultivos agrícolas

Monitoreo de cultivos	Detección de malezas	Predicción de rendimiento
Análisis de imágenes de drones para evaluar la salud de los cultivos.	Identificación y clasificación de diferentes especies de plantas para optimizar el uso de herbicidas.	Uso de datos de imágenes para estimar el rendimiento de los cultivos en función de la salud y la densidad de las plantas.

Para ello la adquisición de imágenes es el primer paso en la visión artificial, esto implica la captura de imágenes mediante diversos sensores, como cámaras, y la conversión de estas señales luminosas a formato digital, los aspectos más importantes en este paso es la calidad y la claridad de las imágenes puesto que son fundamentales para garantizar un análisis eficaz (Upadhyay et al., 2025). Factores como las condiciones de iluminación, la

resolución del sensor y la posición de la cámara deben considerarse para mejorar la calidad de la imagen (Song, Yan, Liu, Gao, Han, Jiang, Jin, Zhu, et al., 2025). En agricultura, la formación precisa de imágenes es esencial para obtener imágenes nítidas de los cultivos y las posibles plagas(X. Wang & Liu, 2024).

Una vez capturadas las imágenes, el siguiente paso es la extracción de características, en este proceso implica identificar y aislar características o patrones específicos dentro de las imágenes que sean relevantes para la tarea en cuestión. Para la identificación de plagas, las características pueden incluir variaciones de color, patrones de textura y formas asociadas con diferentes plagas o sus daños (Han et al., 2024). Las técnicas de extracción de características pueden abarcar desde métodos sencillos, como la detección de bordes y los histogramas de color, hasta algoritmos más complejos que utilizan modelos de aprendizaje automático para una mayor precisión (Montoya et al., 2023). Resaltar estas características permite una clasificación y detección eficaz de plagas a partir de las imágenes, lo cual es crucial para una intervención oportuna en el manejo de cultivos.

Ante los crecientes desafíos del control de plagas y la monitorización de cultivos, la visión artificial presenta una solución viable para mejorar la eficiencia y la productividad (Leal-Lara et al., 2021), ya que, al automatizar la detección de plagas y la monitorización del estado de los cultivos, los profesionales del área de la agricultura pueden tomar decisiones basadas en datos que minimizan las pérdidas y optimizan el uso de recursos. Además, la integración de la visión artificial con otras tecnologías, como drones y dispositivos IoT, facilita la monitorización en tiempo real de los campos agrícolas, garantizando estrategias proactivas de gestión de plagas.

Procesamiento de imágenes digitales

El procesamiento digital de imágenes desempeña un papel fundamental en la extracción de información significativa de imágenes para la detección de plagas en entornos agrícolas. Mediante diversas técnicas, se pueden mejorar la calidad de las imágenes y refinar los datos utilizados para la identificación de plagas, es por ello por lo que se abordarán en esta sección algunas de las técnicas clave empleadas en el procesamiento digital de imágenes.

Calidad de imagen

Mejorar la calidad de la imagen es esencial para una detección eficaz de plagas. Las técnicas clave incluyen:

- **Reducción de ruido** : se utilizan filtros para minimizar las perturbaciones no deseadas en la imagen (Bhujade et al., 2024).
- **Mejora del contraste** : ajuste del brillo para resaltar las características (Haider et al., 2025).
- **Normalización** : estandarizar los datos de la imagen para reducir las variaciones (Jia, Rezaei, Asli, et al., 2025).
- **Filtración**: El filtrado es un proceso que se utiliza para eliminar ruido no deseado o realzar características específicas de una imagen. En el contexto de la detección de plagas, la aplicación de filtros puede ser especialmente beneficiosa. Por ejemplo, los filtros espaciales, como los gaussianos y los de mediana, pueden suavizar el ruido a la vez que conservan los bordes y detalles importantes de la imagen(Bhujade et al., 2024). Además, los filtros de dominio frecuencial pueden ayudar a resaltar ciertas características que podrían indicar la presencia de plagas o daños en los cultivos. La selección del filtro adecuado es importante, ya que influye directamente en la precisión de los pasos de procesamiento posteriores.

Extracción de características

Identificar características específicas en las imágenes facilita la detección precisa de plagas. Los métodos esenciales incluyen:

- **Detección de bordes** : detecta límites dentro de las imágenes para resaltar los contornos de las plagas (Wei et al., 2024).
- **Segmentación** : dividir las imágenes en segmentos para distinguir las plagas del fondo.

La segmentación es la técnica de dividir una imagen en regiones o segmentos significativos según características como el color, la intensidad o la textura

(Haider et al., 2025). Técnicas como el umbral, el crecimiento de regiones y la agrupación (p. ej., la agrupación K-means) permiten distinguir eficazmente las plagas de las plantas sanas. Por ejemplo, la segmentación de imágenes de cultivos puede ayudar a aislar las zonas dañadas, lo que facilita la identificación y evaluación de la gravedad de las infestaciones de plagas. Una segmentación eficaz permite un análisis de datos más preciso y una toma de decisiones más precisa para el control de plagas.

- **Análisis de textura** : evaluación de las características de la superficie de las plantas para detectar señales de plagas (Sahu et al., 2025).

Análisis de datos

Una vez procesadas las imágenes, el análisis de datos es clave:

- **Reconocimiento de patrones**: uso de algoritmos para identificar tipos de plagas basándose en datos visuales.
- **Aprendizaje automático**: implementación de IA para mejorar la precisión de detección a lo largo del tiempo.
- **Métodos estadísticos**: utilización de datos para evaluar las tendencias y los impactos de las poblaciones de plagas.

Técnicas de identificación de enfermedades y plagas

El primer paso en el procesamiento de imágenes es la adquisición de imágenes, que consiste en capturar datos visuales del entorno agrícola. Se pueden obtener imágenes de alta calidad utilizando diversos dispositivos, como cámaras digitales, drones y sensores diseñados específicamente para entornos agrícolas, por ende, la elección del equipo influye significativamente en la calidad y el detalle de las imágenes capturadas. A su vez, factores como la resolución de la cámara, las condiciones de iluminación y el ángulo de captura deben considerarse cuidadosamente para garantizar que las imágenes proporcionen suficiente detalle para su posterior análisis, en definitiva, la adquisición adecuada de imágenes constituye la base de una identificación eficaz de plagas.

Una vez adquiridas las imágenes, se emplean técnicas de preprocesamiento para preparar los datos para el análisis. Los métodos comunes de preprocesamiento incluyen la reducción de ruido, la mejora del contraste y la normalización. Estas técnicas son vitales para mejorar la calidad de la imagen y garantizar que las características relevantes sean visibles y distinguibles. Por ejemplo, los algoritmos de reducción de ruido ayudan a eliminar variaciones no deseadas en los valores de los píxeles, que pueden ocultar detalles importantes para la identificación de plagas. Además, las técnicas de mejora del contraste pueden hacer que las características de interés destaquen más sobre el fondo, facilitando el reconocimiento de las plagas en las imágenes.

Tras el preprocesamiento, los métodos de mejora de imagen refinan aún más los datos visuales. Técnicas como la ecualización de histograma, el filtrado y la detección de bordes desempeñan un papel fundamental para resaltar las características significativas. La ecualización de histograma ajusta la distribución de la intensidad de los píxeles para mejorar el contraste en toda la imagen, haciendo más evidentes las variaciones sutiles. Las técnicas de filtrado, como los detectores de bordes Sobel o Canny, se centran en identificar cambios bruscos de intensidad que suelen corresponder a los límites de los objetos. Estas mejoras son especialmente beneficiosas para la identificación de plagas, ya que ayudan a extraer información crítica de fondos complejos, lo que permite un análisis y una clasificación más precisos de las plagas.

Aprendizaje automático para la detección de enfermedades y plagas

En los últimos años, el aprendizaje automático se ha convertido en una herramienta poderosa para mejorar la detección de plagas en la agricultura. La capacidad de enseñar a las computadoras a identificar y clasificar plagas mediante algoritmos revoluciona la forma en que los ingenieros agrícolas abordan el manejo de plagas y la protección de cultivos.

Los algoritmos de aprendizaje automático se pueden clasificar, a grandes rasgos, en técnicas de aprendizaje supervisado y no supervisado. El aprendizaje supervisado implica entrenar un modelo con datos etiquetados, donde el algoritmo aprende a asignar características de entrada a resultados

objetivo específicos, como la identificación de tipos de plagas. Algunos ejemplos de algoritmos de aprendizaje supervisado populares son:

Máquinas de vectores de soporte (SVM) : las SVM son particularmente efectivas para tareas de clasificación y pueden crear hiperplanos que separan diferentes clases de plagas en función de sus características.

Random Forest : este método de aprendizaje conjunto combina múltiples árboles de decisión para mejorar la precisión de la clasificación, lo que lo hace robusto contra el sobreajuste y mejora la generalización en datos no vistos.

Redes neuronales convolucionales (CNN) : especialmente adecuadas para datos de imágenes, las CNN aprovechan múltiples capas para extraer automáticamente características y clasificar imágenes, lo que permite una alta precisión en la identificación de plagas incluso en entornos complejos.

La extracción de características desempeña un papel fundamental en los modelos de aprendizaje automático para la detección de plagas. Este proceso implica la identificación y cuantificación de patrones o características significativas a partir de imágenes que representan la presencia de plagas. Los métodos tradicionales de extracción de características suelen basarse en la selección manual o la ingeniería de características según la experiencia del dominio. Estas pueden incluir histogramas de color, características de textura y descriptores de forma. Sin embargo, las aplicaciones modernas utilizan con frecuencia técnicas automatizadas de extracción de características, especialmente con enfoques de aprendizaje profundo como las CNN, donde el modelo aprende características relevantes directamente de los datos de entrenamiento sin ingeniería de características explícita.

Una vez extraídas las características relevantes, se emplean técnicas de clasificación para determinar la presencia de plagas y su categoría. Entre los métodos disponibles, k-Vecinos Más Cercanos (k-NN) es un algoritmo de clasificación simple pero eficaz, especialmente útil en escenarios con agrupaciones bien definidas de tipos de plagas según sus características. Los árboles de decisión ofrecen otro método donde los datos se dividen según los valores de las características para crear un modelo que predice los tipos de plagas según las propiedades de las imágenes de entrada.

Más allá de los algoritmos tradicionales, los avances en aprendizaje profundo han dado lugar a técnicas de aprendizaje por transferencia, donde los modelos preentrenados con grandes conjuntos de datos se optimizan para tareas de detección de plagas. Este enfoque reduce significativamente la cantidad de datos de entrenamiento necesarios para una clasificación eficaz de enfermedades y plagas, a la vez que mejora la precisión y el rendimiento.

Mediante la implementación de diversos algoritmos de aprendizaje automático y técnicas robustas de extracción de características, los ingenieros agrícolas pueden mejorar significativamente la capacidad de detección de enfermedades y plagas. Esto no solo se traduce en estrategias de control de plagas más eficaces, sino que también mejora la productividad agrícola general al garantizar intervenciones oportunas y específicas.

Casos de estudios de implementación de modelos de visión por computador en la detección de plagas

A continuación, se presenta revisión de la literatura del 2023 al 2025 donde se muestra los modelos utilizados de visión por computador, la plaga detectada, la métrica de evaluación utilizada para medir la efectividad en la precisión y el porcentaje calculado.

Tabla 1. *Uso de Visión por computador en diferentes cultivos: Casos de estudio*

Nombre del Artículo	Año	Autores	Cultivo	Plaga Detectada	Dataset Utilizado	Modelos de Visión	Métrica	Valor
A New Hybrid ConvViT Model for Dangerous Farm Insect	2025	(Utku et al., 2025a)	General	15 tipos (Áfidos, gusanos, etc.)	1591 imágenes de 15 especies diferentes	ConvViT	Exactitud	93.61%

Nombre del Artículo	Año	Autores	Cultivo	Plaga Detectada	Dataset Utilizado	Modelos de Visión	Métrica	Valor
Detección								
YOLO-LCE: A Lightweight YOLOv8 Model for Agricultural Pest Detection	2025	(Cen et al., 2025)	Algodón / Varios	24 categorías multiobjetivo	Pest24	YOLO-LCE (YOLOv8 imp.)	mAP50	63.9%
YOLO-Pepper: Enhanced model for greenhouse pepper disease and pest detection	2025	(X. Wang et al., 2025)	Pimiento	Antracnosis, ácaros, trips	Custom (8,046 imgs)	YOLO-Pepper	mAP50	94.26%
Lightweight Vision-Transformer Network for Early Insect Pest Identification	2026	(Hong et al., 2026)	Tomate, pepino, fresa y pimiento	Mosca blanca, trips, ácaros	Public In-situ +	Light-HortiNet	mAP50	87.2%

Nombre del Artículo	Año	Autores	Cultivo	Plaga Detectada	Dataset Utilizado	Modelos de Visión	Métrica	Valor
Advancing Food Security through Precision Agriculture: YOLOv8's Role	2025	(Khan et al., 2025)	Varios	Plagas generales IP102	IP102	YOLOv8	mAP50	66.9%
Deep learning based agricultural pest monitoring and classification	2025	(Venkateswara & Padmanabhan, 2025a)	Granos	Plagas de campo diversas	IP102 dataset	CNN	Exactitud	84.95%
MATeacher: Multi-scale Views Learning and Adaptive Unsupervised Weighting	2024	(Li et al., 2024)	Maíz	Barrenador del maíz	Corn borer dataset	MATeacher	mAP	68.2
Efficiency and Scalability of Efficient	2024	(Sharma, 2024)	Varios	12 clases (Hormiga)	Large Pest Dataset	EfficientNetV2-L	Exactitud	88.0%

Nombre del Artículo	Año	Autores	Cultivo	Plaga Detectada	Dataset Utilizado	Modelos de Visión	Métrica	Valor
NetV2-L for Large-Scale Pest				as, moths)				
MA-YOLO: Agricultural pest detection model based on multi-scale fusion	2025	(Zou et al., 2025)	Granos de otoño	Plagas diversas	IP102 Pest24	/ MA-YOLO	Exactitud	73.9%
Yolo-pest: an optimized YoloV8x for detection of small insect pests using smart traps	2025	(Hakim et al., 2025)	Mango Maiz	Mango (moscas de la fruta) y un campo de maíz (gusano cogollero)	Dataset propio	Optimized YOLOv8	Exactitud	94.0%
Efficient Pest Detection Through Advanced Machine	2025	(Subburaj et al., 2025)	Varios	Plagas generales	Public Dataset	Efficient NetB7	Exactitud	93.5%

Nombre del Artículo	Año	Autores	Cultivo	Plaga Detectada	Dataset Utilizado	Modelos de Visión	Métrica	Valor
Learning (Efficient NetB7)								
YOLO-YSTs: Lightweight detection method for yellow sticky traps	2025	(Huang et al., 2025)	Invernadero	Plagas en trampas amarillas	Yellow Sticky Trap	YOLO-YSTs	mAP50	86.8%
Experimental Comparison of YOLOv8, YOLOv9 and YOLOv10 for Caterpillar	2025	(Kodors et al., 2025)	Huertos	Orugas (Caterpillars)	Caterpillar 640	YOLOv8m	mAP50	88,7%
AI-Based Image Profiling and Detection for the Beetle Byte Quintet Using Vision Transfor	2025	(Serfa Juan et al., 2025)	Almacenes	Escarabajos	Beetle Byte Quintet	Vision Transfor mer	Exactitud	99.34%

Nombre del Artículo	Año	Autores	Cultivo	Plaga Detectada	Dataset Utilizado	Modelos de Visión	Métrica	Valor
mer (ViT) in Advanced Stored Product Infestation Monitoring								
MoSViT: Classification of maize leaf diseases and pests	2025	(Chen et al., 2025)	Maíz	Roya, manchas foliares	Maize Disease Set	MoSViT	Exactitud	98.75%

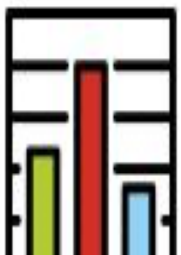
La revisión de la literatura analizada muestra un progreso notable en la identificación automatizada de plagas en cultivos, utilizando principalmente visión artificial y algoritmos de aprendizaje profundo, con especial énfasis en investigaciones desarrolladas entre 2024 y 2026. Los enfoques más utilizados se basan en arquitecturas YOLO (versiones 8, 9 y modelos optimizados) y Transformers de visión (ViT), además de propuestas híbridas como ConvViT. Estas metodologías han logrado altos niveles de eficacia en tareas de clasificación y localización de diversas plagas y enfermedades en cultivos como maíz, mango, pimiento, tomate y cereales.

La evaluación del desempeño se ha sustentado en bases de datos públicas (IP102, Pest24) y conjuntos de datos específicos, demostrando una capacidad de generalización aplicable a condiciones reales. Los resultados reportados son consistentes, con precisiones que frecuentemente superan el 90% y valores de mAP50 por encima del 85% en múltiples estudios. Paralelamente, se identifica un enfoque creciente hacia el desarrollo de modelos ligeros y eficientes, diseñados para su implementación práctica en escenarios como

invernaderos, trampas inteligentes y dispositivos de monitorización en campo.

Beneficios en la agricultura

La integración de las tecnologías de visión artificial en la agricultura ofrece un potencial transformador, pero también conlleva desafíos únicos. Comprender tanto los beneficios como los obstáculos es crucial para los ingenieros agrícolas e investigadores en visión artificial que desean implementar soluciones eficaces en el campo.

Análisis de datos	Eficiencia	Agricultura de Precisión
		
Las tecnologías de visión artificial permiten un análisis detallado de datos agrícolas, lo que conduce a una mejor toma de decisiones, gestión de cultivos y predicciones de rendimiento.	La automatización de tareas mediante visión artificial reduce los costos laborales y aumenta la eficiencia operativa, lo que permite a los agricultores concentrarse en actividades más estratégicas.	La implementación de estas tecnologías permite un monitoreo preciso de la salud de los cultivos y la utilización de los recursos, asegurando condiciones óptimas de crecimiento y minimizando el desperdicio.

La visión artificial ofrece varias ventajas que pueden mejorar significativamente las prácticas agrícolas. Uno de los principales beneficios es una mayor eficiencia en la detección de enfermedades, plagas y el monitoreo de cultivos. El análisis visual automatizado puede identificar rápidamente plagas y enfermedades de las plantas, lo que permite a los agricultores intervenir con mayor rapidez y minimizar los posibles daños. Esta respuesta oportuna puede resultar en un mejor rendimiento de los cultivos y una reducción del uso de pesticidas.

Otro beneficio clave es la agricultura de precisión, que utiliza la visión artificial para analizar las condiciones del campo y gestionar los cultivos de forma más eficaz. Al procesar imágenes de alta resolución, los agricultores pueden monitorear las variaciones en la salud y el crecimiento de las plantas, lo que permite intervenciones específicas según sus necesidades. Esta práctica no solo promueve prácticas agrícolas sostenibles, sino que también optimiza el uso de recursos, reduciendo el desperdicio y los costos.

Además, la visión artificial puede optimizar la toma de decisiones basada en datos, debido a que, con el monitoreo y la recopilación de datos continuos, los agricultores pueden obtener información valiosa sobre las condiciones de los cultivos y las poblaciones de plagas. Esta información respalda estrategias informadas para el manejo de enfermedades y plagas, a su vez que fomenta un enfoque agrícola más adaptable que se adapta al panorama agrícola en constante evolución.

Desafíos enfrentados

Si bien los beneficios de la visión artificial son convincentes, varios desafíos impiden su implementación generalizada. La variabilidad ambiental representa un obstáculo importante, puesto que los campos agrícolas están sujetos a diferentes condiciones ambientales, como variaciones de iluminación, cambios climáticos y la diversidad de especies de cultivos; en consecuencia, estos factores pueden afectar la calidad de los datos de imagen y generar inconsistencias en los resultados de detección de enfermedades y plagas, lo que requiere soluciones robustas para adaptarse a estas variaciones.

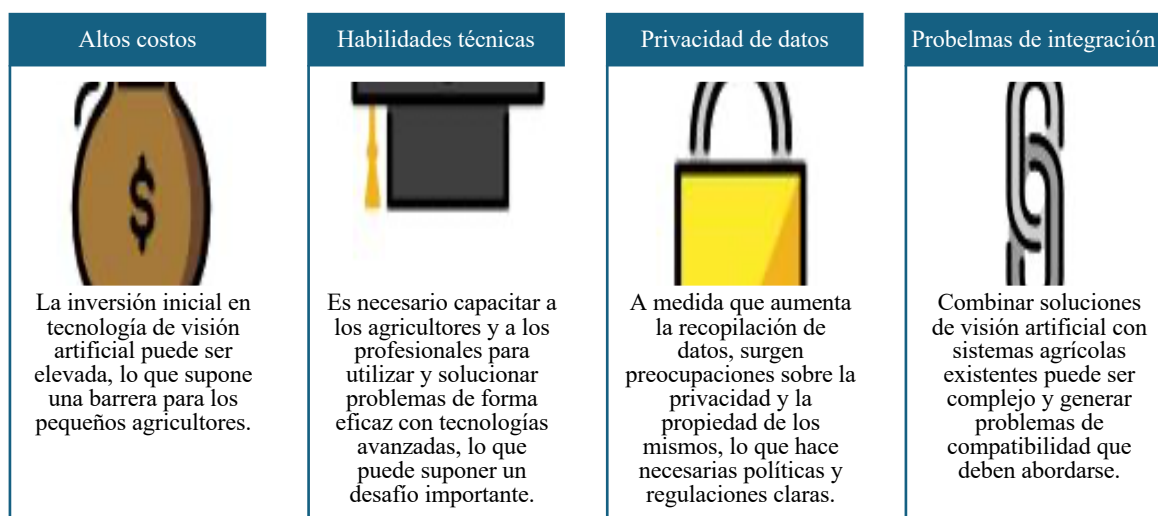
Otro desafío es la calidad de los datos, es decir los conjuntos de datos anotados de alta calidad son esenciales para entrenar modelos de aprendizaje automático. Por lo que es importante recopilar cantidades adecuadas de datos de entrenamiento diversos puede requerir muchos recursos y tiempo y esfuerzo considerables. Además, garantizar que los datos capturen las complejidades de los escenarios agrícolas reales es fundamental para construir sistemas confiables. La insuficiencia de datos puede generar problemas como el sobreajuste o una generalización deficiente en los modelos de identificación de enfermedades y plagas.

Para afrontar estos desafíos, es necesaria la investigación continua y la colaboración entre expertos agrícolas y tecnológicos. Al comprender tanto las ventajas como los obstáculos asociados con la visión artificial, las partes interesadas pueden trabajar en el desarrollo de estrategias y soluciones eficaces que mejoren la productividad agrícola y, al mismo tiempo, aborden los desafíos que se presentan en el campo.

En la agricultura, existe una amplia gama de especies de enfermedades y plagas que varían significativamente en tamaño, color y comportamiento. Muchas enfermedades y plagas pueden parecerse mucho a insectos beneficiosos u otras especies no objetivo, lo que complica el proceso de identificación. Los modelos de aprendizaje automático pueden tener dificultades para generalizar entre diversas clases, especialmente si se han entrenado con un conjunto de datos limitado que no representa adecuadamente el espectro completo de posibles enfermedades y plagas. Recopilar un conjunto de datos completo y bien anotado que abarque diversas especies y sus etapas de vida es esencial para desarrollar un sistema de identificación fiable.

Por otra parte, también presentan desafíos debido a:

Figura 3. Desafíos de implementación de visión artificial



Conclusiones

La visión por computador, junto con técnicas de aprendizaje automático y profundo, se ha convertido en una herramienta sumamente eficaz para la detección automatizada de enfermedades y plagas agrícolas, ya que permite identificar, clasificar y localizar estas amenazas con un alto grado de precisión en diversos cultivos.

En este contexto, los modelos basados en arquitecturas YOLO y Vision Transformers (ViT) se destacan como las soluciones más empleadas y efectivas, dado que logran métricas que superan el 90% en precisión y presentan valores elevados de mAP50 en numerosos estudios. Estos resultados no solo evidencian su capacidad de generalización, sino también su aplicabilidad práctica en escenarios agrícolas reales, tales como invernaderos, trampas inteligentes y sistemas de monitoreo en campo.

Sin embargo, a pesar de los avances significativos alcanzados, todavía persisten desafíos importantes relacionados con la variabilidad ambiental, la diversidad de enfermedades y plagas y la calidad de los conjuntos de datos,

lo cual limita la robustez y escalabilidad de los modelos. De hecho, la carencia de conjuntos de datos amplios, equilibrados y representativos sigue representando un obstáculo clave para la adopción masiva de estas tecnologías.

CAPÍTULO 2: TÉCNICAS DE AUMENTO DE DATOS PARA LA IDENTIFICACIÓN DE ENFERMEDADES Y PLAGAS AGRÍCOLAS MEDIANTE VISIÓN POR COMPUTADOR

Resumen

Este capítulo presenta una revisión sistemática sobre el uso de técnicas de aumento de datos (data augmentation) como estrategia clave para mejorar el rendimiento de los modelos de visión por computador en la identificación automática de enfermedades y plagas agrícolas. Se describen los fundamentos del aumento de datos, su justificación frente a la escasez y desbalance de conjuntos de datos agrícolas, así como las principales técnicas empleadas: transformaciones geométricas, fotométricas, mezcla de imágenes (Mixup, CutMix, Mosaic) y métodos basados en parches. Asimismo, se analizan los efectos de estas técnicas sobre arquitecturas ampliamente utilizadas como YOLO y redes neuronales convolucionales (CNN). La revisión de literatura (2024-2026) evidencia que la combinación adecuada de técnicas de aumento puede incrementar la precisión de detección reducir el sobreajuste y mejorar la generalización en condiciones reales de campo.

Palabras clave: Aumento de datos, Detección de enfermedades y plagas, Visión por computador, Aprendizaje profundo, Agricultura de precisión

Abstract

This chapter presents a systematic review on the use of data augmentation techniques as a key strategy to enhance the performance of computer vision models in the automatic identification of agricultural diseases and pests. It describes the fundamentals of data augmentation and its justification when addressing the scarcity and imbalance of agricultural datasets, as well as the main techniques employed: geometric and photometric transformations, image mixing (Mixup, CutMix, Mosaic), and patch-based methods. Furthermore, the effects of these techniques on widely used architectures, such as YOLO and convolutional neural networks (CNNs), are analyzed. The literature review (2024–2026) demonstrates that an appropriate combination of augmentation techniques can increase detection accuracy, reduce overfitting, and improve generalization under real field conditions.

Keywords: Data Augmentation, Detection of diseases and pests, Computer Vision, Deep Learning, Precision Agriculture

Introducción

La identificación automática de enfermedades y plagas agrícolas mediante modelos de visión por computador representa uno de los avances más prometedores para la agricultura de precisión. Sin embargo, el desarrollo de estos modelos enfrenta un obstáculo estructural que limita su desempeño: la escasez, el desbalance y la poca diversidad de los conjuntos de datos de entrenamiento. En entornos agrícolas reales, las enfermedades y plagas presentan variaciones significativas en tamaño, coloración, postura y contexto de fondo, variaciones que difícilmente quedan representadas en datasets de tamaño moderado. Este desequilibrio entre la diversidad del mundo real y la información disponible para el entrenamiento genera modelos propensos al sobreajuste y con limitada capacidad de generalización (Naseer et al., 2023).

El aumento de datos (data augmentation) se presenta como una solución de bajo costo computacional para abordar esta problemática. Mediante la aplicación de transformaciones controladas sobre las imágenes existentes o la generación de imágenes sintéticas completamente nuevas (Mumuni & Mumuni, 2022), es posible ampliar artificialmente el tamaño y la diversidad

del conjunto de entrenamiento sin necesidad de recopilar y etiquetar datos adicionales. Esta estrategia ha demostrado ser especialmente eficaz en tareas de clasificación y detección de objetos dentro del dominio agrícola, donde la recolección de datos es costosa y está sujeta a estacionalidad (Lu et al., 2022).

En este capítulo se presentan los fundamentos teóricos y prácticos del aumento de datos aplicado a la identificación de enfermedades y plagas agrícolas. En primer lugar, se contextualiza el problema de los datos en el dominio agrícola y se justifica la necesidad del aumento. Posteriormente, se describen en detalle las principales categorías de técnicas: transformaciones geométricas y fotométricas, métodos de mezcla de imágenes, enfoques basados en parches y generación sintética mediante modelos generativos. Asimismo, se analizan las bibliotecas y herramientas más utilizadas en la comunidad científica para implementar estas técnicas, y se revisan casos de estudio recientes que cuantifican el impacto del aumento de datos sobre el rendimiento de los modelos. Finalmente, se discuten los desafíos asociados y se ofrecen recomendaciones prácticas para la implementación en contextos agrícolas latinoamericanos.

Desarrollo

El problema de los datos en la detección de enfermedades y plagas agrícolas

El aprendizaje profundo requiere conjuntos de datos amplios, variados y correctamente balanceados para alcanzar su máximo potencial. En el dominio agrícola, esta condición raramente se cumple de forma natural. La captura de imágenes de enfermedades y plagas está sujeta a factores estacionales, geográficos y logísticos que dificultan la obtención de muestras suficientes de todas las especies de interés. Además, muchas enfermedades y plagas de relevancia económica son morfológicamente similares entre sí o con insectos beneficiosos, lo que exige una mayor riqueza de ejemplos de entrenamiento para que el modelo aprenda fronteras de decisión precisas (Jia, Rezaei, Honarvar, et al., 2025)

Los datasets públicos más utilizados en la comunidad como IP102(2019) y Pest24 (Q. J. Wang et al., 2020) contienen miles de imágenes distribuidas en decenas de categorías. Sin embargo, presentan distribuciones muy

desiguales, debido a que algunas categorías cuentan con cientos de ejemplos mientras que otras apenas superan la decena. Este desbalance extremo induce sesgos sistemáticos en los modelos entrenados, que tienden a clasificar las instancias de clases minoritarias como pertenecientes a las clases mayoritarias. El aumento de datos orientado a las clases con menor representación es, por tanto, una estrategia fundamental para mitigar este problema (Venkateswara & Padmanabhan, 2025b).

Paralelamente, la variabilidad del entorno de captura introduce otra fuente de dificultad, ya que las imágenes de enfermedades y plagas tomadas en campo abierto difieren sustancialmente de las obtenidas en condiciones controladas de laboratorio: varían en iluminación, fondo, resolución y ángulo de captura. Por ende, los modelos entrenados exclusivamente con imágenes de laboratorio suelen mostrar una caída considerable en su rendimiento cuando se despliegan en condiciones reales, por lo que las técnicas de aumento de datos que simulan estas variaciones ambientales permiten construir modelos más robustos y transferibles a escenarios productivos (Song et al., 2025).

Fundamentos del aumento de datos

El aumento de datos es una técnica de regularización que consiste en aplicar transformaciones a las imágenes del conjunto de entrenamiento para generar nuevas instancias que, aunque derivadas de las originales, presentan suficiente variabilidad para ser tratadas como ejemplos independientes por el modelo. Estas transformaciones deben ser semánticamente coherentes, es decir, deben preservar la clase y las características esenciales del objeto de interés —la plaga— aunque modifiquen su apariencia superficial (Upadhyay et al., 2025).

El fundamento teórico del aumento de datos se basa en el principio de invariancia, según el cual un modelo debe ser capaz de reconocer un mismo objeto aun cuando este aparezca bajo diferentes condiciones de observación. En visión por computador, las transformaciones aplicadas durante el data augmentation permiten exponer al modelo a variaciones de orientación, escala, posición e iluminación, favoreciendo el aprendizaje de representaciones más robustas y generalizables (Shorten y Khoshgoftaar,

2019; J. Wang y Perez, 2017) En aplicaciones agrícolas, este principio resulta especialmente relevante, ya que una plaga o enfermedad puede presentarse en diferentes condiciones ambientales y ángulos de captura sin perder su identidad biológica.

Desde el punto de vista práctico, el aumento de datos puede aplicarse de forma offline generando y almacenando las imágenes aumentadas antes del entrenamiento— o de forma online aplicando las transformaciones aleatoriamente en cada época de entrenamiento (Athalye & Arnaout, 2023; De Toro-Castro et al., 2026). El enfoque online es el más utilizado en la actualidad, ya que garantiza que el modelo vea una versión diferente de cada imagen en cada paso de entrenamiento, maximizando la diversidad efectiva del conjunto sin requerir almacenamiento adicional significativo (Utku et al., 2025).

Categorías principales de técnicas de aumento de datos

Transformaciones geométricas

Las transformaciones geométricas modifican la posición, orientación o escala de los objetos en la imagen sin alterar sus propiedades de color o textura. Son las técnicas de aumento más ampliamente utilizadas en visión por computador y han demostrado ser especialmente útiles para la detección de enfermedades y plagas, dado que los insectos pueden aparecer en cualquier orientación sobre la planta (Utku et al., 2025b).

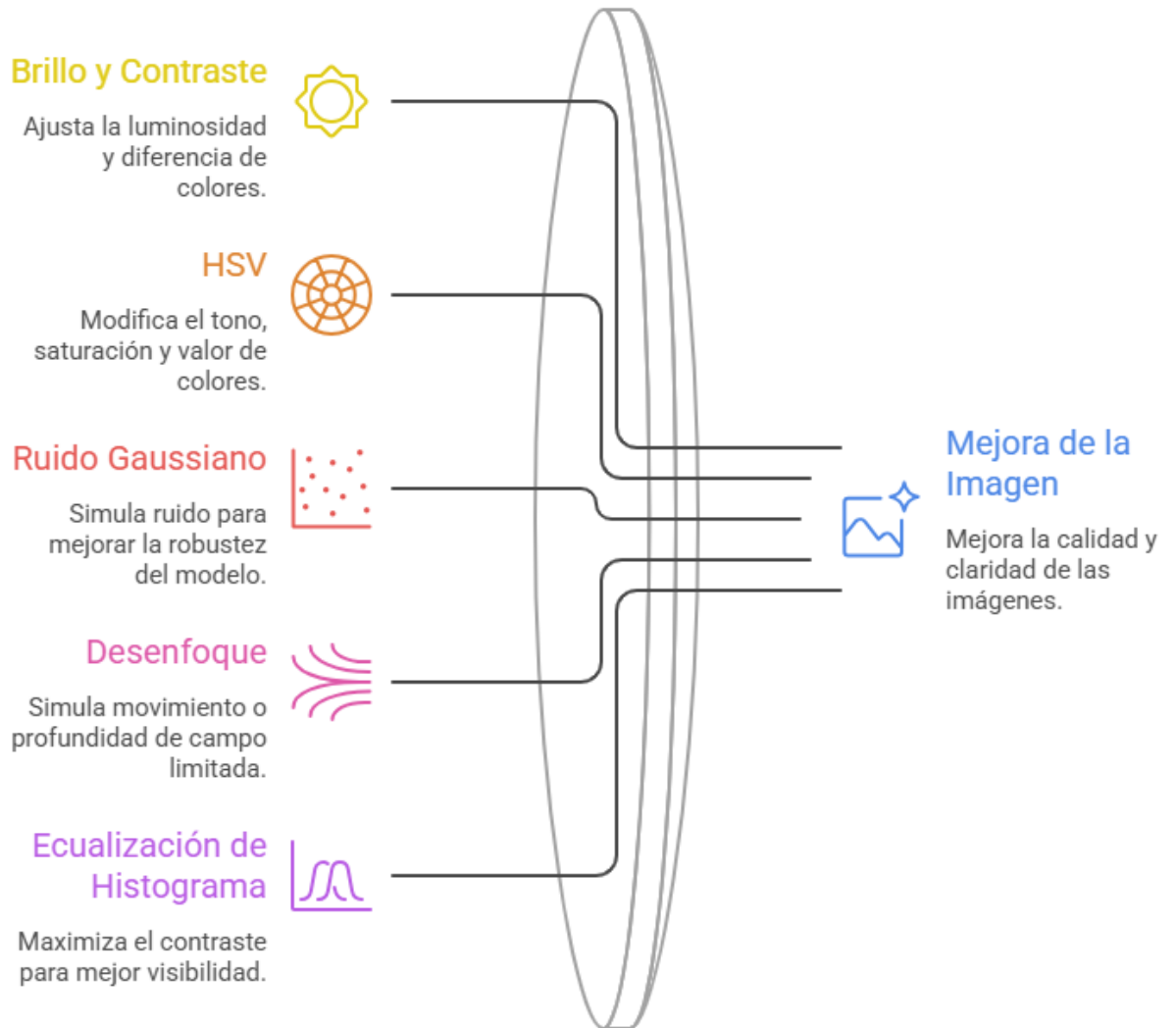
Figura 4. Técnicas de aumento de datos mediante transformaciones geométricas

Característica	 Descripción	 Utilidad	 Implementación	 Beneficio
Horizontal y Vertical	Invierte la imagen a lo largo de los ejes.	Plagas con simetría bilateral.	No requiere parámetros adicionales.	Duplica el tamaño efectivo del dataset.
Volteo/Rotación	Gira la imagen un ángulo arbitrario.	Posiciones no estándar de plagas.	Ajustar o eliminar coordenadas de bounding boxes.	Útil para plagas en posiciones no estándar.
Recorte Aleatorio	Selecciona y extrae sub-regiones.	Representaciones parciales de la plaga.	Fuerza al modelo a aprender representaciones parciales.	Simula escenarios de plagas parcialmente cubiertas.
Zoom y Escalado	Amplía o reduce el objeto de interés.	Capturas a distintas distancias.	Emula capturas a distintas distancias.	Relevante en pipelines que combinan imágenes.
Transformación de Perspectiva	Simula cambios en el ángulo de visión.	Condiciones de captura desde plataformas aéreas.	Reproduce las condiciones de captura.	Reproduce las condiciones de captura desde plataformas aéreas.

Transformaciones fotométricas

Las transformaciones fotométricas alteran las propiedades de color, brillo y contraste de la imagen sin modificar la posición espacial de los objetos. Su objetivo principal es simular la variabilidad en las condiciones de iluminación que se presenta en entornos agrícolas reales, donde la luz solar cambia a lo largo del día y las condiciones climáticas introducen variaciones adicionales (Jia, Rezaei, Asli, et al., 2025).

Figura 5. Técnicas de aumento de datos mediante transformaciones fotométricas



Métodos de mezcla de imágenes

Los métodos de mezcla de imágenes (image mixing) constituyen una categoría más reciente y sofisticada de aumento de datos. En lugar de transformar una única imagen, combinan dos o más imágenes del conjunto de entrenamiento para generar un nuevo ejemplo sintético. Estos métodos

han demostrado ser particularmente eficaces para mejorar la regularización y la calibración de los modelos de detección de objetos

Mixup: genera una nueva imagen como combinación lineal pixel a pixel de dos imágenes del conjunto de entrenamiento, ponderadas por un coeficiente aleatorio $\lambda \sim \text{Beta}(\alpha, \alpha)$. Las etiquetas se combinan con las mismas ponderaciones. Fomenta una transición suave entre clases, reduciendo la sobreconfianza del modelo (Zhang et al., 2018).

CutMix: reemplaza una región rectangular de una imagen con el parche correspondiente de otra imagen del conjunto. Las etiquetas se redistribuyen proporcionalmente al área ocupada por cada imagen. A diferencia de Mixup, preserva la coherencia espacial local de los objetos (Yun et al., 2019).

Mosaico : combina cuatro imágenes en un único mosaico 2×2, exponiendo al modelo a múltiples enfermedades y plagas y contextos en un solo ejemplo de entrenamiento. Esta técnica, introducida en YOLOv4 y perfeccionada en versiones posteriores, ha demostrado mejoras consistentes en la detección de objetos pequeños como insectos (Kim et al., 2026).

Métodos basados en parches y oclusión

Los métodos basados en parches buscan mejorar la robustez del modelo ante oclusiones parciales de las enfermedades y plagas, un escenario muy frecuente en condiciones de campo donde el insecto puede estar cubierto por hojas, tallos u otros insectos.

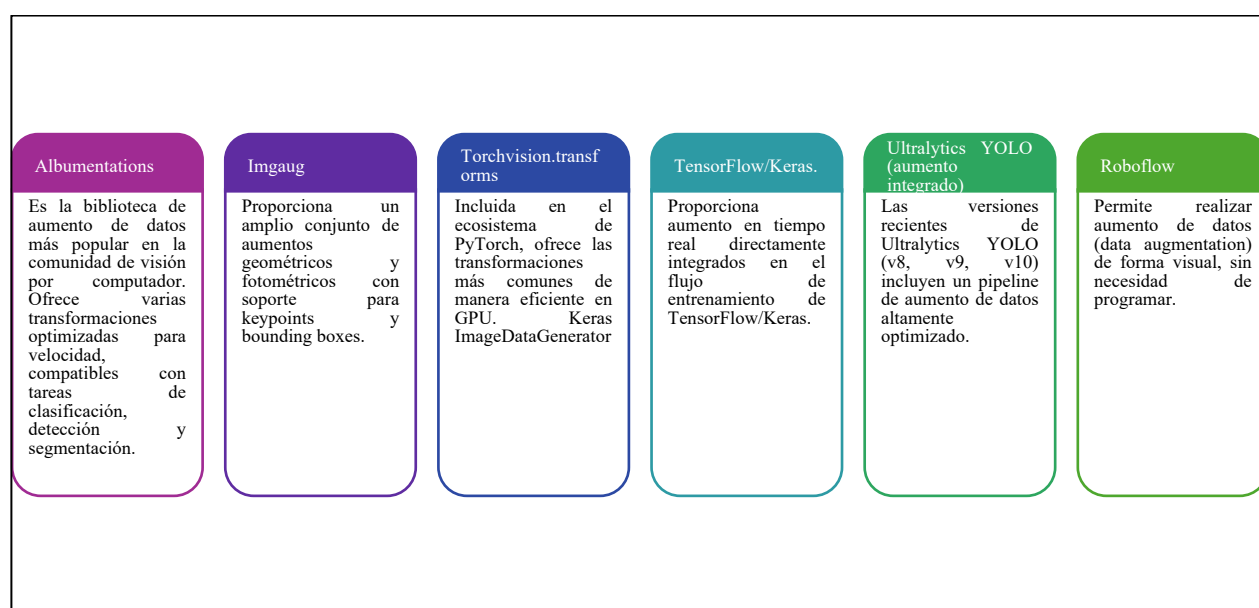
- Cutout / Random Erasing: enmascara aleatoriamente regiones rectangulares de la imagen con píxeles de color constante (generalmente negro o gris). Simula oclusiones parciales y obliga al modelo a aprender representaciones más completas y distribuidas de la plaga (May, 2019; Zhong et al., 2017)

Herramientas y bibliotecas para el aumento de datos

La implementación práctica de las técnicas de aumento de datos se apoya en un conjunto de bibliotecas especializadas que ofrecen interfaces eficientes, compatibles con los principales frameworks de aprendizaje profundo (PyTorch, TensorFlow/Keras). La elección de la herramienta adecuada

depende del tipo de tarea (clasificación vs. detección), la arquitectura del modelo y el volumen del dataset.

Figura 6. Herramientas para el aumento de datos



Impacto del aumento de datos en arquitecturas de detección de enfermedades y plagas

Efecto sobre modelos YOLO

Las arquitecturas YOLO han sido las principales beneficiarias del aumento de datos avanzado en el dominio agrícola. La técnica Mosaic, integrada a partir de YOLOv4, ha demostrado ser el factor diferencial en la detección, al combinar cuatro imágenes en un único ejemplo de entrenamiento, el modelo aprende a detectar en una amplia variedad de contextos y escalas simultáneamente, reduciendo la necesidad de normalización del batch y mejorando la convergencia del entrenamiento (Kim et al., 2026).

Estudios comparativos han demostrado que la eliminación del aumento Mosaic en YOLOv8 puede resultar en una caída de hasta 4 puntos de mAP50

en datasets como IP102, mientras que la combinación de Mosaic con Mixup adiciona aproximadamente 2 puntos de mAP50 frente al uso de Mosaic solo. Asimismo, las transformaciones HSV han mostrado un impacto significativo en la detección de enfermedades y plagas cuya coloración varía entre especies o etapas de desarrollo, como los trips y los ácaros (Wang y Perez, 2017; X. Wang y Liu, 2024; Zheng et al., 2024)

Efecto sobre CNN y modelos de clasificación

En el contexto de la clasificación de enfermedades y plagas mediante CNN, el aumento de datos clásico volteo, rotación, zoom y cambio de brillo, han demostrado ser suficiente para obtener mejoras sustanciales cuando el dataset de partida es pequeño. Estudios como el de Venkateswara y Padmanabhan (2025) reportan incrementos de exactitud al incorporar un pipeline básico de aumento frente a entrenamiento sin aumentos sobre el dataset IP102.

Casos de estudio: aumento de datos en la identificación de enfermedades y plagas

A continuación, se presenta una revisión de la literatura del período 2024-2026 que ilustra el uso de técnicas de aumento de datos en sistemas de identificación de enfermedades y plagas, detallando las transformaciones aplicadas, los modelos empleados y los resultados obtenidos.

Tabla 2. Técnicas de aumento de datos en estudios recientes de identificación de enfermedades y plagas agrícolas (2024-2026).

Nombre del Artículo	Autores	Cultivo Plaga	/	Técnicas Aumento Aplicadas	de Modelo	Exactitud / mAP
Lightweight ViT for Early Insect Pest Identification	(Hong et al., 2026)	Tomate, pepino, mosca blanca, trips	/	Recorte, redimensionamiento, Brillo y contraste	Light-HortiNet	mAP50: 87.2%

Nombre del Artículo	Autores		Cultivo / Plaga	Técnicas de Aumento Aplicadas	Modelo	Exactitud / mAP
A New Hybrid ConvViT Model for Dangerous Farm Insect Detection	(Utku et al., 2025b)		General tipos de insectos	15 Desenfoque, variación de brillo, rotación y adición de ruido	ConvViT	93.61%
YOLO-LCE: Lightweight YOLOv8 for Agricultural Pest Detection	(Cen et al., 2025)		Enfermedades y plagas diversas	Mosaic, Saturación, escalada, HSV, volteo horizontal	YOLO-LCE	mAP50: 63.9%
Deep Learning Agricultural Pest Monitoring	(Venkateswara & Padmanabhan, 2025b)		Plagas diversas	Cambios de exposición, rotación y resolución	CNN	84.95%
Yolo-pest: Detection of Small Insect Pests	(Hakim et al., 2025)		Plagas diversas	Rotación, difuminación, brillo, exposición, volteo, escala de grises, matiz, ruido, giro, saturación, cutmix	YOLOv8x opt.	94.0%
YOLO-YSTs: Yellow Sticky	(Li et al., 2024)		Plagas diversas	volteo vertical, escalado, rotación, recorte, ajuste de brillo, ajuste de	YOLO-YSTs	mAP50: 86.8%

Nombre del Artículo	Autores	Cultivo Plaga	/ Técnicas de Aumento Aplicadas	Modelo	Exactitud / mAP
Trap Pest Detection			saturación y adición de ruido		

El análisis de los casos de estudio revela patrones consistentes en el uso del aumento de datos dentro de la comunidad investigadora. Los estudios que reportan exactitudes superiores al 93% comparten el uso sistemático de al menos tres técnicas de aumento distintas, lo que sugiere que la combinación estratégica de técnicas produce mejores resultados que la aplicación intensiva de una sola transformación.

Beneficios del aumento de datos en la agricultura de precisión

La implementación de técnicas de aumento de datos en pipelines de identificación de enfermedades y plagas agrícolas genera beneficios que trascienden la mejora en métricas de validación. Desde una perspectiva aplicada, estos beneficios impactan directamente en la viabilidad y el costo de implementación de sistemas de monitoreo automatizado.

Figura 7. Beneficio del aumento de datos



Conclusiones

El aumento de datos se consolida como una estrategia indispensable en el desarrollo de sistemas de identificación de enfermedades y plagas agrícolas basados en visión por computador. La revisión de la literatura evidencia que su aplicación sistemática y bien diseñada puede incrementar la precisión de detección, reducir el sobreajuste y mejorar significativamente la generalización en condiciones reales de campo, sin requerir la recolección de datos adicionales.

Las transformaciones geométricas y fotométricas constituyen la base obligatoria de cualquier pipeline de aumento, aportando robustez frente a la variabilidad inherente de los entornos agrícolas.

La selección óptima de técnicas de aumento debe estar guiada por tres criterios: el tipo de arquitectura empleada, las características ecológicas y morfológicas de las enfermedades y plagas objetivo, y las condiciones ambientales del entorno de despliegue.

No existe una política universal de aumento óptima; cada sistema requiere un proceso de validación específico que compare el rendimiento en condiciones controladas y en condiciones de campo real.

Finalmente, el aumento de datos debe entenderse no como un sustituto de la recolección de datos de calidad, sino como un complemento estratégico que maximiza el valor informativo de los datos disponibles.

CAPÍTULO 3: PASO A PASO PARA LA APLICACIÓN DE VISIÓN POR COMPUTADOR EN LA IDENTIFICACIÓN DE ENFERMEDADES Y PLAGAS AGRÍCOLAS

Resumen

Este capítulo presenta una guía metodológica detallada y reproducible para la implementación de sistemas de visión por computador orientados a la identificación automática de enfermedades y plagas agrícolas. El proceso se articula en ocho pasos secuenciales: (1) definición del problema y alcance del sistema; (2) planificación y adquisición de imágenes; (3) construcción y gestión del conjunto de datos; (4) preprocesamiento y aumento de datos; (5) selección y configuración de la arquitectura de aprendizaje profundo; (6) entrenamiento y ajuste fino del modelo; (7) evaluación y validación del desempeño; y (8) despliegue e integración en sistemas agrícolas reales. Para cada paso se describen las actividades concretas, las herramientas recomendadas, los criterios de calidad y los errores frecuentes que deben evitarse. La guía está orientada a investigadores y profesionales del área agrícola y tecnológica que buscan desarrollar soluciones prácticas de detección de enfermedades y plagas con recursos computacionales accesibles.

Palabras clave: Visión por computador, Identificación de enfermedades y plagas, Metodología, Aprendizaje profundo, Agricultura de precisión, Pipeline

Abstract

This chapter presents a detailed and reproducible methodological guide for implementing computer vision systems aimed at the automatic identification of agricultural diseases and pests. The process is structured in eight sequential steps: (1) problem definition and system scope; (2) image acquisition planning; (3) dataset construction and management; (4) preprocessing and data augmentation; (5) deep learning architecture selection and configuration; (6) model training and fine-tuning; (7) performance evaluation and validation; and (8) deployment and integration into real agricultural systems. For each step, specific activities, recommended tools, quality criteria, and common errors to avoid are described. The guide targets researchers and professionals in agricultural and technological domains who seek to develop practical pest detection solutions with accessible computational resources.

Keywords: Computer Vision, Detection, Identification, Agricultural diseases and Pests, Precision Agriculture

Introducción

La implementación de sistemas de visión por computador para la identificación de enfermedades y plagas agrícolas es un proceso que involucra múltiples etapas interdependientes, cada una de las cuales requiere decisiones técnicas precisas que condicionan la calidad del resultado final. Sin embargo, la literatura científica disponible se ha concentrado mayoritariamente en reportar los resultados de modelos específicos, sin ofrecer una guía integrada y reproducible que oriente a investigadores y profesionales a través de todo el proceso, desde la conceptualización del problema hasta el despliegue del sistema en condiciones de campo reales.

Esta ausencia de una metodología unificada genera dificultades prácticas significativas ya que cuando inician en el área cometen errores sistemáticos en la recolección de datos, el etiquetado de imágenes o la configuración del entrenamiento que comprometen la validez de sus resultados (Hussain et al., 2026; Kern et al., 2023); mientras que profesionales del sector agrícola encuentran barreras de entrada que impiden la adopción de tecnologías con alto potencial para mejorar la gestión fitosanitaria de sus cultivos.

Este capítulo busca ofrecer una guía metodológica paso a paso, fundamentada en la evidencia empírica de la literatura reciente y en las mejores prácticas de la comunidad de visión por computador. El proceso se estructura en ocho pasos secuenciales que cubren la totalidad del ciclo de vida del sistema: desde la definición del problema hasta el despliegue operativo. Para cada paso se detallan las actividades concretas que deben realizarse, las herramientas disponibles, los criterios de decisión aplicables y los errores más frecuentes que deben evitarse. La guía está diseñada para ser aplicable con recursos computacionales accesibles, incluyendo entornos de entrenamiento en la nube como Google Colab, y es adaptable a distintos cultivos y regiones geográficas.

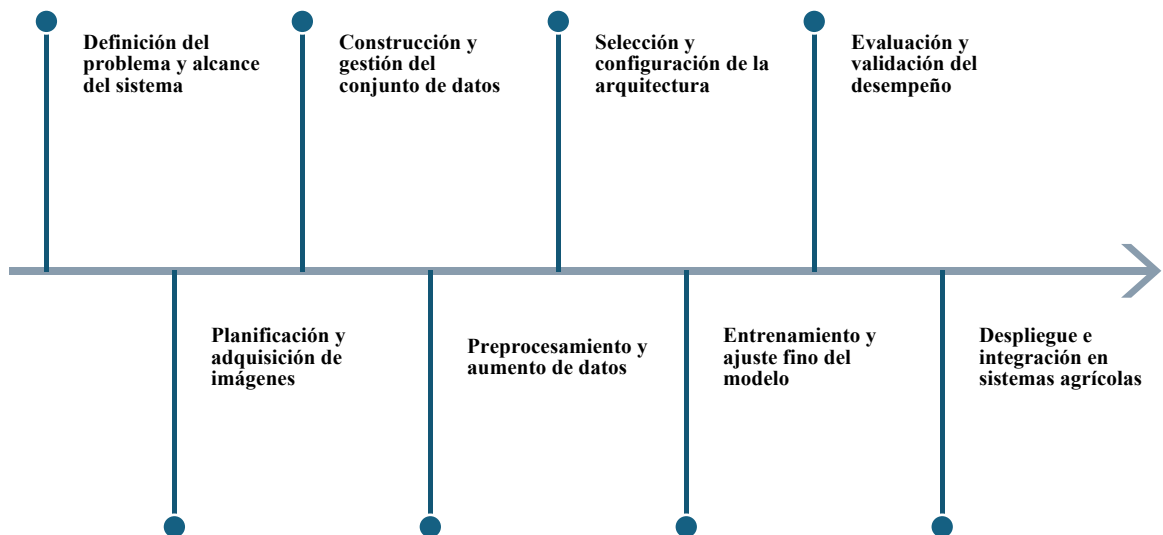
Desarrollo

Visión general del proceso

La aplicación de visión por computador para la identificación de enfermedades y plagas agrícolas sigue un ciclo de vida estructurado en ocho pasos interconectados. Cada paso produce entregables concretos que alimentan el siguiente, de modo que las decisiones tomadas en las etapas iniciales tienen consecuencias directas sobre el desempeño del sistema final. La figura conceptual del proceso puede representarse como un flujo lineal con retroalimentación entre etapas adyacentes: los resultados de la evaluación (Paso 7) pueden motivar ajustes en el entrenamiento (Paso 6), en el aumento de datos (Paso 4), o incluso en la recolección de nuevas imágenes (Paso 2).

Los ocho pasos del proceso son:

Figura 8. Proceso para la aplicación de visión por computador para detección de enfermedades y plagas agrícolas

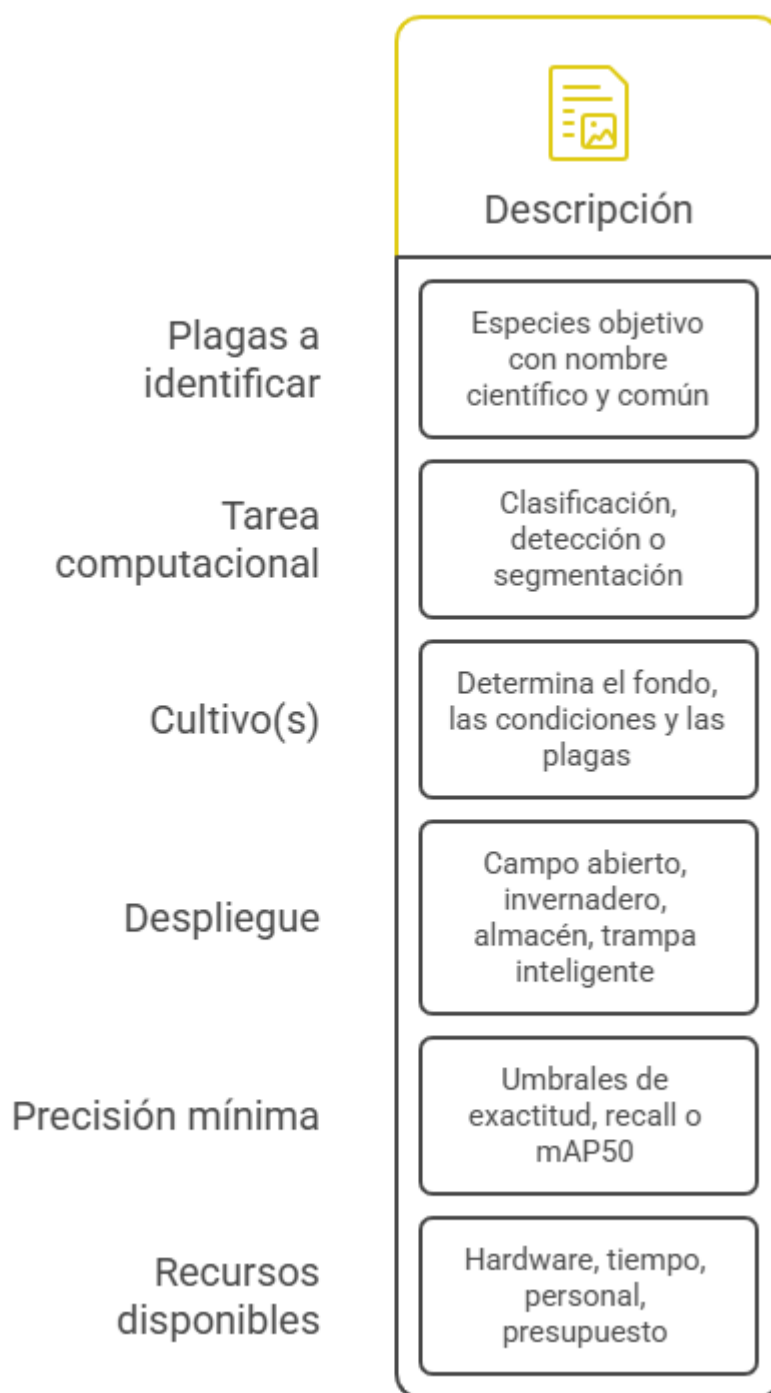


Es importante señalar que, aunque el proceso se presenta de forma secuencial para facilitar su comprensión, en la práctica implica iteraciones frecuentes. Un modelo con bajo rendimiento en la evaluación puede requerir regresar al paso de recolección de datos para ampliar el dataset, o al paso de selección de arquitectura para probar modelos alternativos. Esta naturaleza iterativa es inherente al desarrollo de sistemas de inteligencia artificial y debe ser anticipada en la planificación del proyecto.

Definición del problema

El primer paso consiste en delimitar con precisión el problema que el sistema debe resolver, debido a que una definición poco precisa y no delimitada del problema es la causa más frecuente de fracaso en proyectos de visión por computador aplicada. Antes de recopilar una sola imagen, es imprescindible tomar en consideración los siguientes criterios:

Figura 9. Consideraciones para aplicación de visión por computador



Entregables del Paso 1

Figura 10. Entregables del paso 1



Errores frecuentes

- Definir un número excesivo de clases de enfermedades y plagas sin verificar la disponibilidad de datos para cada una
- No consultar con expertos agronómicos antes de iniciar la recolección

- Confundir la tarea de clasificación con la de detección, generando un diseño de dataset incompatible

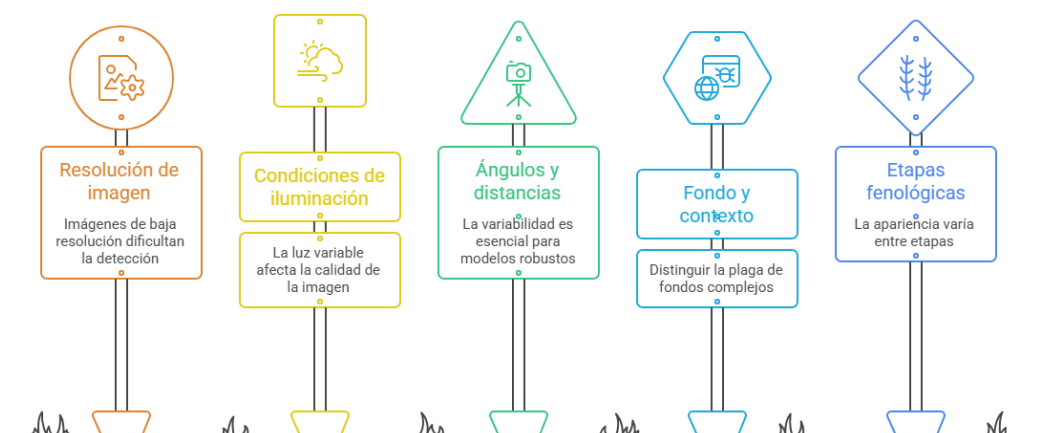
Planificación y adquisición de imágenes

La calidad y diversidad de las imágenes recolectadas determinan en gran medida el rendimiento máximo alcanzable por cualquier modelo. Esta etapa implica diseñar un protocolo de captura y ejecutarlo de forma sistemática para garantizar que el dataset final represente adecuadamente la variabilidad del problema real.

Consideraciones generales para la adquisición de fotos

Para realizar la adquisición de imágenes se debe tomar en consideración:

Figura 11. Ítems a considerar para la adquisición de imágenes



Para la captura de imágenes destinadas a la identificación de enfermedades y plagas mediante visión por computadora, se recomienda emplear cámaras digitales de alta resolución (≥ 12 MP), smartphones con cámara principal, cámaras de dron (multiespectrales o RGB), cámaras macro para insectos pequeños, o cámaras de trampas amarillas pegajosas, según corresponda al entorno de despliegue final. Se sugiere una resolución mínima de 1280×720 píxeles para imágenes de campo y de 2560×1440 para imágenes macro de insectos pequeños, ya que resoluciones menores dificultan la detección de enfermedades y plagas en etapas tempranas. Las capturas deben realizarse en distintas condiciones de iluminación (luz solar directa, sombra, cielo

nublado, amanecer/atardecer), evitando el flash directo por generar reflexiones especulares no representativas del entorno real. Asimismo, debe variarse el ángulo de captura (cenital, oblicuo, lateral) y la distancia a la plaga (macro, media distancia, contexto completo), pues esta variabilidad resulta esencial para entrenar modelos robustos. También conviene incluir distintos fondos (suelo, hojas sanas, hojas dañadas, tallos), de modo que el modelo aprenda a distinguir la plaga de fondos complejos y no únicamente de fondos blancos de laboratorio. Finalmente, se debe recolectar imágenes en las distintas etapas de desarrollo del cultivo (plántula, vegetativa, floración, fructificación) y de la plaga (huevo, larva, ninfa, adulto), dado que su apariencia varía significativamente entre dichas etapas.

Uso de datasets públicos existentes

Antes de iniciar una recolección desde cero, es recomendable revisar los datasets públicos disponibles que pueden servir como base o complemento:

- **IP102:** 75,222 imágenes de 102 especies de insectos plaga. Disponible en GitHub.
- **Pest24:** dataset multiclase con 24 categorías de enfermedades y plagas agrícolas para benchmarking de modelos YOLO.
- **PlantVillage:** 54,000+ imágenes de enfermedades y enfermedades y plagas en hojas de 14 cultivos. Disponible en Kaggle.
- **Roboflow Universe:** repositorio de datasets agrícolas anotados con licencias abiertas, buscable por especie y tarea.
- **GBIF (Global Biodiversity Information Facility):** imágenes georreferenciadas de especies de insectos con identificación taxonómica validada.

Errores frecuentes

- Recolectar todas las imágenes en el mismo día, hora y condición de iluminación

- Capturar únicamente insectos muertos o inmovilizados, que no representan la apariencia en campo
- Ignorar imágenes de insectos beneficiosos (polinizadores, depredadores) que el modelo necesita aprender a no confundir con enfermedades y plagas

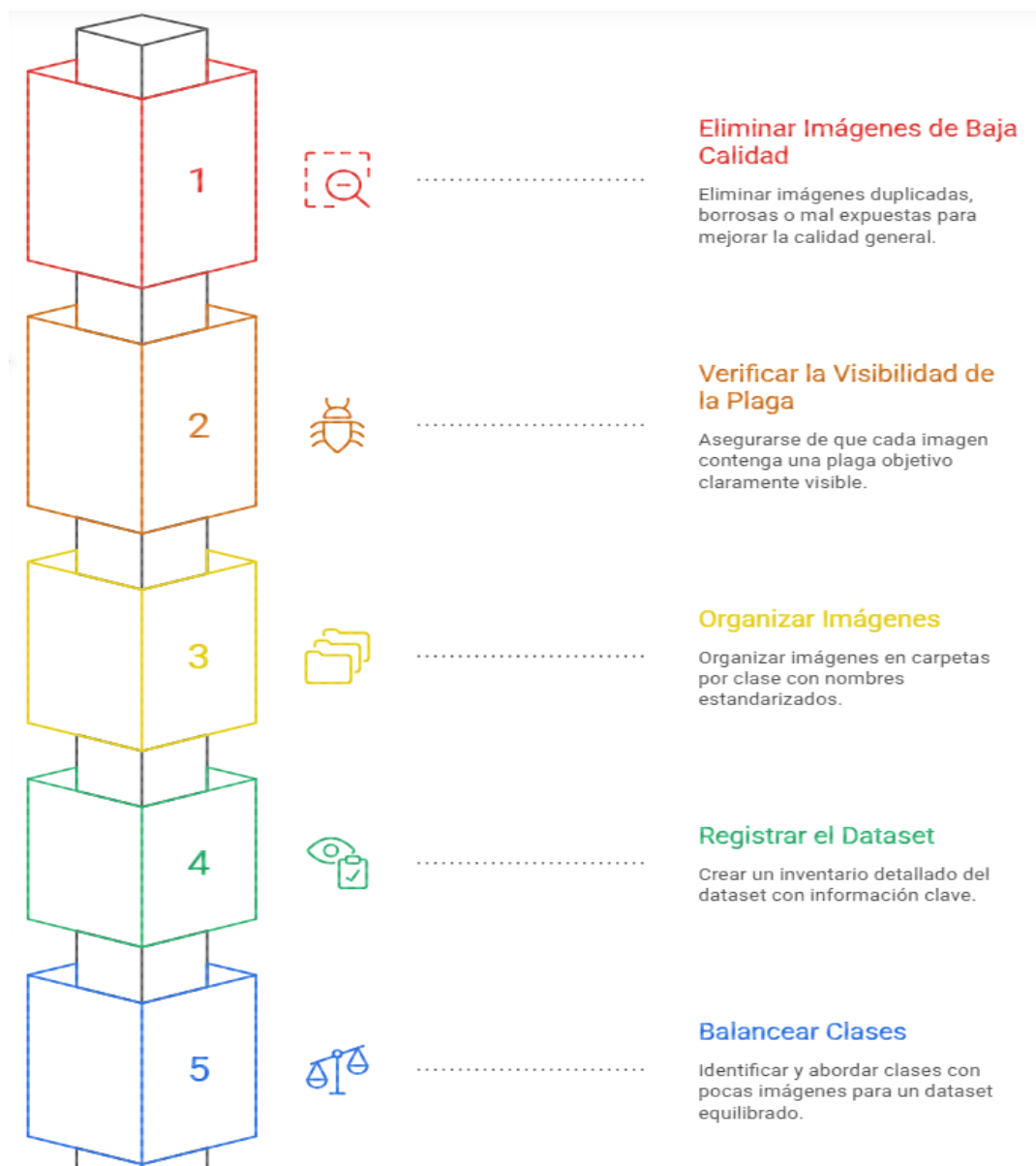
Construcción y gestión del conjunto de datos

Una vez recopiladas las imágenes, deben organizarse, limpiarse, anotarse y dividirse en subconjuntos de entrenamiento, validación y prueba. Este paso es el más laborioso del proceso y el que mayor impacto tiene sobre la calidad del modelo resultante. Datos mal anotados o mal divididos pueden hacer que un modelo aparentemente excelente en validación falle completamente en campo.

Consideraciones generales

Como parte de la curación del dataset, se deben eliminar imágenes duplicadas, borrosas, sobre o subexpuestas, así como verificar que cada imagen contenga al menos una instancia visible de la plaga objetivo. Las imágenes deben organizarse en carpetas por clase con nombres estandarizados, sin espacios ni caracteres especiales, y acompañarse de un inventario que registre número de imágenes por clase, fuente, fecha de captura y resolución. Finalmente, las clases con pocas imágenes deben balancearse mediante recolección adicional o aumento de datos orientado, evitando sesgos hacia las clases mayoritarias, por lo que se deberá tener las siguientes consideraciones:

Figura 12. Consideraciones para la construcción de un dataset



Etiquetado de imágenes

El etiquetado es el proceso de asignar a cada imagen (o región de imagen) la etiqueta de clase correspondiente. Para tareas de detección y segmentación, también implica dibujar los límites del objeto, algunas herramientas recomendadas son:

Tabla 3. *Herramientas para el etiquetado de imágenes*

Herramienta	Descripción
Label Studio	Plataforma open-source para clasificación, detección, segmentación y anotación de texto. Interfaz web intuitiva. Soporta exportación en formatos YOLO, COCO y Pascal VOC.
CVAT	Herramienta de anotación colaborativa especialmente diseñada para bounding boxes y segmentación de instancias. Integra modelos de IA para anotación semiautomática.
Roboflow Annotate	Plataforma en la nube con flujos de trabajo de anotación, revisión de calidad, aumento y exportación integrados. Versión gratuita disponible para proyectos pequeños.
LabelImg	Herramienta de escritorio ligera para anotación de bounding boxes en formato YOLO y Pascal VOC. Ideal para proyectos pequeños sin acceso a internet.
VGG Image Annotator (VIA)	Herramienta basada en navegador sin necesidad de instalación. Soporta polígonos, puntos y bounding boxes. Exporta en CSV y JSON.

División del dataset

El dataset anotado debe dividirse en tres subconjuntos mutuamente excluyentes:

Figura 13. División del dataset

	 Entrenamiento (train)	 Validación (val)	 Prueba (test)
Porcentaje	70-80%	10-15%	10-15%
Uso	Ajustar pesos	Monitorear rendimiento	Evaluación final
Restricciones	Ninguna	No para selección final	No durante el desarrollo
Representación	Diverso y representativo	No para selección final	Condiciones reales de despliegue

La división debe realizarse de forma estratificada, garantizando que cada subconjunto contenga proporciones similares de cada clase. En datasets pequeños, puede emplearse validación cruzada k-fold ($k=5$) como alternativa para maximizar el uso de los datos disponibles.

Errores frecuentes

- Incluir imágenes del mismo evento de captura (mismo insecto, misma sesión) en entrenamiento y prueba.
- No estratificar la división, dejando clases minoritarias sin representación en validación o prueba
- Etiquetar solo las instancias más grandes o visibles, omitiendo las pequeñas
- No documentar el protocolo de etiquetado, impidiendo su replicación

Preprocesamiento y aumento de datos

Antes de introducir las imágenes al modelo, deben someterse a un pipeline de preprocesamiento que las estandarice y optimice para el entrenamiento.

Adicionalmente, se aplican técnicas de aumento de datos para incrementar artificialmente la diversidad del conjunto de entrenamiento. El preprocesamiento se aplica tanto a entrenamiento como a validación y prueba; el aumento solo al conjunto de entrenamiento.

Operaciones de preprocesamiento (train + val + test)

Antes de que el modelo procese cualquier imagen, es necesario aplicar un conjunto de transformaciones previas que garanticen la consistencia y calidad de los datos de entrada. Este preprocesamiento no modifica el contenido visual relevante de las imágenes ni altera las características diagnósticas de las enfermedades y plagas, sino que estandariza las condiciones técnicas bajo las cuales el modelo las recibirá, asegurando que diferencias triviales de formato, resolución o escala de valores no introduzcan ruido en el aprendizaje. Sin este paso, incluso un dataset bien anotado puede producir un modelo inestable o con convergencia lenta.

Figura 14. Operaciones de pre procesamiento de imágenes



Pipeline de aumento de datos (solo train)

El pipeline de aumento debe definirse según el tipo de arquitectura y las características de las enfermedades y plagas objetivo. Se recomienda utilizar la librería Albumentations por su eficiencia y compatibilidad con coordenadas de bounding boxes:

Capa base: volteo horizontal ($p=0.5$), volteo vertical ($p=0.3$), rotación $\pm 45^\circ$ ($p=0.5$), cambio de brillo y contraste ($p=0.4$).

Capa fotométrica: ajuste HSV (hue $\pm 10^\circ$, saturación $\pm 30\%$, valor $\pm 20\%$) para simular variaciones de iluminación. Ruido gaussiano ($p=0.2$) para simular sensores de baja calidad.

Capa geométrica avanzada: transformación de perspectiva ($p=0.3$), GridDistortion ($p=0.2$) para enfermedades y plagas con morfología deformable.

Métodos de mezcla (modelos YOLO): activar Mosaic ($p=1.0$) y Mixup ($p=0.15$) en el archivo de configuración de Ultralytics.

Oclusión: Cutout o RandomErasing con probabilidad 0.2 para simular oclusión parcial por follaje.

Errores Frecuentes

Aplicar normalización con los valores de media y desviación estándar incorrectos para el modelo preentrenado elegido

Aplicar aumento de datos al conjunto de validación o prueba, sobreestimando el rendimiento del modelo

No actualizar las coordenadas de bounding boxes al aplicar transformaciones geométricas, generando anotaciones inconsistentes

Aplicar transformaciones fotométricas tan agresivas que eliminan las características cromáticas diagnósticas de la plaga

Selección y configuración de la arquitectura

La elección de la arquitectura del modelo es una de las decisiones más importantes del proceso. No existe una arquitectura universalmente óptima; la selección debe basarse en el tipo de tarea, el tamaño del dataset, los recursos computacionales disponibles y los requisitos de velocidad de inferencia del sistema final.

YOLOv8 / YOLOv10: arquitecturas de detección de una sola etapa optimizadas para tiempo real. Su variante nano (n) entrena de forma rápida y es la opción recomendada cuando se prioriza la velocidad de inferencia en campo.

EfficientNet-B0 a B4: redes de clasificación con excelente relación precisión-parámetros, por lo que son ideales para identificar la especie de plaga cuando la localización exacta no es requerida y el dataset disponible es pequeño.

ResNet-50 / ResNet-101: arquitecturas de clasificación ampliamente documentadas, robustas para transfer learning.

Vision Transformer (ViT): captura relaciones globales entre regiones de la imagen mediante mecanismos de atención. Supera a las CNN en datasets grandes, pero requiere preentrenamiento extenso para funcionar correctamente con datasets agrícolas de tamaño moderado.

Términos claves

Época

Una época representa una pasada completa del modelo por todas las imágenes del conjunto de entrenamiento. Si el dataset tiene 1.000 imágenes y se entrena durante 100 épocas, el modelo habrá procesado cada imagen exactamente 100 veces, ajustando sus parámetros internos en cada pasada. Más épocas no siempre significan mejor resultado: a partir de cierto punto el modelo deja de mejorar o comienza a memorizar los datos en lugar de aprender patrones generales, fenómeno conocido como sobreajuste.

Optimizador

El optimizador es el mecanismo matemático que decide cómo ajustar los parámetros internos del modelo después de cada error cometido. La elección del optimizador afecta la velocidad de aprendizaje y la estabilidad del entrenamiento.

Learning rate (tasa de aprendizaje)

Es el tamaño del paso que da el optimizador en cada ajuste, un learning rate muy alto hace que el modelo dé saltos demasiado grandes y nunca encuentre

la solución óptima; uno muy bajo hace que el aprendizaje sea innecesariamente lento. En la práctica, se comienza con un valor moderado (0.01) y se reduce progresivamente a medida que el modelo se acerca a la solución, proceso que gestiona automáticamente el learning rate scheduler.

Learning rate scheduler

Es el programa que controla cómo cambia la tasa de aprendizaje a lo largo del entrenamiento, Cosine Annealing, ReduceLROnPlateau y OneCycleLR son distintas estrategias para automatizar esta reducción progresiva.

Tamaño de batch

En lugar de ajustar el modelo después de ver cada imagen individual, el entrenamiento agrupa las imágenes en lotes (batches) y calcula el error promedio del grupo antes de hacer el ajuste. Un batch de 16 significa que el modelo ve 16 imágenes, calcula cuánto se equivocó en promedio y luego ajusta sus parámetros.

Sobreajuste (overfitting)

Es el error más frecuente en el entrenamiento de modelos agrícolas con datasets pequeños. Ocurre cuando el modelo aprende de memoria las imágenes de entrenamiento en lugar de extraer los patrones generales que definen a cada plaga. El resultado es un modelo que obtiene métricas excelentes sobre las imágenes conocidas pero falla al enfrentarse a imágenes nuevas tomadas en campo. Se detecta cuando la pérdida de validación sube mientras la de entrenamiento sigue bajando, y se corrige añadiendo más imágenes, aplicando aumento de datos o reduciendo el número de épocas.

Transfer learning

Es el principio de aprovechar el conocimiento previo de un modelo ya entrenado en millones de imágenes generales para acelerar el aprendizaje sobre un problema específico. La analogía directa con la agronomía es la de un técnico con amplia experiencia general en botánica que necesita especializarse en una plaga nueva: no aprende desde cero, sino que aplica su conocimiento previo y solo necesita ejemplos adicionales del caso específico.

En la práctica, esto reduce de millones a unos pocos centenares las imágenes necesarias para obtener un modelo funcional.

Elección de modelos

Una vez seleccionada la arquitectura, deben configurarse los siguientes componentes:

Función de pérdida

Para clasificación multiclase se utiliza CrossEntropyLoss. En detección, YOLO combina tres pérdidas simultáneas: clasificación, localización (IoU loss) y objetividad. Cuando el dataset presenta clases desbalanceadas, se recomienda Focal Loss con $\gamma=2$ para penalizar más los errores en clases minoritarias.

Optimizador

SGD con momentum 0.937 y weight decay 0.0005 es el estándar para modelos YOLO. Para CNN y ViT con transfer learning se prefiere Adam o AdamW con $lr=1e-4$. En datasets pequeños, Ranger o SAM ofrecen mayor estabilidad en la convergencia.

Learning rate scheduler

YOLO utiliza Cosine Annealing con warmup de 3 épocas para estabilizar el inicio del entrenamiento. Para CNN se recomienda ReduceLROnPlateau con $patience=5$ y $factor=0.5$, que reduce la tasa de aprendizaje cuando la métrica de validación se estanca. OneCycleLR acelera el entrenamiento con transfer learning al combinar warmup y decaimiento en un solo ciclo.

Tamaño de batch

Se recomienda un batch de 32-64 imágenes para GPU con 8-16 GB de VRAM trabajando con resolución 640×640 . Si aparecen errores de memoria (CUDA out of memory), reducir a 16. Cuando la GPU es pequeña, gradient accumulation permite simular un batch mayor sin aumentar el consumo de memoria.

Transfer learning

Se cargan pesos preentrenados en ImageNet o COCO como punto de partida. Las primeras capas se congelan durante 10-15 épocas, permitiendo que solo la cabeza de clasificación o detección se adapte al nuevo dominio. Posteriormente se descongelan progresivamente todas las capas para un ajuste fino completo, reduciendo el tiempo de convergencia y el riesgo de sobreajuste.

Entrenamiento y ajuste del modelo

El entrenamiento es el proceso mediante el cual el modelo ajusta sus parámetros internos para minimizar la función de pérdida sobre el conjunto de entrenamiento. Este proceso requiere monitoreo continuo para detectar problemas como el sobreajuste, el subajuste o la inestabilidad numérica, y ajustar los hiperparámetros en consecuencia.

Configuración del modelo

Número de épocas

Para modelos YOLO con preentrenamiento se recomiendan entre 100 y 300 épocas; para CNN con transfer learning es suficiente con 50 a 100. En ambos casos se recomienda activar early stopping, que detiene el entrenamiento automáticamente cuando el modelo no mejora durante un número determinado de épocas consecutivas (paciencia de 20 a 50), evitando el sobreentrenamiento sin necesidad de monitoreo manual constante.

Warmup

Durante las primeras 3 a 5 épocas el modelo inicia con una tasa de aprendizaje muy baja, aproximadamente diez veces menor a la configurada, y la incrementa gradualmente hasta el valor normal. Este período de calentamiento estabiliza los primeros ajustes de los pesos, especialmente cuando se trabaja con modelos grandes o batches pequeños, previniendo que el modelo tome decisiones bruscas al inicio que sean difíciles de corregir después.

Regularización

Son técnicas que reducen el riesgo de sobreajuste durante el entrenamiento. El dropout apaga aleatoriamente entre el 20% y el 50% de las conexiones de la red en cada paso, obligando al modelo a no depender de ningún grupo específico de neuronas. El weight decay penaliza los pesos muy grandes en el optimizador, favoreciendo soluciones más simples y generalizables. El label smoothing suaviza las etiquetas de clase, lo que mejora la calibración del modelo cuando dos plagas son morfológicamente similares y los límites entre clases no son completamente nítidos.

Seguimiento de experimentos

Herramientas como Weights & Biases (wandb) o TensorBoard registran automáticamente las curvas de pérdida, las métricas por época y ejemplos visuales de predicciones a lo largo del entrenamiento. Esto permite comparar diferentes configuraciones de hiperparámetros, identificar en qué época comenzó el sobreajuste y tomar decisiones informadas sobre cuál versión del modelo guardar como definitiva.

Checkpoint

Un checkpoint es una copia del estado del modelo guardada en un momento específico del entrenamiento. Se recomienda guardar automáticamente el mejor modelo según la métrica de validación (val_mAP50 o val_accuracy) y una copia cada cierto número de épocas. En Google Colab esto es especialmente importante porque las sesiones pueden interrumpirse por inactividad o por límite de tiempo, y sin checkpoints todo el progreso del entrenamiento se pierde.

Evaluación y validación del desempeño

La evaluación del modelo debe realizarse sobre el conjunto de prueba, que no ha sido utilizado en ninguna fase del desarrollo. El objetivo es obtener una estimación imparcial del rendimiento real del sistema antes de su despliegue. Una evaluación incompleta o sobre datos contaminados puede generar una falsa confianza en el sistema.

Para comprender las métricas de evaluación es necesario partir de cuatro conceptos fundamentales que describen los tipos de resultados que puede producir un modelo al analizar una imagen:

Verdadero positivo (TP): el modelo detecta una plaga y efectivamente hay una plaga en ese lugar. Es el resultado ideal: detección correcta.

Verdadero negativo (TN): el modelo no detecta ninguna plaga en una región de la imagen y en efecto no había ninguna. También es un resultado correcto: el modelo supo reconocer la ausencia de plaga.

Falso positivo (FP): el modelo indica que hay una plaga pero en realidad no la hay. En el contexto agrícola equivale a una falsa alarma: el sistema generaría una alerta de infestación donde no existe, lo que puede derivar en aplicaciones innecesarias de plaguicidas.

Falso negativo (FN): el modelo no detecta una plaga que sí está presente en la imagen. Es el error más crítico en agricultura porque significa que una infestación real pasa desapercibida, permitiendo que la plaga se propague sin intervención oportuna.

Tabla 4. *Métricas de Evaluación*

Métrica	Fórmula base	Rango	Ideal para enfermedades y plagas	Limitación principal
Exactitud (Accuracy)	$\frac{TP+TN}{Total}$	0 – 1	Datasets balanceados	Engañosa con clases desbalanceadas
Precisión (Precision)	$\frac{TP}{TP+FP}$	0 – 1	Minimizar falsas alarmas	No mide falsos negativos

Métrica	Fórmula base	Rango	Ideal para enfermedades y plagas	Limitación principal
Exhaustividad (Recall)	$\frac{TP}{TP+FN}$	0 – 1	Detección crítica de plagas	Puede inflar con umbral bajo
F1-Score	$\frac{2 \cdot P \cdot R}{P+R}$	0 – 1	Balance precisión-recall	No considera TN
mAP50	Media AP @ IoU=0.5	0 – 1	Detección con bounding box	Sensible al umbral IoU
mAP50-95	Media AP @ IoU 0.5–0.95	0 – 1	Localización precisa	Exigente; difícil >0.6
AUC-ROC	Área bajo curva ROC	0.5 – 1	Clasificación binaria plaga/no plaga	Menos informativo multiclase

Análisis de errores

Además de las métricas cuantitativas, es fundamental realizar un análisis cualitativo de los errores del modelo:

- **Matriz de confusión:** identificar qué pares de clases se confunden con mayor frecuencia. En enfermedades y plagas, las confusiones entre

especies morfológicamente similares son esperables y deben documentarse.

- **Análisis por clase:** calcular precisión y recall individuales por clase. Las clases con bajo recall son las que el modelo falla en detectar; las de baja precisión generan falsas alarmas frecuentes.
- **Inspección visual de errores:** examinar las imágenes mal clasificadas para identificar patrones: ¿son imágenes de baja calidad? ¿condiciones de iluminación extremas? ¿etapas de desarrollo específicas? Esta inspección guía las mejoras en la siguiente iteración.
- **Análisis de confianza:** verificar la calibración del modelo evaluando si las predicciones con alta confianza corresponden efectivamente a las correctas.

Validación en condiciones reales

La evaluación sobre el conjunto de prueba estático no es suficiente para garantizar el desempeño en campo. Adicionalmente, se recomienda:

- **Prueba en campo:** desplegar el modelo de forma provisional en el entorno real y recopilar predicciones sobre imágenes capturadas en condiciones operativas durante al menos dos semanas.
- **Validación con expertos:** mostrar las predicciones del modelo a entomólogos o fitopatólogos y registrar las discrepancias.
- **Prueba de robustez:** evaluar el modelo con imágenes degradadas artificialmente (ruido, desenfoque, baja resolución, cambios de iluminación extremos) para estimar su comportamiento en condiciones adversas.

Despliegue e integración en sistemas agrícolas

El despliegue es la fase en la que el modelo entrenado y validado se integra en un sistema operativo que puede ser utilizado por agricultores, técnicos o investigadores en condiciones reales. Esta fase requiere decisiones sobre el formato del modelo, la plataforma de despliegue y la interfaz de usuario.

Tabla 5 Plataformas de Despliegue

Plataforma	Descripción
Aplicación móvil	Integrar el modelo en una app Android/iOS usando TensorFlow Lite o CoreML. El agricultor captura la imagen con la cámara del teléfono y recibe la identificación de la plaga en tiempo real sin conexión a internet. Arquitecturas ligeras (YOLOv8n, MobileNetV3) son imprescindibles en este escenario.
API web (REST)	Exponer el modelo como servicio web mediante FastAPI o Flask. El cliente envía la imagen por HTTP y recibe la predicción en JSON. Permite integración con plataformas de gestión agrícola (ERP, sistemas de riego automatizado, alertas por WhatsApp/Telegram).
Dispositivo edge (IoT)	Desplegar en Raspberry Pi 4, NVIDIA Jetson Nano o similar para procesamiento local en trampas inteligentes o cámaras fijas instaladas en campo. Usar TensorRT o ONNX Runtime para optimizar la inferencia en hardware limitado.
Plataforma cloud	AWS SageMaker, Google Vertex AI o Azure ML para inferencia escalable bajo demanda. Adecuado para sistemas que procesan grandes volúmenes de imágenes de drones o sensores remotos.
Integración con drones	Conectar el modelo al sistema de procesamiento en tierra del dron para análisis de imágenes multiespectrales o RGB capturadas en vuelos de monitoreo. Salida de mapas de calor de infestación por parcela.

El despliegue no es el final del ciclo de vida del sistema. Un modelo desplegado debe ser monitoreado continuamente para detectar degradación del rendimiento causada por cambios en las condiciones del campo, aparición de nuevas enfermedades y plagas o variaciones estacionales no contempladas en el dataset original:

- Recopilar y etiquetar las predicciones incorrectas reportadas por los usuarios para enriquecer el dataset
- Establecer un umbral de confianza mínimo por debajo del cual el sistema solicita confirmación humana
- Re-entrenar el modelo periódicamente (cada temporada agrícola) con los nuevos datos recopilados
- Versionar todos los modelos desplegados con registros de fecha, métricas y condiciones de entrenamiento
- Implementar alertas automáticas cuando la distribución de las predicciones se desvía significativamente del patrón histórico.

Conclusiones

La aplicación de visión por computador para la identificación de enfermedades y plagas agrícolas es un proceso sistemático que va mucho más allá del entrenamiento de un modelo de aprendizaje profundo. Los ocho pasos descritos en este capítulo desde la definición precisa del problema hasta el monitoreo post-despliegue conforman un ciclo de vida completo cuya calidad global está determinada por el eslabón más débil: un dataset mal construido compromete el mejor modelo; un modelo excelente mal desplegado no genera ningún impacto en la gestión fitosanitaria real.

La evidencia empírica revisada confirma que las fases de adquisición y gestión del dataset (Pasos 2 y 3) son las que mayor impacto tienen sobre el rendimiento final del sistema, superando incluso la elección de la arquitectura. Un dataset diverso, correctamente anotado y bien dividido permite que arquitecturas relativamente simples como YOLOv8n alcancen exactitudes superiores al 90%, mientras que el mejor modelo disponible entrenado con datos deficientes raramente supera el 70%.

Es fundamental subrayar que la implementación exitosa de estos sistemas requiere una colaboración estrecha entre expertos en tecnología y expertos agronómicos. La visión por computador proporciona las herramientas; el conocimiento entomológico y fitosanitario determina cómo usarlas correctamente para construir sistemas que realmente mejoren la toma de decisiones en campo y contribuyan a una agricultura más eficiente, sostenible y basada en datos.

CAPÍTULO 4: DE LA TEORÍA AL MODELO: APLICACIÓN PARA LA DETECCIÓN DE ENFERMEDADES Y PLAGAS CON PYTHON

Resumen

Este capítulo traduce los fundamentos teóricos de la visión por computador en una implementación práctica y reproducible para la detección y clasificación automática de la severidad de la cenicilla (oídio) en hojas de mango, utilizando Python. Partiendo de los conceptos esenciales del aprendizaje automático y profundo, se presenta un pipeline completo y reproducible de ocho etapas: definición del problema, adquisición de imágenes mediante APIs públicas (GitHub y Kaggle), construcción de un conjunto de datos de 500 imágenes etiquetadas manualmente en cinco niveles de severidad con una herramienta de etiquetado desarrollada en Tkinter, preprocesamiento y extracción de características, diseño y configuración de tres arquitecturas de clasificación (Máquina de Vectores de Soporte, Red Neuronal Convolucional y Vision Transformer), entrenamiento y ajuste de hiperparámetros, evaluación comparativa mediante métricas de exactitud, precisión, recall, F1-score y matriz de confusión, y despliegue final en una aplicación de escritorio desarrollada en Tkinter para uso en campo. Los resultados muestran que el modelo SVM alcanzó la mayor exactitud, seguido del Vision Transformer y la CNN, lo que evidencia las limitaciones de entrenar arquitecturas de aprendizaje profundo con conjuntos de datos reducidos y motiva estrategias futuras de ampliación de datos y transferencia de aprendizaje. El capítulo está orientado a investigadores y estudiantes del área agrícola y tecnológica que buscan iniciar en la implementación práctica de visión por computador sin requerir infraestructura computacional especializada.

Palabras clave: visión por computador, cenicilla, hojas de mango, máquinas de vectores de soporte, redes neuronales convolucionales, Vision Transformer

Abstract

This chapter translates the theoretical foundations of computer vision into a practical, reproducible implementation for the automatic detection and severity classification of powdery mildew on mango leaves using Python. Starting from the essential concepts of machine and deep learning, a complete, reproducible eight-stage pipeline is presented: problem definition, image acquisition through public APIs (GitHub and Kaggle), construction of a 500-image dataset manually labeled into five severity levels using a custom Tkinter labeling tool, preprocessing and feature extraction, design and configuration of three classification architectures (Support Vector Machine, Convolutional Neural Network, and Vision Transformer), training and hyperparameter tuning, comparative evaluation using accuracy, precision, recall, F1-score and confusion-matrix metrics, and final deployment in a Tkinter desktop application for field use. Results show that the SVM model achieved the highest accuracy, followed by the Vision Transformer and the CNN, highlighting the limitations of training deep-learning architectures on small datasets and motivating future data-augmentation and transfer-learning strategies. The chapter is aimed at researchers and students in agricultural and technological fields seeking a practical, hands-on introduction to computer vision without requiring specialized computing infrastructure.

Keywords: computer vision, powdery mildew, mango leaves, support vector machines, convolutional neural networks, Vision Transformer

Introducción

El mango (*Mangifera indica* L.) es uno de los cultivos frutícolas de mayor relevancia económica en zonas tropicales y subtropicales, y su productividad se ve amenazada de forma recurrente por enfermedades fúngicas, entre las cuales la cenicilla (oídio, causada principalmente por *Oidium mangiferae*) es una de las más comunes. Esta enfermedad se manifiesta como un polvo blanquecino sobre hojas, inflorescencias y frutos jóvenes, y su severidad determina el impacto real sobre la cosecha y las decisiones de manejo fitosanitario que debe tomar el agricultor.

El diagnóstico visual tradicional depende de la experiencia del agrónomo o del productor y es susceptible a la subjetividad, a la variabilidad entre observadores y a la falta de disponibilidad de personal capacitado en campo. La visión por computador ofrece una alternativa objetiva y escalable: a partir de una fotografía de la hoja, un modelo de clasificación puede estimar tanto la presencia de la enfermedad como su nivel de severidad, apoyando decisiones oportunas de manejo.

Este capítulo documenta, de forma reproducible, el desarrollo completo de un sistema de detección y clasificación de severidad de cenicilla en hojas de mango en cinco niveles (de resistente a altamente susceptible), construido íntegramente en Python. A diferencia de un enfoque basado en un único modelo, se optó por implementar y comparar tres paradigmas de clasificación con complejidad creciente, una Máquina de Vectores de Soporte (SVM) sobre características diseñadas manualmente, una Red Neuronal Convolucional (CNN) y un Vision Transformer (ViT), con el fin de analizar, con evidencia empírica, cómo se comporta cada familia de modelos cuando se entrena con un conjunto de datos de tamaño modesto (500 imágenes), una condición habitual en proyectos agrícolas de alcance institucional o de tesis, donde no siempre se dispone de miles de imágenes etiquetadas ni de infraestructura de cómputo especializada.

El desarrollo se organiza en ocho etapas que recorren todo el ciclo de vida de un proyecto de visión por computador aplicado: desde la definición del problema hasta el despliegue de una aplicación de escritorio funcional. Cada etapa incluye el código Python efectivamente utilizado, la explicación de las decisiones técnicas tomadas y los resultados obtenidos, de modo que el lector pueda replicar el proceso con su propio conjunto de imágenes.

Desarrollo

Explicación paso a paso, la metodología aplicada para construir el sistema de detección y clasificación de severidad de cenicilla en hojas de mango, siguiendo el pipeline completo: definición del problema, adquisición de imágenes, construcción del conjunto de datos, preprocesamiento, selección de arquitecturas, entrenamiento, evaluación y despliegue.

Definición del problema y alcance del sistema

El problema se definió como una tarea de clasificación multiclase supervisada: dada una imagen de una hoja de mango, el sistema debe asignarla a uno de cinco niveles de severidad de cenicilla, definidos según el porcentaje aproximado de tejido foliar afectado:

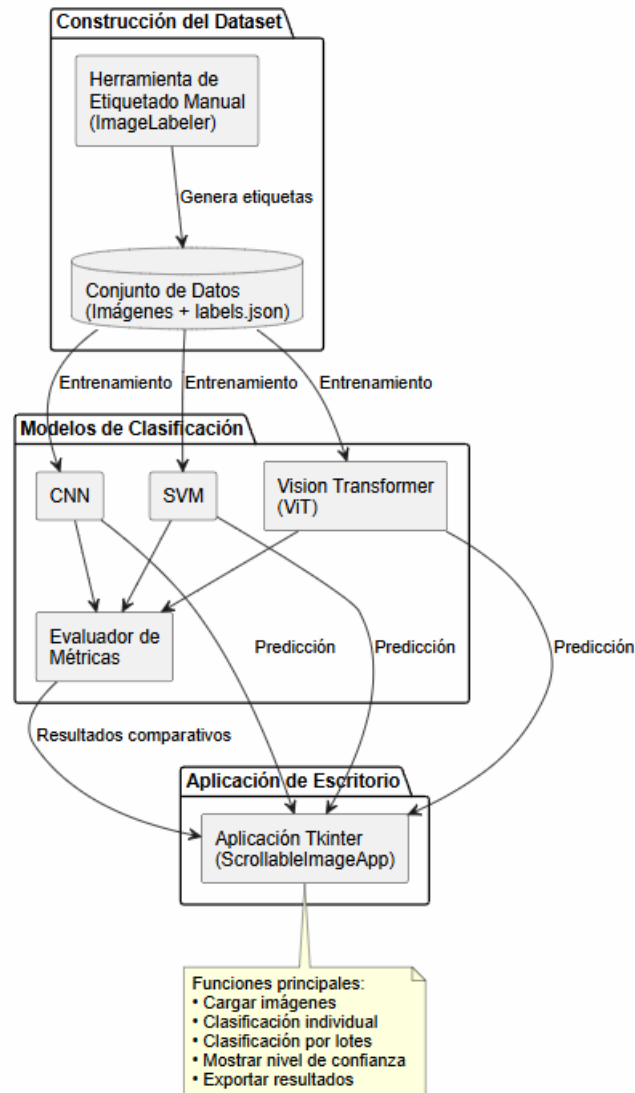
- Clase 1 – Resistente: sin síntomas o muy leves (< 5 % de área afectada).
- Clase 2 – Moderadamente tolerante: síntomas leves (5–25 %).
- Clase 3 – Ligeramente tolerante: síntomas moderados (25–50 %).
- Clase 4 – Susceptible: síntomas severos (50–75 %).
- Clase 5 – Altamente susceptible: síntomas muy severos (> 75 %).

El alcance del proyecto se delimitó en tres componentes principales, el primero corresponde al desarrollo de una herramienta de etiquetado manual, destinada a la construcción del conjunto de datos de entrenamiento, permitiendo generar anotaciones propias sin depender de bases de datos previamente etiquetadas. El segundo componente comprende la implementación de tres modelos de clasificación de imágenes, entrenables de forma local y sin requerir el uso de unidades de procesamiento gráfico (GPU) en la nube, lo que posibilita comparar su desempeño bajo las mismas condiciones de entrenamiento y utilizando el mismo conjunto de datos. Finalmente, el tercer componente consiste en el desarrollo de una aplicación de escritorio, implementada con Tkinter, que integra los tres modelos de clasificación y permite a usuarios sin conocimientos de programación cargar imágenes, tanto de forma individual como por lotes, para obtener la predicción del nivel de severidad de la enfermedad junto con el porcentaje de confianza asociado a cada resultado.

Asimismo, se optó por emplear un enfoque de clasificación de imagen completa (*whole-image classification*), en lugar de técnicas de detección de objetos mediante cajas delimitadoras (*bounding boxes*), debido a que el

objetivo agronómico del proyecto es estimar la severidad global presente en la hoja y no identificar o localizar lesiones individuales.

Figura 15. Arquitectura del proyecto



Planificación y adquisición de imágenes

Para obtener el conjunto de datos inicial, se consideró utilizar imágenes de hojas de mango afectadas por cenicienta disponibles en repositorios públicos. Esta opción permitió contar con una cantidad inicial de imágenes sin tener

que realizar, desde el comienzo, una recopilación propia de fotografías. Para facilitar este proceso, se desarrollaron dos scripts independientes encargados de descargar las imágenes de manera automática las imágenes fueron recolectadas en repositorios de GitHub cree un archivo llamado (descargas.py) donde se utiliza la API de contenidos de GitHub para acceder a carpetas específicas de conjuntos de datos públicos relacionados con enfermedades de las hojas de mango. Dentro de estos repositorios se buscaron principalmente imágenes clasificadas con la categoría **“Powdery Mildew”**, correspondiente a la cenicilla.

Figura 16. API utilizada para la descarga del dataset



```
1 import os
2 import requests
3
4 # URL de la API de GitHub para listar archivos de esa carpeta
5 # api_url = "https://api.github.com/repos/DevyanshMalhotra/MangoLeafDiseaseDetection/contents/dataset/Powdery_Mildew"
6 api_url = "https://api.github.com/repos/IsaacMwendwa/Omdena-Mango-Leaf-Disease-Detection/contents/src/data/MangoLeafBD_Without_Testset_Augmentation/Test/Powdary_Mildew"
7 api_url = "https://api.github.com/repos/IsaacMwendwa/Omdena-Mango-Leaf-Disease-Detection/contents/src/data/MangoLeafBD_Without_Testset_Augmentation/Test/Powdary_Mildew"
8
9 # Crear carpeta "data" si no existe
10 os.makedirs("data", exist_ok=True)
11
12 # Obtener lista de archivos desde GitHub
13 response = requests.get(api_url)
14 files = response.json()
15
16 for file in files:
17     if file["name"].lower().endswith((".jpg", ".jpeg", ".png")):
18         print(f"Descargando: {file['name']}...")
19
20         # Descargar imagen desde el enlace de descarga directa
21         img_data = requests.get(file["download_url"]).content
22
23         # Guardar la imagen en la carpeta
24         with open(os.path.join("data", file["name"]), "wb") as handler:
25             handler.write(img_data)
26
27 print("✅ Descarga Completa. Las imágenes están guardadas en la carpeta 'data'")
28
```

como segundo método para obtener las imágenes, se desarrolló el archivo descargar.py, cuyo objetivo fue facilitar y agilizar la descarga de un conjunto de datos disponible en Kaggle. El script utiliza la API oficial de Kaggle para descargar y descomprimir automáticamente el dataset público **“Mango Leaf Disease Dataset”**. Una vez descargado, el script revisa las carpetas del conjunto de datos y extrae las imágenes que pertenecen a la categoría **“Powdery Mildew”**, correspondiente a la enfermedad de cenicilla.

Las imágenes seleccionadas se almacenan en la carpeta local data/, donde posteriormente son utilizadas para realizar el proceso de etiquetado manual.

Además, el script evita que se sobrescriban imágenes cuando existen archivos con el mismo nombre, agregando automáticamente un número al nombre del archivo.

La creación de este script permitió reducir el tiempo empleado en la descarga y organización de las imágenes, ya que el proceso puede realizarse de forma automática sin tener que descargar y copiar manualmente cada archivo. De esta manera, las imágenes obtenidas desde Kaggle se incorporan al conjunto de datos utilizado posteriormente en el proyecto.

Figura 17. Dataset de Kaggle

```
1 import os
2 import shutil
3 from kaggle.api.kaggle_api_extended import KaggleApi
4
5 # Config
6 DATASET_REF = "aryashah2k/mango-leaf-disease-dataset"
7 DOWNLOAD_DIR = "kaggle_dataset"
8 TARGET_DIR = "data"
9 SEARCH_KEY = "Powdery Mildew" # busca carpetas/archivos que contengan esta palabra (case-insensitive)
10 IMG_EXT = (".jpg", ".jpeg", ".png")
11
12 os.makedirs(TARGET_DIR, exist_ok=True)
13 os.makedirs(DOWNLOAD_DIR, exist_ok=True)
14
15 # Autenticar y descargar + descomprimir el dataset
16 api = KaggleApi()
17 try:
18     api.authenticate()
19 except Exception as e:
20     raise RuntimeError(f"Kaggle authentication failed: {e}")
21
22 print("Descargando y descomprimiendo dataset desde Kaggle...")
23 try:
24     api.dataset_download_files(DATASET_REF, path=DOWNLOAD_DIR, unzip=True)
25 except Exception as e:
26     raise RuntimeError(f"Error descargando dataset: {e}")
27
28 # Buscar y copiar imágenes de las rutas que contengan SEARCH_KEY (case-insensitive)
29 copied = 0
30 search_key = SEARCH_KEY.lower()
31 for root, _, files in os.walk(DOWNLOAD_DIR):
32     if search_key in root.lower():
33         for f in files:
34             if f.lower().endswith(IMG_EXT):
35                 src = os.path.join(root, f)
36                 dst_name = f
37                 dst = os.path.join(TARGET_DIR, dst_name)
38                 # Evitar sobrescribir: si ya existe, añadir sufijo numérico
39                 if os.path.exists(dst):
40                     base, ext = os.path.splitext(dst_name)
41                     i = 1
42                     while os.path.exists(os.path.join(TARGET_DIR, f"{base}_{i}{ext}")):
43                         i += 1
44                     dst = os.path.join(TARGET_DIR, f"{base}_{i}{ext}")
45                 shutil.copy2(src, dst)
46                 copied += 1
47
48 # Si no encontró carpetas con SEARCH_KEY, copiar todas las imágenes como fallback
49 if copied == 0:
50     print(f"No se encontraron rutas que contengan '{SEARCH_KEY}'. Copiando todas las imágenes del dataset...")
51     for root, _, files in os.walk(DOWNLOAD_DIR):
52         for f in files:
53             if f.lower().endswith(IMG_EXT):
54                 src = os.path.join(root, f)
55                 dst_name = f
56                 dst = os.path.join(TARGET_DIR, dst_name)
57                 if os.path.exists(dst):
58                     base, ext = os.path.splitext(dst_name)
59                     i = 1
60                     while os.path.exists(os.path.join(TARGET_DIR, f"{base}_{i}{ext}")):
61                         i += 1
62                     dst = os.path.join(TARGET_DIR, f"{base}_{i}{ext}")
63                 shutil.copy2(src, dst)
64                 copied += 1
65
66 print(f"✅ Descarga y copia completadas. Imágenes guardadas en la carpeta '{TARGET_DIR}' ({copied} archivos).")
67
```

Construcción y gestión del conjunto de datos

Dado que las imágenes obtenidas de los diferentes repositorios no contaban con una clasificación del nivel de severidad, sino que únicamente indicaban la presencia de cenicilla, fue necesario realizar un proceso de etiquetado propio. Para esto se desarrolló el archivo etiquetador.py una aplicación de escritorio creada con Tkinter que permitió organizar y agilizar el proceso de etiquetado de las imágenes.

La herramienta muestra las imágenes almacenadas en la carpeta data/ y permite asignar a cada una de ellas el nivel de severidad correspondiente para facilitar el proceso se establecieron cinco clases de severidad cada una identificada mediante un color y una descripción. La clase puede seleccionarse haciendo clic en la opción correspondiente o utilizando las teclas del 1 al 5, después de lo cual la aplicación avanza automáticamente a la siguiente imagen.

Además, el programa cuenta con botones para desplazarse entre las imágenes mediante las flechas izquierda y derecha, una barra de progreso que permite conocer el porcentaje de imágenes que ya han sido etiquetadas y un panel de estadísticas que muestra en tiempo real la cantidad de imágenes asignadas a cada clase. De esta forma el etiquetado se realizó de manera más ordenada y se pudo llevar un control del avance y de la distribución de las imágenes entre las diferentes clases de severidad.

Figura 18. Construcción y gestión del conjunto de datos



Las etiquetas se persisten en `data/labels.json` como un diccionario `{nombre_de_archivo: clase}`. Durante el desarrollo del proyecto se migró la convención interna de clases de un rango 0–4 a un rango 1–5 (más intuitivo para el usuario final); para no perder el trabajo de etiquetado ya realizado, se implementó un script de migración independiente (`migrar_etiquetas.py`) que detecta automáticamente el formato antiguo, genera una copia de respaldo (`labels_backup.json`) y reescribe las etiquetas en el nuevo formato. La misma lógica de migración se incorporó también dentro de `etiquetador.py` para que la migración ocurra de forma transparente al cargar el archivo.

El conjunto de datos final quedó conformado por 500 imágenes etiquetadas y distribuidas en las cinco clases de severidad, dividido posteriormente en un 80 % para entrenamiento (400 imágenes) y un 20 % para prueba (100 imágenes) mediante muestreo estratificado, con el fin de preservar la proporción de clases en ambos subconjuntos.

Figura 19. Etiquetado por clases

```
1 import json
2 import os
3
4 def migrate_labels():
5     labels_file = "data/labels.json"
6
7     if not os.path.exists(labels_file):
8         print("❌ No se encontró el archivo data/labels.json")
9         return
10
11     # Cargar etiquetas
12     with open(labels_file, 'r') as f:
13         labels = json.load(f)
14
15     print(f"📄 Etiquetas encontradas: {len(labels)}")
16
17     # Verificar si necesita migración
18     needs_migration = False
19     for img_name, class_id in labels.items():
20         if class_id in [0, 1, 2, 3, 4]:
21             needs_migration = True
22             break
23
24     if not needs_migration:
25         print("✅ Las etiquetas ya están en formato 1-5")
26         return
27
28     # Hacer backup
29     backup_file = "data/labels_backup.json"
30     with open(backup_file, 'w') as f:
31         json.dump(labels, f, indent=2)
32     print(f"💾 Backup creado: {backup_file}")
33
34     # Migrar etiquetas
35     migrated = {}
36     for img_name, class_id in labels.items():
37         if class_id in [0, 1, 2, 3, 4]:
38             migrated[img_name] = class_id + 1
39         else:
40             migrated[img_name] = class_id
41
42     # Guardar etiquetas migradas
43     with open(labels_file, 'w') as f:
44         json.dump(migrated, f, indent=2)
45
46     print("✅ Etiquetas migradas exitosamente de 0-4 a 1-5")
47     print(f"📄 Guardado en: {labels_file}")
48
49     # Mostrar distribución
50     stats = {}
51     for class_id in migrated.values():
52         stats[class_id] = stats.get(class_id, 0) + 1
53
54     print("\n📊 Nueva distribución:")
55     for class_id in sorted(stats.keys()):
56         print(f"Clase {class_id}: {stats[class_id]} imágenes")
57
58 if __name__ == "__main__":
59     migrate_labels()
```

Preprocesamiento y aumento de datos

Cada uno de los tres modelos requiere un preprocesamiento distinto de la imagen de entrada, definido en su propio módulo (svm_model.py, cnn_model.py, transformer_model.py):

Para la SVM, en lugar de utilizar los píxeles en bruto, se diseñó una función `extract_features()` que reduce cada imagen (redimensionada a 100×100 píxeles) a un vector de 35 características numéricas, agrupadas en: estadísticas de color por canal RGB —media, desviación estándar, mediana y percentiles 25/75— (15 características); proporción de píxeles muy claros

("blancura", asociada al polvo característico de la cenicilla) y de píxeles de tono grisáceo (2); medidas de textura calculadas sobre la imagen en escala de grises —diferencias horizontales y verticales, varianza local y entropía del histograma— (4); contraste y magnitud de bordes mediante gradientes (2); relaciones entre canales R/G, R/B y G/B (3); un histograma de 8 contenedores de la imagen en escala de grises (8); y un índice de uniformidad de color (1) Para la CNN, las imágenes se redimensionan a 150×150 píxeles y se normalizan al rango $[0, 1]$. Durante el entrenamiento se aplica aumento de datos en tiempo real con ImageDataGenerator (rotaciones de hasta 20° , desplazamientos horizontales y verticales del 20 %, zoom del 20 % y volteo horizontal), con el objetivo de compensar parcialmente el tamaño reducido del conjunto de entrenamiento.

Para el Vision Transformer, las imágenes se redimensionan a 128×128 píxeles; el reescalado a $[0, 1]$ y la división en parches de 16×16 píxeles (lo que produce 64 parches por imagen) se realizan dentro del propio grafo del modelo mediante las capas personalizadas Patches y PatchEncoder, en lugar de hacerlo como un paso externo de preprocesamiento.

Figura 20. Preprocesamiento de imágenes

```
1 def extract_features(self, image_path):
2     try:
3         with Image.open(image_path) as img:
4             img = img.convert('RGB')
5             img = img.resize((100, 100))
6             img_array = np.array(img, dtype=np.float32)
7             features = []
8
9             # Separar canales RGB
10            r, g, b = img_array[:, :, 0], img_array[:, :, 1], img_array[:, :, 2]
11
12            # 1) COLOR RGB (15)
13            for channel in (r, g, b):
14                features.extend([
15                    np.mean(channel),
16                    np.std(channel),
17                    np.median(channel),
18                    np.percentile(channel, 25),
19                    np.percentile(channel, 75),
20                ])
21
22            # 2) MANCHAS (2)
23            whiteness = np.mean((r > 200) & (g > 200) & (b > 200))
24            grayness = np.mean(np.abs(r - g) + np.abs(g - b) + np.abs(r - b) < 30)
25            features.extend([whiteness, grayness])
26
27            # 3) TEXTURA (4)
28            gray = 0.299 * r + 0.587 * g + 0.114 * b
29            h_diff = np.mean(np.abs(np.diff(gray, axis=1)))
30            v_diff = np.mean(np.abs(np.diff(gray, axis=0)))
31            local_var = np.var(gray)
32            hist_entropy, _ = np.histogram(gray, bins=256, range=(0, 255))
33            hist_entropy = hist_entropy / (hist_entropy.sum() + 1e-7)
34            entropy = -np.sum(hist_entropy * np.log2(hist_entropy + 1e-7))
35            features.extend([h_diff, v_diff, local_var, entropy])
36
37            # 4) CONTRASTE Y BORDES (2)
38            contrast = np.std(gray)
39            edge_strength = np.mean(np.abs(np.gradient(gray)[0]) + np.abs(np.gradient(gray)[1]))
40            features.extend([contrast, edge_strength])
41
42            # 5) RELACIONES RGB (3)
43            rg_ratio = np.mean(r) / (np.mean(g) + 1)
44            rb_ratio = np.mean(r) / (np.mean(b) + 1)
45            gb_ratio = np.mean(g) / (np.mean(b) + 1)
46            features.extend([rg_ratio, rb_ratio, gb_ratio])
47
48            # 6) HISTOGRAMA GRIS 8 BINS (8)
49            hist_8, _ = np.histogram(gray, bins=8, range=(0, 255))
50            hist_8 = hist_8 / (hist_8.sum() + 1e-7)
51            features.extend(hist_8.tolist())
52
53            # 7) UNIFORMIDAD (1)
54            color_uniformity = 1 / (np.std(r) + np.std(g) + np.std(b) + 1)
55            features.append(color_uniformity)
56
57            result = np.array(features, dtype=np.float32)
58            if result.size != self.n_features:
59                print(f"⚠ Se obtuvieron {result.size} features, se esperaban {self.n_features}")
60                return np.zeros(self.n_features, dtype=np.float32)
61            return result
62
63    except Exception as e:
64        print(f"⚠ Error procesando {image_path}: {e}")
65        return np.zeros(self.n_features, dtype=np.float32)
```

Selección y configuración de la arquitectura

Se seleccionaron tres arquitecturas de complejidad creciente para poder comparar de forma controlada el efecto del tipo de modelo sobre el desempeño, dado el mismo conjunto de datos:

Máquina de Vectores de Soporte (SVM). Kernel RBF, $C = 10.0$, $\gamma = \text{"scale"}$, $\text{class_weight} = \text{"balanced"}$ (para compensar el posible desbalance entre clases) y $\text{probability} = \text{True}$ para obtener estimaciones de confianza; las características se normalizan previamente con un StandardScaler

Red Neuronal Convolutiva (CNN). Arquitectura inspirada en VGG, con tres bloques convolucionales de 32, 64 y 128 filtros respectivamente (cada bloque con dos capas Conv2D 3×3, BatchNormalization, MaxPooling2D y Dropout creciente 0.25–0.3), seguidos de una capa densa de 512 neuronas con BatchNormalization y Dropout 0.5, y una capa de salida softmax de 5 clases. Se entrena con el optimizador Adam (tasa de aprendizaje 0.0001) y pérdida `sparse_categorical_crossentropy`.

Figura 21. Configuración de modelos



```
1 def create_model_architecture(self):
2     """Arquitectura CNN mejorada (inspirada en VGG)."""
3     model = keras.Sequential([
4         layers.Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(150,150,3)),
5         layers.BatchNormalization(),
6         layers.Conv2D(32, (3,3), activation='relu', padding='same'),
7         layers.BatchNormalization(),
8         layers.MaxPooling2D(2,2),
9         layers.Dropout(0.25),
10
11         layers.Conv2D(64, (3,3), activation='relu', padding='same'),
12         layers.BatchNormalization(),
13         layers.Conv2D(64, (3,3), activation='relu', padding='same'),
14         layers.BatchNormalization(),
15         layers.MaxPooling2D(2,2),
16         layers.Dropout(0.25),
17
18         layers.Conv2D(128, (3,3), activation='relu', padding='same'),
19         layers.BatchNormalization(),
20         layers.Conv2D(128, (3,3), activation='relu', padding='same'),
21         layers.BatchNormalization(),
22         layers.MaxPooling2D(2,2),
23         layers.Dropout(0.3),
24
25         layers.Flatten(),
26         layers.Dense(512, activation='relu'),
27         layers.BatchNormalization(),
28         layers.Dropout(0.5),
29         layers.Dense(5, activation='softmax')
30     ])
31     model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),
32                 loss='sparse_categorical_crossentropy',
33                 metrics=['accuracy'])
34     return model
```

Vision Transformer (ViT). Divide cada imagen en 64 parches de 16×16 píxeles, los proyecta a un espacio de 64 dimensiones mediante una capa densa y les añade una codificación posicional aprendida (PatchEncoder). La secuencia de

parches se procesa con 4 bloques Transformer, cada uno con una capa de MultiHeadAttention de 4 cabezas, normalización por capas (pre-norm) y un MLP interno con activación GELU; finalmente se aplica GlobalAveragePooling1D y una cabeza de clasificación densa ($128 \rightarrow 64 \rightarrow 5$) con activación softmax. Se entrena con Adam (tasa de aprendizaje 0.001).

Entrenamiento y ajuste fino del modelo

Los tres modelos se entrenan sobre la misma partición estratificada 80/20 (400 imágenes de entrenamiento, 100 de prueba), pero con estrategias de ajuste adaptadas a cada arquitectura:

La SVM se entrena en una sola pasada sobre las características ya escaladas; al finalizar, el modelo (`svm_model.pkl`) y el escalador (`svm_scaler.pkl`) se serializan con `joblib`, junto con un archivo de metadatos (`svm_metadata.json`) que registra el número de muestras, la exactitud de prueba y la lista de imágenes válidas utilizadas.

La CNN y el ViT se entrenan de forma iterativa por épocas, con dos callbacks de Keras para evitar el sobreajuste y estabilizar la convergencia: `EarlyStopping` (que detiene el entrenamiento y restaura los mejores pesos si la pérdida de validación deja de mejorar durante 10 épocas en la CNN y 15 en el ViT) y `ReduceLROnPlateau` (que reduce la tasa de aprendizaje a la mitad si la pérdida de validación se estanca durante 5 épocas en la CNN y 7 en el ViT). La CNN se entrena hasta un máximo de 50 épocas con lotes de 16 imágenes; el ViT, hasta 80 épocas con el mismo tamaño de lote.

Los modelos entrenados se guardan en formato HDF5 (`cnn_model.h5`, `transformer_model.h5`, con una copia adicional `transformer_best.h5` para el ViT) junto con archivos de metadatos JSON. Adicionalmente, se creó un script independiente, `entrenar_transformer.py`, que permite reentrenar el Vision Transformer de forma manual y controlada desde la línea de comandos, sin necesidad de abrir la aplicación completa, verificando primero que existan al menos 20 imágenes etiquetadas (el mínimo definido para esta arquitectura, superior a las 10 requeridas por la SVM y la CNN, dado que el ViT necesita más datos para converger).

SVM

```
def train_with_real_images(self, labels=None):
    """Entrenar el SVM con imágenes reales (externo 1-5 + interno 0-4)."""
    if labels is None:
        labels = self.load_labels()
    if not labels or len(labels) < 10:
        print("❌ Necesitas al menos 10 imágenes etiquetadas")
        print("❗ Ejecuta: python etiquetador.py")
        self.is_trained = False
        return False
    print(f"\n{'-'*60}")
    print("📁 Entrenando MODELO SVM CON IMÁGENES REALES")
    print(f"{'-'*60}\n")
    X, y, valid_images, failed_images = [], [], [], []
    print(f"🔄 Procesando {len(labels)} imágenes...")
    for idx, (image_name, class_id) in enumerate(labels.items(), 1):
        try:
            class_id = int(class_id) # por si viniera como texto
        except Exception:
            continue
        image_path = os.path.join("data", image_name)
        if not os.path.exists(image_path):
            failed_images.append((image_name, "Archivo no encontrado"))
            continue
        feats = self.extract_features(image_path)
        if feats is not None and not np.all(feats == 0):
            X.append(feats)
            y.append(class_id - 1) # 1-5 -> 0-4
            valid_images.append(image_name)
        else:
            failed_images.append((image_name, "Error extrayendo características"))
    if idx % 50 == 0:
        print(f"🔄 Procesadas: {idx}/{len(labels)}")
    if len(X) < 10:
        print("❌ ERROR: Solo se procesaron {len(X)} imágenes válidas")
        if failed_images:
            print("⚠️ Imágenes con problemas (primeras 5):")
            for img, reason in failed_images[:5]:
                print(f"❌ {img}: {reason}")
        self.is_trained = False
        return False
    X, y = np.array(X), np.array(y)
    unique, counts = np.unique(y, return_counts=True)
    print(f"\n📊 Distribución de clases (externo 1-5):")
    for c, cnt in zip(unique, counts):
        print(f"  {self.class_names[c+1]}: {cnt} ({cnt/len(y)*100:.1f}%)")
    # División estratificada si es posible
    try:
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42, stratify=y
        )
    except Exception:
        print("⚠️ División sin estratificación (clases muy desbalanceadas)")
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )
    # Escalado
    self.scaler.fit(X_train)
    X_train_scaled = self.scaler.transform(X_train)
    X_test_scaled = self.scaler.transform(X_test)
    # Entrenar SVM
    print(f"\n🔧 Entrenando SVM (RBF, C=10.0, balanced)...")
    self.model = svm.SVC(
        kernel="rbf",
        C=10.0,
        gamma="scale",
        probability=True,
        random_state=42,
        class_weight="balanced",
    )
    self.model.fit(X_train_scaled, y_train)
    # Evaluación
    y_pred_test = self.model.predict(X_test_scaled)
    test_acc = accuracy_score(y_test, y_pred_test)
    print(f"\n📊 Precisión en prueba: (test_acc:.2%)")
    print(f"\n📋 Reporte de Clasificación:")
    print(classification_report(
        y_test, y_pred_test,
        target_names=[self.class_names[i+1] for i in sorted(np.unique(y))],
        zero_division=0
    ))
    print("📊 Matriz de Confusión:")
    print(confusion_matrix(y_test, y_pred_test))
    # Guardar modelo + scaler + metadata
    os.makedirs("modelos", exist_ok=True)
    joblib.dump(self.model, "modelos/svm_model.pkl")
    joblib.dump(self.scaler, "modelos/svm_scaler.pkl")
    metadata = {
        "n_samples": int(len(X)),
        "n_features": int(X.shape[1]),
        "classes": (unique + 1).tolist(), # externo 1-5
        "test_accuracy": float(test_acc),
        "n_train_samples": int(len(X_train)),
        "n_test_samples": int(len(X_test)),
        "valid_images": valid_images[:100],
    }
    with open("modelos/svm_metadata.json", "w", encoding="utf-8") as f:
        json.dump(metadata, f, indent=2, ensure_ascii=False)
    print(f"\n📁 Modelo y metadata guardados en carpeta 'modelos/'.")
    self.is_trained = True
    print(f"{'-'*60}\n")
    return True
```

CNN

```
1 def create_and_train_model(self, labels):
2     if not labels or len(labels) < 10:
3         print("❌ CNN: Necesitas al menos 10 imágenes etiquetadas.")
4         return False
5
6     X, y = self.load_images_from_labels(labels)
7     if len(X) < 10:
8         print("❌ CNN: No hay suficientes imágenes válidas.")
9         return False
10
11     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, stratify=y)
12     datagen = ImageDataGenerator(rotation_range=20,
13                                 width_shift_range=0.2,
14                                 height_shift_range=0.2,
15                                 zoom_range=0.2,
16                                 horizontal_flip=True)
17     print("👉 Construyendo CNN...")
18     self.model = self.create_model_architecture()
19
20     cb = [
21         callbacks.EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True, verbose=1),
22         callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=5, verbose=1)
23     ]
24     print("\n👉 Entrenando modelo CNN...")
25     history = self.model.fit(datagen.flow(X_train, y_train, batch_size=10),
26                             validation_data=(X_val, y_val),
27                             epochs=50,
28                             callbacks=cb,
29                             verbose=1)
30     print("\n👉 Evaluando modelo...")
31     val_loss, val_acc = self.model.evaluate(X_val, y_val, verbose=0)
32     print(f"✅ Precisión validación: (val_acc: {val_acc:.2%})")
33
34     os.makedirs("modelos", exist_ok=True)
35     self.model.save("modelos/cnn_model.h5")
36     metadata = {
37         "samples": len(X),
38         "train_split": len(X_train),
39         "val_split": len(X_val),
40         "val_accuracy": float(val_acc),
41         "classes": [1, 2, 3, 4, 5]
42     }
43
44     with open("modelos/cnn_metadata.json", "w") as f:
45         json.dump(metadata, f, indent=2)
46     print("📁 Modelo y metadata guardados.")
47     self.is_trained = True
48     return True
```

Transformer

```
1 def create_and_train_model(self, labels):
2     """Precondición de modelo: Transformer"""
3     if not labels or len(labels) < 20:
4         print("❌ Transformer: Necesitas al menos 20 imágenes etiquetadas.")
5         return False
6
7     X, y = self.load_images_from_labels(labels)
8     if len(X) < 20:
9         print("❌ Transformer: No hay suficientes imágenes válidas.")
10        return False
11
12    X_train, X_val, y_train, y_val = train_test_split(
13        X, y, test_size=0.2, random_state=42, stratify=y
14    )
15
16    print("\n🟡 Construyendo Vision Transformer...")
17    self.model = self.create_vit()
18
19    cb = [
20        callbacks.EarlyStopping(
21            monitor="val_loss",
22            patience=15,
23            restore_best_weights=True,
24            verbose=1
25        ),
26        callbacks.ReduceLROnPlateau(
27            monitor="val_loss",
28            factor=0.5,
29            patience=7,
30            verbose=1
31        )
32    ]
33
34    print("\n🟢 Entrenando Vision Transformer...")
35    self.model.fit(
36        X_train, y_train,
37        validation_data=(X_val, y_val),
38        epochs=80,
39        batch_size=16,
40        callbacks=cb,
41        verbose=1
42    )
43
44    print("\n🟡 Evaluando modelo...")
45    val_loss, val_acc = self.model.evaluate(X_val, y_val, verbose=0)
46    print(f"🟢 Precisión validación: {val_acc:.2%}")
47
48    os.makedirs("modelos", exist_ok=True)
49    self.model.save("modelos/transformer_model.h5")
50
51    metadata = {
52        "samples": len(X),
53        "train_split": len(X_train),
54        "val_split": len(X_val),
55        "val_accuracy": float(val_acc),
56        "classes": [1, 2, 3, 4, 5]
57    }
58    with open("modelos/transformer_metadata.json", "w") as f:
59        json.dump(metadata, f, indent=2)
60
61    print("🟢 Modelo y metadata guardados.")
62    self.is_trained = True
63    return True
64
```

Evaluación y validación del desempeño

La evaluación comparativa de los tres modelos se centralizó en un módulo dedicado, `metricas.py`, cuya clase `MetricsEvaluator` calcula, a partir de las predicciones sobre el conjunto de prueba: exactitud, precisión, recall y F1-score (ponderados y por clase), matriz de confusión de 5×5, distribución de confianza de las predicciones y, en su versión extendida, el coeficiente Kappa de Cohen y un desglose de verdaderos/falsos positivos y negativos por clase.

Los resultados de exactitud en el conjunto de prueba, registrados en los archivos de metadatos de cada modelo, fueron los siguientes:

Tabla 6. Exactitud del modelo

Modelo	Exactitud (test)
SVM (35 características)	49 %
Vision Transformer	42 %
CNN (VGG-style)	31 %

Para el modelo SVM, el análisis de confianza sobre las 500 imágenes mostró una confianza promedio de predicción de 0.59 (desviación estándar de 0.13), con solo un 4.8 % de predicciones de alta confianza (> 0.8), un 47.2 % de confianza media y un 48 % de confianza baja (< 0.6), lo que confirma que, si bien la SVM obtuvo la mejor exactitud relativa, el sistema en su conjunto opera todavía en un régimen de incertidumbre considerable.

El resultado más relevante desde el punto de vista metodológico es que el modelo más simple (SVM sobre características diseñadas manualmente) superó a las dos arquitecturas de aprendizaje profundo. Esto se explica principalmente por el tamaño reducido del conjunto de datos: 400 imágenes de entrenamiento resultan insuficientes para que una CNN o un Vision Transformer entrenados desde cero (sin pesos preentrenados) aprendan representaciones generalizables, mientras que un modelo de kernel como la SVM, apoyado en características estadísticas y de textura diseñadas específicamente para resaltar el patrón visual de la cenicilla (blancura, uniformidad de color, textura), es más eficiente en régimen de pocos datos. El ViT, al depender de mecanismos de atención que típicamente requieren grandes volúmenes de datos o preentrenamiento previo, y la CNN, más propensa al sobreajuste con pocas muestras por clase, quedaron por debajo de la SVM en este escenario particular.

Despliegue e integración en sistemas agrícolas

El sistema se integró finalmente en una aplicación de escritorio (main.py) construida con Tkinter, pensada para que un técnico agrícola pueda utilizarla sin conocimientos de programación ni conexión a internet, una vez que los modelos ya han sido entrenados y sus archivos (svm_model.pkl, svm_scaler.pkl, cnn_model.h5, transformer_model.h5) se distribuyen junto con la aplicación.

La aplicación ofrece las siguientes funciones principales:

- **Analizar Dataset:** ejecuta los tres modelos sobre todas las imágenes de la carpeta data/ en un hilo separado (mediante threading), evitando que la interfaz se congele durante el análisis, y muestra los resultados en pestañas independientes por modelo (ttk.Notebook).
- **Cargar Imagen:** permite analizar una única fotografía tomada en campo, mostrando una vista previa junto con sus metadatos (dimensiones, tamaño en disco).
- **Configurar Referencias:** permite asociar, para cada una de las cinco clases de severidad, una imagen de referencia (referencias/clase_N.jpg) que sirve de guía visual al usuario para interpretar el resultado del modelo.
- **Resultados detallados:** por cada modelo, una tabla (ttk.Treeview) lista, para cada imagen, la clase predicha, el nivel de severidad, el porcentaje de confianza y un estado cualitativo (Alta/Media/Baja confianza), además de la matriz de confusión y las métricas de evaluación calculadas con metrics.py.
- **Comparativa entre modelos y exportación:** una vista final compara exactitud, precisión, recall, F1-score y confianza promedio de los tres modelos lado a lado, genera una recomendación textual automática y permite exportar todos los resultados a un archivo de texto plano.

Este enfoque de despliegue responde a una limitación práctica frecuente en el contexto agrícola ecuatoriano: la conectividad a internet en zonas rurales

de producción de mango no siempre es confiable, por lo que un sistema que funcione completamente en modo local, con modelos ya entrenados y empaquetados, resulta más adecuado para su adopción en campo que una solución dependiente de una API externa.

Conclusiones

Se desarrolló, de manera íntegra en Python, un sistema completo para la detección y clasificación de severidad de la cenicilla en hojas de mango en cinco niveles, que cubre todo el ciclo de un proyecto de visión por computador aplicado: adquisición de datos desde fuentes públicas, etiquetado manual mediante una herramienta propia, extracción de características y preprocesamiento diferenciado por arquitectura, entrenamiento y ajuste fino de tres modelos de clasificación, evaluación comparativa con métricas estándar y despliegue en una aplicación de escritorio funcional.

De la comparación entre los tres modelos se concluye que, en un escenario de datos limitados (500 imágenes), un modelo clásico como la SVM sobre características diseñadas manualmente (49 % de exactitud) puede superar a arquitecturas de aprendizaje profundo entrenadas desde cero, como la CNN (31 %) y el Vision Transformer (42 %). Este hallazgo es relevante para proyectos agrícolas académicos o institucionales que, como el presente, no siempre cuentan con conjuntos de datos de gran escala.

Referencias bibliográficas

- Aldakheel, E. A., Zakariah, M., & Alabdall, A. H. (2024). Detection and identification of plant leaf diseases using YOLOv4. *Front Plant Sci*, 15, 1355941. <https://doi.org/10.3389/fpls.2024.1355941>
- Athalye, C., & Arnaout, R. (2023). Domain-guided data augmentation for deep learning on medical imaging. *PLOS ONE*, 18(3), e0282532. <https://doi.org/10.1371/JOURNAL.PONE.0282532>
- Bhujade, V. G., Sambhe, V., & Banerjee, B. (2024). Digital image noise removal towards soybean and cotton plant disease using image processing filters. *Expert Systems with Applications*, 246, 123031. <https://doi.org/10.1016/J.ESWA.2023.123031>
- Cen, X., Lu, S., & Qian, T. (2025). YOLO-LCE: A Lightweight YOLOv8 Model for Agricultural Pest Detection. *Agronomy*, 15(9). <https://doi.org/10.3390/agronomy15092022>
- Chen, Y., Wang, A., Liu, Z., Yue, J., Zhang, E., Li, F., & Zhang, N. (2025). MoSViT: a lightweight vision transformer framework for efficient disease detection via precision attention mechanism. *Frontiers in Artificial Intelligence*, 8, 1498025. <https://doi.org/10.3389/FRAI.2025.1498025/BIBTEX>
- De Toro-Castro, A., Lupión, M., González-Ruiz, V., Sanjuan, J. F., & M. Ortigosa, P. (2026). A parallel framework for data input pipelines and online data augmentation in deep learning. *The Journal of Supercomputing* 2026 82:7, 82(7), 389-. <https://doi.org/10.1007/S11227-026-08543-0>
- Ding, W., Abdel-Basset, M., Alrashdi, I., & Hawash, H. (2024). Next generation of computer vision for plant disease monitoring in precision agriculture: A contemporary survey, taxonomy, experiments, and future direction. *Information Sciences*, 665, 120338. <https://doi.org/10.1016/J.INS.2024.120338>

- GitHub. (2019). *GitHub - xpwu95/IP102: IP102: A Large-Scale Benchmark Dataset for Insect Pest Recognition* · GitHub. <https://github.com/xpwu95/IP102>
- Haider, I., Nazir, M., Khan, S. A., 4*, S. A., Ramzan, M., & Habib, M. I. (2025). *Enhanced wheat crop leaf disease classification using multi-level contrast enhancement and modified vision transformers*. <https://doi.org/10.1038/s41598-025-24149-7>
- Hakim, A., Srivastava, A. K., Hamza, A., Owais, M., Habib-ur-Rahman, M., Qadri, S., Qayyum, M. A., Ahmad Khan, F. Z., Mahmood, M. T., & Gaiser, T. (2025). Yolo-pest: an optimized YoloV8x for detection of small insect pests using smart traps. *Scientific Reports* 2025 15:1, 15(1), 14029-. <https://doi.org/10.1038/s41598-025-97825-3>
- Han, F., Guan, X., & Xu, M. (2024). Method of intelligent agricultural pest image recognition based on machine vision algorithm. *Discover Applied Sciences* 2024 6:10, 6(10), 536-. <https://doi.org/10.1007/S42452-024-06224-2>
- Hong, W., Ling, S., Zhu, P., Wang, Z., Zhao, R., Liu, Y., & Dong, M. (2026). Lightweight Vision-Transformer Network for Early Insect Pest Identification in Greenhouse Agricultural Environments. *Insects* 2026, Vol. 17, Page 74, 17(1), 74. <https://doi.org/10.3390/INSECTS17010074>
- Huang, Y., Liu, Z., Zhao, H., Tang, C., Liu, B., Li, Z., Wan, F., Qian, W., & Qiao, X. (2025). YOLO-YSTs: An Improved YOLOv10n-Based Method for Real-Time Field Pest Detection. *Agronomy* 2025, Vol. 15, Page 575, 15(3), 575. <https://doi.org/10.3390/AGRONOMY15030575>
- Hussain, A., Ullah, K., Afaq, M., Munsif, M., Hussain, A., & Baik, S. W. (2026). Quality over quantity: a data-centric survey of annotation errors in object detection datasets. *Artificial Intelligence Review* 2026 59:3, 59(3), 107-. <https://doi.org/10.1007/S10462-026-11502-Z>

- Jia, S., Rezaei, M. H., Asli, B. H. S., Jia, S., Rezaei, M. H., & Asli, B. H. S. (2025). Enhancing Pest Detection in Deep Learning Through a Systematic Image Quality Assessment and Preprocessing Framework. *Journal of Experimental and Theoretical Analyses* 2025, Vol. 3, 3(4), 39. <https://doi.org/10.3390/JETA3040039>
- Kern, C., Eckman, S., Beck, J., Chew, R., Ma, B., & Kreuter, F. (2023). Annotation Sensitivity: Training Data Collection Methods Affect Model Performance. *Findings of the Association for Computational Linguistics: EMNLP* 2023, 14874–14886. <https://doi.org/10.18653/V1/2023.FINDINGS-EMNLP.992>
- Khan, A. T., Jensen, S. M., & Khan, N. (2025). Advancing Food Security through Precision Agriculture: YOLOv8's Role in Efficient Pest Detection and Management. *EAI Endorsed Transactions on AI and Robotics*, 4. <https://doi.org/10.4108/AIRO.8049>
- Kim, G., Kim, J., Hwang, S., & Kim, H. S. (2026). Scale-Aware Mosaic Augmentation and GSIoU-Based Varifocal Loss for Robust Object Detection Under Scale Imbalance. *Electronics* 2026, Vol. 15, Page 1075, 15(5), 1075. <https://doi.org/10.3390/ELECTRONICS15051075>
- Kodors, S., Apeinans, I., Vancans, I., Bartulsons, T., & Zarembo, I. (2025). *EARLY DETECTION OF CATERPILLARS USING ARTIFICIAL INTELLIGENCE*. <https://doi.org/10.22616/ERDev.2025.24.TF112>
- Leal-Lara, D.-D., Barón-Velandia, J., & Rocha-Calderón, C.-E. (2021). Interpretabilidad en el campo de la detección de enfermedades en las plantas: Una revisión. *Revista Facultad de Ingeniería*, 30(58), e13495. <https://doi.org/10.19053/01211129.V30.N58.2021.13495>
- Li, X., Zhang, J., Sun, Y., & Huang, H. (2024). MATEacher: Multi-scale Views Learning and Adaptive Unsupervised Weight for Semi-supervised Pest Region Detection. *2024 6th International Conference on Robotics, Intelligent Control and Artificial*

Intelligence, RICA 2024, 698–703.
<https://doi.org/10.1109/RICA164321.2024.10911445>

Lu, Y., Chen, D., Olaniyi, E., & Huang, Y. (2022). Generative adversarial networks (GANs) for image augmentation in agriculture: A systematic review. *Computers and Electronics in Agriculture*, 200, 107208. <https://doi.org/10.1016/J.COMPAG.2022.107208>

May, P. (2019). *Improved Image Augmentation for Convolutional Neural Networks by Copyout and CopyPairing*. <https://github.com/anonym/anonym>.

Montoya, C., González, R., García, S., & Ardila, L. (2023). *Identification of diseases and/or pests in fruit trees through image processing techniques and artificial intelligence*. <https://revistas.udistrital.edu.co/index.php/visele/article/view/22018>

Mumuni, A., & Mumuni, F. (2022). Data augmentation: A comprehensive survey of modern approaches. En *Array* (Vol. 16). Elsevier B.V. <https://doi.org/10.1016/j.array.2022.100258>

Naseer, M., Bharath, ., Prabakaran, S., Hasan, O., Shafique, . Muhammad, Cohen, D. D., Garcia, J., Ghavamzadeh, M., Petrik, M., & Thomas, P. S. (2023). UnbiasedNets: a dataset diversification framework for robustness bias alleviation in neural networks. *Machine Learning* 2023 113:5, 113(5), 2499–2526. <https://doi.org/10.1007/S10994-023-06314-Z>

Sahu, K., Tiwari, S., Kumar Singh, M., Pahareeya, J., Kumar Shakya, H., Kumar, G., & Selvarajan, S. (2025). *Plant disease detection using a hybrid dilated CNN with attention mechanisms and optimized mask RCNN segmentation* *Scientific Reports*. <https://doi.org/10.1038/s41598-025-26192-w>

Serfa Juan, R. O., Pordesimo, L. O., Campbell, J. F., Armstrong, P. R., & Gerken, A. R. (2025). AI-Based Image Profiling and Detection for the Beetle Byte Quintet Using Vision Transformer (ViT) in Advanced

Stored Product Infestation Monitoring. *Entomologia Experimentalis et Applicata*, 173(12), 1285–1305. <https://doi.org/10.1111/EEA.70018>

Sharma, J. (2024). Efficiency and Scalability of EfficientNetV2-L for Large-Scale Pest Classification. *2024 5th IEEE Global Conference for Advancement in Technology, GCAT 2024*. <https://doi.org/10.1109/GCAT62922.2024.10924022>

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* 2019 6:1, 6(1), 60-. <https://doi.org/10.1186/S40537-019-0197-0>

Song, X., Yan, L., Liu, S., Gao, T., Han, L., Jiang, X., Jin, H., & Zhu, Y. (2025). Agricultural Image Processing: Challenges, Advances, and Future Trends. *Applied Sciences* 2025, Vol. 15, Page 9206, 15(16), 9206. <https://doi.org/10.3390/APP15169206>

Subburaj, S. D. R., Eswaramoorthy, C., Latha, V. G., & Chinnasamy, R. K. P. (2025). Efficient Pest Detection Through Advanced Machine Learning Technique. *Current Agriculture Research Journal*, 12(3), 1127–1134. <https://doi.org/10.12944/CARJ.12.3.08>

Upadhyay, A., Chandel, N. S., Singh, K. P., Chakraborty, S. K., Nandede, B. M., Kumar, M., Subeesh, A., Upendar, K., Salem, A., & Elbeltagi, A. (2025). Deep learning and computer vision in plant disease detection: a comprehensive review of techniques, models, and trends in precision agriculture. *Artificial Intelligence Review* 2025 58:3, 58(3), 92-. <https://doi.org/10.1007/S10462-024-11100-X>

Utku, A., Kaya, M., & Canbay, Y. (2025a). A New Hybrid ConvViT Model for Dangerous Farm Insect Detection. *Applied Sciences (Switzerland)*, 15(5). <https://doi.org/10.3390/app15052518>

Utku, A., Kaya, M., & Canbay, Y. (2025b). A New Hybrid ConvViT Model for Dangerous Farm Insect Detection. *Applied Sciences* 2025, Vol. 15, Page 2518, 15(5), 2518. <https://doi.org/10.3390/APP15052518>

- Venkateswara, S. M., & Padmanabhan, J. (2025a). Deep learning based agricultural pest monitoring and classification. *Scientific Reports* 2025 15:1, 15(1), 8684-. <https://doi.org/10.1038/s41598-025-92659-5>
- Wang, J., & Perez, L. (2017). *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*.
- Wang, Q. J., Zhang, S. Y., Dong, S. F., Zhang, G. C., Yang, J., Li, R., & Wang, H. Q. (2020). Pest24: A large-scale very small object data set of agricultural pests for multi-target detection. *Computers and Electronics in Agriculture*, 175, 105585. <https://doi.org/10.1016/J.COMPAG.2020.105585>
- Wang, X., & Liu, J. (2024). Vegetable disease detection using an improved YOLOv8 algorithm in the greenhouse plant environment. *Scientific Reports* 14, 4261. <https://doi.org/10.1038/s41598-024-54540-9>
- Wang, X., Liu, J., & Chen, Q. (2025). An advanced deep learning method for pepper diseases and pests detection. *Plant Methods*, 21(1), 70. <https://doi.org/10.1186/S13007-025-01387-4>
- Wei, T., Chen, Z., Yu, X., Chapman, S., Melloy, P., & Huang, Z. (2024). *PlantSeg: A Large-Scale In-the-wild Dataset for Plant Disease Segmentation*.
- Yun, S., Han, D., Chun, S., Oh, S. J., Choe, J., & Yoo, Y. (2019). CutMix: Regularization strategy to train strong classifiers with localizable features. *Proceedings of the IEEE International Conference on Computer Vision*, 6022–6031. <https://doi.org/10.1109/ICCV.2019.00612>
- Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). *mixup: Beyond Empirical Risk Minimization*. <https://github.com/facebookresearch/mixup-cifar10>.

- Zheng, Y., Zheng, W., & Du, X. (2024). A lightweight rice pest detection algorithm based on improved YOLOv8. *Scientific Reports* 2024 14:1, 14(1), 29888-. <https://doi.org/10.1038/s41598-024-81587-5>
- Zhong, Z., Zheng, L., Kang, G., Li, S., & Yang, Y. (2017). *Random Erasing Data Augmentation*. <https://github.com/zhunzhong07/Random-Erasing>.
- Zou, X., Tang, Y., Gao, J., Gong, L., Van Mourik, S., Xiong, Y., Lu, Y., Liu, P., & Tan, C. (2025). MA-YOLO: A Pest Target Detection Algorithm with Multi-Scale Fusion and Attention Mechanism. *Agronomy* 2025, Vol. 15, Page 1549, 15(7), 1549. <https://doi.org/10.3390/AGRONOMY15071549>

**Nuvia Aracelly Beltrán Robayo**

Instituto de Investigación, Milagro, Universidad Agraria del Ecuador
Computación, Milagro, Universidad Agraria del Ecuador
nbeltran@uagraría.edu.ec

ORCID: <https://orcid.org/0000-0002-3335-576X>

Nuvia Beltrán Robayo es una ingeniera ecuatoriana cuya trayectoria ha estado marcada por un profundo compromiso con la innovación tecnológica y la educación superior como herramientas de transformación social. Desde el inicio de su carrera profesional ha mostrado un interés constante por vincular la tecnología con el desarrollo productivo y académico del país. Cuenta con una sólida formación académica: es Ingeniera en Computación e Informática, Magister en Auditoría de Tecnologías de la Información y Magister en Big Data y Ciencias de Datos. Actualmente cursa un doctorado en Informática, lo que refleja su vocación por la excelencia académica y la actualización permanente en las ciencias de la computación y el análisis de datos.

Darwin Roberto Pow Chon Long Vásquez

Computación, Milagro, Universidad Agraria del Ecuador
dpow@uagraría.edu.ec

ORCID: <https://orcid.org/0009-0000-7860-2106>

Darwin Roberto Pow Chon Long Vásquez es un ingeniero ecuatoriano nacido en Quevedo, provincia de Los Ríos, cuya trayectoria ha estado marcada por un profundo compromiso con el desarrollo del sector agropecuario y la gestión estratégica como herramientas para el fortalecimiento productivo del país. Su formación combina las ciencias agronómicas con la visión empresarial, orientando su labor hacia la investigación aplicada y la vinculación entre la producción agrícola y la administración eficiente de recursos.

Cuenta con una sólida formación académica: es Ingeniero Agrónomo y Magister en Administración de Empresas, lo que refleja su vocación por integrar el conocimiento técnico del campo agropecuario con herramientas de gestión y toma de decisiones estratégicas.

Su trayectoria profesional inició con un año de experiencia en Reybanpac, donde se introdujo en la dinámica operativa del sector agroindustrial. Posteriormente se desempeñó en la Hacienda Zulema, primero como Supervisor de Proceso y luego como Administrador, consolidando su experiencia en la gestión integral de fincas y procesos productivos agropecuarios. Su trayectoria continuó en Agripac, empresa en la que laboró durante 10 años, período en el que profundizó su experiencia en el sector agroindustrial ecuatoriano. Más adelante se vinculó a la International Biotechnology Organization (IBO), donde trabajó durante 2 años ampliando su visión hacia la biotecnología aplicada al agro. Desde hace 8 años forma parte de la Universidad Agraria del Ecuador, institución en la que continúa desempeñándose actualmente, combinando el ejercicio profesional con la investigación académica y la docencia.

**Mario Manuel Cárdenas Rodríguez**

Computación, Milagro, Universidad Agraria del Ecuador
mcardenas@uagraría.edu.ec

ORCID: <https://orcid.org/0000-0002-1824-8474>

Mario Cárdenas Rodríguez es un docente universitario ecuatoriano con más de 30 años de trayectoria en la enseñanza superior. Cuenta con una sólida formación académica que incluye un Máster en Docencia Superior Universitaria y un Máster en Tecnologías de la Información con mención en Transformación Digital e Innovación. Complementa este perfil con un Diplomado en Investigación Educativa y un Diplomado en Derecho Digital y Legaltech, además de su formación de base como Ingeniero en Informática y Tecnólogo en Programación de Sistemas. A lo largo de su extensa carrera docente, ha combinado su experiencia en el aula con un interés constante por la actualización tecnológica y jurídica aplicada al entorno digital, reflejando un perfil profesional que integra la pedagogía universitaria con las nuevas tendencias en tecnología, innovación y transformación digital.

**Jhonny Joffre Choez Burgos**

Computación, Guayaquil, Universidad Agraria del Ecuador
jchoez@uagraría.edu.ec

ORCID: <https://orcid.org/0000-0003-3001-4561>

Jhonny Joffre Choez Burgos es Ingeniero en Computación e Informática y Magister en Seguridad de la Información. Con esta formación se ha especializado en el diseño, implementación y monitoreo de sistemas de seguridad, así como en la administración de bases de datos y sistemas empresariales.

Su vínculo con la Universidad Agraria del Ecuador se ha dado en dos etapas: entre 2018 y 2019 como asistente técnico de investigación, en el proyecto "Modelo Cloud Computing y establecimiento de metodologías para almacenamiento de información"; y desde 2025 como docente ocasional, a cargo de las asignaturas de Robótica, Emprendimiento Empresarial, Simulación de Circuitos, Sistemas Operativos, Lenguaje de Programación I, Agromática I y Titulación I.

En el ámbito profesional, se desempeñó como Jefe de Sistemas del grupo empresarial Proceplas-Diferrec-Reyplas entre 2020 y 2025, a cargo de la administración de bases de datos, el desarrollo de sistemas y el monitoreo de cámaras y equipos móviles; actualmente presta servicios de asesoría tecnológica a esa misma organización. Previamente trabajó en Cendcap S.A. (2020-2022) como jefe de sistemas y docente interno, encargado del circuito cerrado de seguridad y la capacitación en informática básica; en el Instituto Tecnológico Bolivariano (2017-2018) como docente de Excel; y en Atiempo S.A. (2019-2020) como agente de soporte técnico.

**Mario Alberto Ibarra Martínez**

Computación, Milagro, Universidad Agraria del Ecuador
miibarra@uagraría.edu.ec

ORCID: <https://orcid.org/0009-0004-1721-5953>

Mario Alberto Ibarra Martínez es un docente universitario ecuatoriano, actualmente profesor en la carrera de Computación de la Facultad de Ciencias Agrarias de la Universidad Agraria del Ecuador, con más de 10 años de trayectoria en la enseñanza superior. Es Ingeniero Industrial por la Escuela Superior Politécnica del Litoral (ESPOL), Máster en Seguridad, Higiene Industrial y Salud Ocupacional por la Universidad de Guayaquil, y Máster en Ingeniería de Software y Sistemas Informáticos por la Universidad Internacional de La Rioja (España). Esta formación le permite integrar el rigor analítico propio de la ingeniería industrial con enfoques modernos aplicados al ámbito tecnológico y educativo. Ha desarrollado un estilo pedagógico innovador orientado a la aplicación práctica de los conceptos matemáticos en la resolución de problemas de ingeniería, fomentando el pensamiento crítico y el uso de herramientas digitales, e impulsando en varias asignaturas el desarrollo de software vinculado a los contenidos de cátedra.

A lo largo de su carrera ha dirigido redes de aprendizaje, proyectos integradores y trabajos de titulación, además de participar en la gestión académica coordinando proyectos de vinculación con la sociedad. Su interés por la investigación se refleja en la publicación de varios artículos científicos indexados relacionados con las matemáticas, la inteligencia artificial y las tecnologías educativas, entre ellos trabajos difundidos en revistas como Reciamuc y CienciaLatina. Antes de dedicarse por completo a la docencia universitaria, acumuló experiencia profesional en los ámbitos administrativo y de seguridad industrial, desempeñándose en instituciones como Kimberly-Clark Ecuador, Artes Gráficas Senefelder S.A. y el Benemérito Cuerpo de Bomberos de Guayaquil, experiencia que complementa su perfil interdisciplinario entre la ingeniería, la educación y los sistemas computacionales.

**Luis Andrés Rodríguez Valle**

Computación, Milagro, Universidad Agraria del Ecuador
luis.rodriguez.valle@uagraría.edu.ec

ORCID: <https://orcid.org/0009-0004-1721-5953>

Luis Andrés Rodríguez Valle es un estudiante ecuatoriano de la carrera de Computación, cuya formación combina la experiencia práctica laboral con el desarrollo de proyectos propios de programación.

En el ámbito tecnológico, se define como programador junior con facilidad de aprendizaje y fluidez para resolver problemas de programación. Su portafolio en GitHub refleja un perfil orientado al desarrollo web y la aplicación de inteligencia artificial: ha trabajado con JavaScript y React en proyectos como una cartelera de cine interactiva, y con Python en un sistema de análisis de sentimientos. También ha explorado la IA aplicada al agro, con un proyecto de predicción de cosecha de maíz, además de desarrollos en HTML y CSS, entre ellos una tienda virtual de productos y su portafolio personal.



ISBN: 978-9942-53-217-6



Compás
capacitación e investigación