UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN, E INNOVACIÓN TECNOLÓGICA

CARRERA DE TECNOLOGÍA DE LA INFORMACIÓN

DISEÑO DE UN SISTEMA DE CONTROL DE CALIDAD CON REDES NEURONALES PARA CLASIFICAR GUINEO SEGÚN SU ESTADO.

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

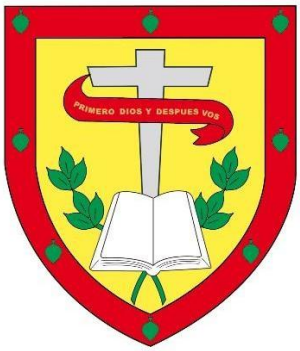
AUTOR: JERICK DAVID LLANGARI PEÑARANDA

DIRECTOR: ING. CESAR REMIGIO VEGA ABAD, MSC.

LA TRONCAL - ECUADOR

2025

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICA DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN, E INNOVACIÓN TECNOLÓGICA

CARRERA DE TECNOLOGÍA DE LA INFORMACIÓN

DISEÑO DE UN SISTEMA DE CONTROL DE CALIDAD CON REDES NEURONALES PARA CLASIFICAR GUINEO SEGÚN SU ESTADO.

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

AUTOR: JERICK DAVID LLANGARI PEÑARANDA

DIRECTOR: ING. CESAR REMIGIO VEGA ABAD, MSC.

LA TRONCAL - ECUADOR

2025

DIOS, PATRIA, CULTURA Y DESARROLLO

UNIDAD ACADÉMICA DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

UNIDAD DE TITULACIÓN

La Troncal, 27 de agosto de 2025

Sección: U.A. de Informática, Ciencias de la Computación e Innovación Tecnológica

Asunto: Certificación y aprobación de presentación del Trabajo de Titulación

Señor Ingeniero

Marcos Orellana Parra. PhD
Responsable de la Unidad de Titulación

Ingeniería en Tecnologías de la Información
De mi consideración.

Reciba un cordial saludo y mis mejores deseos de éxito en sus funciones.

El suscrito, en calidad de tutor del trabajo de titulación, certifica que el trabajo titulado: "DISEÑO DE UN SISTEMA DE CONTROL DE CALIDAD CON REDES NEURONALES PARA CLASIFICAR GUINEO SEGÚN SU ESTADO", desarrollado

por el estudiante **Jerick David Llangari Peñaranda**, con numero de cedula 0957152804, ha sido guiado y revisado de manera periódica, cumpliendo con las normativas estatutarias establecidas por la Universidad Católica de Cuenca.

Particular que pongo en su conocimiento para los fines legales consiguientes. Sin otro particular me suscribo de Usted.

Atentamente,

Firmado electrónicamente por:
CESAR REMIGIO VEGA ABAD
Validar únicamente con FirmaEC

Ing. Cesar Remigio Vega Abad, MSC.
TUTOR

DECLARACIÓN DE AUTORÍA Y RESPONSABILIDAD

Jerick David Llangari Peñaranda portador(a) de la cédula de ciudadanía N.º **0957152804**. Declaro ser el autor de la obra: "**Diseño de un sistema de control de calidad con redes neuronales para clasificar guineo según su estado"**, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

La Troncal, **29 de agosto del 2025**

F: ……

Jerick David Llangari Peñaranda

C.I. 0957152804

AGRADECIMIENTO

Quiero expresar mi profundo agradecimiento a Dios, por darme la sabiduría y la paz necesarias para culminar esta etapa con propósito y convicción.

A mis padres y familia, quienes han sido mi mayor inspiración. Gracias por su amor, confianza y por cada palabra de aliento que me impulsó a seguir adelante.

Asimismo, expreso mi más sincero agradecimiento a la Universidad Católica de Cuenca, por abrirme las puertas y permitirme formarme como profesional.

A mis docentes, por compartir sus conocimientos, por su guía constante y por inculcarme la disciplina y el compromiso en cada etapa de mi formación.

A mis compañeros de carrera, con quienes compartí experiencias, aprendizajes y esfuerzos, demostrando que el trabajo en equipo siempre multiplica los resultados.

Este trabajo no solo representa el fruto de años de esfuerzo, sino también el reflejo de cada persona y momento que me ayudó a construirlo. Culmino esta etapa con gratitud, convicción y el firme compromiso de seguir aportando con responsabilidad y pasión al mundo que me rodea.

DEDICATORIA

Dedico mi tesis principalmente a Dios, por darme la fortaleza, la sabiduría y la perseverancia necesaria para alcanzar esta meta.

A mis padres, por su esfuerzo y apoyo incondicional, y por ser los pilares fundamentales en cada etapa de mi vida.

A mis seres queridos, que, con cada palabra de aliento, compañía y fe en mí, me ayudaron a levantarme en momentos difíciles.

Esta tesis es el fruto del trabajo constante, pero también del amor, la paciencia y el respaldo de quienes han estado a mi lado desde el comienzo. A todos ellos, mi más profunda gratitud.

Resumen

La clasificación manual de guineos (Musa Cavendish) es propensa a errores y subjetividad, lo que genera pérdidas económicas en la agroindustria. Este trabajo presenta un sistema de control de calidad mediante el uso de redes neuronales convolucionales para la clasificación de guineos según su estado utilizando visión artificial. El sistema distingue dos categorías: “Bueno” y “Malo”. Se utilizó una red neuronal convolucional ResNet18 con aprendizaje por transferencia, entrenada con imágenes fotografiadas en un entorno controlado. El modelo se incorporó en una aplicación de escritorio desarrollada con OpenCV y Tkinter, que permite clasificar en tiempo real mediante el uso de una cámara web, incorporando un prefiltros HSV para mejorar la detección. Las métricas de evaluación mostraron una exactitud superior al 90%, validando así su rendimiento. El sistema disminuye la intervención humana, mejora la inspección de calidad y representa un prototipo eficaz para futuras implementaciones a mayor escala en el sector alimenticio.

Palabras clave: Visión artificial, redes neuronales convolucionales, clasificación de frutas, control de calidad, ResNet18.

Abstract

Manual classification of bananas (Musa Cavendish) is prone to errors and subjectivity, leading to economic losses in the agroindustry. This paper presents a quality control system based on convolutional neural networks for classifying bananas according to their condition through computer vision. The system distinguishes between two categories: “Good” and “Bad.” A ResNet18 convolutional neural network with transfer learning was used, trained with images photographed in a controlled environment. The model was integrated into a desktop application developed with OpenCV and Tkinter, enabling real-time classification using a webcam and incorporating HSV pre- filters to improve detection. The evaluation metrics showed an accuracy level above 90%, thus validating its performance. The system reduces human intervention, improves quality inspection, and serves as an effective prototype for future larger-scale implementations in the food industry.

Keywords: Computer vision, convolutional neural networks, fruit classification, quality control, ResNet18.

INDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

INTRODUCCIÓN

En el sector de la industria alimenticia, el control de calidad es un elemento fundamental para asegurar la satisfacción del cliente y la eficiencia de la cadena de suministro. La clasificación de frutas es uno de los procesos clave y se ha realizado tradicionalmente de manera manual, este método, aunque es efectivo, es lento, costoso y propenso a cometer errores. Además, factores como el cansancio del personal o la inconsistencia al momento de la selección pueden resultar en una clasificación inadecuada, lo que genera pérdidas económicas significativas afectando la calidad del producto final que se comercializa.

Frente a este problema, los avances en la inteligencia artificial y visión por computadora ofrecen soluciones sólidas e innovadoras. Tecnologías como las redes neuronales convolucionales (CNN) han mostrado una capacidad que sobresale al momento de analizar y clasificar imágenes con un nivel alto de precisión, superando en algunos casos las capacidades humanas en tareas de inspección visual.

El propósito de este proyecto es desarrollar un sistema de control de calidad automatizado para la clasificación de guineos según su estado, utilizando redes neuronales convolucionales. El objetivo general es implementar un prototipo funcional que, a través de imágenes tomadas por una cámara, pueda distinguir entre guineos en “buen estado” y “mal estado”.

Este trabajo se centra en la clasificación binaria del guineo y se desarrolla como un prototipo tecnológico para el sector agroindustrial. El sistema final se implementa en una aplicación de escritorio que realiza la clasificación en tiempo real, así mismo se demuestra la viabilidad de integrar soluciones de inteligencia artificial de bajo costo en procesos de control de calidad.

CAPITULO I

1. Marco Referencial

1.1. Planteamiento del Problema

En el área de alimentación, una de las principales tareas es la clasificación de frutas, puesto que impacta de una manera directa en la calidad del producto como también al momento de la degustación y aprobación de los consumidores, estos procesos se realizan siempre de una manera manual. Esto conlleva muchos imprevistos los cuales son: las opiniones del personal cuando se hace la selección, el cansancio y desgaste de todo el personal, estos factores y más han aumentado de una manera significativa los errores. Esto puede llevar a la distribución posteriormente a la venta de frutas en mal estado.

La implementación de un sistema en redes neuronales convolucionales ayudaría a disminuir estos problemas, brindándonos un método efectivo para la clasificación de frutas. En este contexto, la implementación de este tipo de tecnología encara varios desafíos los cuales son en la selección de las características visuales de las frutas, el entrenamiento de los modelos para que sean precisos y también la recolección de datos.

Por ello, es necesario investigar y poder desarrollar e implementar soluciones que mejoren los sistemas actuales de clasificación de frutas con el uso de tecnologías avanzadas de inteligencia artificial. Esto nos ayudará a optimizar el proceso de producción, también impulsará a la innovación de la industria alimenticia, disminuyendo las pérdidas económicas causadas por una clasificación errónea en los productos

1.2. Formulación del Problema

¿Cómo se puede implementar un sistema de visión artificial usando las redes neuronales convolucionales que clasifique guineos según su estado visual (bueno o dañado) para apoyar el control de calidad?

1.3. Antecedentes de la Investigación

En los últimos años, la incorporación de la inteligencia artificial en la agricultura ha tenido un crecimiento significativo, en especial con los procesos de la clasificación y control de calidad de frutas, modelos de aprendizaje profundo como son las redes neuronales convolucionales (CNN). Estas han demostrado una gran precisión en la clasificación visual de productos agrícolas. Según Cecotti et al. [1] se desarrolló un sistema para la clasificación de frutas usando CNN junto con técnicas de aumento de datos, obteniendo así una precisión superior al 94%. De una forma similar, Mishra et al. [2] usaron la arquitectura ResNet para evaluar la madurez de los guineos, logrando unos resultados altamente confiables mediante aprendizaje por transferencia.

La técnica de análisis de color se ha fortalecido como una técnica muy eficaz para identificar las etapas de maduración en frutas. En el estudio realizado por Kalia et al. [3], mostraron en su investigación que se puede diferenciar entre frutas de buena calidad y otras dañadas al combinar las características del color y su textura con los algoritmos de aprendizaje automático, como KNN y SVM.

Sin embargo, a pesar de estos avances globales en este ámbito, se detecta una falta de uso de estas tecnologías al guineo ecuatoriano (Cavendish), sobre todo en entornos locales y de tiempo real. Por lo tanto, esta investigación busca cubrir esta necesidad mediante un sistema funcional que integre CNN y se adapte a un entorno de bajo costo y gran impacto académico.

1.4. Justificación de la investigación

La clasificación adecuada de frutas como el guineo es un proceso clave en la industria alimentaria, puesto que garantiza la calidad del producto y satisface las demandas que tiene el mercado. Los métodos manuales hay veces pueden ser ineficientes, costosos y propensos a errores. En este contexto, un sistema basado en redes neuronales convolucionales ofrece una solución precisa y automatizada, siendo capaz de clasificar guineos como “buenos” o “dañados” de forma rápida y confiable.

Este sistema tiene el potencial de optimizar la cadena de suministro, reducir el desperdicio de productos no aptos y así poder mejorar la satisfacción del consumidor.

Además, representa un aporte un gran avance de la inteligencia artificial aplicada a la agroindustria.

En este contexto, el guineo constituye uno de los principales rubros de exportación en el Ecuador. En 2023 se exportaron alrededor de 354,63 millones de cajas, lo que representó un incremento del 4,52% respecto al año anterior, siendo la Unión Europea uno de los destinos prioritarios con el 28,42% del volumen total [4]. Las estadísticas oficiales del periodo 2019-2023 indican que los envíos bananeros estuvieron principalmente dirigidos a Rusia y Estados Unidos, con una participación promedio del 35% [5]. Este sector también alcanzó un valor FOB de $3,79 mil millones en 2023, un crecimiento cercano al 15% respecto a 2022 [5].

Para mantener esta posición, el sector tiene que cumplir ciertos estándares técnicos: fruta sana y homogénea conforme a las normas internacionales, certificación fitosanitaria respaldada por AGROCALIDAD con inspecciones previas al embarque [6]. El cumplimiento de estos parámetros evidencia que la competitividad del guineo ecuatoriano depende de garantizar la calidad y su trazabilidad, lo que justifica la aplicación de sistema automáticos de clasificación basados en visión artificial.

1.5. Objetivos

1.5.1. Objetivo General

Desarrollar un sistema de control de calidad para la clasificación del guineo utilizando redes neuronales convolucionales, basado en imágenes completas capturadas por cámara.

1.5.2. Objetivos Específicos

Diseñar y entrenar un modelo de red neuronal convolucional (ResNet18) con imágenes de guineos clasificados como “Bueno” o “Malo”.

Validar el modelo entrenado clasificando imágenes de guineos para verificar su funcionamiento antes del desarrollo de la aplicación de escritorio.

Evaluar el desempeño del sistema mediante métricas de exactitud, precisión y pérdida durante el entrenamiento.

1.6. Limitaciones

El sistema desarrollado corresponde a un prototipo funcional en fase experimental. Su desempeño es óptimo solo bajo condiciones controladas de iluminación y fondo neutro.

Por otro lado, la base de datos empleada es limitada en comparación con los estándares internacionales, lo cual puede afectar la capacidad de generalización del modelo. Además, la red neuronal puede disminuir su precisión ante frutas parcialmente visibles o en entornos no entrenados. Aunque se consideró el análisis colorimétrico como un enfoque alternativo, el sistema final no lo utiliza de una manera explícita, salvo un filtro HSV básico para detectar la presencia del guineo.

1.7. Delimitaciones

Este trabajo se desarrolla como un prototipo de apoyo tecnológico para el sector de la agroindustria, centrado en el proceso de clasificación del guineo mediante visión computacional. Esta investigación se lleva a cabo en los espacios académicos de la Universidad Católica de Cuenca, extensión La Troncal, y se enfoca de manera exclusiva en la clasificación binaria del guineo: “Bueno” o “Malo”.

Se emplea una red neuronal convolucional tipo ResNet18 y no se utiliza colorimetría como una técnica principal, aunque se implementa un filtro HSV simple como condición previa para la presencia de la fruta en la imagen.

CAPÍTULO II

2. Marco teórico

Este capítulo ofrece los fundamentos teóricos y sus antecedentes relevantes al tema de la clasificación automática de guineos. En primer lugar, se introduce el marco de visión por computador y aprendizaje profundo aplicados a la agricultura, con énfasis en las redes neuronales convolucionales (CNN) como una herramienta para la clasificación de imágenes. En segundo lugar, se describen las principales causas del deterioro en la calidad del guineo, incluyendo enfermedades y defectos de postcosecha, para comprender qué tipos de daños debe detectar un sistema de visión.

La integración de las redes neuronales en el sector de la agroindustria ha revolucionado en el control de calidad de las frutas. Este tipo de técnica ha permitido automatizar la inspección visual de los productos considerando los atributos como son el color, forma y sus defectos, esto ayuda a reducir la subjetividad y los errores que están asociados a la evaluación humana. Párraga et al. [7], muestran en su investigación que la visión artificial impulsada por inteligencia artificial (IA) mejora de una manera significativa la eficiencia, ayudando a agilizar los procesos de selección y disminuyendo la intervención manual en comparación con métodos tradicionales.

Por otro lado, la visión por computadora se ha consolidado como una de las herramientas más esenciales en la inspección de frutas. En los últimos años varias investigaciones evidencian un panorama amplio donde algoritmos de aprendizaje automático son aplicados en la detección y clasificación de frutas a nivel industrial.

Según Naranjo et al. [8], resumen en su investigación el uso de redes neuronales convolucionales (CNN) en tareas como clasificación de varias frutas, control de calidad externo y el conteo de frutos, mostrando logros significativos frente a las técnicas convencionales. La implementación de estas técnicas responde a la necesidad de automatizar los procesos y manejar grandes volúmenes de producción, manteniendo esa solidez en la evaluación de parámetros de calidad en diversos tipos de frutas.

Los modelos CNN representan el estado del arte en la inspección automática de la calidad externa de frutas, los avances recientes en la clasificación de frutas mediante CNN, abarcando parámetros visuales como color, tamaño, forma y defectos.

Los modelos CNN representan el estado del arte en la inspección automática de la calidad externa de frutas, los avances recientes en la clasificación y clasificación de frutas mediante CNN, abarcando parámetros visuales como color, tamaño, forma y defectos. Según el estudio de Chuquimarca et al. [9], destaca que la inspección no destructiva basada en visión artificial alcanza precisiones muy altas al momento de identificar frutas de primera calidad, cumpliendo así con los estándares internacionales que consideran madurez, geometría y defectos. En general, al momento de usar las CNN para el control de calidad, logra evaluaciones más rápidas, objetivas y replicables en entornos industriales.

La utilización de la visión artificial ha mostrado grandes resultados en el control de calidad del guineo. Según Ryan A. León et al. [10], se desarrolló un algoritmo de visión artificial basado en redes neuronales convolucionales, en específico YOLO V8, para identificar mandarinas Murcott en mal estado en tiempo real. Esto demuestra el potencial que tienen estas tecnologías para aplicarse en otros frutos como el guineo. Asimismo, el grado de madurez del guineo constituye un parámetro fundamental al momento para determinar su calidad y aceptación comercial.

Tradicionalmente se evalúa mediante inspección visual. Esta tarea resulta propensa a la subjetividad humana. Entrando en este contexto, las redes neuronales convolucionales han sobresalido como una solución efectiva para automatizar dicha evaluación. Según Arunima et al. [11], desarrollaron un algoritmo basado en CNN para clasificar el estado de maduración postcosecha de guineos de la variedad Nendran, logrando alrededor de un 95% de exactitud en la clasificación de los niveles de madurez.

Este enfoque digital acelera la clasificación de los frutos en comparación con la inspección humana, garantizando consistencia al aplicar los estándares visuales, por ejemplo, la escala de color de la cáscara definidos para exportación y mercado local.

La implementación de CNN en la evaluación de madurez mejora la objetividad y permite un procesamiento por lotes más rápidos, beneficiando la logística de postcosecha.

Este tipo de soluciones también ha sido exitoso en la clasificación automática de otras frutas tropicales, en este caso, el mango. En el estudio realizado por Zheng et al. [12], propusieron un sistema de clasificación para mangos basado en una CNN optimizada para determinar categorías de calidad comercial del fruto. Este modelo alcanzó un 97.37% de precisión al identificar el grado de madurez de los mangos, con una tasa de error promedio de solo el 2.63%.

Este nivel de acierto, es comparable al de expertos humanos, evidenciando así que las CNN pueden capturar eficazmente diferencias de color, tamaño y textura asociadas a la madurez y calidad del mango. La automatización de este proceso permite clasificar grandes volúmenes de fruta de una manera consistente, mejorando así la eficiencia en plantas empacadoras y reduciendo las pérdidas por clasificación errónea. Asimismo, la identificación de frutas en estado de descomposición ha alcanzado niveles casi perfectos gracias al Deep Learning. Afsharpour et al. [13] propusieron un método robusto para detectar fruta podrida y simultáneamente identificar la especie vegetal, enfrentando escenarios con datos limitados y condiciones adversas. Su modelo logró una precisión extraordinaria del 99.93% al distinguir frutos sanos de los deteriorados. Además, incorporó técnicas que mantuvieron su rendimiento incluso bajo iluminación intensa. Mediante mapas de activación de clase (CAM), el sistema resaltó regiones claves que permitieron interpretar visualmente las diferencias entre fruta fresca y podrida, aportando así la interpretabilidad al diagnóstico automatizado.

Según Aviles Amador et al. [14], la Sigatoka negra (Mycosphaerella fijiensis) es una de las enfermedades foliares más severas que afecta al cultivo de guineo en Ecuador. Esta enfermedad compromete el área foliar activa de la planta, reduciendo así su fotosíntesis y debilitando su desarrollo general. Como resultado, se puede observar una disminución de hasta 50% en la producción, lo que representa una amenaza considerable para la rentabilidad del cultivo y la estabilidad de la cadena de

suministro. Los autores destacan que, si bien el uso de fungicidas ha sido la práctica más extendida para su control, en la actualidad se están explotando varias alternativas más sostenibles como el uso de microorganismos endófitos, los cuales muestran potencial para mejorar el manejo de esta enfermedad en sistemas agrícolas tropicales.

Durante los últimos años, el uso de redes neuronales convolucionales (CNN) ha emergido como una estrategia efectiva para la detección temprana de enfermedades foliares en los cultivos tropicales como son el guineo. Este tipo de técnicas han demostrado su capacidad para identificar patrones visuales asociados a distintos grados de infección, incluso en etapas iniciales, donde los síntomas suelen pasarse desapercibidos para el ojo humano. Esta implementación no solo permite un diagnóstico más preciso, sino también una intervención oportuna que puede reducir de manera significativa las pérdidas de producción.

Un ejemplo relevante es el estudio de Nyambo et al. [15], en el cual se desarrolló un modelo de aprendizaje profundo para clasificar las etapas de infección de la Sigatoka negra en hojas de guineo. Este sistema, basado tanto en una arquitectura CNN básica como en VGG16, logro una precisión de hasta el 96% durante el entrenamiento y un 89% en la fase de la validación, poniendo en evidencia el potencial de estos modelos para asistir en el manejo fitosanitario del cultivo.

Por otro lado, Jiménez et al. [16] evaluaron la efectividad de distintas arquitecturas avanzadas, como EfficientNetB0, ResNet50 y VGG19, en la detección temprana de enfermedades foliares en guineo. Los resultados mostraron precisiones que oscilaban entre el 87% y el 89%, confirmando así la versatilidad de las CNN en diferentes configuraciones y su aplicabilidad en escenarios reales de producción agrícola. Estas investigaciones refuerzan la viabilidad de integrar sistemas de visión artificial en las estrategias de monitoreo y el control de enfermedades, para optimizar la sanidad vegetal y la productividad del cultivo.

El uso de modelos ligeros y eficientes también ha cobrado relevancia en la detección de enfermedades foliares del guineo mediante las técnicas de visión por computadora. En este contexto, se ha implementado la arquitectura MobileNetV2 en conjunto con imágenes capturadas por drones, esto permite una evaluación aérea y no invasiva de los cultivos. Linero-Ramos et al. [17] evaluaron la estabilidad de un

conjunto de datos multiespectral para la clasificación de la Sigatoka negra en hojas de banano, alcanzando una precisión del 86.5% al distinguir entre hojas sanas y hojas afectadas, demostrando así la viabilidad de emplear redes móviles en entornos de monitoreo agrícola a gran escala.

Adicionalmente, Hussain et al. [18] propusieron un enfoque más robusto mediante una variante mejorada del algoritmo YOLO, denominada KHO-YOLOv8, la cual fue entrenada con más de 5.000 muestras de hojas infectadas por diferentes enfermedades, incluida la Sigatoka. Este modelo logro una precisión del 96.5% en la clasificación multiclase, logrando evidenciar su capacidad para identificar distintos tipos de patologías en el cultivo con alta exactitud y eficiencia, lo que lo convierte en una solución prometedora para aplicaciones agrícolas inteligentes a gran escala.

La arquitectura ResNet ha tenido un impacto importante en el desarrollo de modelos de clasificación de imágenes gracias a su mecanismo de conexiones residuales o skip connections. Según Xu et al. [19] para facilitar el entrenamiento de redes neuronales profundas, al reducir el efecto del desvanecimiento del gradiente.

Este tipo de mecanismo ha permitido un avance significativo en el desempeño de modelos para la visión por computadora y análisis de imágenes médicas.

Especialmente, ResNet-18 es considerada la versión más ligera de la familia ResNet, lo que la hace más adecuada para hacer tareas de clasificación rápida en contextos donde se requiere eficiencia sin sacrificar precisión. En el estudio realizado por Shao et al. [20], se evaluaron diferentes modelos preentrenados sobre conjuntos de datos de componentes electrónicos, donde ResNet-18 alcanzó un 99.5% de exactitud con un número moderado de parámetros, posicionándose como una buena alternativa competitiva frente a arquitecturas complejas como VGG-16 y más optimizadas como MobileNet-V2.

En cambio, Hassan et al. [21] señalan que modelos como VGG y GoogleNet ofrecen un buen rendimiento, pero demandan mayor capacidad computacional. En cambio, arquitecturas eficientes como MobileNet o EfficientNet son más adecuadas para sistemas de bajo consumo, aunque estas pueden presentar limitaciones de precisión en tareas multicategoría como la clasificación de frutas. Asimismo, Behera et

al. [22] aplicaron ResNet-18 en un estudio sobre la clasificación del estado de madurez de la papaya, comparándolo con modelos como VGG-16, VGG-19 y ResNet-50.

Aunque VGG-19 alcanzó una precisión del 100%, ResNet-18 ha demostrado ser igualmente eficaz, confirmando su utilidad en las aplicaciones agrícolas donde se requiere un equilibrio entre el rendimiento y eficiencia computacional.

El transfer learning consiste en reutilizar modelos de visión por computador entrenándolos previamente sobre grandes conjuntos de datos genéricos, como ImageNet, e ir ajustándolo (fine-tuning) a tareas específicas con datasets reducidos.

Esta estrategia resulta especialmente relevante en el contexto agrícola, donde la recolección de imágenes etiquetadas suele ser limitada por las restricciones logísticas y de recursos. Según Al Sahili y Awad [23], entrenar redes profundas desde cero exige grandes volúmenes de datos, mientras que el uso de modelos preentrenados permite alcanzar una alta precisión sin necesidad de construir datasets extensivos desde el campo.

En la práctica, los modelos preentrenados sobre bases de datos extensas como ImageNet conservan parámetros optimizados, mejor conocidos como pesos que encapsulan representaciones visuales generales útiles, como bordes, formas y texturas. Estas representaciones pueden ser reutilizadas en otras tareas mediante transferencia de aprendizaje. Por ejemplo, en el modelo PapayaFreshNet, Sar et al. [24] usaron ResNet- 50 como extractor de características jerárquicas en imágenes de papaya, alcanzando una precisión del 97.7% en la clasificación de frescura, sin necesidad de aplicar métodos destructivos.

En la clasificación de frutas tropicales, diversos estudios han reportado resultados exitosos mediante la transferencia de aprendizaje. Behera et al. [25] emplearon múltiples arquitecturas preentrenadas, entre ellas VGG-16, VGG-19 y ResNet-18, para identificar el estado de madurez de papayas. En su investigación, VGG-19 alcanzó un rendimiento perfecto del 100%, mientras que los demás modelos asimismo tuvieron resultados destacados, demostrando la eficiencia del enfoque transferido. De forma similar, Appe et al. [26] diseñaron un sistema para la clasificación de madurez de tomates utilizando un modelo basado en VGG-16 con las

capas superiores reemplazadas por una red neuronal densa, obteniendo también una alta precisión.

Estos resultados dejan en claro que el transfer learning permite reducir de una manera considerable el tiempo de entrenamiento y la dependencia de grandes volúmenes de datos, convirtiéndose en una herramienta clave en sistemas de visión artificial aplicada en la agroindustria.

El presente capítulo se ha definido los fundamentos conceptuales y tecnológicos que sustentan la propuesta de un sistema automático de clasificación de guineos mediante visión artificial. Se ha evidenciado como la combinación de redes neuronales convolucionales, técnicas de transferencia de aprendizaje y arquitecturas eficientes como ResNet-18 representa una solución factible, precisa y adaptable para enfrentar los desafíos del control de calidad en la agroindustria.

Asimismo, se ha puesto en contexto la importancia del guineo, sus principales enfermedades como la Sigatoka negra y los retos postcosecha que justifican la necesidad de sistemas automatizados. Las aplicaciones exitosas de CNN en frutas tropicales, junto con la implementación de estos sistemas en entornos embebidos, ratifican el potencial de la inteligencia artificial para transformar la inspección visual agrícola tradicional hacia un enfoque más robusto, replicable y en tiempo real.

CAPÍTULO III

3. Marco metodológico

3.1. Enfoque de la Investigación

Esta investigación se enmarca en un enfoque cuantitativo, ya que se está trabajando con datos digitales provenientes de imágenes de guineo, los cuales son convertidos en matrices numéricas que son procesadas por un modelo de red neuronal. Se hace uso de herramientas matemáticas, estadísticas y computacionales para analizar, clasificar y evaluar los resultados del sistema.

La investigación también es de tipo aplicada, ya que propone una solución tecnológica a un problema real en el ámbito de la agroindustria: la clasificación automatizada de guineos mediante visión computacional e inteligencia artificial. Esta solución se implementará a través del uso de redes neuronales convolucionales (CNN) para clasificar visualmente el estado del guineo (bueno o malo).

3.2. Nivel de la Investigación

El nivel de esta investigación es propositivo, ya que tiene como objetivo final el diseño y desarrollo de un sistema funcional que pueda implementarse en entornos reales de clasificación de frutas.

También, posee un componente descriptivo, dado que se realiza un análisis visual de los guineos en diferentes condiciones (mal estado y en buen estado), evaluando patrones visuales como textura, color, manchas y el deterioro que puedan ser detectados por el modelo entrenado.

3.3. Población y Muestra

La población está conformada por imágenes digitales de guineos, tanto en buen estado como en mal estado. La muestra utilizada corresponde a un conjunto de imágenes capturadas manualmente mediante cámara de un teléfono móvil en condiciones controladas de iluminación y fondo.

Se utilizó un muestreo no probabilístico por conveniencia, debido a que las imágenes fueron recolectadas en función de su disponibilidad y accesibilidad en

contextos académicos, sin un criterio aleatorio. Las imágenes fueron clasificadas manualmente para así poder etiquetar los datos necesarios y entrenar el modelo.

3.4. Métodos de Investigación

Se empleó el método deductivo, puesto que parte de los fundamentos teóricos del aprendizaje profundo, redes neuronales convolucionales y visión por computadora para aplicarlos a un caso específico: la clasificación del guineo.

También se aplicó el método analítico, ya que el sistema fue descompuesto en diferentes etapas: adquisición de datos (imágenes), preprocesamiento, modelado, entrenamiento del modelo, validación del sistema y evaluación del rendimiento.

3.5. Técnicas e Instrumentos de Recolección

Una de las técnicas que se usó fue la observación directa, a través de la captura de imágenes de guineos en distintas condiciones visuales. Como uno de los instrumentos principales se usó una cámara de celular, permitiendo obtener imágenes con una resolución adecuada para el entrenamiento.

Como una de las herramientas computacionales se utilizaron Python y sus librerías especializadas: PyTorch y OpenCV, para la construcción del modelo, procesamiento de datos e integración de la aplicación de escritorio.

3.6. Tratamiento de la información

Las imágenes fueron organizadas en carpetas por clase: “Bueno” y “Malo”. Después, se realizó un preprocesamiento que incluyó redimensionamiento a 224x224 píxeles, normalización de los valores de color y técnicas de aumento de datos (rotación, inversión, variación de brillo).

El modelo fue entrenado utilizando una arquitectura ResNet18, validado y evaluado mediante métricas como: pérdida (loss), precisión (accuracy) y la visualización gráfica del desempeño por épocas.

Finalmente, el modelo fue integrado en una aplicación de escritorio desarrollada con Tkinter y OpenCV, capaz de recibir imágenes en tiempo real desde la

cámara, pudiendo así realizar la predicción, mostrando el resultado y guardando la imagen clasificada según corresponda.

CAPÍTULO IV

4. Resultados y Análisis

4.1. Introducción General del Sistema

Este capítulo presenta los resultados y el análisis de un sistema automático de clasificación de guineos. El sistema se ha creado utilizando métodos de visión artificial y redes neuronales convolucionales para diferenciar entre guineos en buen y mal estado a través de imágenes.

Se describirán los elementos empleados, la base de datos utilizada, el procedimiento de entrenamiento del modelo **ResNet18**, así como la valoración de su desempeño. Además, se detallará el funcionamiento de la aplicación gráfica desarrollada con **Tkinter**, que permite la clasificación en tiempo real, y el módulo de clasificación masiva por lote.

4.2. Descripción del Dataset

La recolección de datos se generó de manera manual mediante la fotografía de guineos reales. Las fotografías fueron clasificadas en dos grupos “Bueno” y “Malo”. Se presenta a continuación la distribución de imágenes en los subconjuntos de entrenamiento y validación.

Tabla 4.1: Distribución del dataset

Carpetas	Cantidad de Imágenes
Training = Malo	1750
Training = Bueno	1278
Validation = Malo	1750
Validation = Bueno	1278

Fuente: Elaboración propia, (2025).

Esta distribución permite que el modelo aprenda de forma equitativa las características visuales de ambas clases, evitando sesgos en el proceso de clasificación

4.3. Proceso de Entrenamiento del Modelo

El modelo fue entrenado utilizando el script “modelo_guineo_clasificador.py”, que implementa una arquitectura ResNet18 que se adapta a dos clases: “Bueno” y “Malo”.

El modelo se cargó con pesos preentrenados en ImageNet y se ajustó con una nueva capa final (fully connected) que disminuye la salida a dos neuronas, correspondiente a las clases que se mencionaron anteriormente.

Las imágenes que se tomaron manualmente fueron cargadas desde carpetas organizadas por clase y se les aplicaron varias técnicas de transformaciones como redimensionamiento, rotación, inversión horizontal y ajuste de color, con el fin de mejorar la generalización del modelo.

4.4. Configuración inicial

```
# ==== CONFIGURACIÓN ====
clases_guineo = ['Malo', 'Bueno']       #Clases para predecir
base_dir = 'C:/Clasificacion_guineo'    #Carpeta base del proyecto
train_dir = os.path.join(base_dir, 'Training')    #Carpeta de entrenamiento
val_dir = os.path.join(base_dir, 'Validation')    #Carpeta de validación
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')  #Usar GPU si está disponible
```

Figura 4.1: Fragmento de código – Configuración inicial del sistema

Fuente: Elaboración propia, (2025).

Tabla 4.2: Explicación de configuración inicial del script de entrenamiento.

Línea de Código	Descripción
clases_guineo = ['Malo', 'Bueno']	Establece las dos categorías objetivo que el modelo debe predecir: guineos en buen estado y guineos en mal estado.
base_dir = 'C:/Clasificacion_guineo'	Se establece la ruta base donde se encuentran todas las carpetas del proyecto.
train_dir = os.path.join(base_dir, 'Training')	Se crea la ruta completa hacia la carpeta de imágenes de entrenamiento.

val_dir = os.path.join(base_dir, 'Validation')	Se crea la ruta completa hacia la carpeta de imágenes de validación.
device = torch.device ('cuda' if torch.cuda.is_available () else 'cpu')	Detecta si hay GPU disponible (CUDA) y si hay se usará para acelerar el entrenamiento.

Fuente: Elaboración propia, (2025).

4.5. Dataset personalizado

```python
# ==== DATASET PERSONALIZADO ====
class GuineoDataset(torch.utils.data.Dataset):
    def __init__(self, data_dirs, transform):
        self.transform = transform
        self.data_dirs = data_dirs
        self.clases = list(data_dirs.keys())
        # Crear un diccionario con las imágenes de cada clase
        self.imagenes = {
            clase: [f for f in os.listdir(data_dirs[clase]) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
            for clase in self.clases
        }

    def __len__(self):
        # Devuelve el número total de imágenes en todas las clases
        return sum(len(self.imagenes[c]) for c in self.clases)

    def __getitem__(self, idx):
        # Selecciona una clase aleatoria y una imagen de esa clase
        clase = random.choice(self.clases)
        # Asegura que el índice no se salga del rango de imágenes de la clase
        idx = idx % len(self.imagenes[clase])
        nombre_imagen = self.imagenes[clase][idx]
        ruta_imagen = os.path.join(self.data_dirs[clase], nombre_imagen)
        imagen = Image.open(ruta_imagen).convert('RGB')
        # Aplica las transformaciones y devuelve la imagen y su etiqueta
        return self.transform(imagen), self.clases.index(clase)
```

Figura 4.2: Fragmento del script modelo_guineo_clasifcador.py – Definición de GuineoDataset.

Fuente: Elaboración propia, (2025).

Tabla 4.3: Clase personalizada GuineoDataset para cargar imágenes.

Línea de Código	Descripción
class GuineoDataset(torch.utils.data.Dataset):	Define una clase que hereda de Dataset para manejar imágenes de manera personalizada.
def __init__(...)	Constructor que recibe rutas y transformaciones, y prepara las clases.
self.imagenes = {...}	Crea un diccionario con las imágenes disponibles por clase.
def __len__(self):	Devuelve la cantidad total de imágenes del dataset.
def __getitem_(self, idx):	Selecciona una clase aleatoria, aplica transformaciones y devuelve imagen + etiqueta.
Image.open(...).convert('RGB')	Abre la imagen y la convierte a formato RGB.
self.transform(imagen)	Aplica transformaciones como redimensionamiento o normalización.

Fuente: Elaboración propia, (2025).

4.6. Transformaciones de imágenes

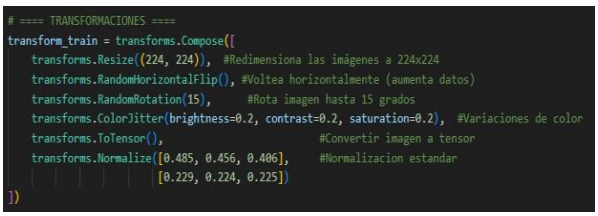


Figura 4.3: Transformaciones aplicadas al conjunto de entrenamiento (transform_train).

Fuente: Elaboración propia, (2025).

Tabla 4: Descripción de las transformaciones aplicadas al conjunto de entrenamiento (transform_train).

Transformación	Descripción
Resize ((224, 224))	Redimensiona todas las imágenes a 224x224 píxeles.
RandomHorizontalFlip()	Realiza un volteo horizontal aleatorio para aumentar la variedad de datos.
RandomRotation (15)	Aplica una rotación aleatoria de hasta ±15 grados.
ColorJitter(...)	Ajusta aleatoriamente brillo, contraste y saturación.
ToTensor ()	Convierte la imagen PIL a un tensor para PyTorch.
Normalize ([...], [...])	Normaliza los valores de píxeles con la media y desviación estándar de ImageNet.

Fuente: Elaboración propia, (2025).

4.7. Carga de datos (DataLoader)

```
# Crear DataLoaders para cargar los datos en lotes (batches de 8 imágenes)
dl_train = DataLoader(train_dataset, batch_size=8, shuffle=True)
dl_val = DataLoader(val_dataset, batch_size=8, shuffle=True)
```

Figura 4.4: Creación de los DataLoader para entrenamiento y validación.

Fuente: Elaboración propia, (2025).

Tabla 4.5: Parámetros y propósito de los DataLoader definidos para entrenamiento y validación

Componente	Descripción
dl_train	DataLoader para el conjunto de entrenamiento, con lotes de 8 imágenes y barajado aleatorio (shuffle=True).
dl_val	DataLoader para el conjunto de validación, también con lotes de 8 imágenes y orden aleatorio.

Fuente: Elaboración propia, (2025).

4.8. Definición del modelo

```
# ==== DEFINICIÓN DEL MODELO ====
modelo = torchvision.models.resnet18(weights='IMAGENET1K_V1')
modelo.fc = nn.Linear(modelo.fc.in_features, len(clases_guineo))
modelo = modelo.to(device)
```

Figura 4.5: Definición y adaptación del modelo ResNet18 para la clasificación de guineos.

Fuente: Elaboración propia, (2025).

Tabla 4.6: Adaptación de la arquitectura ResNet18 para clasificación binaria.

Línea de código	Descripción
torchvision.models.resnet18(...)	Carga un modelo ResNet18 preentrenado en ImageNet.
modelo.fc = nn.Linear(...)	Sustituye la capa final del modelo por una personalizada para dos clases.
modelo.to(device)	Envía el modelo a GPU o CPU dependiendo de la disponibilidad del sistema.

Fuente: Elaboración propia, (2025).

4.9. Optimización y función de pérdida

```
# ==== OPTIMIZADOR Y PÉRDIDA ====
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(modelo.parameters(), lr=1e-4)
```

Figura 4.6: Configuración del optimizador y la función de pérdida en el entrenamiento.

Fuente: Elaboración propia, (2025).

Tabla 4.7: Parámetros de optimización utilizados para entrenar el modelo.

Elemento	Descripción
loss_fn = nn.CrossEntropyLoss()	Define la función de pérdida utilizada para tareas de clasificación multiclase. Es adecuada para modelos que generan probabilidades en la salida.
optimizer = optim.Adam(...)	Asigna el optimizador **Adam**, que ajusta los pesos del modelo con una tasa de aprendizaje de 0.0001 (1e-4), facilitando una convergencia más estable.

Fuente: Elaboración propia, (2025).

4.10. Función de entrenamiento

```
# ==== FUNCIÓN DE ENTRENAMIENTO ====
def entrenar(epochs):
    for epoca in range(epochs):
        print(f"\n🔁 Epoch {epoca+1},
        modelo.train()  # Poner model
        running_loss = 0.0
```

Figura 4.7: Inicio de la función de entrenamiento del modelo.

Fuente: Elaboración propia, (2025).

Tabla 4.8: Inicio del proceso de entrenamiento por épocas

Línea de código	Descripción
def entrenar(epochs):	Define la función encargada de ejecutar el entrenamiento del modelo durante un número definido de épocas.
for epoca in range(epochs):	Bucle principal que recorre cada época de entrenamiento.
modelo.train()	Coloca el modelo en modo entrenamiento, activando capas como *dropout* o *batch normalization* si existieran.
running_loss = 0.0	Inicializa la variable para acumular la pérdida total durante una época.

Fuente: Elaboración propia, (2025).

4.11. Registro de métricas y resultados del entrenamiento

Durante el entrenamiento del modelo, se registraron tres métricas clave por cada época:

- Pérdida del entrenamiento.
- Pérdida de la validación.
- Exactitud de la validación.

Estas métricas nos permiten evaluar si el modelo mejora con el tiempo de ejecución, si se presentan sobreajustes y también como evoluciona su capacidad de generalización. La captura de estos valores se realizó automáticamente dentro del ciclo de entrenamiento del script "modelo_guineo_clasificador.py", y se mostró en la consola por cada época.

En la siguiente tabla se presenta un resumen de las métricas registradas en las primeras 9 épocas. La décima época no se pudo capturar para la visualización por razones de tiempo, pero sus métricas fueron registradas internamente.

Tabla 4.9: Resumen de métricas por época durante el entrenamiento del modelo.

Época	Pérdida Entrenamiento	Pérdida Validación	Exactitud Validación
1	0.0408	0.3782	0.9156
2	0.0370	0.3271	0.9216
3	0.0091	0.3746	0.9276
4	0.0068	0.4001	0.9220
5	0.0251	0.4387	0.8944
6	0.0121	0.5230	0.8422
7	0.0275	0.3346	0.9135
8	0.0108	0.4103	0.9085
9	0.0023	0.5213	0.9004

Fuente: Elaboración propia, (2025).

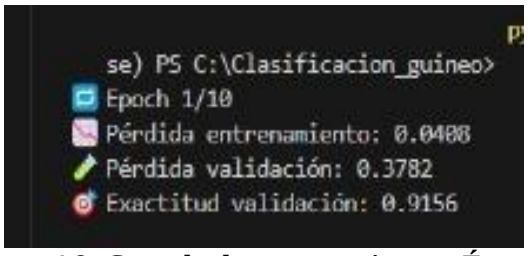


Figura 4.8: Consola de entrenamiento – Época 1.

Fuente: Elaboración propia, (2025).

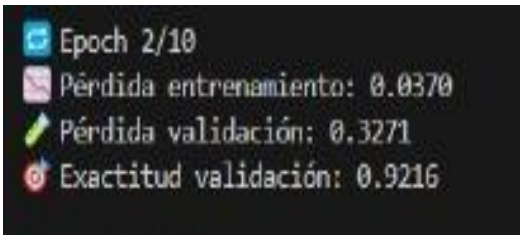


Figura 4.9: Consola de entrenamiento – Época 2.

Fuente: Elaboración propia, (2025).

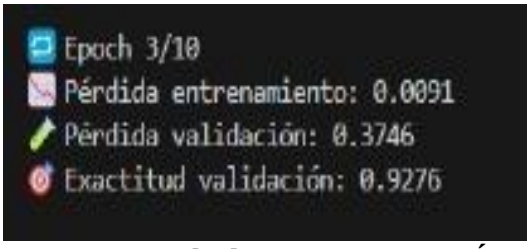


Figura 4.10: Consola de entrenamiento – Época 3.

Fuente: Elaboración propia, (2025).

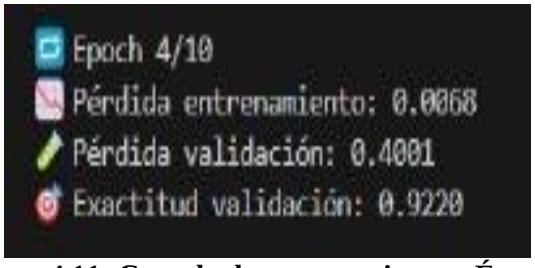


Figura 4.11: Consola de entrenamiento – Época 4.

Fuente: Elaboración propia, (2025).

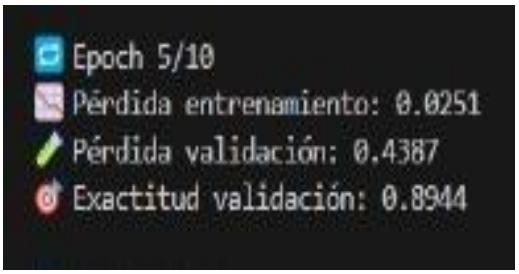


Figura 4.12: Consola de entrenamiento – Época 5

Fuente: Elaboración propia, (2025).

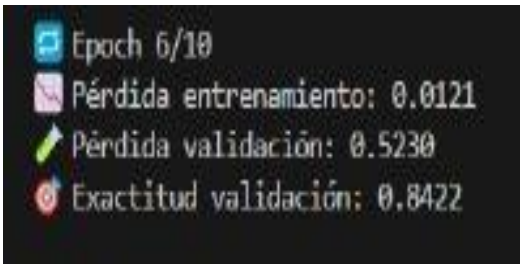


Figura 4.13: Consola de entrenamiento – Época 6.

Fuente: Elaboración propia, (2025).

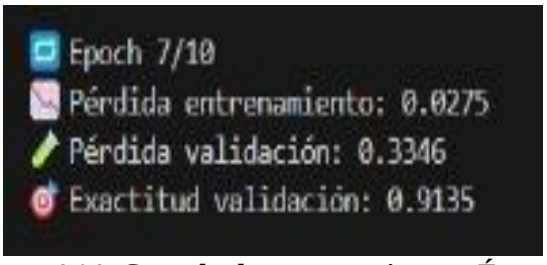


Figura 4.14: Consola de entrenamiento – Época 7.

Fuente: Elaboración propia, (2025).

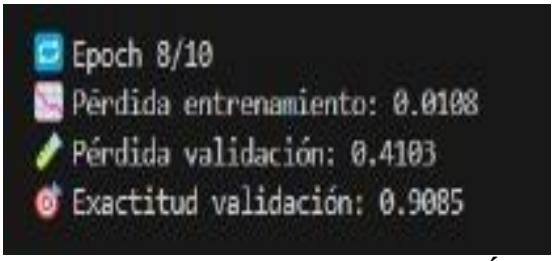


Figura 4.15: Consola de entrenamiento – Época 8.

Fuente: Elaboración propia, (2025).

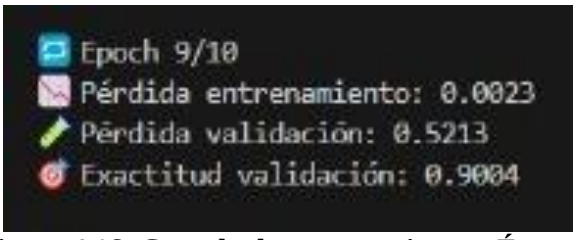


Figura 4.16: Consola de entrenamiento – Época 9

Fuente: Elaboración propia, (2025).

4.12. Registro gráfico del entrenamiento

Además del registro por épocas, el script generó un gráfico que permite observar visualmente el comportamiento del entrenamiento.

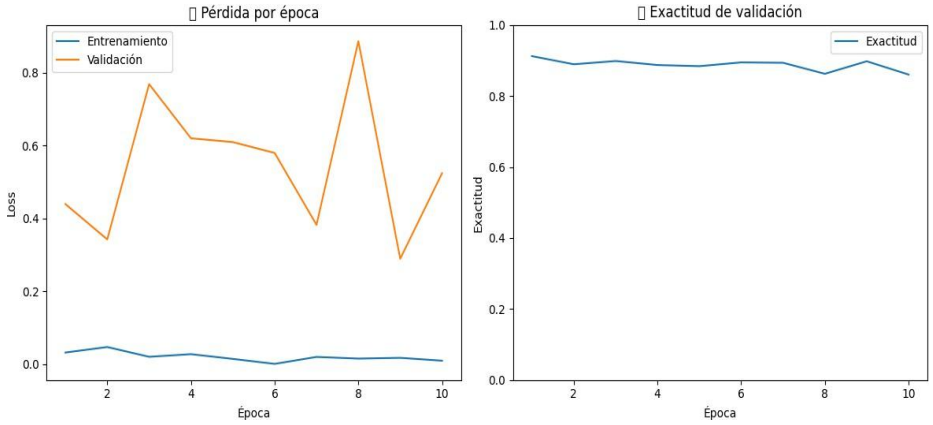


Figura 4.17: Gráfica de pérdida y exactitud del modelo por época.

Fuente: Elaboración propia, (2025).

4.13. Análisis de resultados

En esta sección se analizará el desempeño del modelo en el proceso de entrenamiento, que muestra una buena convergencia y una sólida capacidad para generalizar. El entrenamiento se redujo rápidamente en las primeras fases y se mantuvo estable en niveles bajos, lo que indica que el modelo aprendió adecuadamente las características del conjunto de datos.

Además, en la pérdida de validación se mostró variaciones moderadas, estas son características típicas de las tareas de clasificación con imágenes reales. Sin embargo, estas variaciones no afectan considerablemente en el rendimiento general.

La precisión de la validación se mantuvo constantemente alta, variando entre el 90% y el 92% en las etapas finales del entrenamiento. Este rango demuestra que el

modelo no solo aprendió a clasificar correctamente las imágenes que se mostró en el entrenamiento, sino que también logra mantener un rendimiento sólido ante imágenes que no se han cargado durante el entrenamiento. Por lo cual, se concluye que el sistema presenta confiabilidad y efectividad para diferenciar entre guineos en buen estado y guineos en mal estado, con un margen de error bajo.

4.14. Clasificación Masiva de Imágenes

Para evaluar el rendimiento del modelo entrenado, se utilizó un conjunto de imágenes independientes que no formaron parte del proceso de entrenamiento y tampoco de validación. Teniendo en cuenta eso, se desarrolló un script que se puso de nombre “Resultado.py”, el cual permite realizar una clasificación masiva (por lotes) de imágenes. Este script automatiza el proceso de inferencia sobre múltiples imágenes, organizando y clasificando en subcarpetas de salida según la clase predicha por el modelo: “Bueno” o “Malo”, dependiendo de la calidad visual del guineo.

Estas clases son las mismas utilizadas durante el proceso de entrenamiento, la automatización implementada no solo facilita la evaluación a gran escala, sino que también asegura un seguimiento eficiente de los resultados, analizando de forma clara la distribución de predicciones realizadas por el sistema.

Se recopilaron 48 imágenes reales de guineos, recolectadas manualmente y que no fueron utilizadas durante el entrenamiento, las cuales fueron clasificadas automáticamente por el sistema y organizadas según su categoría, con la siguiente distribución:

Tabla 4.10: Distribución de imágenes reales utilizadas en la clasificación masiva.

Clase	Cantidad de imágenes
Bueno	22
Malo	26
Total	**48**

Fuente: Elaboración propia, (2025).

4.15. Estructura de Carpetas Utilizadas

El script “Resultado.py” funciona sobre una estructura de carpetas diseñada para poder facilitar el procesamiento, la organización y el almacenamiento de los resultados, una de las ventajas del sistema es su capacidad para generar de manera automática las rutas necesarias en caso de que no existieran, esto garantiza un entorno limpio y ordenado durante la ejecución del programa.

A continuación, se detallan las carpetas que se usaron para el proceso:

Tabla 4.11: Estructura de carpetas utilizadas por el script Resultado.py.

Carpeta	Descripción
Resultado/	Contiene las imágenes reales sin clasificar que serán evaluadas por el modelo.
Clasificados/Bueno/	Generada automáticamente para almacenar las imágenes clasificadas como guineo en buen estado.
Clasificados/Malo/	Generada automáticamente para almacenar las imágenes clasificadas como guineo en mal estado.

Fuente: Elaboración propia, (2025)

Se muestran a continuación las imágenes correspondientes al proceso que se llevó a cabo:

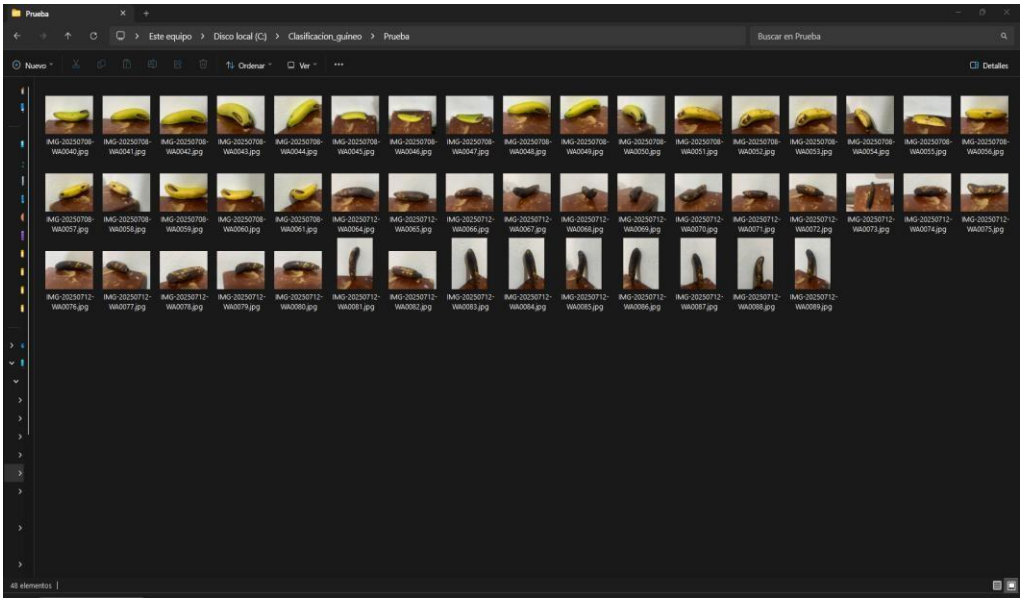


Figura 4.18: Carpeta de imágenes utilizadas para la prueba masiva del sistema.

Fuente: Elaboración propia, (2025).

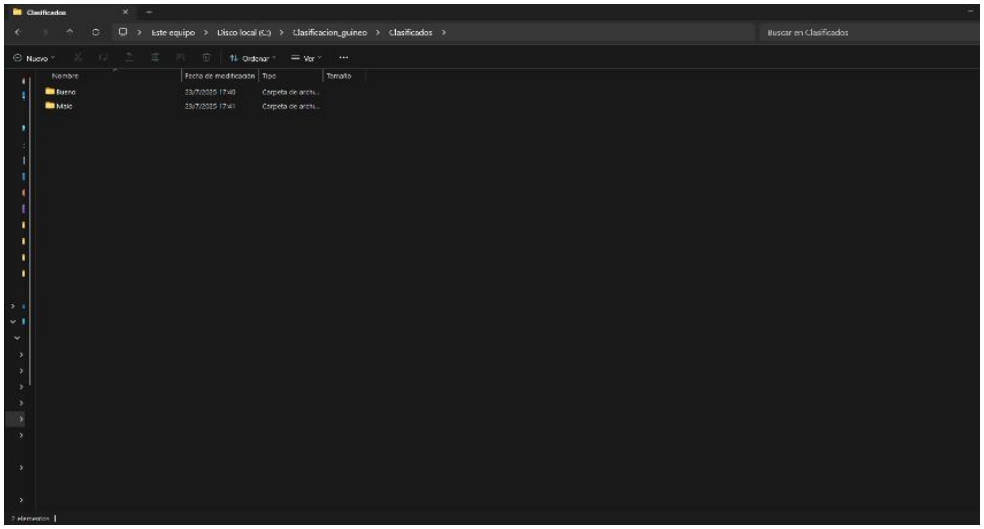


Figura 4.19: Resultado de la clasificación automática de imágenes.

Fuente: Elaboración propia, (2025).

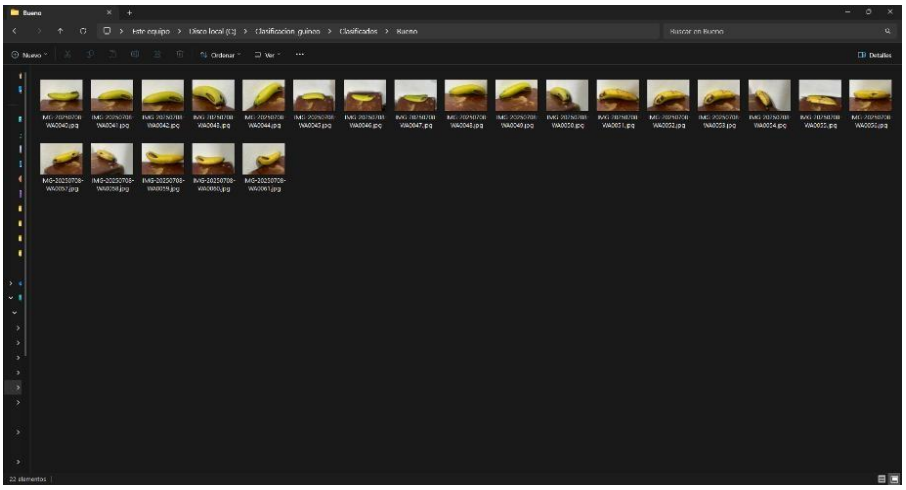


Figura 4.20: Carpeta con imágenes clasificadas como “Bueno”

Fuente: Elaboración propia, (2025).

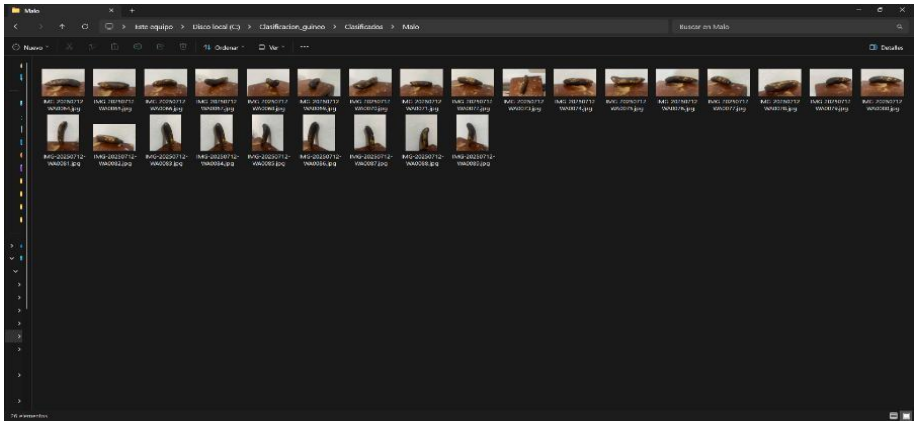


Figura 4.21:Carpeta con imágenes clasificadas como “Malo”

Fuente: Elaboración propia, (2025).

4.16. Fragmento del Código: Clasificación y Organización

A continuación, se mostrará un fragmento del script “Resultado.py”, que se encarga de recorrer todas las imágenes, aplicar las transformaciones que son necesarias, ejecutar el modelo ya entrenado, obtener la predicción y mover cada imagen a la carpeta correspondiente:

```python
# === CLASIFICACIÓN Y ORGANIZACIÓN ===
buenos = 0
malos = 0
total = 0
registro = []

for archivo in os.listdir(carpeta_imagenes):
    if archivo.lower().endswith(('.jpg', '.jpeg', '.png')):
        ruta = os.path.join(carpeta_imagenes, archivo)
        img = Image.open(ruta).convert("RGB")
        input_tensor = transform(img).unsqueeze(0).to(device)

        with torch.no_grad():
            salida = modelo(input_tensor)
            confianza = torch.nn.functional.softmax(salida, dim=1)[0]
            _, pred = torch.max(salida, 1)
            clase = clases[pred.item()]
            conf_score = confianza[pred.item()].item()

        if clase == 'Bueno':
            destino = os.path.join(carpeta_salida_bueno, archivo)
            buenos += 1
        else:
            destino = os.path.join(carpeta_salida_malo, archivo)
            malos += 1

        copy2(ruta, destino)
        total += 1
        registro.append(f"{archivo} - {clase} - {conf_score*100:.2f}%")
```

Figura 4.22:Fragmento del código Resultado.py – Clasificación de imágenes por lote

Fuente: Elaboración propia, (2025).

Tabla 4.12: Explicación del fragmento de código para clasificación masiva.

Línea de código	Descripción
for archivo in os.listdir(...)	Recorre cada imagen dentro de la carpeta Resultado/.
Image.open(...).convert("RGB")	Abre la imagen y la convierte al formato RGB.
transform(img)	Aplica las transformaciones necesarias para que la imagen sea compatible con el modelo.
modelo(input_tensor)	Ejecuta la red neuronal para obtener la predicción.
torch.max(...)	Obtiene la clase con mayor probabilidad.
confianza = ...	Calcula la probabilidad asociada a la predicción mediante softmax.

Fuente: Elaboración propia, (2025).

4.17. Registro de Resultado

El script también genera un archivo de texto llamado “registro_resultado.txt”, donde se almacena de forma detallada:

- La fecha de ejecución del script.
- El total de imágenes clasificadas.
- La cantidad de imágenes por clase.

El nombre de cada imagen, la clase que se predijo y el porcentaje de confianza del modelo

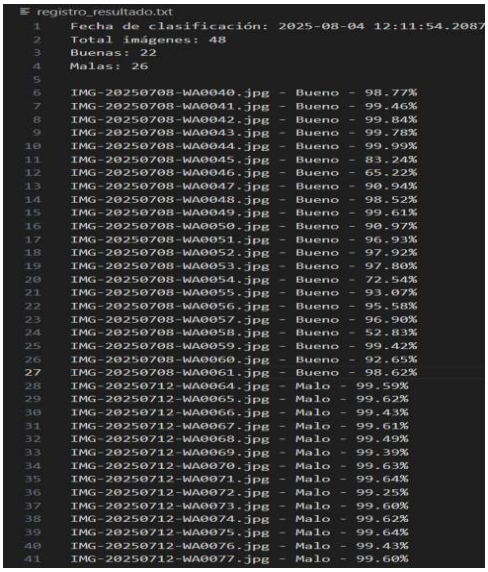


Figura 4.23: Fragmento del archivo generado registro_resultado.txt.

Fuente: Elaboración propia, (2025).

Este registro nos permite observar y validar que el sistema no solo clasifica, sino que también proporciona el nivel de certeza con la toma de decisión.

4.18. Visualización de Resultados

También el mismo script “Resultado.py” incluye una función que genera un gráfico de barras que representa la cantidad de imágenes que se clasificaron por cada clase “Bueno” o “Malo”, lo que nos permite evaluar de manera visual la distribución final tras la clasificación automática:

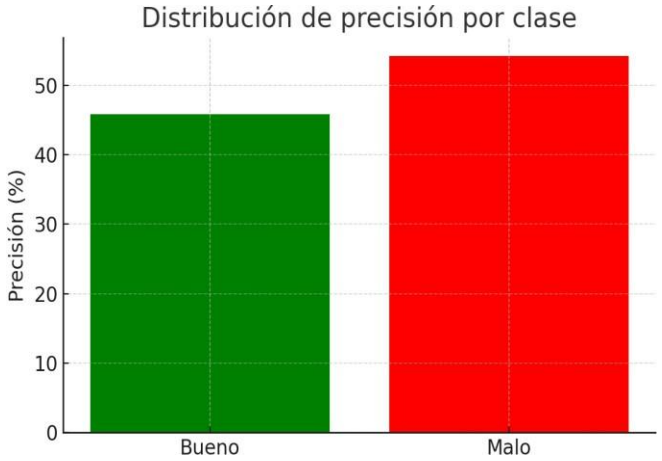


Figura 4.24: Gráfico de distribución por clase tras clasificación masiva.

Fuente: Elaboración propia, (2025).

Nota: Este gráfico de barras se muestra al finalizar el proceso, pero solo se guarda si el usuario lo desea, esta opción nos ofrece flexibilidad al momento de realizar pruebas o generar reportes.

4.19. Clasificación con la aplicación Tkinter

Esta es la última fase del sistema desarrollado y su implementación fue en una interfaz gráfica de usuario (GUI) utilizando la biblioteca Tkinter, dando la facilidad de su uso por parte de cualquier persona sin conocimientos técnicos. Esta aplicación ejecutable permite la clasificación en tiempo real a través de una cámara conectada a la laptop, mostrando en la pantalla el resultado de la clasificación y ofreciendo opciones para guardar la imagen y el resultado en un historial.

4.19.1. Interfaz de Clasificación en Tiempo Real

La aplicación fue convertida en un ejecutable (.exe) para su fácil distribución. Al ejecutar el archivo “BananoScan.exe” desde el escritorio, se activa la cámara conectada y se despliega el interfaz.

Figura 4.25: Clasificación en tiempo real de un guineo en buen estado mediante la ejecución del aplicativo GUINEO.exe

Fuente: Elaboración propia, (2025).

En esta imagen se puede observar la interfaz en tiempo real de la aplicación, donde un guineo es categorizado exitosamente como “Bueno”, con una confianza del 99.0%.

Figura 4.26: Clasificación en tiempo real de un guineo en mal estado mediante la ejecución del aplicativo GUINEO.exe

Fuente: Elaboración propia, (2025).

En esta imagen se puede observar la interfaz en tiempo real de la aplicación, donde un guineo es categorizado exitosamente como “Malo”, con una confianza del 99.7%.

4.19.2. Registro de Resultados y Carpeta Generada.

Al darle clic en “Guardar Imagen”, el sistema guarda la imagen en una carpeta automáticamente generada en el escritorio con el nombre de “resultado_clasificado”, en esta carpeta se crean dos sub carpetas llamado: “Bueno”, “Malo”.

También se crea un archivo llamado “historial_clasificaciones.txt”, en este archivo (.txt) se registrará la fecha, hora, resultado y nombre del archivo guardado.

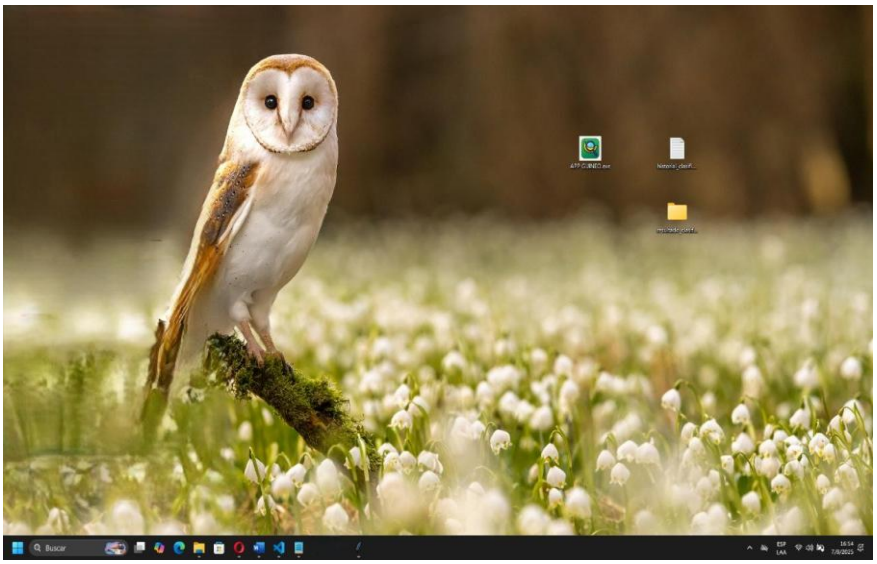


Figura 4.27: Ícono de la app, historial y carpeta generada.

Fuente: Elaboración propia, (2025).

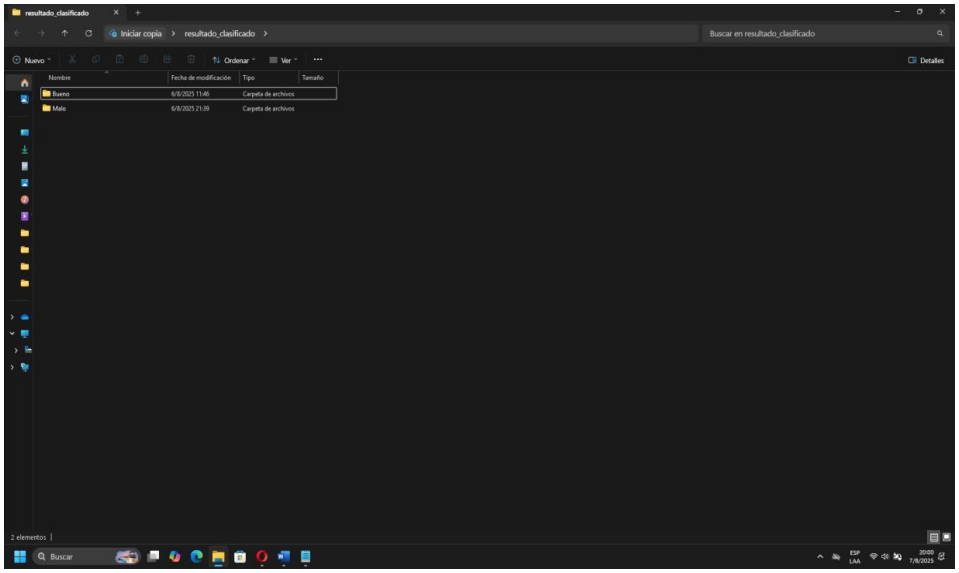


Figura 4.28: Estructura interna de resultado_clasificado/ con las subcarpetas Bueno y Malo.

Fuente: Elaboración propia, (2025).

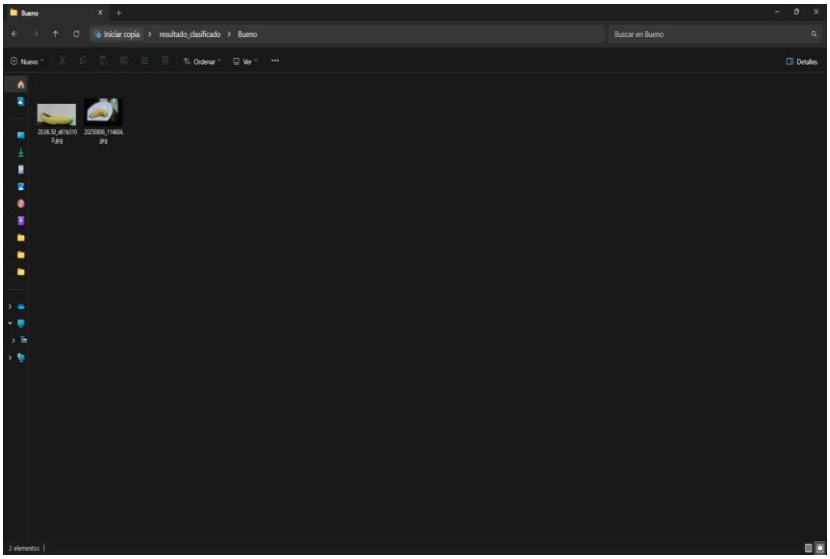


Figura 4.29:Vista de los resultados clasificados en la carpeta Bueno.

Fuente: Elaboración propia, (2025).

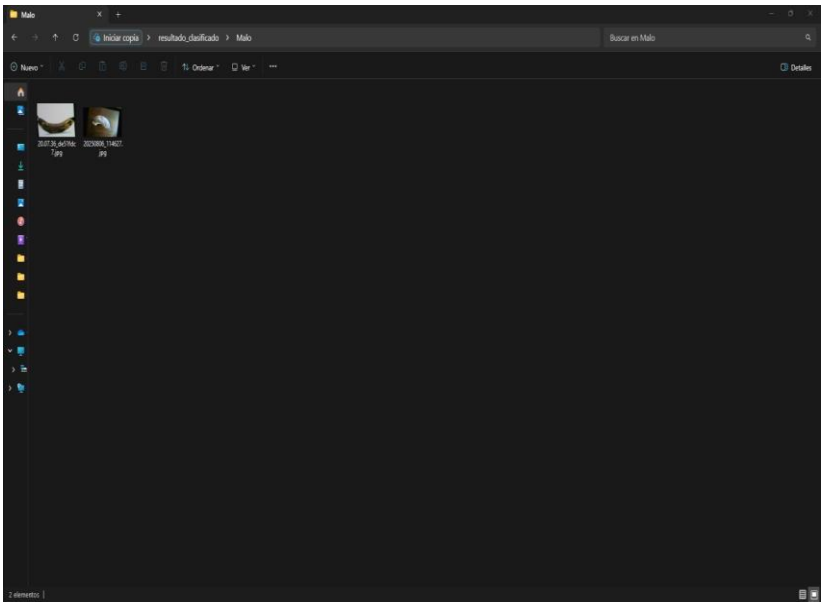


Figura 4.30: Vista de los resultados clasificados en la carpeta Malo.

Fuente: Elaboración propia, (2025).

Figura 4.31: Historial generado automáticamente por la app.

Fuente: Elaboración propia, (2025).

4.19.3. Fragmento de Código de Clasificación con Tkinter

Esta aplicación fue desarrollada en el archivo "app_guineo_nuevo.py", este script permite tomar foto, clasificarla, guardar los resultados y registrar el historial de manera automática. Para mejorar el rendimiento de la aplicación y su uso, se añadieron:

4.19.4. Carga del modelo y transformaciones (robusto para .py y .exe)

```python
# === CONFIGURACIÓN (ruta robusta para .py y .exe) ===
def resource_path(rel):
    # 1) PyInstaller onefile (carpeta temporal)
    meipass = getattr(sys, "_MEIPASS", None)
    if meipass:
        p = _os.path.join(meipass, rel)
        if _os.path.exists(p):
            return p
    # 2) Si es .exe, busca al lado del ejecutable
    if getattr(sys, "frozen", False):
        p = _os.path.join(_os.path.dirname(sys.executable), rel)
        if _os.path.exists(p):
            return p
    # 3) Modo .py: junto al script
    return _os.path.join(_os.path.dirname(_os.path.abspath(__file__)), rel)

modelo_path = resource_path("modelo_guineo.pth")
clases = ['Malo', 'Bueno']
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# === CARGAR MODELO ===
modelo = torchvision.models.resnet18(weights=None)
modelo.fc = torch.nn.Linear(modelo.fc.in_features, len(clases))
modelo.load_state_dict(torch.load(modelo_path, map_location=device))
modelo.eval().to(device)

# === TRANSFORMACIONES ===
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Figura 4.32: Carga robusta del modelo y configuración de transformaciones para ejecución en ".py" y ".exe."

Fuente: Elaboración propia, (2025).

Tabla 4.13: Carga del modelo y preprocesamiento.

Línea de código	Descripción
resource_path("modelo_guineo.pth")	Ubica el archivo del modelo tanto en desarrollo (.py) como en él .exe (PyInstaller).
clases = ['Malo','Bueno']	Define las etiquetas de salida del modelo.
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')	Selecciona GPU si está disponible; si no, CPU.
torchvision.models.resnet18(weights=None)	Crea la arquitectura ResNet18 sin pesos preentrenados.
modelo.fc = torch.nn.Linear(..., len(clases))	Ajusta la capa final para 2 clases.
modelo.load_state_dict(torch.load(modelo_ path, map_location=device))	Carga los pesos entrenados en el dispositivo correcto.
modelo.eval().to(device)	Pone el modelo en modo inferencia y lo mueve a CPU/GPU.
transforms.Compose([...])	Define el pipeline de preprocesado.
transforms.Resize((224,224))	Redimensiona la imagen al tamaño esperado.
transforms.ToTensor()	Convierte a tensor PyTorch.
transforms.Normalize(mean=[...], std=[...])	Normaliza con los parámetros estándar de ImageNet.

Fuente: Elaboración propia, (2025).

4.19.5. Selector de cámara con vista previa

```python
def seleccionar_camara(root, max_indices=8):
    def abrir_para_preview(i, w=640, h=360):
        for be in (cv2.CAP_DSHOW, cv2.CAP_MSMF, 0):  # intenta DirectShow, MSMF y por defecto
            cap = cv2.VideoCapture(i, be) if be != 0 else cv2.VideoCapture(i)
            if not cap.isOpened():
                continue
            cap.set(cv2.CAP_PROP_FRAME_WIDTH,  w)
            cap.set(cv2.CAP_PROP_FRAME_HEIGHT, h)
            cap.set(cv2.CAP_PROP_FPS, 10)
            cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
            ok, frame = cap.read()
            if ok:
                return cap, frame
            cap.release()
        return None, None

    for i in range(max_indices):
        cap, frame = abrir_para_preview(i)
        if cap is None:
            continue

        win = tk.Toplevel(root)
        win.title(f"Vista previa - Cámara {i}")

        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        imgtk = ImageTk.PhotoImage(Image.fromarray(frame_rgb).resize((384, 288)))
        tk.Label(win, image=imgtk).pack(padx=8, pady=8)
        usar = tk.BooleanVar(False)

        tk.Button(win, text="✅ Usar esta cámara",
                  command=lambda: (usar.set(True), win.destroy())).pack(side="left", padx=8, pady=
        tk.Button(win, text="❌ Probar otra", command=win.destroy)\
            .pack(side="right", padx=8, pady=8)

        win.wait_window()
        cap.release()
        if usar.get():
            return i
    return -1
```

Figura 4.33: Selector de cámara con vista previa

Fuente: Elaboración propia, (2025).

Tabla 4.14: Selector de cámara con vista previa y cambio

Línea de código	**Descripción**
indice = seleccionar_camara(root)	Se abre el selector y devuelve el índice de la cámara elegida por el usuario.
def seleccionar_camara(root, max_indices=8):	Recorre las posibles cámaras y muestra una vista previa por cada una.
cap = cv2.VideoCapture(i, be)	Intenta abrir el índice “i” usando backends DSHOW, MSMF o el default.
cap.set(cap_prop_frame_width/height/fps/buffersi ze, …)	Fija una resolución baja y buffer mínimo para que la **vista previa** sea fluida.

Línea de código	Descripción
cap.set(cap_prop_fourcc, videowriter_fourcc(*'yuy2'))	Solicita un formato común (si el dispositivo lo soporta) para evitar pantallas negras.
frame = cap.read()	Lee un fotograma de prueba para componer la vista previa.
ventana_preview = tk.Toplevel(root)	Ventana temporal con la pregunta "¿Deseas usar esta cámara?".
image.fromarray(frame_rgb).resize((384,288), image.nearest)	Redimensiona el snapshot para mostrarlo en la ventana de selección.
ImageTk.PhotoImage(...); tk.Label(image=...)	Renderiza la imagen dentro de Tkinter.
usar = tk.BooleanVar(value=False)	Variable de control del botón elegido (usar / probar otra).
def aceptar()/rechazar(): ... destroy()	Manejan los botones "Usar" y "Probar otra" y cierran la ventana.
ventana_preview.wait_window(); cap.release()	Espera la decisión del usuario y libera la cámara de preview.
if usar.get(): return i	Si se aceptó, retorna el índice i como selección final.
self.btn_cambiar = tk.Button(..., command=self.cambiar_camara)	Botón en la app para volver a abrir el **selector** sin reiniciar.
idx = seleccionar_camara(self.root); cap = abrir_camara(idx)	Selecciona y abre la nueva cámara para el flujo principal.

Fuente: Elaboración propia, (2025).

4.19.6. Clasificación con prefiltros HSV

```python
def clasificar(self):
    if self.ultima_frame is None:
        return

    # 1) Prefiltro HSV: detectar "amarillo" del guineo y área mínima
    frame_resized = cv2.resize(self.ultima_frame, (224, 224))
    hsv = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2HSV)

    bajos = np.array([15, 40, 40])     # tono/saturación/valor mínimos
    altos = np.array([40, 255, 255])   # tono/saturación/valor máximos
    mascara = cv2.inRange(hsv, bajos, altos)
    area = cv2.countNonZero(mascara)

    if area < 1000:
        self.resultado.set("Resultado: NO DETECTADO")
        self.resultado_label.config(fg="gray")
        self.thumb_label.config(image='')
        self.btn_guardar.config(state=tk.DISABLED)
        return

    # 2) Inferencia con el modelo (ResNet18)
    img_rgb = cv2.cvtColor(self.ultima_frame, cv2.COLOR_BGR2RGB)
    input_tensor = transform(Image.fromarray(img_rgb)).unsqueeze(0).to(device)

    with torch.no_grad():
        salida = modelo(input_tensor)
        prob = torch.nn.functional.softmax(salida, dim=1)[0]
        pred_idx = torch.argmax(prob).item()
        conf = prob[pred_idx].item()

    self.ultima_pred = clases[pred_idx]
    self.resultado.set(f"Resultado: {self.ultima_pred.upper()} ({conf*100:.1f}%)")
    self.resultado_label.config(fg=("green" if self.ultima_pred == "Bueno" else "red"))
```

Figura 4.34: Clasificación con prefiltros HSV (evitar falsos positivos sin fruta)

Fuente: Elaboración propia, (2025).

Tabla 4.15: Flujo de clasificación con prefiltros HSV y presentación del resultado.

Línea de código	Descripción
frame_resized = cv2.resize(self.ultima_frame, (224, 224))	Ajusta el frame al tamaño de entrada de la red.
hsv = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2HSV)	Pasa a HSV para segmentar el color del guineo.
bajos = np.array([15,40,40]); altos = np.array([40,255,255])	Rango aproximado de **amarillo** (H,S,V) para la fruta.
mascara = cv2.inRange(hsv, bajos, altos)	Máscara binaria de los píxeles que caen en el rango.
area = cv2.countNonZero(mascara)	Cuenta de píxeles “positivos” (área detectada).

if area < 1000: ... return	Si el área es pequeña → **NO DETECTADO** (no se infiere).
img_rgb = cv2.cvtColor(self.ultima_frame, cv2.COLOR_BGR2RGB)	Convierte a RGB para PIL/torchvision.
img_pil = Image.fromarray(img_rgb)	Crea imagen PIL a partir del frame.

input_tensor = transform(img_pil).unsqueeze(0).to(device)	Preprocesa y añade dimensión de batch; mueve a CPU/GPU.
with torch.no_grad(): salida = modelo(input_tensor)	Inferencia sin gradientes (más rápido y liviano).
confianza =torch.nn.functional.softmax(salida, dim=1)[0]	Probabilidades normalizadas por clase.
_, pred = torch.max(salida, 1)	Índice de la clase con mayor probabilidad.
pred_idx = pred.item(); conf_score = confianza[pred_idx].item()	Índice final y porcentaje de confianza.
self.ultima_pred = clases[pred_idx]	Traduce índice a etiqueta (“Bueno” / “Malo”).
self.resultado.set(f"Resultado: ... {conf_score*100:.1f}%")	Muestra el resultado en la interfaz.
self.resultado_label.config(fg="green"/"red")	Colorea el texto (verde si “Bueno”, rojo si “Malo”).
winsound.Beep(1000 if ... else 600, 200)	Señal auditiva distinta según la clase.
img_thumbnail = img_pil.resize((120,120)); ImageTk.PhotoImage(...)	Miniatura de la captura mostrada bajo el resultado.
self.btn_guardar.config(state=tk.NORMAL)	Habilita el botón “Guardar Imagen”.

Fuente: Elaboración propia, (2025).

4.20. Manual de Usuario – Aplicación de Clasificación de Guineo.

4.20.1. Descripción General del Sistema.

La aplicación de escritorio que se desarrolló permite clasificar guineos en tiempo real mediante el uso de una cámara conectada a la laptop. Esta aplicación está construida en Python con la biblioteca Tkinter y ejecuta el modelo de red neuronal convolucional que ha sido entrenado con anterioridad (ResNet18) para que pueda diferenciar entre guineos "Buenos" y "Malos". El que utiliza esta aplicación puede tomar fotos en vivo, visualizar la clasificación de manera instantánea con su porcentaje de confianza y también guardar las imágenes clasificadas en carpetas separadas automáticamente. También, se mantiene un registro cronológico en un archivo de texto para facilitar el seguimiento del proceso.

4.20.2. Ejecución del archivo .exe

Para hacer más fácil el uso del sistema sin necesidad de un entorno de desarrollo, se generó un archivo ejecutable (.exe) mediante la herramienta PyInstaller. Este ejecutable se encuentra bajo el nombre "BananoScan.exe". Al momento de ejecutar, se abre una ventana con la interfaz gráfica desde la cual se puede ejecutar la clasificación.

Para facilitar el uso del sistema sin necesidad de un entorno de desarrollo, se generó un archivo ejecutable (.exe) mediante la herramienta PyInstaller. El ejecutable se encuentra bajo el nombre 'BananoScan.exe'. Al iniciarlo, se abre una ventana con la interfaz gráfica desde la cual se puede operar la clasificación.

Pasos para ejecutar el sistema:

4.20.3. Pasos para ejecutar el sistema.

En esta sección se describe la manera en la que se tiene que ejecutar el sistema desde el ejecutable. Al momento de iniciar el sistema buscara las cámaras disponibles y se mostrará una vista previa para que el usuario elija la cámara que desee utilizar, después de seleccionar, se abrirá el interfaz principal, desde donde se puede realizar la captura, ver el resultado de la clasificación y guardar en la estructura de carpetas que se genera automáticamente.

A continuación, se detallan los pasos a seguir:

1. Ubicar el archivo “BananoScan.exe” que se encuentra en el escritorio y dar doble clic para que se abra el sistema.

Figura 4.35: Ícono del ejecutable en el Escritorio.

Fuente: Elaboración propia, (2025).

2. Aparecerá el selector de cámara con vista previa, si se ve la imagen, pulsa el botón “Usar esta cámara” y si no es la correcta, pulsa “Probar otra” y sigue probando hasta encontrar la cámara que necesitas.

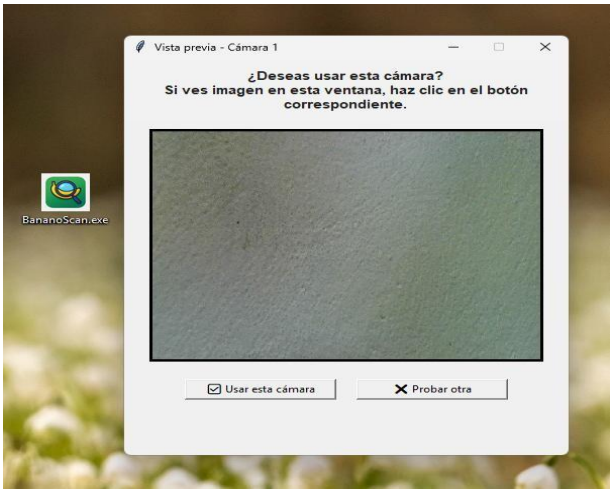


Figura 4.36: Vista previa correcta — usar esta cámara.

Fuente: Elaboración propia, (2025).

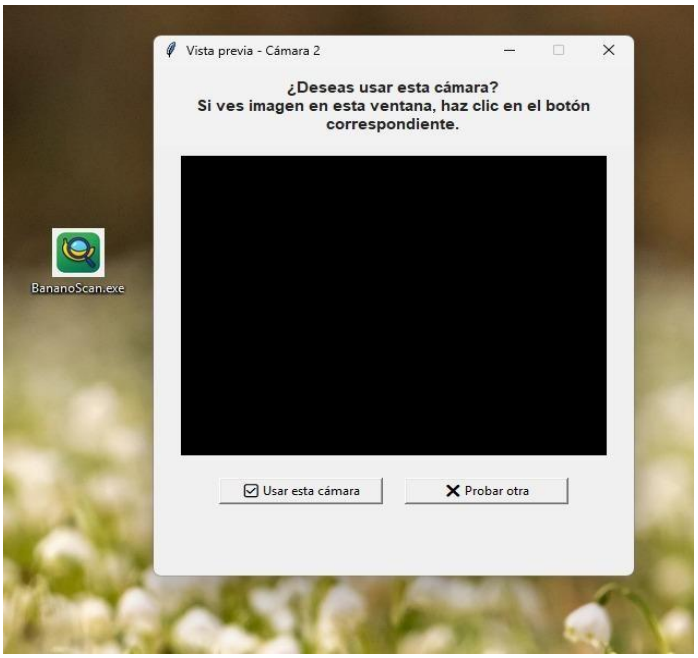


Figura 4.37: Vista previa sin señal — probar otra cámara.

Fuente: Elaboración propia, (2025).

3. Presiona “Clasificar” para que el programa procese el fotograma actual, si detecta que hay fruta en este caso guineo, se clasificará como “Bueno” o “Malo” con su % de confianza, pero si no detecta fruta se mostrará “Resultado: No detectado”.

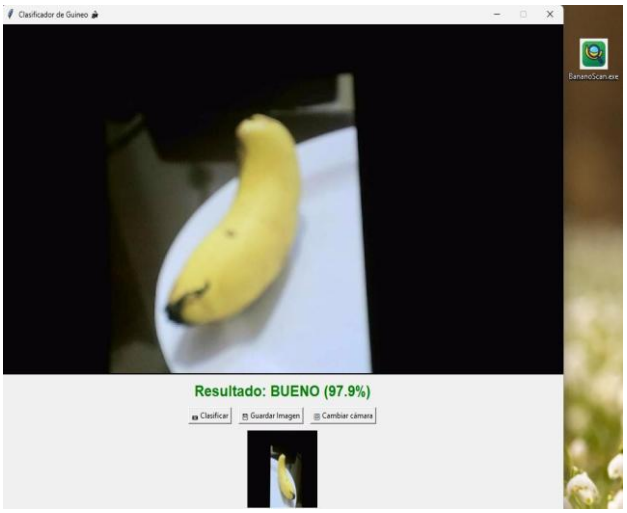


Figura 4.38: Clasificación usando foto en pantalla del teléfono. Resultado: BUENO (97.9%).

Fuente: Elaboración propia, (2025).

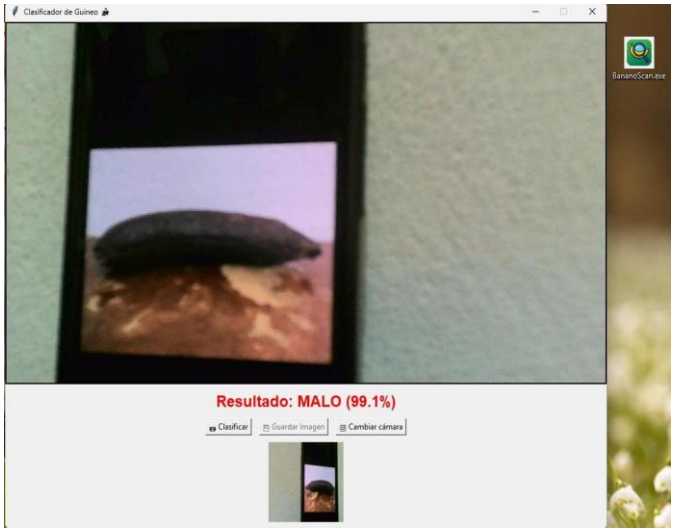


Figura 4.39: Clasificación usando foto en pantalla del teléfono. Resultado: MALO (99.1%).

Fuente: Elaboración propia, (2025).

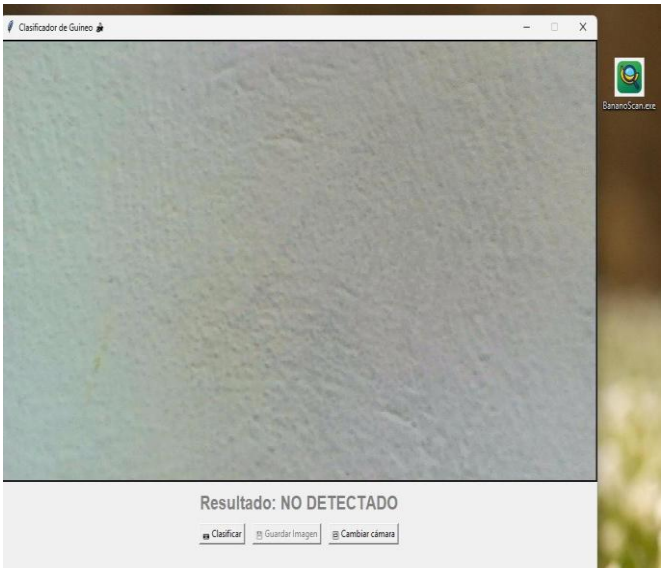


Figura 4.40: Demostración sin fruta- Resultado: NO DETECTADO

Fuente: Elaboración propia, (2025).

4. Para guardar la foto del guineo detectado, presiona el botón “Guardar Imagen”, al momento de presionar el botón se guardará automáticamente el resultado en la carpeta “resultado_clasificado” dependiendo de la clasificación se guardará dentro de dos subcarpetas llamadas “Bueno” y “Malo”, así mismo se genera un archivo de texto llamado “historial_clasificaciones.txt” con la fecha, clase y nombre de la foto.

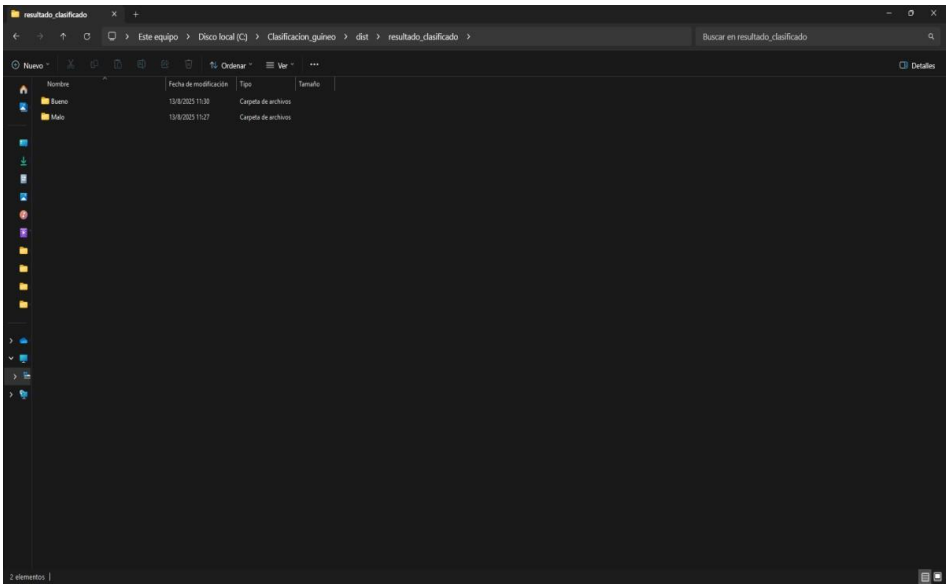


Figura 4.41: Estructura resultado_clasificado/ con subcarpetas Bueno y Malo

Fuente: Elaboración propia, (2025).

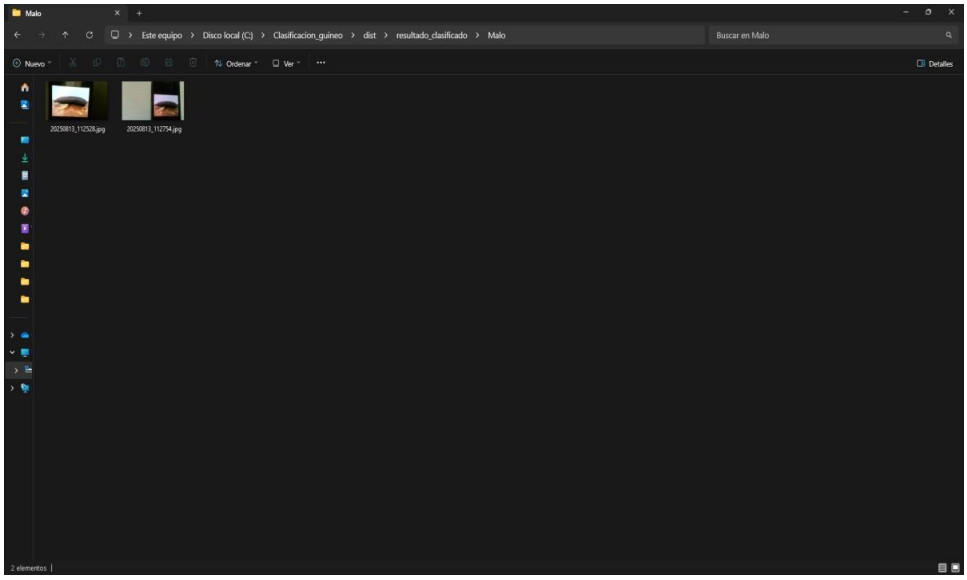


Figura 42: Subcarpeta “Malo”. Imágenes clasificadas como Bueno, nombradas con sello de tiempo.

Fuente: Elaboración propia, (2025).

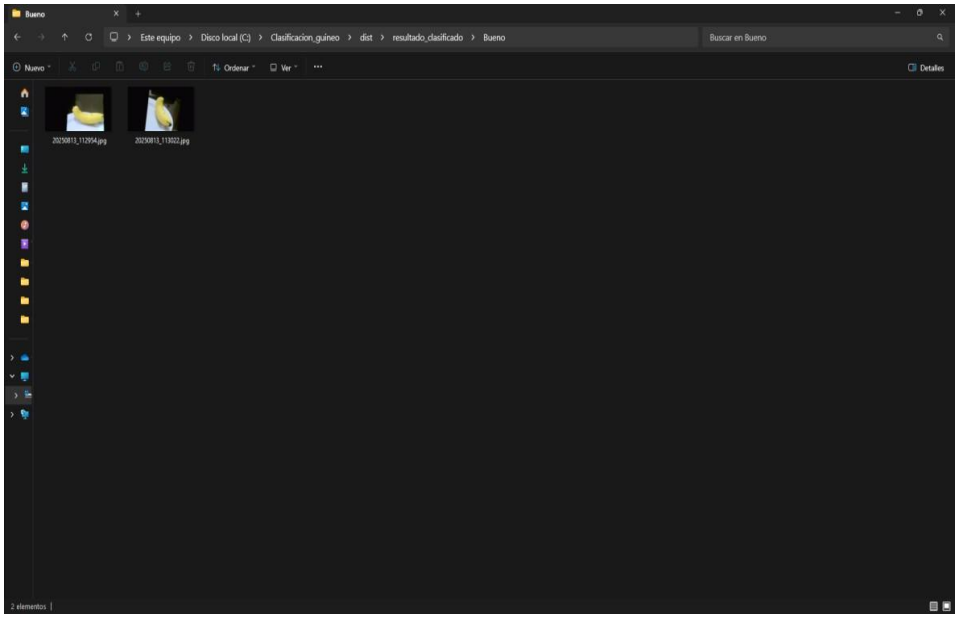


Figura 4.43: Subcarpeta “Bueno”. Imágenes clasificadas como Bueno, nombradas con sello de tiempo

Fuente: Elaboración propia, (2025).

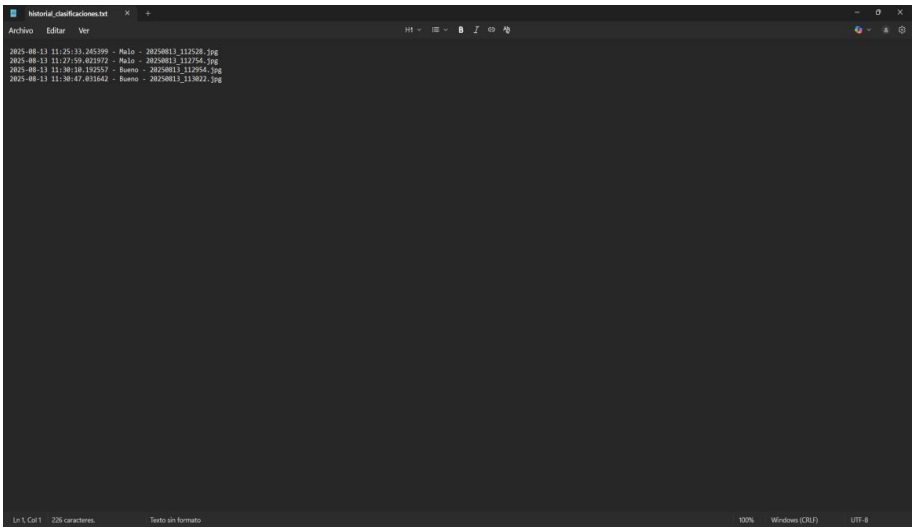


Figura 44Historial de clasificaciones. Archivo historial_clasificaciones.txt con fecha/hora, clase y nombre del archivo.

Fuente: Elaboración propia, (2025).

Si la cámara que se eligió no muestra imagen o no es la que se desea usar, puede utilizar el botón “Cambiar cámara” desde el interfaz para volver a seleccionar otro dispositivo sin tener que salir del programa principal.

Para obtener mejores resultados, se recomienda trabajar con una buena iluminación y mantener el guineo centrado, al momento de guardar, se creará o se actualizará la carpeta “resultado_clasificado” con subcarpetas por clase (Bueno/Malo) y se registrará lo que se guardó en “historial_clasificaciones.txt”.

4.20.4. Requisitos técnicos del sistema

Para ejecutar el “BananoScan.exe” no es necesario tener Python instalado, ya que todas las dependencias vienen ya incluidas en el ejecutable, pero si se recomiendan tener los siguientes requisitos:

Tabla 16: Requisitos mínimos para el funcionamiento del sistema.

Requisito	Especificación
Sistema operativo	Windows 10 o superior (64-bit)
RAM	Tener un mínimo de 4 GB (recomendado 8 GB)

CPU	Intel Core i5 / AMD Ryzen 5 o superior
Resolución de pantalla	1280×720 o mayor
Cámara	Tener una cámara integrada o una cámara USB, con drivers actualizados
Espacio en disco	Tener 300 MB para el ejecutable y los archivos generados que son las carpetas de resultados e historial.

Fuente: Elaboración propia, (2025).

4.20.5. Limitaciones

Aunque el sistema es funcional para una clasificación básica, presenta algunas limitaciones:

- Requiere una buena iluminación para tener resultados óptimos.
- La cámara tiene que estar bien enfocada al guineo para evitar clasificaciones incorrectas.
- No detecta múltiples guineos en una misma imagen.
- El modelo está entrenado solo para clasificar dos clases (Bueno/Malo); no reconoce otras frutas, ni otros estados.

CAPÍTULO V

5. Conclusiones y Recomendaciones

En este capítulo se presentan las conclusiones de este proyecto de clasificación automática de guineos basado en visión artificial y redes neuronales convolucionales, así como las recomendaciones para trabajos futuros. Las conclusiones generales de este trabajo realizado se presentarán primero, seguido de las conclusiones específicas. A continuación, se describen las validaciones técnicas que se efectuaron para evaluar el sistema y al final se sugieren varias recomendaciones para su mejora.

5.1. Conclusiones Generales.

En el presente proyecto se logró desarrollar un sistema automatizado para la clasificación de guineos (Musa Cavendish) mediante visión artificial y redes neuronales convolucionales, cumpliendo con los objetivos que se propusieron con anterioridad. Se demostró que era factible integrar técnicas de aprendizaje profundo en el control de calidad, sustituyendo la inspección visual manual por un método rápido, objetivo y reproducible. El sistema alcanzó un desempeño elevado en condiciones controladas, clasificando guineos como “buenos” o “malos” con una exactitud superior al 90%, lo que pone en evidencia una reducción significativa de la subjetividad y errores humanos en el proceso. En general, la aplicación de la arquitectura ResNet18 preentrenada permitió obtener un modelo eficiente y también preciso, validando que incluso con un dataset limitado es posible entrenar un algoritmo.

5.2. Conclusiones Específicas por Área.

5.2.1. Modelo CNN (ResNet18).

La elección de la arquitectura ResNet18, combinada con pesos preentrenados en ImageNet, fue una decisión correcta, ya que el modelo se adaptó de una manera eficaz a la clasificación binaria de guineos, extrayendo características distintivas como la textura, color y los defectos que tenía la cáscara. Esto permitió lograr una precisión alta con un conjunto de parámetros moderados, confirmando así que ResNet18 ofrece un balance adecuado entre rendimiento y eficiencia para esta aplicación.

5.2.2. Proceso de Entrenamiento y Validación.

Dado que el conjunto de datos era limitado, la estrategia de entrenamiento se basó en el uso de técnicas de aumento de datos para mejorar la robustez del modelo.

El modelo demostró un aprendizaje eficiente desde el inicio, alcanzando de manera rápida una exactitud de validación superior al 90% y manteniéndose en ese nivel de rendimiento. Aunque en las fases finales se observaron leves signos de sobreajustes como fluctuaciones en la pérdida y un estancamiento en la exactitud, el desempeño general fue muy sólido.

Se hizo una prueba final con 48 imágenes nuevas que confirmó la confiabilidad del sistema, ya que la mayoría fueron clasificadas correctamente, demostrando su capacidad para predecir y generalizar con datos nunca antes vistos.

5.2.3. Desempeño y Precisión del Sistema.

El sistema ha demostrado ser un prototipo de control de calidad funcional y fiable. El rendimiento del modelo tuvo una precisión superior del 90%, el análisis de la curva de aprendizaje confirma que la red neuronal distingue de manera eficaz entre guineos en buen y mal estado.

Además, la estrategia de usar un filtro de color “HSV” con una red neuronal convolucional fue clave para la fiabilidad del sistema, ya que la “CNN” aprendió a identificar los defectos visuales clave, mientras que el filtro eliminó de manera exitosa los falsos positivos, garantizando así que el sistema solo analice la fruta que se le muestre.

5.2.4. Aplicación e Implementación.

La integración del modelo entrenado en la aplicación de escritorio llamada “BananoScan” demuestra que esta investigación no quedó solo en la parte teórica: fue convertida en una herramienta útil. Por medio de la interfaz gráfica desarrollada con Tkinter, cualquier persona puede usar el sistema sin conocimientos técnicos, clasificando guineos al instante usando una cámara convencional.

Las pruebas que se realizaron ayudaron a confirmar que el sistema identifica con precisión el estado de la fruta y muestra el resultado junto con su nivel de confianza, por lo mismo, permitiendo guardar las evidencias y generando un historial

útil para su seguimiento. Esto confirma que el prototipo no solo es funcional en entornos de desarrollo, cumpliendo con el objetivo de crear una herramienta que sirva de apoyo para el control de calidad.

Se destaca que el mejor rendimiento se obtuvo bajo condiciones de una buena iluminación y un fondo neutro, tal como se dijo al principio; dentro de estos parámetros, la aplicación dio unos resultados rápidos y consistentes, lo cual agilizó el procedimiento de evaluación visual.

En resumen, se lograron todos los objetivos establecidos: se diseñó y entrenó un modelo CNN que sea eficiente, se validó su funcionamiento con resultados satisfactorios y se desarrolló una aplicación funcional que fue capaz de clasificar guineos en tiempo real. El sistema que se desarrolló es una exitosa prueba de concepto de cómo la inteligencia artificial y la visión por computadora pueden mejorar los procesos tradicionales en el sector agroindustrial, ayudando a reducir los errores humanos y mejorando la productividad.

Sin embargo, aunque los resultados fueron positivos, estos se limitan a un entorno controlado del prototipo. Para aplicar estos resultados en entornos industriales, será necesario tratar ciertas limitaciones, las cuales se explicarán en las siguientes recomendaciones.

5.3. Recomendaciones.

A partir de los resultados obtenidos y las limitaciones que se identificaron durante la implementación y desarrollo, se plantean a continuación varias recomendaciones para facilitar su estabilidad y abrir nuevas líneas de investigación. Estas sugerencias no solo buscan mejorar el desempeño técnico del sistema, sino también su adaptabilidad en condiciones reales. Para ello, se proponen las siguientes recomendaciones:

5.3.1. Dataset más robusto y diverso.

Aumentar el número de imágenes de entrenamiento añadiendo imágenes de guineos con diferentes grados de madurez, fondos y condiciones de iluminación, ya que al tener un conjunto de datos más grande y variado mejorara la capacidad de generalización del modelo y reducir el sobreajuste. Además, se recomienda incorporar

o generar datos sintéticos que contengan alteraciones en el fondo y la posición de la fruta, esto con el fin de fortalecer el aprendizaje del modelo más allá de las condiciones controladas actuales.

5.3.2. Procesamiento avanzado de imágenes.

Integrar técnicas avanzadas de procesamiento de imágenes para complementar a la red neuronal, por ejemplo, se podría utilizar la segmentación de fondo o la detección de objetos para separar el guineo del entorno cuando se necesite analizar múltiples frutas de manera simultánea o en escenarios menos controlados, esto permitiría al sistema manejar casos en los que esté más de un guineo al momento de clasificar o con fondos más complejos, evitando clasificaciones erróneas debido a objetos. Además, al aplicar la corrección de iluminación y balance de color ayudaría a mantener la precisión del modelo bajo condiciones variables.

5.3.3. Optimización para despliegue real.

Para que el sistema funcione de manera efectiva fuera del entorno en el que se desarrolló, es necesario adaptarlo a las condiciones reales. Esto implica ajustar el rendimiento del modelo para que se pueda ejecutar en dispositivos con recursos limitados, como tablets o laptops básicas. Además, llevar a cabo pruebas pilotos en entornos reales de empaquetado o selección de fruta, sería una gran ventaja para comprobar el desempeño del sistema bajo condiciones no controladas, como la manipulación constante de los frutos, variaciones ambientales y las condiciones de iluminación no controladas, estas pruebas ayudarían a mejorar los tiempos de respuesta y detectar oportunidades de mejora en la interacción del usuario con el sistema.

5.3.4. Extensión funcional y escalabilidad.

Cuando la funcionalidad básica ha sido validada, el sistema puede hacer trabajos más especializados que aporten un valor operativo. Por ejemplo, se puede incluir la identificación de los grados de madurez, los defectos específicos o incluso los síntomas de enfermedades u hongos, en lugar de limitarse a una clasificación binaria. Esto ayudaría a que la toma de decisiones en los procesos de control de selección, empaque y control fitosanitario sea más precisa. Además, la metodología que se desarrolló tiene la capacidad de adaptarse a otras frutas de exportación como el

aguacate, mango o papaya, lo que abre la puerta a una solución que puede transferirse a diferentes sectores agrícolas. Esta capacidad de expansión fortalece el impacto del sistema y promueve el uso de la inteligencia artificial como una herramienta útil en la agroindustria.

5.3.5. Mejoras en la interfaz para la experiencia del usuario.

Aunque la aplicación de escritorio ha demostrado su eficacia, la utilidad del sistema no depende de su precisión técnica, sino también de qué tan sencillo resulta para el usuario. Por lo tanto, se recomienda que se mejore la interfaz gráfica para hacerla más intuitiva para distintos operarios. El desarrollo para versiones móviles o web permitiría utilizar el sistema directamente en el mundo real, sin necesidad de tener equipos especializados. También se pueden incorporar métricas en tiempo real, como la tasa de acierto o el tiempo promedio por muestra, esto facilitaría el monitoreo del rendimiento y la toma de decisiones.

Además, se puede ofrecer opciones de configuración personalizadas, como ajustes de sensibilidad o modos de opción, permitiendo adaptar el sistema a diferentes contextos de uso. Estas mejoras no solo aumentarían la eficacia, sino también fortalecerían el sistema como una herramienta confiable, moderna y lista para ser implementada.

En conclusión, este trabajo establece los fundamentos para sistemas inteligentes de clasificación de frutas en entornos locales; sin embargo, para implementarlo a gran escala se requerirá mejora constante. Las recomendaciones dichas anteriormente tienen como objetivo guiar a futuras mejoras, señalando la importancia de tener más datos, tener mayor robustez ante la variabilidad y adaptarse a las condiciones reales de operación. Gracias a estos avances, el sistema tiene una gran posibilidad de desarrollarse como un prototipo académico a una herramienta comercial que tenga un gran impacto en la cadena del suministro del guineo, ayudando a reducir pérdidas por clasificaciones erróneas y aumentar la competitividad de la industria del guineo al incorporar tecnologías de inteligencia artificial.

Referencias

[1] H. Cecotti, A. Rivera, M. Farhadloo y M. A. Pedroza, «Grape detection with convolutional neural networks,» Expert Systems with Applications, vol. 159, p. 113588, 2020.

[2] R. Mishra, S. Goyal, T. Choudhury y T. Sarkar, «Banana ripeness classification using transfer learning techniques,» In 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), pp. 1-6, 2022.

[3] P. Kalia, A. Garg y A. Kumar, «Fruit quality evaluation using Machine Learning: A review,» In 2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT) , vol. 1, p. 952–956, 2019.

[4] PRO ECUADOR, «PRO ECUADOR,» 29 01 2024. [En línea]. Available: https://www.proecuador.gob.ec/banano-incrementa-sus-exportaciones-al-cierre-del-2023/. [Último acceso: 02 10 2025].

[5] Corporación Financiera Nacional (CFN), «Ficha Sectorial Banano 2019–2023,» 09 2024. [En línea]. Available: https://www.cfn.fin.ec/wp-content/uploads/2024/10/Ficha-Sectorial-Banano.pdf. [Último acceso: 02 10 2025].

[6] AGROCALIDAD, «Manual de certificación fitosanitaria para exportación,» 17 06 2024. [En línea]. Available: https://www.agrocalidad.gob.ec/wp-content/uploads/2024/07/Resolución-0175-Manual-de-Certificación-Fitosanitaria-actualizado-jun-2024.pdf. [Último acceso: 02 10 2025].

[7] S. R. P.-B. a. M. A. Coral-Ygnacio, «Implementaciones de selección visual en frutas: una revisión sistemática de literatur,» Revista Científica de Sistemas e Informática, vol. 4, nº 1, pp. 1-24, 2024.

[8] J. Naranjo-Torres, M. Mora, R. Hernández-García, R. J. Barrientos, C. Fredes y A. Valenzuela, «A review of convolutional neural network applied to fruit image processing,» Applied Sciences, vol. 10, nº 10, p. 3443, 2020.

[9] L. E. Chuquimarca, B. X. Vintimilla y S. A. Velastin, «A review of external quality inspection for fruit grading using CNN models,» Artificial Intelligence in Agriculture,

2024.

[10] R. A. León León, D. A. De La Cruz Ramos y Á. D. Rubio Infantes, «Desarrollo de un algoritmo de visión artificial con redes neuronales convolucionales (YOLO V8) para el control de calidad de la mandarina Murcott,» en Memorias de la Vigésima Tercera Conferencia Iberoamericana en Sistemas, Cibernética e Informática (CISCI 2024), 2024.

[11] P. L. Arunima, P. P. Gopinath, P. G. Lekshmi y M. Esakkimuthu, «Digital assessment of post-harvest Nendran banana for faster grading: CNN-based ripeness classification model,» Postharvest Biology and Technology, vol. 214, p. 112972, 2024.

[12] B. Zheng y T. Huang, «Mango grading system based on optimized convolutional neural network,» Mathematical Problems in Engineering, vol. 2021, nº 1, p. 2652487, 2021.

[13] P. Afsharpour, T. Zoughi, M. Deypir y M. J. Zoqi, «Robust deep learning method for fruit decay detection and plant identification: enhancing food security and quality control,» Frontiers in Plant Science, vol. 15, p. 1366395, 2024.

[14] AVILES AMADOR y Wilson Agustin, «Impacto de los microorganismos endófitos en el manejo de la sigatoka negra (Mycosphaerella fijiensis) en el cultivo de banano (Musa × paradisiaca) en el Ecuador,» BS thesis. BABAHOYO: UTB, 2025.

[15] D. Nyambo, E. Kambo, J. Leo y M. Ally, «A deep learning model for classifying Black Sigatoka disease in banana leaves based on infection stages,» Indian Journal of Science and Technology., vol. 17, nº 37, p. 3889–3897, 2024.

[16] N. Jiménez, S. Orellana, O. B. Mazon, A. W. Rivas y M. I. Ramírez, «Detection of leaf diseases in banana crops using deep learning techniques,» AI, vol. 6, nº 3, p. 61, 2025.

[17] R. R. Linero, R. C. Parra, V. A. Espinosa, R. J. Gómez y M. Gongora, «Assessment of dataset scalability for classification of Black Sigatoka in banana crops using UAV-based multispectral images and deep learning techniques,» Drones, vol. 8, nº 9, p. 503, 2024.

[18] M. Hussain, S. Ahmad, S. Ullah, H. Sattar y M. A. Rafiq, «Multi-class banana leaf disease detection via KHO-YOLOv8,» SSRN, p. 5295011, 2025.

[19] G. Xu, X. Wang, X. Wu, X. Leng y Y. Xu, «Development of skip connection in deep neural networks for computer vision and medical image analysis: A survey,» arXiv preprint, nº arXiv:2405.0172, 2024.

[20] Y. Shao, L. Zhou, F. Tang, X. Shi, D. Chen y S. Xia, «Comparative performance of finetuned ImageNet pre-trained models for electronic component classification,» arXiv preprint, nº arXiv:2506.19330, 2025.

[21] E. Hassan, S. A. Ghazalah, N. El-Rashidy, T. A. El-Hafeez y M. Y. Shams, «Sustainable deep vision systems for date fruit quality assessment using attention-enhanced deep learning models,» Frontiers in Plant Science, vol. 16, p. 1521508, 2025.

[22] S. K. Behera, A. K. Rath y P. K. Sethy, «Maturity status classification of papaya fruits based on machine learning and transfer learning approach,» Information Processing in Agriculture, vol. 8, nº 2, pp. 244-250, 2021.

[23] Z. Al Sahili y M. Awad, «The power of transfer learning in agricultural applications: AgriNet,» Frontiers in Plant Science, vol. 13, p. 992700, 2022.

[24] A. Sar, T. Choudhury, T. Sarkar y K. Kotecha, «Papayafreshnet: a hybrid deep learning framework for non-destructive freshness classification of papayas using convolutional and transformer networks,» Discover Food, vol. 5, nº 1, p. 97, 2025.

[25] S. K. Behera, A. K. Rath y P. K. Sethy, «Maturity status classification of papaya fruits based on machine learning and transfer learning approach,» Information Processing in Agriculture, vol. 8, nº 2, pp. 244-250, 2021.

[26] S. N. Appe, G. Arulselv y G. Balaji, «Tomato ripeness detection and classification using VGG based CNN models,» International Journal of Intelligent Systems and Applications in Engineering, vol. 11, nº 1, pp. 296-302, 2023.

AUTORIZACIÓN DE PUBLICACIÓN EN EL REPOSITORIO INSTITUCIONAL

Jerick David Llangari Peñaranda portador(a) de la cédula de ciudadanía N.º **0957152804**. En calidad de autor/a y titular de los derechos patrimoniales del proyecto de titulación **"Diseño de un sistema de control de calidad con redes neuronales para clasificar guineo según su estado"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos y no comerciales. Autorizo además a la Universidad Católica de Cuenca, para que realice la publicación de este proyecto de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

La Troncal**, 27 de octubre de 2025**

F: ……

Jerick David Llangari Peñaranda

C.I. **0957152804**