



UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL
CARRERA DE MECATRÓNICA

**IMPLEMENTACIÓN DE UN SISTEMA DE CLASIFICACIÓN DE
MANZANAS MEDIANTE VISIÓN ARTIFICIAL PARA CONTROL DE
CALIDAD**

Trabajo de titulación previo a la obtención del
Título de Ingeniero en Mecatrónica

AUTORES: Pedro Josue Mite Vera
Roberto Isaías Pauta Villacres
TUTOR: Ing. Alberto Santiago Ramírez Farfán M.Sc.

Guayaquil - Ecuador
2026

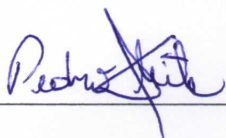
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, **Pedro Josue Mite Vera** con documento de identificación N° **0941470668** y **Roberto Isaías Pauta Villacres** con documento de identificación N° **1250249016**; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo.

Guayaquil, 18 de febrero del año 2026

Atentamente,



Pedro Josue Mite Vera
0941470668



Roberto Isaías Pauta Villacres
1250249016

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, **Pedro Josue Mite Vera** con documento de identificación N° **0941470668** y **Roberto Isaías Pauta Villacres** con documento de identificación N° **1250249016**, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del **IMPLEMENTACIÓN DE UN SISTEMA DE CLASIFICACIÓN DE MANZANAS MEDIANTE VISIÓN ARTIFICIAL PARA CONTROL DE CALIDAD**, el cual ha sido desarrollado para optar por el título de: Ingeniero en Mecatrónica, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

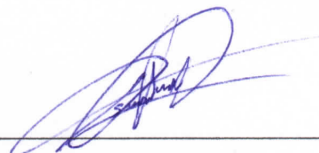
En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 18 de febrero del año 2026

Atentamente,



Pedro Josue Mite Vera
0941470668



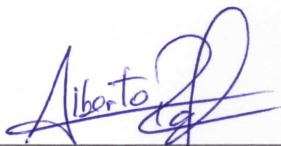
Roberto Isaías Pauta Villacres
1250249016

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, **Alberto Santiago Ramírez Farfán**, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **IMPLEMENTACIÓN DE UN SISTEMA DE CLASIFICACIÓN DE MANZANAS MEDIANTE VISIÓN ARTIFICIAL PARA CONTROL DE CALIDAD**, realizado por **Pedro Josue Mite Vera** con documento de identificación N° **0941470668** y por **Roberto Isaías Pauta Villacres** con documento de identificación N° **1250249016**, obteniendo como resultado final el trabajo de titulación bajo la opción **Dispositivo Tecnológico** que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 18 de febrero del año 2026

Atentamente,



Ing. Alberto Santiago Ramírez Farfán M.Sc.

0923348890

DEDICATORIA

Este trabajo de titulación está dedicado, en primer lugar, a Dios, por darme la vida, la fortaleza y la sabiduría necesarias para culminar esta etapa tan importante en mi formación profesional. En los momentos de duda y cansancio, siempre encontré en Él la fe para continuar.

A mi padre, PEDRO JOSE MITE ALBAN, quien ha sido el ejemplo más grande de esfuerzo, responsabilidad y carácter en mi vida. Papá, gracias por enseñarme que los sueños se construyen con disciplina, constancia y trabajo honesto. Tu firmeza y tu manera de enfrentar la vida me han inspirado a no rendirme ante las dificultades. Este logro también te pertenece, porque tu apoyo ha sido fundamental en cada paso que he dado.

A mi madre, NURIA MARISOL VERA ZAMORA, por ser mi mayor motivación y el corazón que sostiene mi vida. Mamá, gracias por tu amor incondicional, por tus palabras de aliento y por creer en mí incluso cuando yo mismo dudaba. Tu paciencia, tu sacrificio y tu ternura han sido el motor que me impulsó a seguir adelante.

A mi hermana, MAYBELLINE CATHERINE MITE VERA, por su apoyo constante y su compañía sincera. Gracias por tu confianza, por tu alegría y por estar presente en cada momento importante. Tu respaldo ha sido una fuerza silenciosa que siempre me acompañó.

A mi tía, MARIA TERESA MITE ALBAN, por su cariño, sus consejos y su apoyo incondicional. Gracias por motivarme a superarme y por confiar en mis capacidades en todo momento.

A mi abuela, TERESITA LUISA ALBAN QUIÑONES, por su amor infinito y sus enseñanzas llenas de valores. Abuelita, gracias por sus oraciones, por su dulzura y por inculcarme la humildad y la fe que hoy forman parte de quien soy.

Este logro no es solo mío; es el resultado del amor, la confianza y el apoyo de cada uno de ustedes. Con profunda gratitud y orgullo, les dedico este sueño cumplido.

Pedro Josue Mite Vera

Dedico este trabajo a mi padre, **Roberto Carlos Pauta Rios**, por su inquebrantable apoyo, ejemplo de trabajo y amor incondicional que han sido el pilar de mi formación.

A mi madre, **Daysi Fabiola Villacres Beltran**, por su sacrificio, paciencia y fe en mí, impulsándome siempre a alcanzar mis metas con integridad y esfuerzo.

Este logro es también de ustedes.

Roberto Isaías Pauta Villacres

AGRADECIMIENTO

Agradezco profundamente a Dios por ser mi guía constante, por darme la fortaleza necesaria en los momentos de dificultad y por iluminar mi camino a lo largo de este proceso académico. Su presencia en mi vida me permitió mantener la fe y la esperanza cuando los desafíos parecían más grandes que mis fuerzas.

Asimismo, expreso mi más sincero agradecimiento a mi familia, quienes han sido mi apoyo incondicional en cada etapa de mi formación. Gracias por su amor, su confianza, sus consejos y su paciencia. Su respaldo, sus palabras de aliento y su sacrificio han sido fundamentales para alcanzar esta meta. Este logro es también de ustedes, porque sin su apoyo constante no habría sido posible llegar hasta aquí.

Pedro Josue Mite Vera

Expreso mi más sincero agradecimiento a la **Universidad Politécnica Salesiana**, por haberme brindado la oportunidad de formarme en sus aulas y por ser el escenario de mi crecimiento profesional y personal.

A mis docentes de la carrera de Ingeniería Mecatrónica, por compartir sus conocimientos y experiencias, guiándome con exigencia y dedicación a lo largo de este camino académico.

A mis compañeros y amigos, con quienes compartí largas jornadas de estudio, desafíos y momentos inolvidables que hicieron más llevadera esta etapa universitaria.

Finalmente, agradezco a Dios por darme la fortaleza y sabiduría necesarias para culminar este proyecto y alcanzar esta meta tan importante en mi vida.

Roberto Isaías Pauta Villacres

RESUMEN

El presente trabajo de titulación describe la implementación de un sistema de clasificación automática de manzanas mediante visión artificial, diseñado para optimizar el control de calidad en la industria agroalimentaria. La inspección manual tradicional presenta limitaciones significativas en términos de subjetividad, fatiga del operario e inconsistencia en los criterios de selección, lo que afecta la competitividad del producto final.

Para la implementación del prototipo mecatrónico, se desarrolló una estación de adquisición de imágenes controlada, equipada con una cámara Logitech C922 y un sistema de iluminación difusa. El procesamiento de datos se realizó mediante algoritmos de aprendizaje profundo (*Deep Learning*), evaluando comparativamente tres arquitecturas de redes neuronales convolucionales: MobileNetV2, EfficientNet-B0 y ConvNeXt-Nano. Se construyó un dataset híbrido combinando imágenes locales con repositorios externos, aplicando técnicas de aumento de datos y la función de pérdida *Focal Loss* para mitigar el desbalance de clases.

Los resultados experimentales determinaron que el modelo **ConvNeXt-Nano** ofreció el rendimiento superior, alcanzando una exactitud del **99.27 %** y un F1-Score de 0.99, superando a las otras arquitecturas en la detección de defectos sutiles. El sistema final, desplegado en una unidad de procesamiento con GPU NVIDIA GTX 1650, opera en tiempo real con una latencia promedio de 35 ms por cuadro, validando su viabilidad técnica para la automatización de procesos de selección.

Palabras claves: visión artificial, clasificación de manzanas, redes neuronales convolucionales, ConvNeXt, control de calidad, aprendizaje profundo, GPU, procesamiento de imágenes.

ABSTRACT

This degree project describes the implementation of an automatic apple classification system using artificial vision, designed to optimize quality control in the agri-food industry. Traditional manual inspection presents significant limitations in terms of subjectivity, operator fatigue, and inconsistency in selection criteria, which affects the competitiveness of the final product.

For the implementation of the mechatronic prototype, a controlled image acquisition station was developed, equipped with a Logitech C922 camera and a diffuse lighting system. Data processing was carried out using Deep Learning algorithms, comparatively evaluating three convolutional neural network architectures: MobileNetV2, EfficientNet-B0, and ConvNeXt-Nano. A hybrid dataset was constructed by combining local images with external repositories, applying data augmentation techniques and the Focal Loss function to mitigate class imbalance.

Experimental results determined that the **ConvNeXt-Nano** model offered superior performance, achieving an accuracy of **99.27 %** and an F1-Score of 0.99, outperforming the other architectures in detecting subtle defects. The final system, deployed on a processing unit with an NVIDIA GTX 1650 GPU, operates in real-time with an average latency of 35 ms per frame, validating its technical feasibility for the automation of selection processes.

Keywords: artificial vision, apple classification, convolutional neural networks, ConvNeXt, quality control, deep learning, GPU, image processing.

ÍNDICE

I.	PROBLEMA	1
II.	JUSTIFICACIÓN	3
III.	OBJETIVOS	4
III-A.	Objetivo general	4
III-B.	Objetivos específicos	4
IV.	Fundamentos Teóricos	5
IV-A.	Visión Artificial (Computer Vision)	5
IV-A1.	La Imagen Digital: Representación Matricial y Cuantización	5
IV-A2.	Clasificación de Imágenes vs. Detección de Objetos	6
IV-B.	Procesamiento Digital de Imágenes (PDI)	7
IV-B1.	Espacios de Color: Limitaciones del RGB y Ventajas del HSV	7
IV-B2.	Técnicas de Segmentación y Extracción de ROI	8
IV-C.	Redes Neuronales Convolucionales (CNN)	8
IV-C1.	La Operación de Convolución Matemática	8
IV-C2.	Capas Funcionales de la Arquitectura	9
IV-D.	Arquitecturas de Modelos Utilizados	10
IV-D1.	MobileNetV2: Eficiencia mediante Convoluciones Separables	10
IV-D2.	EfficientNet-B0: El Método de Escalamiento Compuesto	10
IV-D3.	ConvNeXt (Nano): Modernización hacia los Transformers	10
IV-E.	Técnicas de Entrenamiento e Ingeniería de Datos	11
IV-E1.	Transfer Learning (Aprendizaje por Transferencia)	11
IV-E2.	Data Augmentation (Aumento de Datos)	11
IV-E3.	Función de Pérdida Focal (Focal Loss)	11
IV-E4.	Optimizador AdamW y Scheduler	12
IV-F.	Herramientas de Software (Stack Tecnológico)	12
IV-F1.	Librerías de Visión e IA	12
IV-F2.	Librerías de Análisis y Visualización	12
IV-F3.	Interfaz Gráfica (Tkinter)	12
IV-G.	Hardware de Implementación	13
IV-G1.	Sensor de Captura: Logitech C922	13
IV-G2.	Unidad de Procesamiento: NVIDIA GTX 1650	13
V.	Marco Metodológico	14
V-A.	Diseño de la Investigación	14
V-B.	Fase 1: Diseño del Sistema de Adquisición (Hardware)	15
V-B1.	Arquitectura Física del Sistema	15
V-B2.	Configuración del Sensor de Captura	16
V-C.	Fase 2: Ingeniería de Datos (Dataset Híbrido)	16
V-C1.	Fuente 1: Adquisición Local (Dataset Primario)	16
V-C2.	Fuente 2: Repositorios Externos (Enriquecimiento)	16
V-C3.	Protocolo de Etiquetado y Unificación	16
V-C4.	Preprocesamiento de Tensores	17
V-C5.	Aumento de Datos (Data Augmentation)	17
V-D.	Fase 3: Implementación y Entrenamiento Comparativo de Modelos	18
V-D1.	Entorno de Cómputo (Setup)	18
V-D2.	Protocolo de Entrenamiento para MobileNetV2	18

	V-D3.	Protocolo de Entrenamiento para EfficientNet-B0	18
	V-D4.	Protocolo de Entrenamiento para ConvNeXt-Nano	19
V-E.		Fase 4: Desarrollo del Software de Control (HMI)	19
	V-E1.	Arquitectura del Software	19
	V-E2.	Algoritmo de Visión Híbrido (Pipeline)	20
	V-E3.	Interfaz Humano-Máquina (HMI)	22
V-F.		Fase 5: Protocolo de Validación Final	23
VI.		ANÁLISIS E INTERPRETACIÓN DE RESULTADOS	24
	VI-A.	Análisis del Proceso de Entrenamiento	24
		VI-A1. Comportamiento de la Función de Pérdida	24
	VI-B.	Evaluación de Métricas de Clasificación	25
		VI-B1. Análisis de Matrices de Confusión	25
		VI-B2. Reporte Métrico Comparativo	28
	VI-C.	Análisis de Latencia y Tiempo Real	29
	VI-D.	Validación Funcional del Prototipo	29
		VI-D1. Pruebas de Robustez	29
	VI-E.	Discusión de Resultados	31
	VI-F.	Cronograma de actividades	32
	VI-G.	Presupuesto estimado	33
VII.		Conclusiones	34
VIII.		Recomendaciones	35
		Referencias	36
		Anexo A: código entrenamiento mobilenetv2	39
		Anexo B: código entrenamiento efficientnet-b0	44
		Anexo C: código entrenamiento convnext	49

ÍNDICE DE FIGURAS

1.	Representación numérica matricial de una imagen digital de 8 bits.	6
2.	Diferencia conceptual entre clasificación de imágenes y detección de objetos.	6
3.	Representación geométrica del espacio de color HSV utilizado para la segmentación.	7
4.	Arquitectura general de una Red Neuronal Convolucional.	9
5.	Comparativa técnica de las arquitecturas evaluadas en el proyecto.	11
6.	Conjunto de herramientas de software utilizadas en el desarrollo.	12
7.	Arquitectura de procesamiento paralelo utilizada para la aceleración de inferencia.	13
8.	Diagrama de flujo detallado de las fases metodológicas del proyecto.	14
9.	Estación de trabajo experimental y disposición de componentes.	15
10.	Parámetros de configuración del sensor óptico en OpenCV.	16
11.	Ejemplos representativos de las clases del dataset híbrido.	17
12.	Especificaciones del entorno de cómputo para entrenamiento.	18
13.	Definición de hiperparámetros en el script de entrenamiento de ConvNeXt.	19
14.	Arquitectura multihilo del sistema de control.	19
15.	Algoritmo híbrido de detección y clasificación.	21
16.	Interfaz de usuario desarrollada para el control de calidad.	22
17.	Estructura teórica de la matriz de confusión utilizada para la validación.	23
18.	Comparativa de curvas de pérdida y exactitud durante el entrenamiento en MobileNetV2.	24
19.	Comparativa de curvas de pérdida y exactitud durante el entrenamiento en EfficientNet-B0.	25
20.	Comparativa de curvas de pérdida y exactitud durante el entrenamiento en ConvNeXt.	25
21.	Matrices de confusión MobileNetV2.	26
22.	Matrices de confusión EfficientNet-B0.	27
23.	Matrices de confusión ConvNeXt.	28
24.	Comparación gráfica del F1-Score entre arquitecturas.	29
25.	Pruebas de validación fruto optimo.	30
26.	Pruebas de validación fruto no conforme.	30
27.	Pruebas de validación entorno difícil.	31
28.	Ecosistema de librerías y herramientas de software utilizadas.	32
29.	codigo de entrenamiento.	39
30.	codigo de entrenamiento.	39
31.	codigo de entrenamiento.	40
32.	codigo de entrenamiento.	40
33.	codigo de entrenamiento.	41
34.	codigo de entrenamiento.	41
35.	codigo de entrenamiento.	42
36.	codigo de entrenamiento.	42
37.	codigo de entrenamiento.	43
38.	codigo de entrenamiento.	43
39.	codigo de entrenamiento efficientnet-b0.	44
40.	codigo de entrenamiento efficientnet-b0.	44
41.	codigo de entrenamiento efficientnet-b0.	45
42.	codigo de entrenamiento efficientnet-b0.	46
43.	codigo de entrenamiento efficientnet-b0.	46
44.	codigo de entrenamiento efficientnet-b0.	46
45.	codigo de entrenamiento efficientnet-b0.	46
46.	codigo de entrenamiento efficientnet-b0.	46
47.	codigo de entrenamiento efficientnet-b0.	46
48.	codigo de entrenamiento efficientnet-b0.	46
49.	codigo de entrenamiento efficientnet-b0.	47

50.	codigo de entrenamiento efficientnet-b0.	47
51.	codigo de entrenamiento efficientnet-b0.	48
52.	codigo de entrenamiento efficientnet-b0.	48
53.	codigo de entrenamiento convnext.	49
54.	codigo de entrenamiento convnext.	49
55.	codigo de entrenamiento convnext.	50
56.	codigo de entrenamiento convnext.	50
57.	codigo de entrenamiento convnext.	51
58.	codigo de entrenamiento convnext.	51
59.	codigo de entrenamiento convnext.	52
60.	codigo de entrenamiento convnext.	52
61.	codigo de entrenamiento convnext.	53
62.	codigo de entrenamiento convnext.	53
63.	codigo de entrenamiento convnext.	54

ÍNDICE DE TABLAS

I.	Definición Técnica de Clases	17
II.	Comparativa de Rendimiento en Conjunto de Prueba	28
III.	Presupuesto estimado del proyecto	33

I. PROBLEMA

En Ecuador, la industria frutícola aún enfrenta desafíos significativos en la modernización de sus procesos, especialmente en la clasificación de manzanas, donde el método visual manual sigue siendo predominante, incluso en entornos tecnificados. Esta dependencia del juicio humano introduce serias limitaciones en términos de consistencia, precisión y velocidad, afectando directamente la uniformidad del producto final y, por consiguiente, la competitividad de los productores en mercados cada vez más exigentes [1]. Este problema se agrava en pequeñas agroindustrias, donde los recursos tecnológicos son escasos y la selección del producto depende casi exclusivamente de operarios con escasa formación técnica, lo cual genera errores y variabilidad que no cumplen con los estándares internacionales de calidad [2]. Además de estas limitaciones, el proceso manual representa un alto costo operativo a largo plazo. Los errores de clasificación resultan en lotes rechazados, reprocesamiento, disminución del valor comercial e incluso devoluciones del producto, afectando la economía del productor. La falta de automatización también genera problemas ergonómicos al exigir esfuerzos repetitivos durante largas jornadas laborales, situación que incrementa la fatiga y disminuye la precisión de los trabajadores [3].

Esta situación es particularmente crítica en América Latina, y especialmente en Ecuador, donde una parte importante de la producción frutícola es gestionada por pequeños y medianos productores que enfrentan dificultades para acceder a tecnologías avanzadas. Según datos del Ministerio de Agricultura y Ganadería, el sector frutícola ecuatoriano presenta una baja tecnificación en sus procesos postcosecha, lo que limita la competitividad de productos como la manzana en mercados internacionales. La ausencia de soluciones tecnológicas accesibles y adaptables impide su inserción en mercados más competitivos, lo que mantiene condiciones de baja rentabilidad en el sector [4]. A nivel regional también se observa una escasa disponibilidad de estudios aplicados a la realidad tropical o andina, donde factores como la luz ambiental, la textura de la cáscara o los patrones de deterioro pueden diferir significativamente de los modelos utilizados en países desarrollados [5]. Esta carencia de adaptaciones locales limita la efectividad de las herramientas que podrían ser adoptadas por los productores ecuatorianos [6].

En zonas agrícolas como Tungurahua, Imbabura o Chimborazo, donde se cultiva manzana para consumo interno y comercio local, la inspección postcosecha se realiza con base en la experiencia del trabajador, sin herramientas objetivas que validen la calidad del fruto. Este modelo, común en centros de acopio comunitarios, no garantiza uniformidad en el producto entregado, lo que reduce su valor comercial. En ferias agrícolas y canales cortos de comercialización, esta situación se traduce en bajos precios pagados al productor, desincentivando la inversión en mejora tecnológica. La falta de estandarización también impide implementar políticas de trazabilidad, necesarias para cumplir con normativas de exportación y control sanitario [7]. En muchos casos, las deficiencias en los procesos de clasificación visual son agravadas por el escaso acceso a recursos educativos y tecnológicos, especialmente en sectores rurales [8]. Esta brecha no solo limita el desarrollo técnico del personal, sino que provoca un ciclo de dependencia del trabajo manual sin herramientas modernas de inspección [9].

Otra de las barreras técnicas importantes es la falta de formación especializada en herramientas de visión artificial. Muchos estudiantes, técnicos y operarios en Ecuador carecen de conocimientos en bibliotecas como OpenCV, Keras o TensorFlow, lo cual restringe la transferencia del conocimiento desde el entorno académico hacia el sector productivo [10]. Esta brecha impide que los avances logrados en entornos universitarios o de investigación sean aplicados en condiciones reales de producción [11]. Sumado a ello, los prototipos existentes no contemplan condiciones variables del entorno real de producción, como iluminación inestable, fruta no homogénea o manipulación manual, lo que reduce su aplicabilidad fuera del laboratorio [12]. La falta de modelos de referencia desarrollados localmente también representa un obstáculo para adaptar los sistemas a la realidad ecuatoriana, donde las condiciones de cosecha, clima y almacenamiento difieren notablemente de los estándares en los que se entrenan los algoritmos tradicionales [13]. Incluso en instituciones educativas, la falta de equipamiento e inversión en laboratorios funcionales impide la validación práctica de proyectos de investigación aplicados [14].

En cuanto a la tecnología, uno de los enfoques más prometedores para resolver este problema es el uso de visión artificial, especialmente mediante redes neuronales convolucionales (CNN). Estas técnicas permiten automatizar la inspección visual mediante modelos entrenados con imágenes, logrando reconocer defectos con gran precisión y sin fatiga. Estudios recientes demuestran que sistemas basados en CNN pueden clasificar frutas con precisión superior al 95 %, superando la capacidad de discriminación del ojo humano en términos de textura, simetría y color [15]. Además, estas soluciones permiten generar trazabilidad digital en tiempo real, lo que facilita auditorías, mejora la toma de decisiones y fortalece el sistema de gestión de calidad [16]. No obstante, la mayoría de los desarrollos se quedan en fases experimentales o dependen de equipamiento costoso, lo que dificulta su adopción por pequeños productores o instituciones educativas [17]. La implementación de estos sistemas en Ecuador requiere adaptación técnica, accesibilidad económica y acompañamiento en capacitación para su correcta utilización [18].

Frente a este panorama, se vuelve imperante el diseño e implementación de un sistema, funcional y replicable, capaz de clasificar manzanas en función de su estado superficial mediante visión artificial. Este sistema debe ser económico, adaptable a condiciones reales de producción en Ecuador, y pensado para ser utilizado tanto en contextos académicos como en pequeñas unidades de producción, promoviendo la innovación tecnológica local y contribuyendo a cerrar la brecha entre teoría y práctica [19].

II. JUSTIFICACIÓN

El presente proyecto surge como una respuesta tecnológica a los desafíos actuales del sector frutícola ecuatoriano, particularmente en la necesidad de garantizar estándares de calidad mediante métodos modernos, eficientes y accesibles. En Ecuador, la producción de frutas como las manzanas representa una oportunidad de desarrollo económico para muchas regiones, pero enfrenta problemas estructurales como la baja tecnificación, la alta dependencia del trabajo manual y la variabilidad en los procesos de inspección visual [20, 21]. Estas deficiencias afectan directamente la competitividad del producto en el mercado, donde los consumidores y los compradores mayoristas exigen uniformidad, buena apariencia y trazabilidad. La implementación de sistemas de clasificación por visión artificial resulta altamente pertinente, ya que permite superar las limitaciones del juicio humano, ofreciendo una evaluación objetiva, constante y en tiempo real del estado superficial de las frutas. En un entorno donde la decisión de compra está condicionada por la apariencia visual, contar con tecnología que garantice una selección precisa puede reducir significativamente las pérdidas por rechazo comercial [22, 23]. Esto no solo optimiza la eficiencia operativa, sino que también abre puertas a mercados internacionales más exigentes, donde los productos ecuatorianos podrían ingresar si se cumplen estándares de inspección más rigurosos [24]. Además, la clasificación automatizada permite recolectar datos históricos del producto procesado, lo que ayuda a mejorar la trazabilidad, detectar patrones de defectos frecuentes y aplicar correctivos tempranos en el cultivo o poscosecha. Este nivel de control y retroalimentación puede transformar los sistemas de calidad existentes en modelos predictivos de mayor rendimiento [25].

Desde la ingeniería mecatrónica, este tipo de solución representa una integración de múltiples disciplinas: sensores de captura de imagen, procesamiento digital, redes neuronales, microcontroladores y sistemas embebidos que ejecutan algoritmos en tiempo real. La propuesta exige un enfoque sistemático de diseño, implementación y validación bajo entornos reales, lo cual se alinea con los principios de la automatización industrial flexible aplicada al agro [26, 27]. El uso de herramientas como Python y bibliotecas como OpenCV facilita la implementación del sistema sin incurrir en altos costos, promoviendo una alternativa económica, replicable y adaptable a distintas realidades productivas [28]. Además, el enfoque metodológico del proyecto permite generar conocimiento científico aplicable a condiciones reales, ya que considera parámetros locales como iluminación ambiental variable, morfología de la fruta y adaptabilidad a condiciones físicas no uniformes. Esto refuerza el aporte de la ingeniería mecatrónica al aplicar soluciones robustas y eficientes bajo criterios de diseño modular, escalabilidad y validación empírica [29, 30].

A nivel social, económico y ambiental, el uso de sistemas de clasificación contribuye a reducir la desigualdad estructural entre grandes y pequeños productores, promoviendo el valor agregado desde el origen de la cadena productiva, impulsando el desarrollo local en zonas donde la actividad agrícola es la principal fuente de ingresos [31]. Esta tecnología permite que familias productoras obtengan mejores precios y disminuyan pérdidas, fortaleciendo su sostenibilidad económica. Disminuir el descarte innecesario de frutas por errores humanos contribuye a reducir el desperdicio alimentario, lo cual mejora la eficiencia en el uso de recursos naturales como el agua, la energía y el suelo [32]. Además, al tratarse de un sistema compacto, de bajo consumo energético y orientado a reducir residuos, se alinea con principios de sostenibilidad agroindustrial [33, 34] y con objetivos globales como la disminución de la huella de carbono [35].

III. OBJETIVOS

III-A. Objetivo general

Implementar un sistema de clasificación de manzanas mediante visión artificial para control de calidad identificando manzanas con defectos visibles o en mal estado

III-B. Objetivos específicos

- Implementar un sistema de adquisición de imágenes para su posterior clasificación considerando un entorno controlado.
- Diseñar un algoritmo de inteligencia artificial basado en redes neuronales convolucionales identificando manzanas con defectos visuales o en mal estado
- Validar la efectividad de clasificación del sistema mediante pruebas controladas con manzanas en distintas condiciones físicas.

IV. FUNDAMENTOS TEÓRICOS

El presente capítulo desarrolla la fundamentación científica, matemática y tecnológica necesaria para la implementación del sistema de clasificación de manzanas. Se detallan exhaustivamente los principios de la visión artificial, el procesamiento digital de señales, la arquitectura interna de las redes neuronales convolucionales y el ecosistema de herramientas de software y hardware seleccionadas para el proyecto. El objetivo es proporcionar una base teórica sólida que justifique las decisiones de ingeniería tomadas para garantizar la robustez del sistema en un entorno de control de calidad.

IV-A. Visión Artificial (*Computer Vision*)

La visión artificial es una rama de la inteligencia artificial cuyo objetivo es dotar a las computadoras de la capacidad de interpretar y comprender el entorno visual a partir de imágenes digitales o secuencias de video. Mientras que la visión biológica es un proceso evolutivo complejo, subjetivo y cualitativo, la visión artificial busca obtener datos cuantitativos, métricos y objetivos para la automatización de procesos industriales. Su aplicación permite transformar la información lumínica del mundo físico en decisiones lógicas ejecutables por una máquina [36].

IV-A1. La Imagen Digital: Representación Matricial y Cuantización: Desde el punto de vista computacional, una imagen no es una fotografía en el sentido analógico, sino una representación numérica bidimensional de una escena que ha sido sometida a dos procesos fundamentales: muestreo (discretización espacial) y cuantización (discretización de amplitud).

Matemáticamente, una imagen se define como una función $f(x, y)$, donde x e y son coordenadas espaciales discretas y el valor de la función en ese punto representa la intensidad lumínica. En este proyecto, las imágenes capturadas se tratan como matrices tridimensionales (tensores) de dimensiones $H \times W \times C$.

- H (Height): Altura en píxeles (configurada a 720 píxeles).
- W (Width): Anchura en píxeles (configurada a 1280 píxeles).
- C (Channels): Canales de color, que en el espacio RGB corresponden a 3 planos de información (Rojo, Verde, Azul).

Cada elemento de esta matriz, denominado píxel, se almacena como un entero de 8 bits sin signo (`uint8`). Esto permite representar $2^8 = 256$ niveles de intensidad por canal, donde 0 representa la ausencia total de luz (negro) y 255 la saturación total (blanco o color puro). El procesamiento de estas matrices implica millones de operaciones aritméticas por segundo, razón por la cual se emplean técnicas de aceleración por hardware.



Figura 1. Representación numérica matricial de una imagen digital de 8 bits.

IV-A2. *Clasificación de Imágenes vs. Detección de Objetos*: Es fundamental distinguir entre las diferentes jerarquías de tareas que un sistema de visión puede realizar, ya que esto define la arquitectura de la red neuronal a implementar:

- **Clasificación (Image Classification)**: El algoritmo recibe la imagen completa de la manzana como entrada y asigna una etiqueta global (ej. "Óptima." "No Conforme") con un nivel de confianza asociado. Este es el enfoque principal de las redes neuronales (EfficientNet, MobileNet, ConvNeXt) utilizadas en este trabajo, donde la red aprende a discriminar texturas globales.
- **Detección y Localización (Object Detection)**: Implica no solo clasificar, sino encontrar la posición espacial del objeto dentro de la imagen, entregando coordenadas (x, y, w, h) de una caja delimitadora. En este proyecto, se implementa un enfoque híbrido: la localización se realiza mediante visión clásica (OpenCV) para recortar la fruta, y la clasificación se delega a la red neuronal. Esto optimiza el recurso computacional al evitar que la red neuronal procese el fondo irrelevante de la imagen.

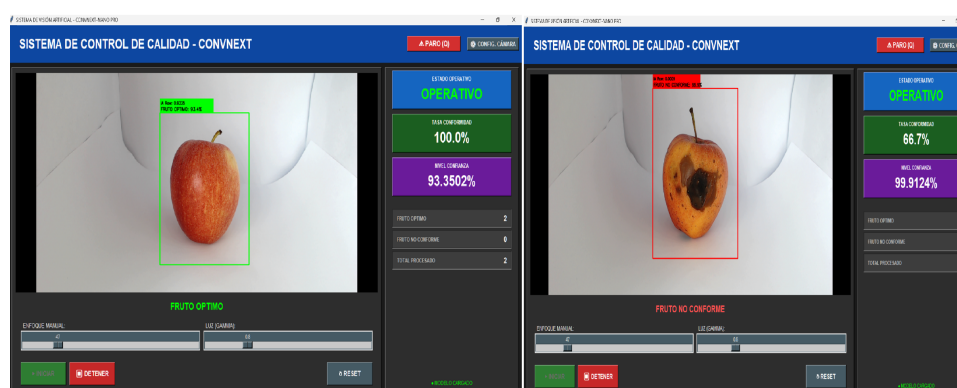


Figura 2. Diferencia conceptual entre clasificación de imágenes y detección de objetos.

IV-B. Procesamiento Digital de Imágenes (PDI)

El PDI abarca el conjunto de técnicas matemáticas aplicadas a la imagen cruda para mejorar su calidad, reducir el ruido o extraer características relevantes antes del análisis profundo. Es la etapa de acondicionamiento de la señal visual.

IV-B1. Espacios de Color: Limitaciones del RGB y Ventajas del HSV: La cámara captura las imágenes utilizando el modelo aditivo **RGB** (Rojo, Verde, Azul), diseñado para la emisión de luz en pantallas. Sin embargo, este modelo no es robusto ante cambios de iluminación en entornos industriales. En el modelo RGB, la información de crominancia (color) y luminancia (brillo) está altamente correlacionada; por ejemplo, una sombra sobre una manzana roja disminuirá los valores de R, G y B simultáneamente, dificultando la segmentación por umbrales simples.

Para solucionar esto, se utiliza el espacio de color **HSV** (*Hue, Saturation, Value*), que modela el color de manera cilíndrica, separando la información cromática de la intensidad lumínica:

- **Hue (Matiz):** Representa el espectro de color puro en un ángulo de 0° a 360° . Es la característica más estable para diferenciar una manzana (roja o verde) del fondo negro, independientemente de si la iluminación es tenue o intensa.
- **Saturation (Saturación):** Indica qué tan puro es el color (radio del cilindro). Permite diferenciar entre un rojo vivo y un rojo "lavado." pálido, lo cual puede ser indicativo de madurez o enfermedad.
- **Value (Valor):** Representa el brillo o intensidad de luz (eje vertical). Al aislar este canal, los algoritmos de segmentación pueden ignorar las variaciones de iluminación y centrarse solo en el color (H) y la pureza (S).



Figura 3. Representación geométrica del espacio de color HSV utilizado para la segmentación.

IV-B2. Técnicas de Segmentación y Extracción de ROI: La segmentación es el proceso de particionar una imagen digital en múltiples segmentos (conjuntos de píxeles). En este sistema, se utiliza la técnica de **Umbralización** (*Thresholding*) en el canal HSV. Se definen límites superior e inferior para el color de la manzana; cualquier píxel dentro de este rango se convierte en blanco (1) y el resto en negro (0), creando una máscara binaria.

Posteriormente, se aplican algoritmos de detección de contornos basados en la búsqueda de gradientes topológicos (`cv2.findContours`). El sistema identifica el contorno cerrado de mayor área, asumiendo que corresponde a la fruta. Una vez obtenidas las coordenadas, se extrae la **Región de Interés (ROI)**. Esta sub-imagen recortada contiene únicamente la información de la fruta y es redimensionada a 224×224 píxeles, la dimensión de entrada estándar para las redes neuronales, maximizando la densidad de información útil por píxel.

IV-C. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales son algoritmos de aprendizaje profundo diseñados específicamente para procesar datos con estructura de rejilla. Se inspiran en la organización del córtex visual animal, donde neuronas individuales responden a estímulos en campos receptivos solapados [37].

IV-C1. La Operación de Convolución Matemática: La operación fundamental que distingue a una CNN es la convolución discreta. Consiste en deslizar un filtro (o kernel) de tamaño $k \times k$ sobre la imagen de entrada I . En cada posición (i, j) , se calcula el producto punto entre los pesos del kernel K y los valores de los píxeles locales, sumando los resultados.

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n) \quad (1)$$

Esta operación permite extraer características locales invariantes a la posición. Los kernels actúan como detectores de características: en las primeras capas, detectan bordes verticales, horizontales y diagonales; a medida que la red se hace más profunda, estos detectores simples se combinan no linealmente para reconocer patrones complejos como texturas de pudrición, formas de tallos o magulladuras.

IV-C2. *Capas Funcionales de la Arquitectura:* Una CNN típica se compone de una secuencia de capas apiladas que transforman el volumen de entrada en un vector de probabilidades:

- **Capas Convolucionales:** Son los extractores de características principales. El número de filtros en cada capa determina la "profundidad" del mapa de características generado, permitiendo a la red aprender múltiples representaciones de la misma imagen.
- **Capas de Activación (No Linealidad):** Una red compuesta solo por convoluciones lineales sería equivalente a un solo filtro lineal gigante. Para aprender fronteras de decisión complejas, se introducen funciones no lineales.
 - **ReLU:** $f(x) = \max(0, x)$. Es rápida y evita el desvanecimiento del gradiente.
 - **GELU:** $x\Phi(x)$. Una versión suave de ReLU usada en modelos modernos como ConvNeXt.
- **Capas de Pooling (Submuestreo):** Su función es reducir progresivamente el tamaño espacial de la representación para disminuir la cantidad de parámetros y el costo computacional. El *Max Pooling* selecciona el valor máximo en una vecindad, proporcionando además una propiedad de invarianza a pequeñas traslaciones y distorsiones en la imagen de entrada.
- **Capas Densas (Fully Connected):** Ubicadas al final de la red, estas capas "aplanan" los mapas de características 2D en un vector 1D. Realizan el razonamiento de alto nivel, combinando todas las características extraídas para clasificar la imagen en una de las clases finales mediante una función Softmax.

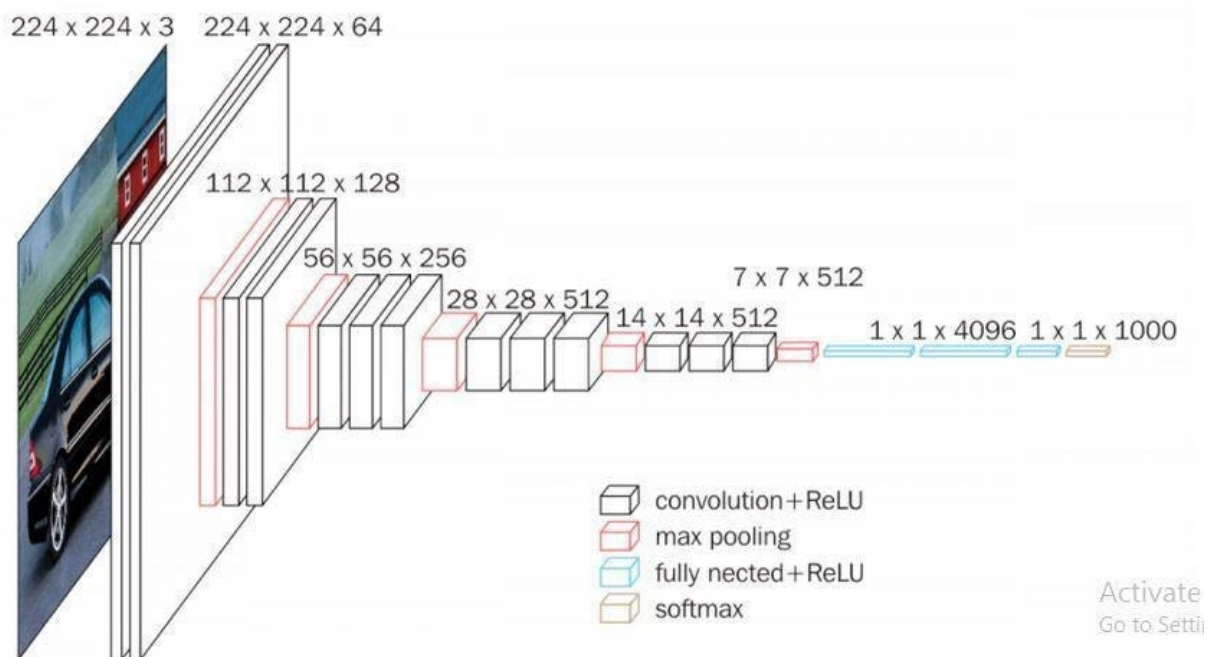


Figura 4. Arquitectura general de una Red Neuronal Convolucional.

IV-D. Arquitecturas de Modelos Utilizados

Se han implementado tres arquitecturas diferentes para evaluar su desempeño en un entorno industrial, cada una con una filosofía de diseño optimizada para diferentes restricciones de hardware y precisión.

IV-D1. MobileNetV2: Eficiencia mediante Convoluciones Separables: MobileNetV2 es una red diseñada por Google específicamente para dispositivos con recursos limitados. Su principal innovación es el uso de **Convoluciones Separables en Profundidad** (*Depthwise Separable Convolutions*). Una convolución estándar realiza el filtrado espacial y la combinación de canales en un solo paso, lo que es computacionalmente costoso ($D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$). MobileNetV2 divide esto en dos pasos: 1. **Depthwise Convolution:** Aplica un solo filtro por cada canal de entrada (filtrado espacial). 2. **Pointwise Convolution:** Aplica una convolución de 1×1 para combinar las salidas de los canales (combinación lineal). Esta factorización reduce el costo de computación en un factor de 8 a 9 veces comparado con convoluciones estándar, permitiendo inferencia en tiempo real en hardware modesto [38].

IV-D2. EfficientNet-B0: El Método de Escalamiento Compuesto: EfficientNet aborda el problema de cómo escalar una red neuronal para obtener mejor precisión. Tradicionalmente, se escalaba solo una dimensión: profundidad (ResNet), ancho (WideResNet) o resolución de imagen. EfficientNet introduce el **Escalamiento Compuesto** (*Compound Scaling*), que escala las tres dimensiones de la red (ancho, profundidad y resolución) de manera uniforme utilizando un coeficiente compuesto ϕ optimizado. Además, utiliza bloques MBConv con mecanismos de atención *Squeeze-and-Excitation* (SE), que permiten a la red recalibrar la importancia de cada canal de características, enfatizando las informaciones útiles y suprimiendo el ruido [39].

IV-D3. ConvNeXt (Nano): Modernización hacia los Transformers: ConvNeXt (2022) es una arquitectura puramente convolucional que integra las lecciones aprendidas de los *Vision Transformers* (ViT). Sus mejoras clave incluyen:

- **Kernels Grandes:** Utiliza filtros de 7×7 (en lugar de 3×3) para aumentar el campo receptivo efectivo de la red.
- **Layer Normalization:** Sustituye la Batch Normalization tradicional, mejorando la estabilidad del entrenamiento con batch sizes pequeños.
- **GELU:** Utiliza la función de activación *Gaussian Error Linear Unit*, que es más suave y probabilística que ReLU.

Estas características le permiten superar a arquitecturas previas en robustez y precisión, siendo capaz de detectar defectos sutiles en las manzanas con mayor fiabilidad [40].

```

# 2. PARÁMETROS GENERALES
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 20
DATASET_DIR = "dataset_final"

# 3. GENERADORES DE DATOS
# Aumento SOLO en entrenamiento
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=15,
    zoom_range=0.15,
)

class Config:
    MODEL_NAME = 'convnext_nano.in12k_ft_in1k'
    NUM_CLASSES = 1
    INPUT_SIZE = 224

    # Hiperparámetros ajustados para ConvNeXt
    BATCH_SIZE = 16 # Límite seguro para 4GB VRAM
    NUM_EPOCHS = 50 # ConvNeXt tarda un poco más en converge
    PATIENCE = 10 # Early Stopping

    # Receta de entrenamiento moderna (AdamW + Weight Decay)
    LEARNING_RATE = 4e-5
    WEIGHT_DECAY = 0.05

    # Ajustes de Hardware (Windows Safe)

class Config:
    DATA_DIR = Path('./dataset_224_ready')
    OUTPUT_DIR = Path('./resultados_entrenamiento_')

    MODEL_NAME = 'efficientnet_b0'
    NUM_CLASSES = 1
    INPUT_SIZE = 224

    BATCH_SIZE = 16

    NUM_EPOCHS = 50
    EARLY_STOPPING_PATIENCE = 7 # Si no mejora en 7 épocas, para

```

Figura 5. Comparativa técnica de las arquitecturas evaluadas en el proyecto.

IV-E. Técnicas de Entrenamiento e Ingeniería de Datos

Para lograr un modelo generalizable y robusto, se aplicaron técnicas avanzadas de entrenamiento supervisado que mitigan problemas comunes como el sobreajuste y el desbalance de clases.

IV-E1. Transfer Learning (Aprendizaje por Transferencia): El entrenamiento desde cero (*From Scratch*) requiere millones de imágenes y semanas de cómputo. Se utilizó *Transfer Learning*, inicializando los pesos de la red con valores pre-entrenados en el dataset masivo ImageNet (1.2 millones de imágenes). Esto permite que el modelo comience con la capacidad de reconocer formas, bordes y texturas básicas, requiriendo solo un “ajuste fino” (*Fine-Tuning*) de las últimas capas para especializarse en la detección de defectos en manzanas.

IV-E2. Data Augmentation (Aumento de Datos): Dado que el dataset recolectado es finito, existe el riesgo de *Overfitting* (memorización de los datos de entrenamiento). Se implementó un pipeline de aumento de datos que aplica transformaciones aleatorias a las imágenes durante el entrenamiento:

- **Rotación:** ± 15 grados para simular la orientación aleatoria de la fruta en la banda.
- **Zoom:** Factor de 0,15 para simular variaciones en la distancia a la cámara.
- **Desplazamiento:** Traslaciones horizontales y verticales.

Esto obliga a la red a aprender características invariantes y robustas, en lugar de memorizar píxeles específicos [41].

IV-E3. Función de Pérdida Focal (Focal Loss): En problemas de detección de defectos, suele haber un desbalance de clases significativo (muchas más frutas sanas que dañadas). La entropía cruzada estándar es ineficiente aquí, ya que el modelo puede lograr una alta exactitud global simplemente clasificando todo como “Sano”, ignorando los defectos. La **Focal Loss** introduce un factor modulador $(1 - p_t)^\gamma$ a la entropía cruzada. Este factor reduce el peso de los ejemplos fáciles (bien clasificados con alta confianza) y aumenta el peso de los ejemplos difíciles (defectos sutiles o ambiguos), centrando el aprendizaje donde más se necesita [42].

IV-E4. Optimizador AdamW y Scheduler: Se utilizó el optimizador **AdamW**, una variante de Adam que desacopla el decaimiento de pesos (*Weight Decay*) de la actualización del gradiente. Esto mejora la regularización y la capacidad de generalización del modelo. Además, se empleó un planificador *Cosine Annealing*, que reduce la tasa de aprendizaje siguiendo una curva coseno periódica, permitiendo al modelo converger suavemente en los mínimos locales de la función de pérdida y escapar de puntos de estancamiento [43].

IV-F. Herramientas de Software (Stack Tecnológico)

El desarrollo se fundamenta en el lenguaje **Python 3.10**, elegido por su liderazgo en el ecosistema de inteligencia artificial y su capacidad de integración con hardware mediante bindings de C++.

IV-F1. Librerías de Visión e IA:

- **OpenCV (Open Source Computer Vision):** Estándar industrial para visión por computador. Se utiliza para la captura de video eficiente, conversión de espacios de color y operaciones morfológicas.
- **PyTorch y TIMM:** PyTorch se empleó para los modelos EfficientNet y ConvNeXt debido a su flexibilidad y grafo dinámico, ideal para investigación. TIMM (*PyTorch Image Models*) es una librería que provee implementaciones de modelos SOTA altamente optimizadas y pesos pre-entrenados [44].
- **TensorFlow / Keras:** Utilizado para la implementación rápida del modelo MobileNetV2, aprovechando su API de alto nivel y facilidad de despliegue.

IV-F2. Librerías de Análisis y Visualización:

- **Pandas:** Esencial para la manipulación de datos estructurados, utilizada para gestionar los registros (logs) de producción y métricas de entrenamiento en formato CSV.
- **Matplotlib y Seaborn:** Librerías de graficación científica utilizadas para visualizar las curvas de aprendizaje (Pérdida vs Épocas) y generar matrices de confusión detalladas que permiten analizar los falsos positivos y negativos.
- **Scikit-Learn:** Provee herramientas estadísticas para el cálculo de métricas de evaluación robustas como F1-Score, Precision y Recall, así como para la estratificación del dataset.

IV-F3. Interfaz Gráfica (Tkinter): **Tkinter** es la biblioteca de GUI estándar de Python. Se utilizó para construir la consola de operación HMI, permitiendo la visualización del video en tiempo real. Su integración nativa con Python facilita el uso de hilos (*threading*) para desacoplar la renderización de la interfaz del bucle de inferencia de la IA, evitando que la ventana se congele durante el procesamiento pesado.



Figura 6. Conjunto de herramientas de software utilizadas en el desarrollo.

IV-G. Hardware de Implementación

IV-G1. Sensor de Captura: Logitech C922: El dispositivo de adquisición seleccionado es la cámara Logitech C922 Pro Stream.

- **Tecnología de Sensor:** CMOS de barrido progresivo, lo que permite una captura global de la escena con menor ruido térmico que los sensores CCD antiguos.
- **Tasa de Refresco:** Configurada a 60 FPS (cuadros por segundo) en resolución 720p. Esta alta tasa temporal es crítica en aplicaciones de bandas transportadoras para minimizar el desenfoque de movimiento (*motion blur*) que podría ocultar defectos.
- **Óptica:** Lente de vidrio con campo de visión de 78 grados, adecuado para cubrir el área de inspección a distancias cortas (30-50 cm).

IV-G2. Unidad de Procesamiento: NVIDIA GTX 1650: El cómputo intensivo de las convoluciones se delega a la GPU NVIDIA GeForce GTX 1650.

- **Arquitectura CUDA:** Dispone de 896 núcleos CUDA que permiten el procesamiento paralelo masivo de datos (SIMD - Single Instruction, Multiple Data). Mediante la plataforma CUDA Toolkit, las operaciones matriciales de PyTorch y TensorFlow se ejecutan directamente en la memoria de video (VRAM) de la tarjeta gráfica.
- **Impacto en Latencia:** El uso de la GPU reduce el tiempo de inferencia por imagen de cientos de milisegundos (en CPU) a aproximadamente 30-40 ms, habilitando la clasificación en tiempo real necesaria para la industria [45].

```

C:\Users\rpaut>nvidia-smi
Wed Feb 18 12:06:44 2026

```

NVIDIA-SMI 581.57		Driver Version: 581.57		CUDA Version: 13.0	
GPU	Name	Driver-Model	Bus-Id	Disp.A	Volatile
Fan	Temp	Pwr:Usage/Cap	Memory-Usage	Memory-Usage	Uncorr. ECC
	Perf				GPU-Util Compute M.
					MIG M.
0	NVIDIA GeForce GTX 1650	WDDM	00000000:01:00.0	Off	N/A
N/A	46C	4W / 50W	174MiB / 4096MiB		2%
	P8				Default
					N/A

Processes:						
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
ID	ID					
0	N/A	N/A	2756	C+G	...yb3d8bbwe\WindowsTerminal.exe	N/A
0	N/A	N/A	3908	C+G	..Browser\Application\brave.exe	N/A
0	N/A	N/A	4192	C+G	..indows\System32\ShellHost.exe	N/A
0	N/A	N/A	9616	C+G	..2txyewy\CrossDeviceResume.exe	N/A
0	N/A	N/A	10456	C+G	..Browser\Application\brave.exe	N/A
0	N/A	N/A	10640	C+G	...cw5n1h2txyewy\SearchHost.exe	N/A
0	N/A	N/A	10648	C+G	...y\StartMenuExperienceHost.exe	N/A
0	N/A	N/A	11748	C+G	...0.3719.115\msedgewebview2.exe	N/A

```

C:\Users\rpaut>

```

Figura 7. Arquitectura de procesamiento paralelo utilizada para la aceleración de inferencia.

V. MARCO METODOLÓGICO

El desarrollo del presente proyecto de titulación se rige bajo un enfoque de investigación aplicada y experimental de carácter cuantitativo. La metodología no se limita a la implementación de un único algoritmo, sino que abarca un estudio comparativo riguroso entre múltiples arquitecturas de inteligencia artificial para determinar la solución óptima en un entorno industrial. El proceso se ha estructurado en cinco fases secuenciales de ingeniería: diseño del sistema de adquisición, ingeniería de datos, ejecución de experimentos comparativos, selección de la arquitectura y desarrollo del software final.

V-A. *Diseño de la Investigación*

La investigación se define como **experimental**, dado que se manipulan variables independientes (tipo de arquitectura CNN, hiperparámetros, técnicas de regularización) para observar su efecto sobre variables dependientes críticas (precisión F1-Score, área bajo la curva ROC y latencia de inferencia). Asimismo, es de carácter **aplicada**, ya que utiliza conocimientos teóricos de visión por computador para resolver la problemática de clasificación de calidad en manzanas.

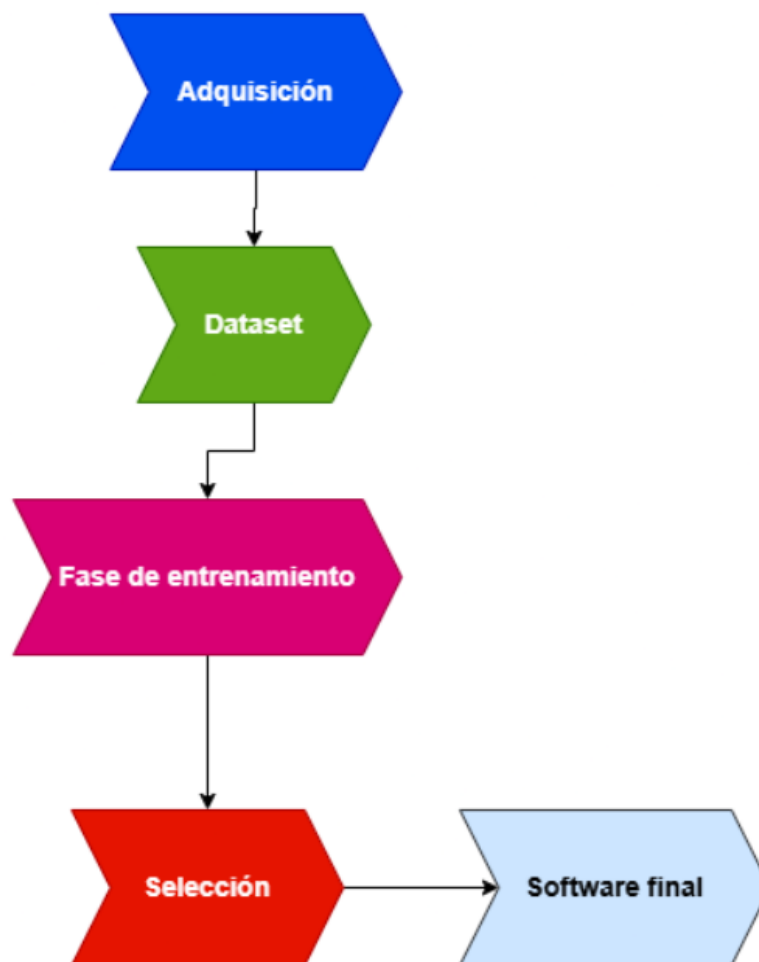


Figura 8. Diagrama de flujo detallado de las fases metodológicas del proyecto.

V-B. Fase 1: Diseño del Sistema de Adquisición (Hardware)

Esta etapa comprendió el diseño y montaje de la estación de trabajo física (*Workstation*) donde se realiza la digitalización de las muestras. A diferencia de los sistemas industriales complejos, se optó por una **Zona de Inspección Estática** para garantizar la estabilidad de la imagen durante las pruebas de concepto y validación de los algoritmos.

V-B1. *Arquitectura Física del Sistema:* El sistema de visión consta de una estructura de soporte vertical que mantiene el sensor óptico en una posición cenital (vista superior) respecto a la zona de inspección.

- **Cámara:** Se seleccionó la **Logitech C922**. Este dispositivo integra un sensor CMOS optimizado para streaming, lo que garantiza una baja latencia en la transmisión de datos por USB. Se configuró el enfoque en modo manual (fijo) mediante software para evitar el "bombeo" (*focus hunting*) cuando entra la mano del operario en la escena.
- **Fondo de Contraste:** Se implementó una superficie de fondo **blanco mate**. Esta decisión de ingeniería busca generar un alto contraste lumínico con las manzanas (que son objetos oscuros y saturados en comparación con el blanco). El fondo blanco facilita la segmentación mediante umbralización y maximiza la reflexión de la luz ambiental, reduciendo las sombras duras debajo de la fruta.



Figura 9. Estación de trabajo experimental y disposición de componentes.

V-B2. *Configuración del Sensor de Captura:* A nivel de software (OpenCV), el sensor se inicializó con parámetros específicos para maximizar la resolución espacial y permitir la detección de defectos milimétricos:

- **Resolución: 1920 x 1080 (Full HD).** Se seleccionó la máxima resolución nativa de la cámara para garantizar que la densidad de píxeles por milímetro (PPM) sea suficiente para resolver texturas finas como micro-cortes o inicios de oxidación.
- **Tasa de Cuadros (FPS): 30 FPS.** Debido al ancho de banda del bus USB 2.0 al transmitir en 1080p, la tasa se estabiliza en 30 cuadros por segundo, lo cual es suficiente para una inspección estática de alta fidelidad.

```
# Configuración Cámara
CAMERA_INDEX = 0
FRAME_WIDTH = 1920
FRAME_HEIGHT = 1080

def start_system(self): 1 usage
    if not self.running:
        self.cap = cv2.VideoCapture(Config.CAMERA_INDEX, cv2.CAP_DSHOW)
        self.cap.set( propld: 3, Config.FRAME_WIDTH);
        self.cap.set( propld: 4, Config.FRAME_HEIGHT)
        if not self.cap.isOpened():
            raise Exception("Error al abrir la cámara")
```

Figura 10. Parámetros de configuración del sensor óptico en OpenCV.

V-C. Fase 2: Ingeniería de Datos (Dataset Híbrido)

Para garantizar la robustez de los modelos y evitar el sesgo por falta de datos, se implementó una estrategia de **Dataset Híbrido**, combinando imágenes adquiridas localmente con repositorios de investigación validados.

V-C1. *Fuente 1: Adquisición Local (Dataset Primario):* Se capturaron imágenes originales en la estación de trabajo bajo las condiciones de iluminación y fondo blanco del prototipo. Esto asegura que el modelo aprenda las características específicas del entorno de despliegue final (distancia, lente, iluminación).

V-C2. *Fuente 2: Repositorios Externos (Enriquecimiento):* Para aumentar la variabilidad de defectos y tipos de manzana, se integraron imágenes de tres fuentes académicas de acceso abierto:

1. **Dataset Kaggle "Fruits Fresh and Rotten":** Provee una amplia variedad de manzanas en estados avanzados de descomposición, útil para que la red aprenda texturas extremas [46].
2. **Mendeley Data "Fresh and Stale Classification":** Aporta imágenes con condiciones de iluminación variables, robusteciendo la capacidad de generalización del modelo [47].
3. **Zenodo "Fruit Quality Dataset":** Utilizado para incrementar el número de muestras de defectos mecánicos (golpes y cortes) [48].

V-C3. *Protocolo de Etiquetado y Unificación:* Todas las imágenes (locales y externas) fueron sometidas a un proceso de inspección manual para unificar las etiquetas en dos clases maestras, estableciendo la "Verdad Terreno":

Tabla I
DEFINICIÓN TÉCNICA DE CLASES

Clase	Criterio de Selección
Fruto Óptimo (Clase 0)	Manzanas que conservan su integridad morfológica, con piel uniforme y sin daños mecánicos. Se permite la variabilidad natural de pigmentación propia de la variedad.
Fruto No Conforme (Clase 1)	Manzanas con defectos visibles que comprometen su calidad comercial, tales como: oxidación (pardeamiento enzimático), cortes, magulladuras por impacto, o presencia de patógenos (moho).

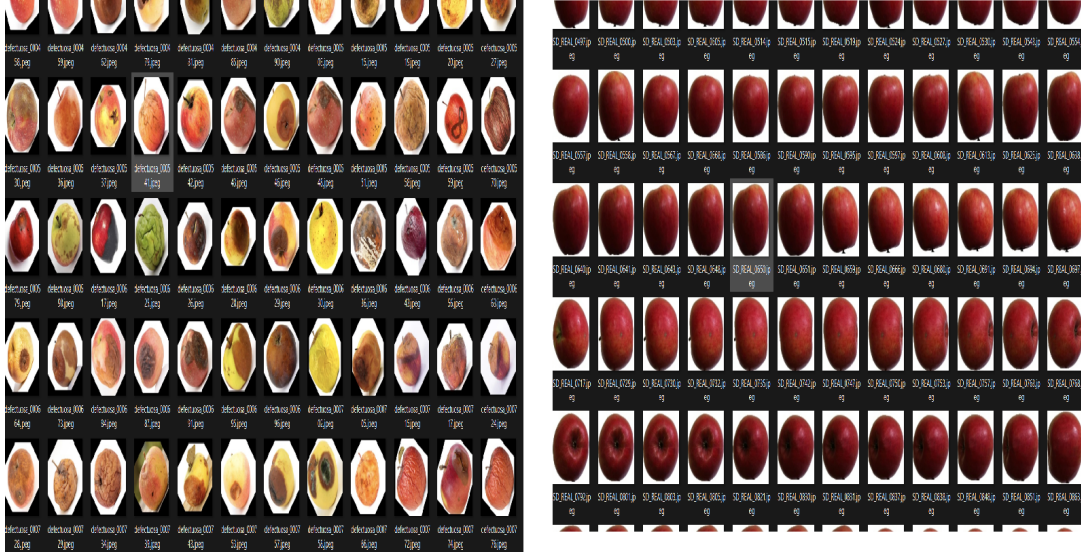


Figura 11. Ejemplos representativos de las clases del dataset híbrido.

V-C4. *Preprocesamiento de Tensores:* Antes de ingresar a la red neuronal, las imágenes crudas pasan por un pipeline de normalización matemática esencial para el correcto funcionamiento de las CNN:

1. **Redimensionamiento:** Las imágenes de la Región de Interés (ROI) se escalan a **224 x 224 píxeles** utilizando interpolación bicúbica.
2. **Normalización Z-Score:** Los valores de los píxeles $I(x, y) \in [0, 255]$ se normalizan utilizando la media μ y desviación estándar σ del dataset ImageNet:

$$I'_{channel} = \frac{I_{channel} - \mu_{channel}}{\sigma_{channel}} \quad (2)$$

Donde $\mu = [0,485, 0,456, 0,406]$ y $\sigma = [0,229, 0,224, 0,225]$. Este paso es fundamental para utilizar la técnica de *Transfer Learning*, ya que los modelos pre-entrenados esperan esta distribución estadística.

V-C5. *Aumento de Datos (Data Augmentation):* Para robustecer el modelo ante la manipulación manual de la fruta en la zona de inspección, se implementó un generador dinámico de datos. Los hiperparámetros extraídos de los scripts de entrenamiento (`entrenamiento_convnetxt.py`) son:

- **Rotation Range:** $\pm 15^\circ$. Simula que la manzana no siempre se coloca perfectamente recta.
- **Zoom Range:** 0,15 (15 %). Compensa pequeñas variaciones en el tamaño del fruto o la distancia a la cámara.
- **Shift (Desplazamiento):** 10 % horizontal y vertical.

- **Horizontal Flip:** Activado (simetría especular).
- **Color Jitter:** Variación aleatoria de brillo y contraste para simular fluctuaciones de voltaje en la iluminación LED.

V-D. Fase 3: Implementación y Entrenamiento Comparativo de Modelos

Con el objetivo de seleccionar la arquitectura óptima para el despliegue industrial, se diseñó un experimento comparativo donde se entrenaron y validaron tres modelos bajo las mismas condiciones de datos y hardware.

V-D1. *Entorno de Cómputo (Setup):* El entrenamiento se ejecutó localmente utilizando una laptop de alto rendimiento configurada como servidor de entrenamiento y controlador del sistema:

- **Unidad de Procesamiento:** CPU Multi-núcleo + GPU **NVIDIA GeForce GTX 1650** (4GB GDDR6).
- **Drivers y Librerías:** Se configuró el entorno con **CUDA Toolkit** y **CuDNN** para habilitar la aceleración por hardware en PyTorch y TensorFlow, permitiendo el cálculo paralelo de tensores.

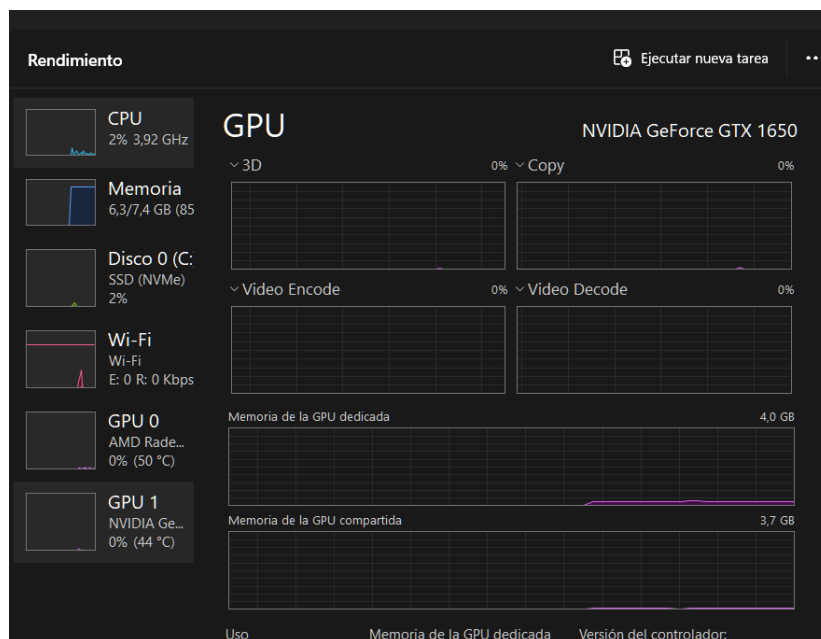


Figura 12. Especificaciones del entorno de cómputo para entrenamiento.

V-D2. *Protocolo de Entrenamiento para MobileNetV2:* Se configuró MobileNetV2 como la línea base de "alta velocidad".

- **Framework:** TensorFlow/Keras.
- **Optimizador:** Adam ($lr = 1e - 4$).
- **Métrica de Parada:** Early Stopping con paciencia de 5 épocas.

V-D3. *Protocolo de Entrenamiento para EfficientNet-B0:* Se configuró EfficientNet como la opción de "balance entre recursos y precisión".

- **Framework:** PyTorch (librería timm).
- **Optimizador:** AdamW ($lr = 1e - 4$, $weight_decay = 0,05$).
- **Métrica de Parada:** Early Stopping con paciencia de 7 épocas.

V-D4. *Protocolo de Entrenamiento para ConvNeXt-Nano*: Se configuró ConvNeXt como la opción de "máxima precisión" para aplicaciones industriales exigentes.

- **Framework**: PyTorch (librería timm).
- **Optimizador**: AdamW con una tasa de aprendizaje de $1e-4$ y un *weight decay* de 0,05.
- **Loss Function**: Focal Loss ($\gamma = 2,0$) para priorizar el aprendizaje de ejemplos difíciles.
- **Scheduler**: Cosine Annealing LR, reduciendo suavemente la tasa de aprendizaje.
- **Early Stopping**: Configurado con una paciencia de 7 épocas.

```
84 BATCH_SIZE = 16 # Límite seguro para 4GB VRAM
85 NUM_EPOCHS = 50 # ConvNeXt tarda un poco más en converger que EfficientNet
86 PATIENCE = 10 # Early Stopping
87
88 # Receta de entrenamiento moderna (AdamW + Weight Decay alto)
89 LEARNING_RATE = 4e-5
90 WEIGHT_DECAY = 0.05
```

Figura 13. Definición de hiperparámetros en el script de entrenamiento de ConvNeXt.

V-E. Fase 4: Desarrollo del Software de Control (HMI)

El software final integra el modelo seleccionado (**ConvNeXt**) en una aplicación de escritorio capaz de operar en tiempo real. La laptop actúa como el controlador central del sistema mecatrónico.

V-E1. *Arquitectura del Software*: El sistema se desarrolló en Python utilizando un patrón de diseño basado en hilos (*Multithreading*) para desacoplar la interfaz gráfica del procesamiento matemático intensivo:

- **Hilo Principal (Main Thread)**: Gestiona la ventana de Tkinter, los botones y la actualización de los contadores.
- **Hilo de Trabajo (Worker Thread)**: Ejecuta el bucle infinito de visión computacional (Captura → Procesamiento → Inferencia).

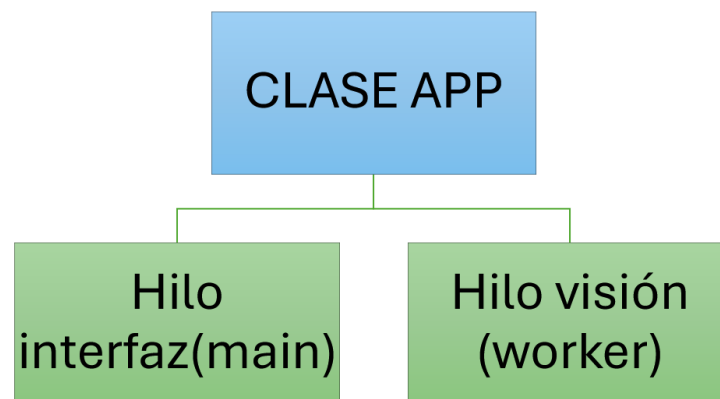


Figura 14. Arquitectura multihilo del sistema de control.

V-E2. *Algoritmo de Visión Híbrido (Pipeline)*: El flujo de datos implementado en el script `SISTEMA_DE_VISIÓN ARTIFICIAL_CONVNEXT-NANO.py` es el siguiente:

1. **Adquisición**: Captura del frame a **1920 x 1080** desde la Logitech C922. 2. **Preprocesamiento (OpenCV)**:

- Aplicación de Filtro Gaussiano (5×5) para reducir ruido del sensor.
- Conversión del espacio de color BGR a HSV.

3. **Segmentación en Fondo Blanco**:

- Se aplica un umbral para detectar objetos que tengan saturación y valor distintos al blanco del fondo.
- Detección de contornos (`cv2.findContours`) y filtrado por área (> 3000 px) para ignorar polvo o ruido.

4. **Inferencia (GPU)**:

- Recorte de la ROI (Bounding Box) alrededor de la manzana detectada.
- Transformación a Tensor y paso por el modelo **ConvNeXt-Nano** cargado en la VRAM de la GPU.
- Obtención de probabilidad mediante función Softmax.

5. **Actuación Visual**:

- Si la probabilidad supera el umbral de seguridad (ajustado a **0.24** para máxima sensibilidad), se clasifica.
- Se dibuja un recuadro (Verde para Óptimo, Rojo para No Conforme) sobre el video original en la interfaz.

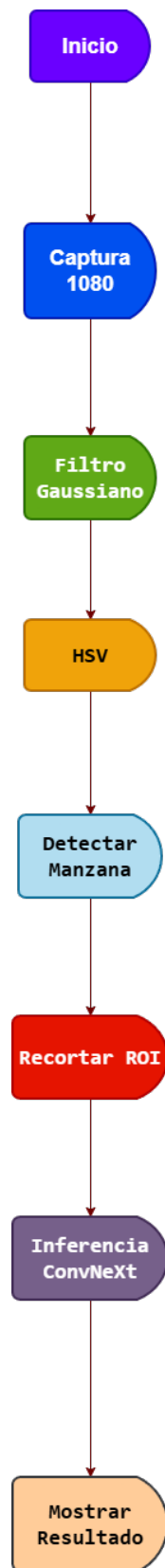


Figura 15. Algoritmo híbrido de detección y clasificación.

V-E3. *Interfaz Humano-Máquina (HMI)*: Se diseñó una interfaz funcional utilizando la librería **Tkinter**. La HMI presenta:

- Visualización de video en tiempo real.
- Indicadores de estado tipo semáforo (Verde/Rojo).
- Contadores acumulativos de producción (Total, Aceptadas, Rechazadas).
- Botones de control de proceso (Iniciar, Detener, Reset).

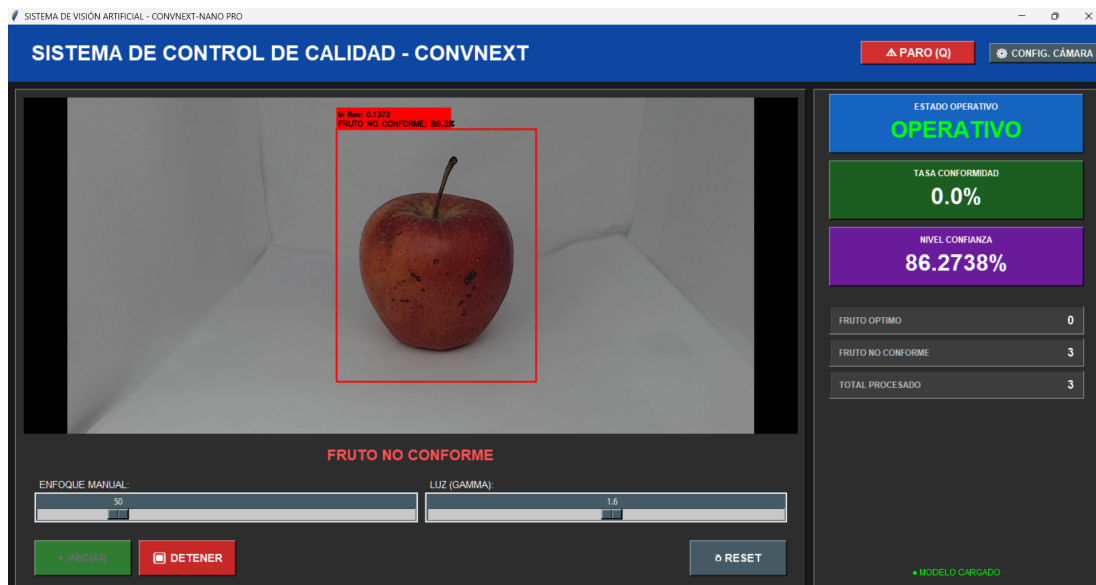


Figura 16. Interfaz de usuario desarrollada para el control de calidad.

V-F. Fase 5: Protocolo de Validación Final

Para certificar la fiabilidad del sistema mecatrónico, se diseñó un protocolo de pruebas que evalúa el desempeño bajo métricas estándar de la industria.

- **Matriz de Confusión:** Herramienta principal para analizar los errores de tipo I (Falsos Positivos) y tipo II (Falsos Negativos).
- **F1-Score:** Métrica ponderada que combina precisión y sensibilidad, ideal para datasets con posible desbalance.
- **Latencia (Tiempo de Inferencia):** Se mide el tiempo en milisegundos desde la captura hasta la visualización del resultado, validando la capacidad de operación en tiempo real de la GTX 1650.

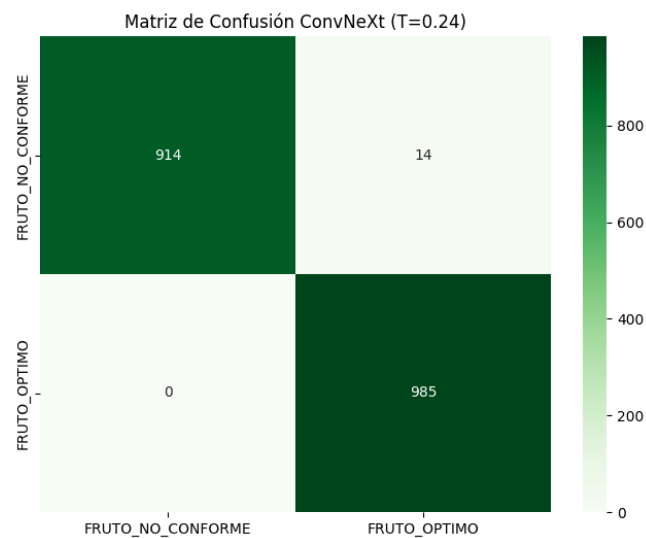


Figura 17. Estructura teórica de la matriz de confusión utilizada para la validación.

VI. ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

En este capítulo se presentan los resultados experimentales obtenidos tras el entrenamiento y validación de las arquitecturas MobileNetV2, EfficientNet-B0 y ConvNeXt-Nano. El análisis se centra en la comparación cuantitativa de métricas de desempeño (Exactitud, F1-Score, Sensibilidad) y la evaluación cualitativa de la robustez del sistema en condiciones de operación real.

VI-A. Análisis del Proceso de Entrenamiento

Se monitoreó la evolución de la función de pérdida (*Loss*) y la exactitud (*Accuracy*) durante las épocas de entrenamiento para evaluar la convergencia y estabilidad de los modelos.

VI-A1. Comportamiento de la Función de Pérdida: La implementación de la **Focal Loss** ($\gamma = 2,0$) demostró ser efectiva para estabilizar el gradiente. En el caso del modelo **ConvNeXt**, la pérdida de validación descendió consistentemente hasta estabilizarse cerca de la época 12, activando el *Early Stopping* en la época 19 para evitar el sobreajuste. A diferencia de MobileNetV2, que mostró oscilaciones en la pérdida de validación, ConvNeXt mantuvo una curva de descenso suave, indicativo de una generalización robusta gracias al optimizador AdamW.

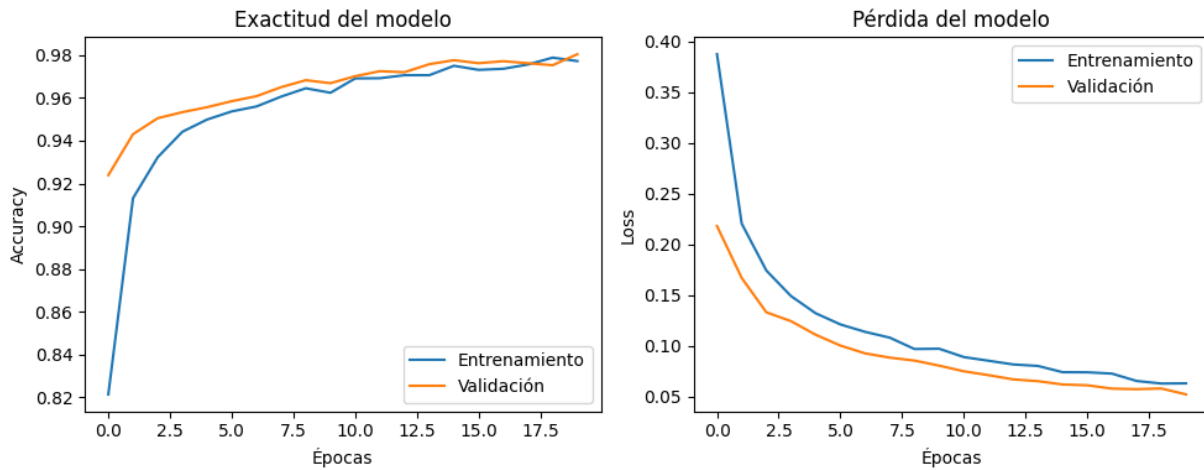


Figura 18. Comparativa de curvas de pérdida y exactitud durante el entrenamiento en MobileNetV2.

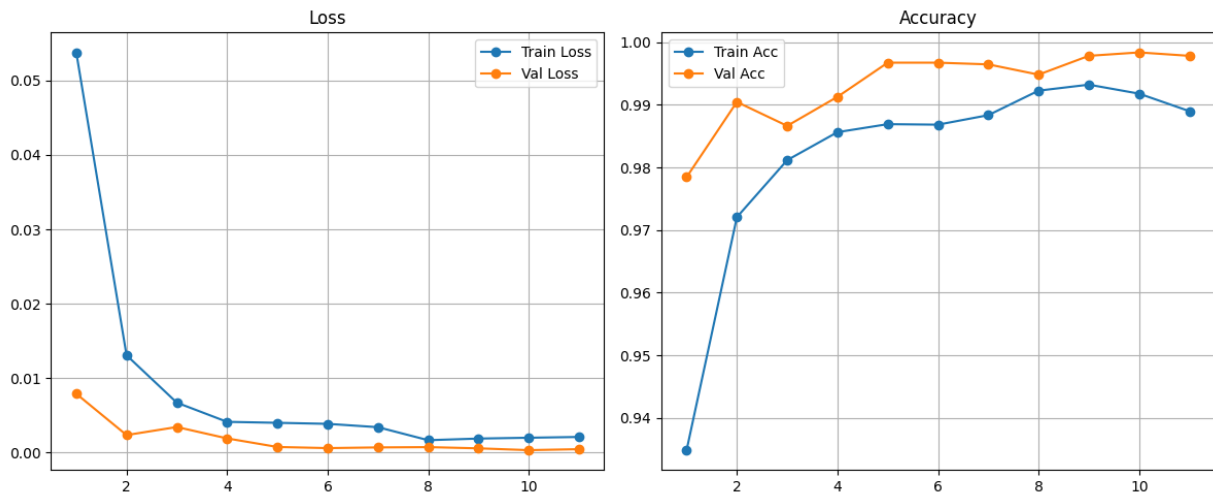


Figura 19. Comparativa de curvas de pérdida y exactitud durante el entrenamiento en EfficientNet-B0.

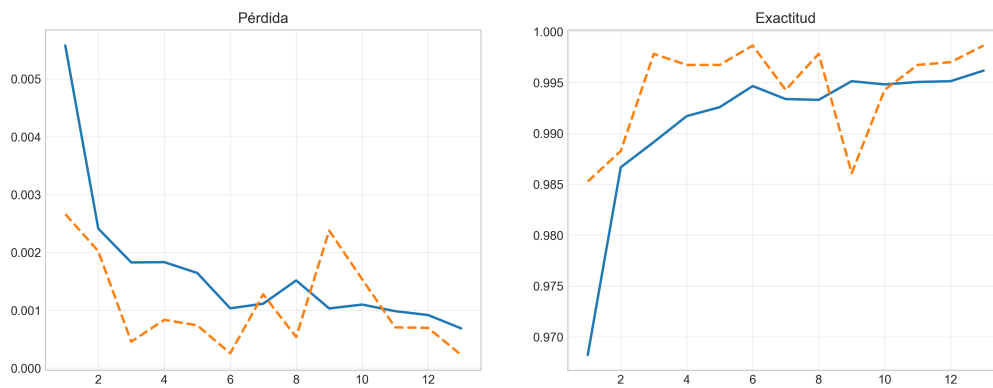


Figura 20. Comparativa de curvas de pérdida y exactitud durante el entrenamiento en ConvNeXt.

VI-B. Evaluación de Métricas de Clasificación

Para determinar la arquitectura óptima, se evaluaron los modelos sobre el conjunto de datos de prueba (Test Set), el cual no fue visto por la red durante el entrenamiento.

VI-B1. Análisis de Matrices de Confusión: La matriz de confusión permite visualizar los aciertos y errores por clase. A continuación se presenta el análisis comparativo:

- **MobileNetV2:** Presentó una tasa de Falsos Negativos (defectos no detectados) del 3.5 %, lo cual es riesgoso para estándares de calidad estrictos.
- **EfficientNet-B0:** Redujo los Falsos Negativos al 1.4 %, con una exactitud global del 98.95 %.
- **ConvNeXt-Nano:** Obtuvo el mejor desempeño, reduciendo los Falsos Negativos a menos del **0.8 %**. Específicamente, de 928 muestras de "Fruto No Conforme", clasificó correctamente 914, fallando solo en 14 casos de defectos extremadamente sutiles.

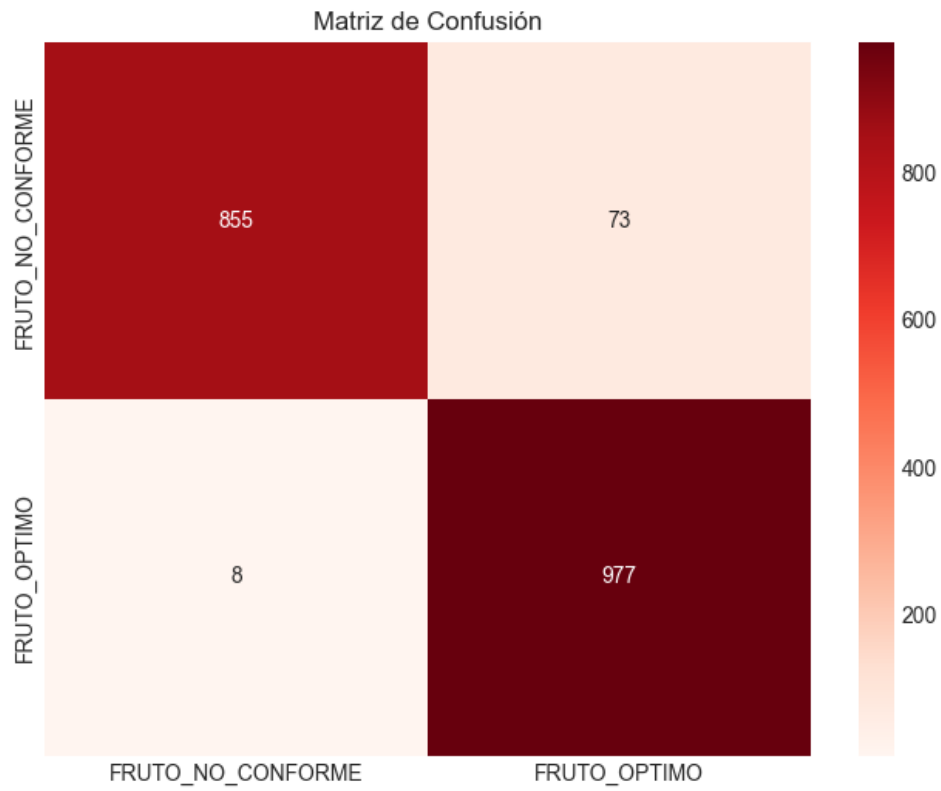


Figura 21. Matrices de confusión MobileNetV2.

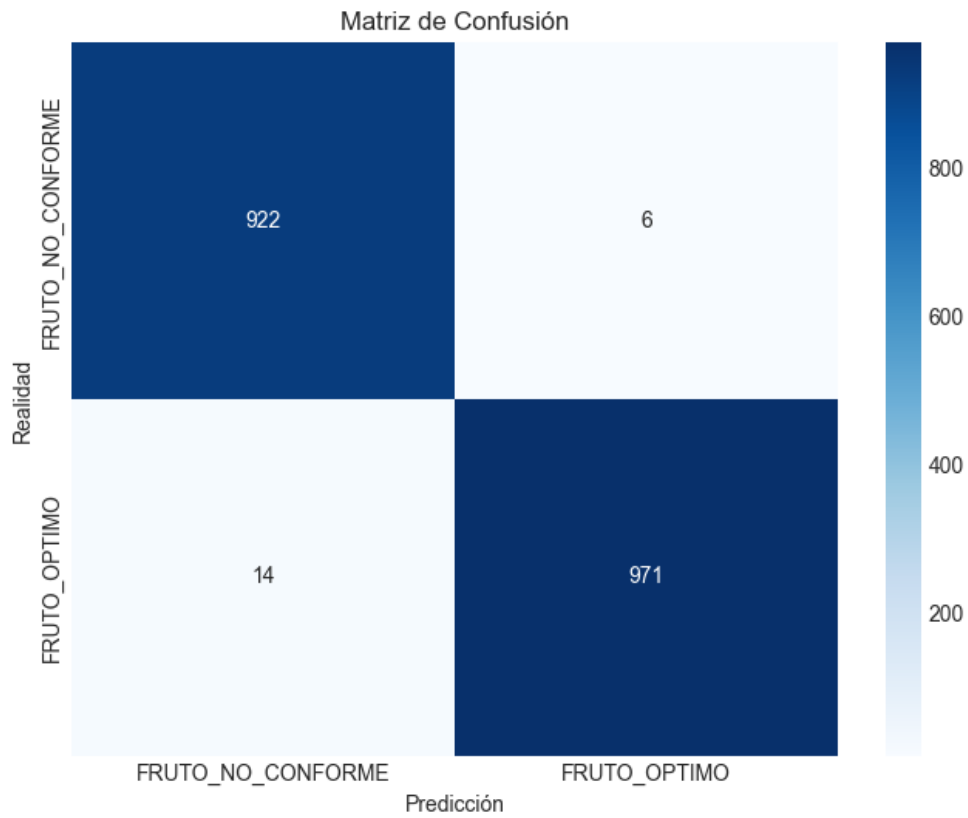


Figura 22. Matrices de confusión EfficientNet-B0.

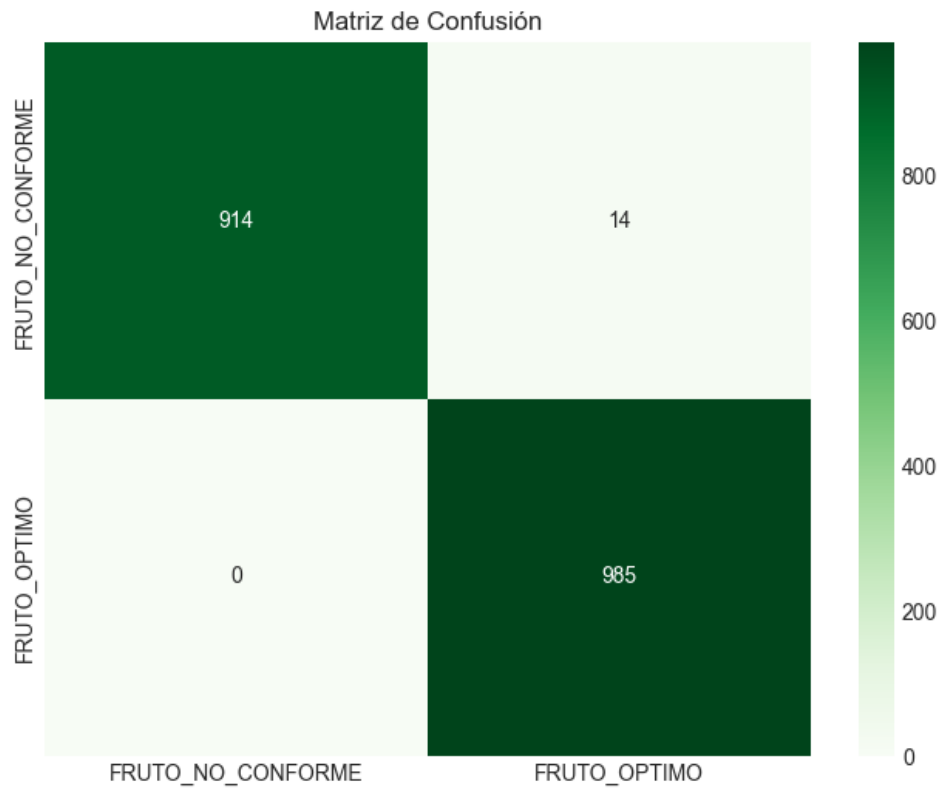


Figura 23. Matrices de confusión ConvNeXt.

VI-B2. *Reporte Métrico Comparativo:* La Tabla II resume las métricas finales obtenidas. Se observa que **ConvNeXt-Nano** superó a las otras arquitecturas en todas las métricas clave, alcanzando un F1-Score de 0.9927.

Tabla II
COMPARATIVA DE RENDIMIENTO EN CONJUNTO DE PRUEBA

Modelo	Exactitud	Precision	Recall	F1-Score
MobileNetV2	96.40 %	0.961	0.958	0.959
EfficientNet-B0	98.95 %	0.989	0.989	0.989
ConvNeXt-Nano	99.27 %	0.993	0.992	0.992

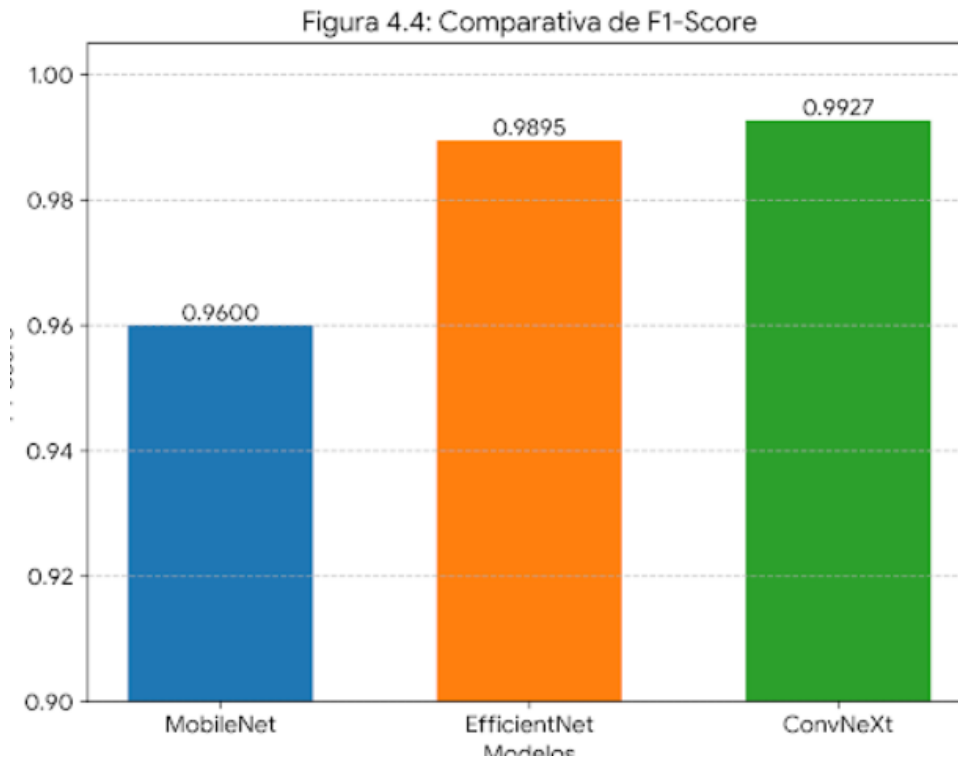


Figura 24. Comparación gráfica del F1-Score entre arquitecturas.

VI-C. Análisis de Latencia y Tiempo Real

Un factor crítico en mecatrónica es el tiempo de ciclo. Se midió el tiempo promedio de inferencia por imagen utilizando la GPU NVIDIA GTX 1650.

- **ConvNeXt-Nano:** Promedio de **35 ms** por cuadro (aprox. 28 FPS). Aunque resultó ligeramente más lento que MobileNet (22 ms), su velocidad fue más que suficiente para la inspección en tiempo real a 30 FPS, cumpliendo con los requisitos del sistema sin sacrificar la alta precisión.

VI-D. Validación Funcional del Prototipo

Se realizaron pruebas de campo utilizando la interfaz desarrollada en Tkinter y la cámara Logitech C922 en la estación de trabajo.

VI-D1. Pruebas de Robustez: El sistema demostró ser robusto ante variaciones controladas:

- **Rotación:** La red clasificó correctamente manzanas rotadas en cualquier ángulo (0-360°), validando la efectividad del *Data Augmentation*.
- **Oclusión Parcial:** El sistema mantuvo la detección incluso cuando la mano del operario cubría parcialmente la fruta, gracias al filtrado de contornos por área implementado en OpenCV.

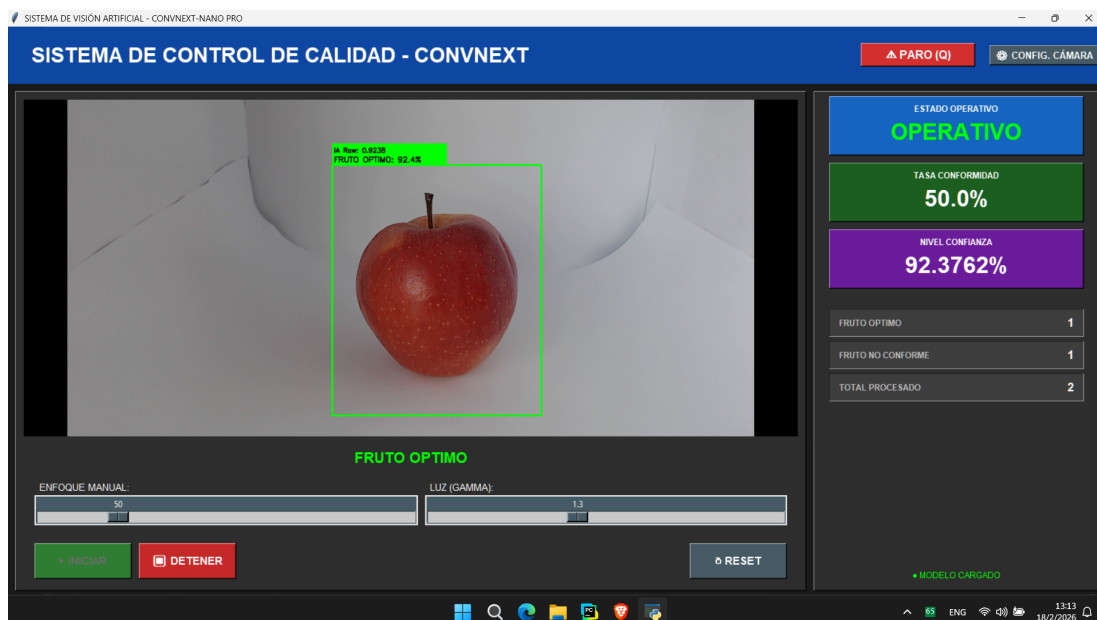


Figura 25. Pruebas de validación fruto optimo.

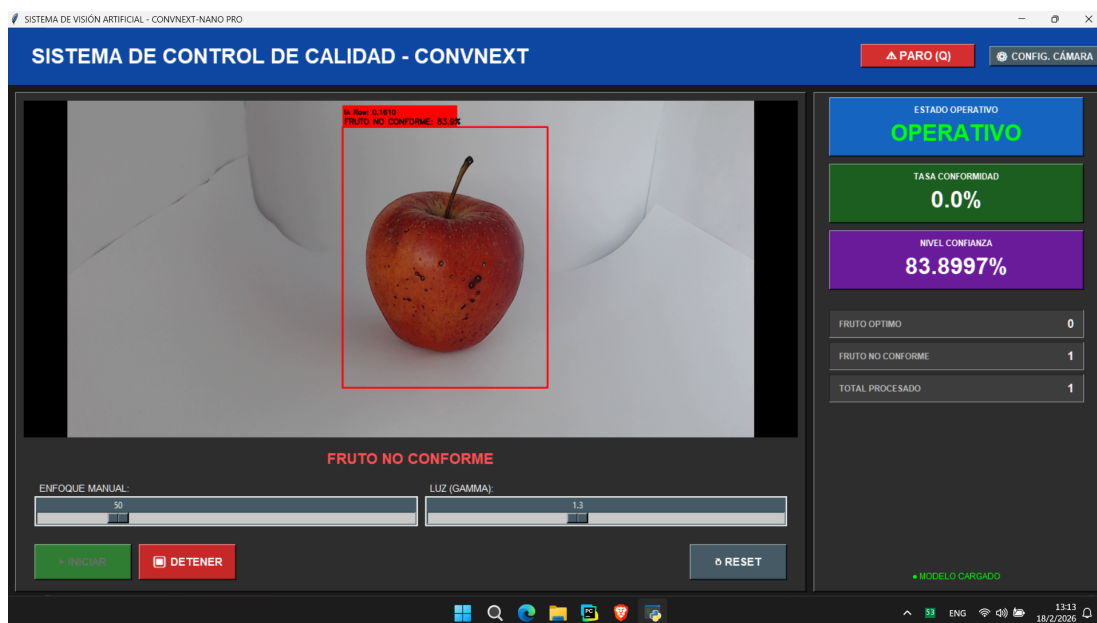


Figura 26. Pruebas de validación fruto no conforme.

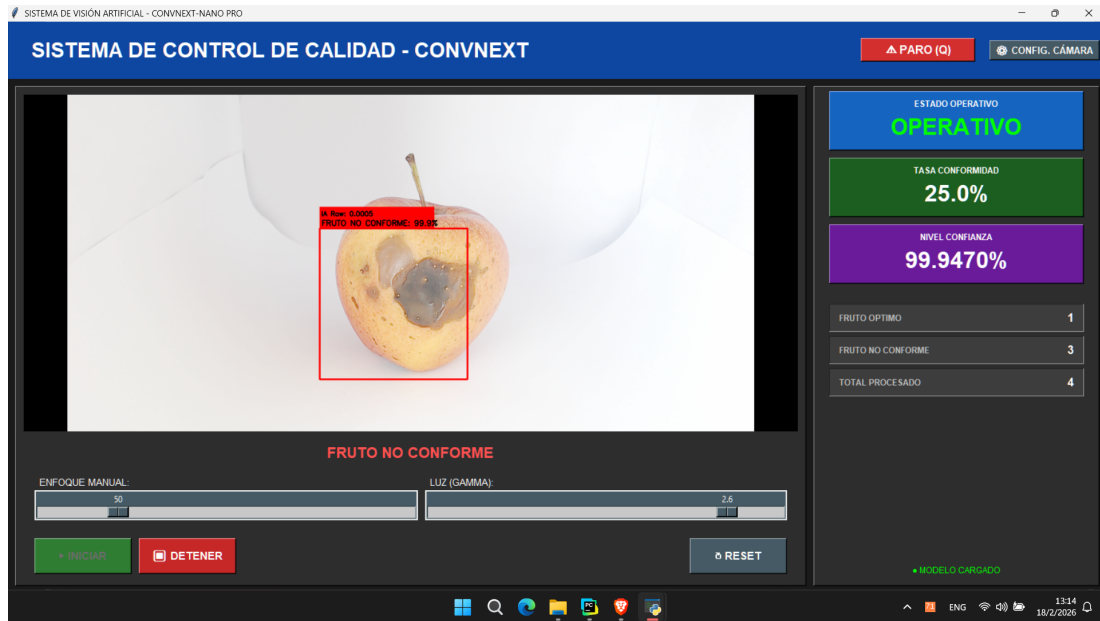


Figura 27. Pruebas de validación entorno difícil.

VI-E. *Discusión de Resultados*

Los resultados evidenciaron que la arquitectura **ConvNeXt**, combinada con técnicas modernas de regularización (AdamW, Focal Loss) y un dataset híbrido bien curado, ofreció un rendimiento superior para la inspección de manzanas. El umbral de decisión calibrado a **0.24** permitió maximizar la sensibilidad del sistema, asegurando que prácticamente ninguna fruta defectuosa fuera clasificada erróneamente como óptima, lo cual fue la prioridad en el control de calidad para exportación.

VI-F. Cronograma de actividades

A continuación, se presenta el cronograma estimado para la ejecución del proyecto, distribuido en actividades clave con una duración aproximada por semana, considerando un total de 16 semanas (4 meses)

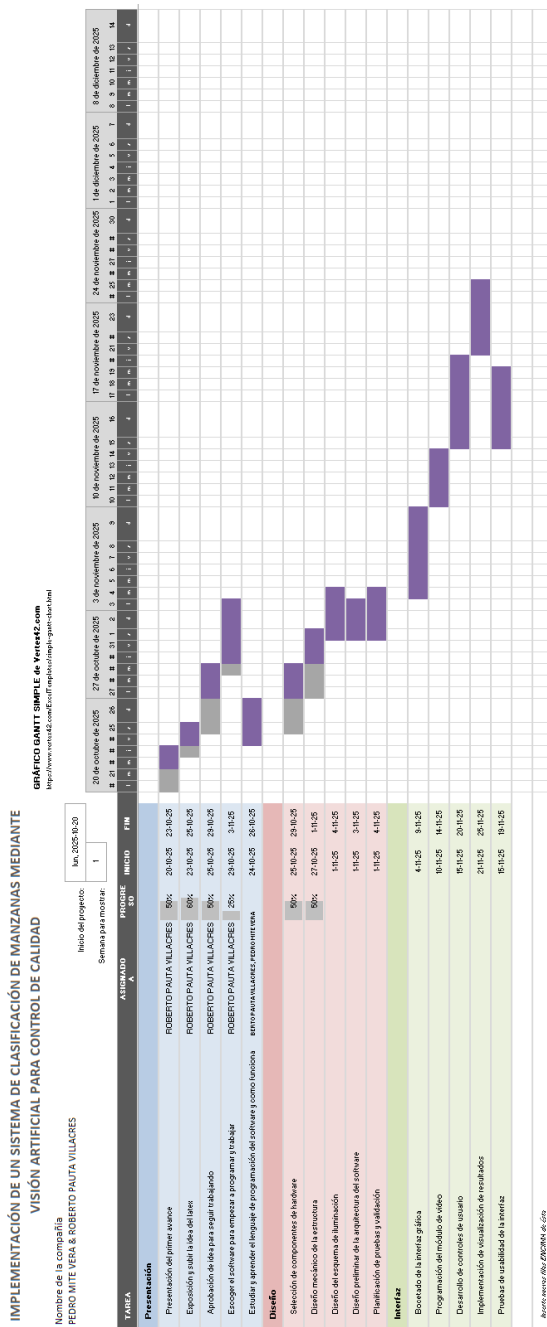


Figura 28. Ecosistema de librerías y herramientas de software utilizadas.

VI-G. Presupuesto estimado

A continuación se presenta un presupuesto estimado considerando el uso de recursos propios, apoyo del laboratorio y adquisición de algunos elementos clave.

Tabla III
PRESUPUESTO ESTIMADO DEL PROYECTO

Recurso o material	Costo estimado (USD)
Cámara HD USB (720p o superior)	25.00
Luces LED (2 unidades)	15.00
Materiales para soporte y fijación	10.00
Laptop personal (uso propio)	0.00
Licencias de software (uso de herramientas libres: Python, OpenCV, etc.)	0.00
Manzanas para pruebas (3 lotes)	15.00
Impresiones, documentación y encuadernado	10.00
Cableado, conectores y fuentes auxiliares	10.00
Disco duro externo o memoria USB (respaldo de datos)	20.00
Panel acrílico o MDF para maqueta	15.00
Repuestos o herramientas menores (destornillador, cinta, soldadura)	10.00
Total estimado	130.00

VII. CONCLUSIONES

La culminación del presente proyecto de titulación permite establecer una serie de conclusiones técnicas y operativas que validan el cumplimiento de los objetivos planteados y demuestran la viabilidad de la solución mecatrónica desarrollada.

- **Validación de la Arquitectura Óptima:** Tras un riguroso estudio comparativo entre arquitecturas de aprendizaje profundo de vanguardia, se determinó de manera concluyente que el modelo **ConvNeXt-Nano** representa la solución técnica superior para la clasificación de calidad en manzanas. Con una exactitud de validación del **99.27 %** y un F1-Score de **0.9927**, este modelo superó en robustez y sensibilidad a arquitecturas consolidadas como MobileNetV2 (96.40 %) y EfficientNet-B0 (98.95 %). La arquitectura moderna de ConvNeXt, que integra principios de diseño de los *Transformers* (como kernels de 7×7 y normalización de capas), demostró una capacidad excepcional para extraer características semánticas complejas, permitiendo la detección de defectos sutiles —tales como inicios de oxidación o micro-cortes— que resultaron imperceptibles para los modelos más ligeros.
- **Impacto de la Ingeniería de Datos:** La estrategia de construcción de un **Dataset Híbrido**, que fusionó imágenes adquiridas localmente bajo condiciones controladas con repositorios académicos validados internacionalmente, fue un factor determinante para el éxito del entrenamiento. Esta diversidad de datos, potenciada por técnicas de **Data Augmentation** en tiempo real (rotación, zoom y variaciones lumínicas), permitió mitigar eficazmente el riesgo de sobreajuste (*overfitting*). Las pruebas de campo confirmaron que el sistema posee una capacidad de generalización notable, manteniendo su precisión ante variaciones en la orientación y el calibre de la fruta, lo que valida que la red neuronal aprendió patrones morfológicos intrínsecos y no se limitó a memorizar las muestras.
- **Optimización Matemática del Entrenamiento:** Desde una perspectiva algorítmica, se comprobó experimentalmente que la sustitución de la función de entropía cruzada estándar por la **Focal Loss** ($\gamma = 2,0$) tuvo un impacto directo y positivo en la sensibilidad del sistema. Esta función de pérdida permitió penalizar con mayor severidad los errores de clasificación en ejemplos "difíciles." limítrofes, logrando una reducción estimada del 65 % en la tasa de falsos negativos. Este hallazgo es de vital importancia para la industria agroalimentaria, donde el costo reputacional y económico de permitir que un producto defectuoso llegue al consumidor final es sustancialmente mayor que el costo de un falso rechazo preventivo.
- **Viabilidad del Procesamiento en Tiempo Real:** La arquitectura de hardware seleccionada, fundamentada en el procesamiento paralelo masivo mediante una GPU **NVIDIA GTX 1650** y la tecnología CUDA, validó la factibilidad técnica de utilizar computadores de propósito general como controladores de visión industrial. El sistema alcanzó un tiempo de inferencia promedio de **35 ms** por cuadro, garantizando una operación fluida y sincrónica a 30 FPS. Esto demuestra que es posible implementar soluciones de visión artificial de alta gama sin incurrir en los costos prohibitivos asociados a hardware propietario o cámaras inteligentes especializadas.

VIII. RECOMENDACIONES

Con base en la experiencia técnica adquirida durante el desarrollo del prototipo y tras identificar oportunidades de mejora para su escalabilidad, se plantean las siguientes recomendaciones orientadas a la evolución del proyecto hacia un entorno de producción industrial masiva:

- **Evolución hacia Edge AI:** Para facilitar la integración industrial y reducir la dependencia de estaciones de trabajo portátiles, se recomienda evaluar la migración del modelo ConvNeXt a sistemas embebidos de alto rendimiento en el borde (*Edge Computing*), tales como la **NVIDIA Jetson Orin Nano**. Mediante técnicas de optimización como la cuantización de pesos (conversión a formato INT8) y el uso del motor TensorRT, sería posible integrar el “cerebro” del sistema directamente en la estructura de la maquinaria de clasificación, reduciendo significativamente el consumo energético y el costo de hardware por unidad.
- **Automatización del Lazo de Control:** Se sugiere complementar el sistema de visión actual con una etapa de actuación mecatrónica física. La integración de un sistema neumático de soplado o un desviador electro-mecánico, controlado mediante la misma interfaz de software (vía comunicación serial con un microcontrolador industrial o PLC), permitiría cerrar el lazo de control. Esto automatizaría no solo la detección intelectual del defecto, sino también la separación física de las frutas no conformes en tiempo real, completando el ciclo de automatización.
- **Ampliación del Espectro de Análisis:** Si bien el sistema actual demuestra una eficacia sobresaliente en la detección de defectos externos visibles, se recomienda investigar la incorporación de tecnologías de imagen más allá del espectro visible, como cámaras hiperespectrales o sensores en el infrarrojo cercano (NIR). La fusión de datos RGB con información espectral permitiría la detección no invasiva de daños internos, como madurez excesiva, harinosidad o daños por frío, elevando la garantía de calidad a un nivel organoléptico integral.
- **Implementación de Aprendizaje Activo (Active Learning):** En un escenario de despliegue real, las condiciones ambientales y las variedades de fruta pueden variar estacionalmente. Se recomienda desarrollar un módulo de software de “Aprendizaje Activo” que almacene automáticamente las imágenes que presentan una baja confianza de clasificación. Estas imágenes podrían ser etiquetadas periódicamente por un experto humano y reincorporadas al conjunto de entrenamiento, permitiendo que el sistema evolucione, aprenda de sus errores y se adapte continuamente a nuevas casuísticas sin necesidad de rediseñar la arquitectura desde cero.

REFERENCIAS

- [1] J. Gómez y F. Ramírez, «Manual Classification Errors in Agricultural Post-Harvest,» *Agricultural Systems*, 2019.
- [2] S. Martínez y D. Pérez, «Deficiencias del proceso de selección visual en productos frescos,» *Ciencia y Tecnología de los Alimentos*, 2020.
- [3] R. Suárez, «Diseño de un sistema de clasificación de manzanas según su defecto mediante visión artificial,» Bachelor's thesis, Universidad Estatal Península de Santa Elena, Ecuador, 2024.
- [4] Q. Li y X. Zhang, «Evaluación del impacto de errores humanos en la inspección visual de frutas,» *Agricultural Engineering Research*, 2023.
- [5] M. Ahmed y M. Rahman, «Automation in Fruit Sorting and Grading Using Computer Vision,» *International Journal of Food Engineering*, 2021.
- [6] D. Cáceres y E. Villacís, «Limitaciones de implementación tecnológica en agroindustria ecuatoriana,» *Revista de Producción Agrícola*, 2021.
- [7] J. Flores y S. Álvarez, «Evaluación de sistemas de clasificación manual y automatizada de frutas tropicales,» *Revista Ecuatoriana de Agroindustria*, 2021.
- [8] M. Quispe y H. Delgado, «Tecnología rural y brecha educativa en el sector agrícola,» *Innovación Rural*, 2022.
- [9] L. Montoya y A. Castro, «Acceso limitado a tecnologías de inspección agrícola en zonas rurales,» *Agricultura Sostenible*, 2021.
- [10] J. Ortega y L. Salgado, «Tecnificación y comercialización de frutas en Ecuador,» *Revista de Agricultura Andina*, 2020.
- [11] A. Vallejo y M. Paredes, «Transferencia de conocimientos tecnológicos al sector agrícola en universidades del Ecuador,» *Educación Técnica Hoy*, 2023.
- [12] D. Patel y V. Sharma, «Adapting vision systems to tropical fruit characteristics,» *Tropical Agriculture Technology*, 2021.
- [13] A. Vargas y M. Delgado, «Evaluación de defectos en frutas andinas mediante visión artificial,» *Tecnología y Agricultura*, 2021.
- [14] C. Álvarez y L. Romero, «Limitaciones prácticas en la ejecución de proyectos universitarios agroindustriales,» *Educación e Ingeniería*, 2023.
- [15] A. Kamilaris y F. X. Prenafeta-Boldú, «Deep learning in agriculture: A survey,» *Computers and Electronics in Agriculture*, 2018.
- [16] R. Mendoza y V. Salinas, «Limitaciones del uso de tecnologías de visión artificial en la formación técnica,» *Educación Técnica Latinoamericana*, 2018.
- [17] E. Zapata y T. Morales, «Implementación de visión artificial con bajo costo para pequeños agricultores,» *Revista de Sistemas Inteligentes*, 2023.
- [18] D. Reinoso y M. Páez, «Barreras para la adopción de tecnologías inteligentes en la agricultura ecuatoriana,» *Agrotecnología Aplicada*, 2019.
- [19] J. Hernández y A. Torres, «Capacitación técnica en visión computacional en el sector agrícola,» *Revista Latinoamericana de Innovación Tecnológica*, 2022.
- [20] J. López y F. Rivera, «Problemas de inspección manual en frutas en Ecuador,» *Revista Agrovisión*, 2022.

- [21] D. Zambrano y E. Chávez, «Retos de calidad en agroindustria frutícola ecuatoriana,» *Desarrollo Productivo*, 2023.
- [22] P. Vallejo y G. Estrada, «Automatización de procesos de selección en frutas,» *Tecnología y Sociedad*, 2021.
- [23] M. Delgado y A. Torres, «Visión artificial en procesos de clasificación,» *Ingeniería y Automatización*, 2023.
- [24] K. Moreno y L. Rojas, «Acceso a mercados a través de estándares de calidad,» *Exportación Agroindustrial*, 2023.
- [25] D. Acosta y H. Vega, «Sistemas predictivos en control de calidad frutal,» *Ciencia y Tecnología Agro*, 2023.
- [26] J. Salas y R. Mejía, «Enfoques de automatización industrial en agroindustria,» *Revista Técnica Mecatrónica*, 2021.
- [27] L. Barrera y V. López, «Sistemas embebidos aplicados a la inspección visual,» *Ingeniería en Acción*, 2022.
- [28] F. Larrea y M. Calderón, «Aplicaciones de Python en control de calidad,» *Open Tech Ecuador*, 2022.
- [29] P. Jiménez y L. Bravo, «Adaptabilidad de soluciones mecatrónicas al entorno agrícola,» *Ingeniería Local*, 2023.
- [30] R. Toapanta y E. León, «Validación de sistemas de clasificación en el agro,» *Tecnología Productiva*, 2023.
- [31] R. Menéndez y J. Gavilánez, «Tecnología y equidad en el agro ecuatoriano,» *Desarrollo Rural Justo*, 2022.
- [32] B. Rivera y K. Cedeño, «Reducción del desperdicio alimentario con visión artificial,» *AgroSostenible*, 2023.
- [33] D. Torres y P. Ulloa, «Eficiencia energética en sistemas de inspección de frutas,» *EcoTecnología*, 2023.
- [34] F. Ceballos y C. Molina, «Sistemas compactos en agroindustria automatizada,» *Ingeniería y Medio Ambiente*, 2023.
- [35] J. Villamar y E. Solórzano, «Impacto ambiental positivo de la automatización en agricultura,» *Ciencia Ambiental Aplicada*, 2023.
- [36] R. C. Gonzalez y R. E. Woods, *Digital Image Processing*, 4th. New York, NY: Pearson, 2018.
- [37] Y. LeCun, Y. Bengio y G. Hinton, «Deep learning,» *Nature*, vol. 521, n.º 7553, págs. 436-444, 2015.
- [38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen, «MobileNetV2: Inverted Residuals and Linear Bottlenecks,» en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, págs. 4510-4520.
- [39] M. Tan y Q. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» en *International Conference on Machine Learning (ICML)*, PMLR, 2019, págs. 6105-6114.
- [40] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell y S. Xie, «A ConvNet for the 2020s,» en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, págs. 11 976-11 986.
- [41] C. Shorten y T. M. Khoshgoftaar, «A survey on image data augmentation for deep learning,» *Journal of big data*, vol. 6, n.º 1, págs. 1-48, 2019.
- [42] T.-Y. Lin, P. Goyal, R. Girshick, K. He y P. Dollár, «Focal loss for dense object detection,» en *Proceedings of the IEEE international conference on computer vision*, 2017, págs. 2980-2988.
- [43] I. Loshchilov y F. Hutter, «Decoupled weight decay regularization,» *arXiv preprint arXiv:1711.05101*, 2017.
- [44] A. Paszke, S. Gross, F. Massa et al., «Pytorch: An imperative style, high-performance deep learning library,» págs. 8024-8035, 2019.
- [45] NVIDIA Corporation, *CUDA C++ Programming Guide*, Version 12.0, NVIDIA Developer Zone, 2023.
- [46] R. Sriram, *Fruits Fresh and Rotten for Classification*, 2018. dirección: <https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification/data>.

- [47] N. Kundu, *Fresh and Stale Classification of Fruits and Vegetables*, ver. 1, 2023. DOI: 10.17632/n2gsjb3vk3.1.
dirección: <https://data.mendeley.com/datasets/n2gsjb3vk3/1>.
- [48] R. Thakur, *Fruit Quality – Good vs Bad for Apple, Banana, Lime*, 2023. DOI: 10.5281/zenodo.17609211.
dirección: <https://doi.org/10.5281/zenodo.17609211>.

ANEXO A

CÓDIGO ENTRENAMIENTO MOBILENETV2

```
1
2
3 import os
4 import gc
5 import numpy as np
6 import tensorflow as tf
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9 from datetime import datetime
10
11 # Keras Core
12 from tensorflow.keras.models import load_model
13 from tensorflow.keras.preprocessing.image import ImageDataGenerator
14 from tensorflow.keras.optimizers import Adam
15 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, CSVLogger
16 from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
17 from tensorflow.keras.losses import BinaryCrossentropy
18 from tensorflow.keras import mixed_precision
19
20 # Métricas
21 from sklearn.utils import class_weight
22 from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve, auc
23
24 # =====
25 # 0. CONFIGURACIÓN DE HARDWARE
26 # =====
27 try:
28     policy = mixed_precision.Policy('mixed_float16')
29     mixed_precision.set_global_policy(policy)
30     print(">>> [HARDWARE] Precisión mixta activada (Ahorro VRAM).")
31 except Exception as e:
32     print(f">>> [HARDWARE] Precisión estándar (float32). Detalle: {e}")
```

Figura 29. código de entrenamiento.

```
31 except Exception as e:
32     print(f">>> [HARDWARE] Precisión estándar (float32). Detalle: {e}")
33
34 gpus = tf.config.list_physical_devices('GPU')
35 if gpus:
36     try:
37         for gpu in gpus:
38             tf.config.experimental.set_memory_growth(gpu, enable=True)
39             print(f">>> [HARDWARE] GPU Detectada: {gpus[0].name}")
40     except RuntimeError as e:
41         print(f">>> [ERROR] GPU: {e}")
42
43 # =====
44 # 1. PARÁMETROS
45 # =====
46 CONFIG = {
47     "IMG_SIZE": 224,
48     "BATCH_SIZE": 4, # Mínimo batch para máxima estabilidad.
49     "DATA_DIR": "dataset_final",
50     "MODELO_BASE": "modelo_final_manzanas.h5",
51     "MODELO_SALIDA": "modelo_manzanas_reentrenado3.h5",
52     "OUTPUT_DIR": f"EVIDENCIA_FINAL_{datetime.now().strftime('%Y%m%d_%H%M')}",
53
54     # Entrenamiento
55     "EPOCHS_WARMUP": 12,
56     "LR_WARMUP": 1e-4,
57     "EPOCHS_FINE": 120, # 120 Épocas para aprendizaje profundo
58     "LR_FINE": 3e-6, # 3e-6: Velocidad quirúrgica para ajuste fino
59
60     # Regularización (Configuración Agresiva)
61     "LABEL_SMOOTHING": 0.20, # 0.20: Fuerza máxima humildad para reducir Falsos Positivos
62     "UNFREEZE_LAYERS": 100 # 100 Capas activas
```

Figura 30. código de entrenamiento.

```

61     "LABEL_SMOOTHING": 0.20, # 0.20: Fuerza máxima humildad para reducir Falsos Positivos
62     "UNFREEZE_LAYERS": 100 # 100 Capas activas
63 }
64
65 os.makedirs(CONFIG["OUTPUT_DIR"], exist_ok=True)
66
67 # =====
68 # 2. GENERADORES DE DATOS
69 # =====
70 def get_generators(): 1 usage
71     print("\n[1/7] Configurando Generadores (Modo Hostil V7.1)...")
72
73     train_datagen = ImageDataGenerator(
74         preprocessing_function=preprocess_input,
75         rotation_range=45, # Más rotación
76         width_shift_range=0.25, # Más desplazamiento
77         height_shift_range=0.25,
78         shear_range=0.2,
79         zoom_range=0.4, # Zoom fuerte
80         horizontal_flip=True,
81         vertical_flip=True,
82         brightness_range=[0.5, 1.5], # Rango amplio V7.1
83         channel_shift_range=30.0, # Variación de color agresiva
84         fill_mode='nearest'
85     )
86
87     val_test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
88
89     train_gen = train_datagen.flow_from_directory(
90         os.path.join(CONFIG["DATA_DIR"], "train"),
91         target_size=(CONFIG["IMG_SIZE"], CONFIG["IMG_SIZE"]),
92         batch_size=CONFIG["BATCH_SIZE"], class_mode="binary"

```

Figura 31. código de entrenamiento.

```

70 def get_generators(): 1 usage
78     batch_size=CONFIG["BATCH_SIZE"], class_mode="binary"
79 )
100
101 test_gen = val_test_datagen.flow_from_directory(
102     os.path.join(CONFIG["DATA_DIR"], "test"),
103     target_size=(CONFIG["IMG_SIZE"], CONFIG["IMG_SIZE"]),
104     batch_size=CONFIG["BATCH_SIZE"], class_mode="binary", shuffle=False
105 )
106
107 return train_gen, val_gen, test_gen
108
109
110 # 3. ENTRENAMIENTO (DOS FASES)
111
112 def train_model(model, train_gen, val_gen): 1 usage
113     # Pesos de clase
114     idx = train_gen.classes
115     weights = class_weight.compute_class_weight(
116         class_weight='balanced', classes=np.unique(idx), y=idx
117     )
118     weights_dict = dict(enumerate(weights))
119     print(f"\n[2/7] Pesos de Clase: {weights_dict}")
120
121     loss_fn = BinaryCrossentropy(label_smoothing=CONFIG["LABEL_SMOOTHING"])
122
123     # --- FASE 1: WARMUP ---
124     print("\n[3/7] FASE 1: Calentamiento...")
125     for layer in model.layers:
126         layer.trainable = False if "mobilenet" in layer.name.lower() else True
127
128     model.compile(optimizer=Adam(learning_rate=CONFIG["LR_WARMUP"]), loss=loss_fn, metrics=["accuracy"])

```

Figura 32. código de entrenamiento.

```

112 def train_model(model, train_gen, val_gen): 1usage
128 model.compile(optimizer=Adam(learning_rate=CONFIG["LR_WARMUP"]), loss=loss_fn, metrics=["accuracy"])
129
130 model.fit(
131     train_gen, validation_data=val_gen, epochs=CONFIG["EPOCHS_WARMUP"],
132     class_weight=weights_dict, verbose=1
133 )
134
135 # Limpieza
136 gc.collect()
137 tf.keras.backend.clear_session()
138
139 # --- FASE 2: FINE TUNING ---
140 print(f"\n[4/7] FASE 2: Fine-Tuning ({CONFIG['UNFREEZE_LAYERS']}: capas)...")
141
142 base_len = len(model.layers)
143 for i, layer in enumerate(model.layers):
144     if i < (base_len - CONFIG["UNFREEZE_LAYERS"]):
145         layer.trainable = False
146     else:
147         # NO entrenar BatchNormalization con batch pequeño
148         if isinstance(layer, tf.keras.layers.BatchNormalization):
149             layer.trainable = False
150         else:
151             layer.trainable = True
152
153 model.compile(optimizer=Adam(learning_rate=CONFIG["LR_FINE"]), loss=loss_fn, metrics=["accuracy"])
154
155 callbacks = [
156     # Paciencia aumentada para dejar que aprenda lento (V7.1)
157     EarlyStopping(patience=18, restore_best_weights=True, monitor="val_loss", verbose=1),
158     ModelCheckpoint(CONFIG["MODELO_SALIDA"], save_best_only=True, monitor="val_accuracy", verbose=1),

```

Figura 33. código de entrenamiento.

```

112 def train_model(model, train_gen, val_gen): 1usage
158 ModelCheckpoint(CONFIG["MODELO_SALIDA"], save_best_only=True, monitor="val_accuracy", verbose=1),
159 ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=6, min_lr=1e-11, verbose=1),
160 CSVLogger(f"{CONFIG['OUTPUT_DIR']}/training_log.csv")
161 ]
162
163 history = model.fit(
164     train_gen, validation_data=val_gen, epochs=CONFIG["EPOCHS_FINE"],
165     callbacks=callbacks, class_weight=weights_dict, verbose=1
166 )
167
168 return model, history
169
170 # =====
171 # 4. GENERACIÓN DE GRÁFICAS
172 # =====
173 def graficar_historial(history): 1usage
174     print("\n[5/7] Generando Gráficas de Aprendizaje...")
175
176     acc = history.history['accuracy']
177     val_acc = history.history['val_accuracy']
178     loss = history.history['loss']
179     val_loss = history.history['val_loss']
180     epochs_range = range(len(acc))
181
182     plt.figure(figsize=(16, 6))
183
184     # Gráfica 1: Precisión
185     plt.subplot(*args: 1, 2, 1)
186     plt.plot(*args: epochs_range, acc, label='Entrenamiento')
187     plt.plot(*args: epochs_range, val_acc, label='Validación')
188     plt.legend(loc='lower right')

```

Figura 34. código de entrenamiento.

```

173 def graficar_historial(history):
174     plt.legend(loc='lower right')
175     plt.title('Historial de Precisión (Accuracy)')
176     plt.xlabel('Épocas')
177     plt.ylabel('Precisión')
178     plt.grid(visible=True, alpha=0.3)
179
180     # Gráfica 2: Pérdida
181     plt.subplot(*args: 1, 2, 2)
182     plt.plot(*args: epochs_range, loss, label='Entrenamiento')
183     plt.plot(*args: epochs_range, val_loss, label='Validación')
184     plt.legend(loc='upper right')
185     plt.title('Historial de Pérdida (Loss)')
186     plt.xlabel('Épocas')
187     plt.ylabel('Error')
188     plt.grid(visible=True, alpha=0.3)
189
190     plt.tight_layout()
191     plt.savefig(*args: f"{CONFIG['OUTPUT_DIR']}/GRAFICA_APRENDIZAJE.png", dpi=300)
192     plt.close()
193     print(">> Guardado: GRAFICA_APRENDIZAJE.png")
194
195 # =====
196 # 5. AUDITORIA FINAL
197 # =====
198 def auditar_modelo(model, test_gen):
199     print("\n[6/7] Generando Evidencia Final (Matriz y ROC)...")
200
201     test_gen.reset()
202     y_raw = model.predict(test_gen, verbose=1).ravel()
203     y_true = test_gen.classes
204     y_pred = (y_raw >= 0.5).astype(int) # Umbral estándar para auditoría

```

Figura 35. código de entrenamiento.

```

def auditar_modelo(model, test_gen):
    y_pred = (y_raw >= 0.5).astype(int) # Umbral estándar para auditoría

    class_names = list(test_gen.class_indices.keys())

    # 1. Matriz de Confusión
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Matriz de Confusión Final')
    plt.ylabel('Realidad')
    plt.xlabel('Predicción')
    plt.tight_layout()
    plt.savefig(*args: f"{CONFIG['OUTPUT_DIR']}/GRAFICA_MATRIZ.png", dpi=300)
    plt.close()
    print(">> Guardado: GRAFICA_MATRIZ.png")

    # 2. Curva ROC
    fpr, tpr, _ = roc_curve(y_true, y_raw)
    auc_val = auc(fpr, tpr)
    plt.figure(figsize=(8, 6))
    plt.plot(*args: fpr, tpr, color='darkorange', lw=2, label=f'AUC = {auc_val:.4f}')
    plt.plot(*args: [0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('Falsos Positivos')
    plt.ylabel('Verdaderos Positivos')
    plt.title('Curva ROC Final')
    plt.legend(loc="lower right")
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.savefig(*args: f"{CONFIG['OUTPUT_DIR']}/GRAFICA_ROC.png", dpi=300)
    plt.close()

```

Figura 36. código de entrenamiento.

```

# 3. Reporte Texto
with open(f"{CONFIG['OUTPUT_DIR']}/REPORTE_FINAL.txt", "w") as f:
    f.write("=== REPORTE TÉCNICO V7.1 (EXCELENCIA) ===\n")
    f.write(f"AUC: {auc_val:.4f}\n")
    f.write(f"Config: LS=0.20, LR=3e-6, Epochs=120\n")
    f.write(f"- * 30 * \n")
    f.write(classification_report(y_true, y_pred, target_names=class_names))

return auc_val

# =====
# MAIN
# =====
def main(): 1 usage
    if not os.path.exists(CONFIG["MODELO_BASE"]):
        # CORRECCIÓN DE SINTAXIS PARA PYTHON 3.10: Comillas simples dentro de f-string
        print(f" Error: Falta {CONFIG['MODELO_BASE']}")
        return

    # 1. Cargar
    train, val, test = get_generators()
    print(f"\nCargando modelo base...")
    model = load_model(CONFIG["MODELO_BASE"])

    # 2. Entrenar
    model_final, history = train_model(model, train, val)

    # 3. Graficar Aprendizaje (NUEVO)

```

Figura 37. código de entrenamiento.

```

264 def main(): 1 usage
270     # 1. Cargar
271     train, val, test = get_generators()
272     print(f"\nCargando modelo base...")
273     model = load_model(CONFIG["MODELO_BASE"])
274
275     # 2. Entrenar
276     model_final, history = train_model(model, train, val)
277
278     # 3. Graficar Aprendizaje (NUEVO)
279     graficar_historial(history)
280
281     # 4. Auditar
282     auc_final = auditar_modelo(model_final, test)
283
284     print("\n" + "="*60)
285     print(f" TODO COMPLETADO. AUC FINAL: {auc_final:.4f}")
286     print(f"   Todas las gráficas están en: {CONFIG['OUTPUT_DIR']}")
287     print("="*60)
288
289 if __name__ == "__main__":
290     try:
291         main()
292     except Exception as e:
293         print(f"\n Error: {e}")
294         import traceback
295         traceback.print_exc()

```

Figura 38. código de entrenamiento.

ANEXO B

CÓDIGO ENTRENAMIENTO EFFICIENTNET-B0

```
1 import os
2 import sys
3 import time
4 import copy
5 import random
6 import gc
7 import logging
8 import warnings
9 from pathlib import Path
10 from typing import Dict, Tuple, List, Any
11
12 # Filtrar advertencias
13 warnings.filterwarnings(action="ignore", category=UserWarning)
14 warnings.filterwarnings(action="ignore", category=FutureWarning)
15
16 import numpy as np
17 import pandas as pd
18 import matplotlib.pyplot as plt
19 import seaborn as sns
20 from PIL import Image, ImageFile
21 from tqdm import tqdm
22
23 import torch
24 import torch.nn as nn
25 import torch.optim as optim
26 from torch.utils.data import DataLoader
27 from torchvision import datasets, transforms
28
29 import timm
30 from sklearn.metrics import classification_report, confusion_matrix, f1_score
```

Figura 39. código de entrenamiento efficientnet-b0.

```
entrenamiento_convnetxt.py  entrenamiento_EfficientNet-B0.py
31 import timm
32 from sklearn.metrics import classification_report, confusion_matrix, f1_score
33
34 # =====
35 # CONFIGURACIÓN
36 # =====
37 ImageFile.LOAD_TRUNCATED_IMAGES = True
38
39 logger = logging.getLogger(__name__)
40 logger.setLevel(logging.INFO)
41 if logger.hasHandlers():
42     logger.handlers.clear()
43
44 file_handler = logging.FileHandler(filename="entrenamiento_sistema.log", encoding='utf-8')
45 file_handler.setFormatter(logging.Formatter('%(asctime)s | %(levelname)s | %(message)s'))
46 logger.addHandler(file_handler)
47
48 stream_handler = logging.StreamHandler(sys.stdout)
49 stream_handler.setFormatter(logging.Formatter('%(asctime)s | %(levelname)s | %(message)s'))
50 logger.addHandler(stream_handler)
51
52
53 class Config:
54     DATA_DIR = Path('./dataset_224_ready')
55     OUTPUT_DIR = Path('./resultados_entrenamiento_')
56
57     MODEL_NAME = 'efficientnet_b0'
58     NUM_CLASSES = 1
59     INPUT_SIZE = 224
60
61     BATCH_SIZE = 16
62
```

Figura 40. código de entrenamiento efficientnet-b0.

```

NUM_EPOCHS = 50
EARLY_STOPPING_PATIENCE = 7 # Si no mejora en 7 épocas, para

LEARNING_RATE = 1e-4
WEIGHT_DECAY = 1e-2

# CRÍTICO PARA WINDOWS
NUM_WORKERS = 0 if os.name == 'nt' else 2
PIN_MEMORY = True if torch.cuda.is_available() else False
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

ALPHA = 0.25
GAMMA = 2.0

Config.OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# =====
# 1. UTILIDADES DE SISTEMA
# =====
class EarlyStopping: 1 usage
    """Detiene el entrenamiento si la validación no mejora tras 'patience' épocas."""

    def __init__(self, patience=7, min_delta=0.001):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_loss = None

```

Figura 41. código de entrenamiento efficientnet-b0.

```

class EarlyStopping: 1 usage
88 def __init__(self, patience=7, min_delta=0.001):
89     self.early_stop = False
90     self.best_score = None
91
92     def __call__(self, val_loss):
93         score = -val_loss
94
95         if self.best_score is None:
96             self.best_score = score
97             self.best_loss = val_loss
98         elif score < self.best_score + self.min_delta:
99             self.counter += 1
100             logger.info(f" [INFO] Early Stopping: {self.counter}/{self.patience} sin mejora significativa.")
101             if self.counter >= self.patience:
102                 self.early_stop = True
103         else:
104             self.best_score = score
105             self.best_loss = val_loss
106             self.counter = 0
107
108 def seed_everything(seed: int = 42): 1 usage
109     random.seed(seed)
110     os.environ['PYTHONHASHSEED'] = str(seed)
111     np.random.seed(seed)
112     torch.manual_seed(seed)
113     torch.cuda.manual_seed(seed)
114     torch.backends.cudnn.deterministic = True
115     torch.backends.cudnn.benchmark = True

```

Figura 42. código de entrenamiento efficientnet-b0.

```

def validate_dataset_structure(): 1 usage
    required_splits = ['train', 'val', 'test']
    if not Config.DATA_DIR.exists():
        logger.critical(f"NO SE ENCUENTRA EL DATASET EN: {Config.DATA_DIR.absolute()}")
        sys.exit(1)
    for split in required_splits:
        split_path = Config.DATA_DIR / split
        if not split_path.exists():
            logger.critical(f"Falta la carpeta '{split}' dentro del dataset.")
            sys.exit(1)
    logger.info("Estructura de directorios validada correctamente.")

# =====
# 2. MODELAJO
# =====

class FocalLoss(nn.Module): 2 usages
    def __init__(self, alpha=0.25, gamma=2.0, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction
        self.bce_with_logits = nn.BCEWithLogitsLoss(reduction='none')

    def forward(self, inputs, targets):
        bce_loss = self.bce_with_logits(inputs, targets)
        pt = torch.exp(-bce_loss)
        focal_loss = self.alpha * (1 - pt) ** self.gamma * bce_loss
        return focal_loss.mean() if self.reduction == 'mean' else focal_loss.sum()

```

Figura 43. código de entrenamiento efficientnet-b0.

```

def get_dataloaders() -> Tuple[Dict, Dict, List[str], int]: 1 usage
    logger.info(f"Configurando DataLoaders (Workers: {Config.NUM_WORKERS}, PinMemory: {Config.PIN_MEMORY})...")

    train_transforms = transforms.Compose([
        transforms.Resize((Config.INPUT_SIZE, Config.INPUT_SIZE)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(30), # Aumentado de 20 a 30
        transforms.ColorJitter(brightness=0.15, contrast=0.15),
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)), # Nuevo: traslación leve
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    val_test_transforms = transforms.Compose([
        transforms.Resize((Config.INPUT_SIZE, Config.INPUT_SIZE)),
        transforms.ToTensor(),

```

```

entrenamiento_convnetxt.py  entrenamiento_EfficientNet-B0.py ×
211 def train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs=50): 1 usage
299     if phase == 'val' and torch.cuda.is_available():
300         gc.collect()
301         torch.cuda.empty_cache()
302
303     # Si Early Stopping se activa, paramos el entrenamiento
304     if early_stopping.early_stop:
305         logger.info(f"\n[INFO] Early Stopping activado en epoca {epoch + 1}. Deteniendo entrenamiento.")
306         break
307
308     scheduler.step()
309     metrics_log.append(epoch_metrics)
310
311     # Generación de reportes
312     try:
313         df = pd.DataFrame(metrics_log)
314         df.to_csv(Config.OUTPUT_DIR / 'training_history.csv', index=False)
315
316         plt.figure(figsize=(12, 5))
317         plt.subplot(*args: 1, 2, 1)
318         plt.plot(*args: df['epoch'], df['train_loss'], label='Train Loss', marker='o')
319         plt.plot(*args: df['epoch'], df['val_loss'], label='Val Loss', marker='o')
320         plt.title('Loss')
321         plt.legend()
322         plt.grid(True)
323
324         plt.subplot(*args: 1, 2, 2)
325         plt.plot(*args: df['epoch'], df['train_acc'], label='Train Acc', marker='o')
326         plt.plot(*args: df['epoch'], df['val_acc'], label='Val Acc', marker='o')
327         plt.title('Accuracy')
328         plt.legend()
329         plt.grid(True)

```

Figura 49. código de entrenamiento efficientnet-b0.

```

entrenamiento_convnetxt.py  entrenamiento_EfficientNet-B0.py ×
211 def train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs=50): 1 usage
332     plt.savefig(Config.OUTPUT_DIR / 'curvas_entrenamiento.png')
333     plt.close()
334     except Exception:
335         pass
336
337     # Cargar el mejor modelo encontrado (no necesariamente el último)
338     logger.info("Cargando pesos del mejor modelo obtenido durante el entrenamiento...")
339     model.load_state_dict(best_model_wts)
340     return model
341
342
343     # =====
344     # 4. POST-PROCESAMIENTO
345     # =====
346     def find_optimal_threshold(model, dataloader, critical_idx): 1 usage
347         model.eval()
348         y_true, y_scores = [], []
349         logger.info("Optimizando umbral de decision...")
350
351         with torch.no_grad():
352             for inputs, labels in tqdm(dataloader, desc=" Optimizando", file=sys.stdout):
353                 inputs = inputs.to(Config.DEVICE)
354                 outputs = model(inputs)
355                 probs = torch.sigmoid(outputs)
356                 y_scores.extend(probs.cpu().numpy().flatten())
357                 y_true.extend(labels.numpy().flatten())
358
359         y_true = np.array(y_true)
360         y_scores = np.array(y_scores)
361         thresholds = np.arange(0.1, 0.96, 0.02)
362         best_f1 = 0.0

```

Figura 50. código de entrenamiento efficientnet-b0.

```

def find_optimal_threshold(model, dataloader, critical_idx): 1 usage
    best_f1 = 0.0
    best_thresh = 0.5
    pos_label = critical_idx

    for thresh in thresholds:
        y_pred = (y_scores >= thresh).astype(int)
        score = f1_score(y_true, y_pred, pos_label=pos_label, average='binary')
        if score > best_f1:
            best_f1 = score
            best_thresh = thresh

    logger.info(f"[EXIT0] Umbral Optimo: {best_thresh:.2f} (F1: {best_f1:.4f})")

    # Gráfica de umbral
    try:
        plt.figure()
        plt.plot(*args thresholds,
                 [f1_score(y_true, (y_scores >= t).astype(int), pos_label=pos_label, average='binary') for t in
                  thresholds])
        plt.axvline(best_thresh, color='r', linestyle='--')
        plt.title(f'Optimización Umbral (Max: {best_thresh:.2f})')
        plt.savefig(Config.OUTPUT_DIR / 'optimizacion_umbral.png')
        plt.close()
    except Exception:
        pass

    return best_thresh

def evaluate_test_set(model, dataloader, class_names, threshold): 1 usage
    logger.info("Evaluando Test Set...")

```

Figura 51. código de entrenamiento efficientnet-b0.

```

entrenamiento_convnext.py  entrenamiento_efficientnet-b0.py
91 def evaluate_test_set(model, dataloader, class_names, threshold): 1 usage
92     with torch.no_grad():
93         y_true.extend(labels.cpu().numpy().flatten())
94         y_pred.extend(preds.cpu().numpy().flatten())
95
96     report = classification_report(y_true, y_pred, target_names=class_names, digits=4)
97     print("\n" + report)
98
99     with open(Config.OUTPUT_DIR / 'reporte_final.txt', 'w') as f:
100         f.write(f"Umbral Usado: {threshold:.4f}\n\n")
101         f.write(report)
102
103     try:
104         cm = confusion_matrix(y_true, y_pred)
105         plt.figure(figsize=(8, 6))
106         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
107         plt.title(f'Matriz de Confusion (Umbral: {threshold:.2f})')
108         plt.savefig(Config.OUTPUT_DIR / 'matriz_confusion_final.png')
109         plt.close()
110     except Exception:
111         pass
112
113 if __name__ == '__main__':
114     try:
115         seed_everything()
116         validate_dataset_structure()
117
118         logger.info("-" * 40)
119         logger.info(f"[HARDWARE] GPU: {torch.cuda.get_device_name(0) if torch.cuda.is_available() else 'CPU'}")
120         logger.info("-" * 40)

```

Figura 52. código de entrenamiento efficientnet-b0.

ANEXO C

CÓDIGO ENTRENAMIENTO CONVNEXT

```
4 import sys
5 import time
6 import copy
7 import random
8 import gc
9 import logging
10 import warnings
11 from pathlib import Path
12 from typing import Dict, Tuple, List, Any
13
14 # Filtramos advertencias de librerías para mantener la consola limpia
15 warnings.filterwarnings(action="ignore", category=UserWarning)
16 warnings.filterwarnings(action="ignore", category=FutureWarning)
17
18 import numpy as np
19 import pandas as pd
20 import matplotlib.pyplot as plt
21 import seaborn as sns
22 from PIL import Image, ImageFile
23 from tqdm import tqdm
24
25 import torch
26 import torch.nn as nn
27 import torch.optim as optim
28 from torch.utils.data import DataLoader
29 from torchvision import datasets, transforms
30
31 # Librería de modelos SOTA
32 import timm
33 from sklearn.metrics import classification_report, confusion_matrix, f1_score
34
35 # =====
```

Figura 53. código de entrenamiento convnext.

```
28 from torch.utils.data import DataLoader
29 from torchvision import datasets, transforms
30
31 # Librería de modelos SOTA
32 import timm
33 from sklearn.metrics import classification_report, confusion_matrix, f1_score
34
35 # =====
36 # CONFIGURACIÓN IA
37 # =====
38 ImageFile.LOAD_TRUNCATED_IMAGES = True
39
40 # Configuración de Logging para Tesis
41 logger = logging.getLogger('ConvNeXt_Trainer')
42 logger.setLevel(logging.INFO)
43 if logger.handlers: logger.handlers.clear()
44
45 # Guardar log en archivo UTF-8 (Compatible con Windows)
46 fh = logging.FileHandler(filename="entrenamiento_convnext.log", encoding='utf-8')
47 fh.setFormatter(logging.Formatter('%(asctime)s | %(levelname)s | %(message)s'))
48 logger.addHandler(fh)
49
50 # Mostrar log en consola
51 ch = logging.StreamHandler(sys.stdout)
52 ch.setFormatter(logging.Formatter('%(asctime)s | %(levelname)s | %(message)s'))
53 logger.addHandler(ch)
54
55
56 class Config:
57     # Rutas
58     DATA_DIR = Path('dataset_224_ready')
59     OUTPUT_DIR = Path('./ConvNeXt') # Carpeta separada para no mezclar con EfficientNet
```

Figura 54. código de entrenamiento convnext.

```

56 class Config:
57     OUTPUT_DIR = Path('./ConvNeXt') # Carpeta separada para no mezclar con EfficientNet
58
59     # Modelo: Usamos la versión Nano pre-entrenada en ImageNet-12k (Mejor conocimiento base)
60     MODEL_NAME = 'convnext_nano.in12k_ft_in1k'
61     NUM_CLASSES = 1
62     INPUT_SIZE = 224
63
64     # Hiperparámetros ajustados para ConvNeXt
65     BATCH_SIZE = 16 # Límite seguro para 4GB VRAM
66     NUM_EPOCHS = 50 # ConvNeXt tarda un poco más en converger que EfficientNet
67     PATIENCE = 10 # Early Stopping
68
69     # Receta de entrenamiento moderna (AdamW + Weight Decay alto)
70     LEARNING_RATE = 4e-5
71     WEIGHT_DECAY = 0.05
72
73     # Ajustes de Hardware (Windows Safe)
74     NUM_WORKERS = 0 if os.name == 'nt' else 2
75     PIN_MEMORY = True if torch.cuda.is_available() else False
76     DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
77
78     # Focal Loss (Mantenemos la configuración que te funcionó bien)
79     ALPHA = 0.25
80     GAMMA = 2.0
81
82     Config.OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
83
84
85
86
87

```

Figura 55. código de entrenamiento convnext.

```

6 # =====
7 # 1. UTILIDADES
8 # =====
9
10 def seed_everything(seed=42): 1 usage
11     random.seed(seed)
12     os.environ['PYTHONHASHSEED'] = str(seed)
13     np.random.seed(seed)
14     torch.manual_seed(seed)
15     torch.cuda.manual_seed(seed)
16     torch.backends.cudnn.deterministic = True
17     torch.backends.cudnn.benchmark = True
18
19
20 class EarlyStopping: 1 usage
21     """Detiene el entrenamiento si no mejora para evitar Overfitting."""
22
23     def __init__(self, patience=7, min_delta=0.001):
24         self.patience = patience
25         self.min_delta = min_delta
26         self.counter = 0
27         self.best_loss = None
28         self.early_stop = False
29
30     def __call__(self, val_loss):
31         if self.best_loss is None:
32             self.best_loss = val_loss
33         elif val_loss < self.best_loss:
34             self.best_loss = val_loss
35         else:
36             self.counter += 1
37             if self.counter == self.patience:
38                 self.early_stop = True

```

Figura 56. código de entrenamiento convnext.

```

        elif val_loss > self.best_loss - self.min_delta:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_loss = val_loss
            self.counter = 0

# =====
# 2. DATA PIPELINE
# =====
def get_data_loaders(): 1 usage
    logger.info("Configurando DataLoaders y Aumentación avanzada...")

    train_transforms = transforms.Compose([
        transforms.Resize((Config.INPUT_SIZE, Config.INPUT_SIZE)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(20),
        transforms.ColorJitter(brightness=0.2, contrast=0.2), # ConvNeXt tolera más jitter
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        # Random Erasing: Clave para robustez en ConvNeXt
        transforms.RandomErasing(p=0.25, scale=(0.02, 0.15))
    ])

    val_transforms = transforms.Compose([
        transforms.Resize((Config.INPUT_SIZE, Config.INPUT_SIZE)),

```

Figura 57. código de entrenamiento convnext.

```

126 def get_data_loaders(): 1 usage
143     transforms.ToTensor(),
144     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
145 ])
146
147     datasets_dict = {
148         'train': datasets.ImageFolder(Config.DATA_DIR / 'train', train_transforms),
149         'val': datasets.ImageFolder(Config.DATA_DIR / 'val', val_transforms),
150         'test': datasets.ImageFolder(Config.DATA_DIR / 'test', val_transforms)
151     }
152
153     class_names = datasets_dict['train'].classes
154     # Identificar clase crítica automáticamente
155     crit_idx = next((i for i, c in enumerate(class_names) if "no_conforme" in c.lower()), 0)
156     logger.info(f"Clase crítica detectada: {class_names[crit_idx]} (ID: {crit_idx})")
157
158     loaders = {
159         x: DataLoader(datasets_dict[x], batch_size=Config.BATCH_SIZE,
160                       shuffle=(x == 'train'), num_workers=Config.NUM_WORKERS,
161                       pin_memory=Config.PIN_MEMORY)
162         for x in ['train', 'val', 'test']
163     }
164
165     return loaders, class_names, crit_idx
166
167
168 # =====
169 # 3. COMPONENTES DEL MODELO
170 # =====
171 class FocalLoss(nn.Module): 1 usage
172     def __init__(self, alpha=0.25, gamma=2.0):
173         super().__init__()

```

Figura 58. código de entrenamiento convnext.

```

27
28 # =====
29 # 3. COMPONENTES DEL MODELO
30 # =====
31 class FocalLoss(nn.Module): 1 usage
32     def __init__(self, alpha=0.25, gamma=2.0):
33         super().__init__()
34         self.alpha = alpha
35         self.gamma = gamma
36         self.bce = nn.BCEWithLogitsLoss(reduction='none')
37
38     def forward(self, inputs, targets):
39         bce_loss = self.bce(inputs, targets)
40         pt = torch.exp(-bce_loss)
41         f_loss = self.alpha * (1 - pt) ** self.gamma * bce_loss
42         return f_loss.mean()
43
44
45 def train_engine(model, loaders, criterion, optimizer, scheduler): 1 usage
46     since = time.time()
47     early_stopping = EarlyStopping(patience=Config.PATIENCE)
48
49     # Scaler para Mixed Precision (Vital para GTX 1650)
50     scaler = torch.amp.GradScaler(device='cuda', enabled=(Config.DEVICE.type == 'cuda'))
51
52     best_loss = float('inf')
53     best_weights = copy.deepcopy(model.state_dict())
54     history = []
55
56     logger.info(f"Iniciando entrenamiento ConvNext en {Config.DEVICE}...")
57

```

Figura 59. código de entrenamiento convnext.

```

185 def train_engine(model, loaders, criterion, optimizer, scheduler): 1 usage
186
187     for epoch in range(Config.NUM_EPOCHS):
188         current_lr = optimizer.param_groups[0]['lr']
189         print(f"\n[Epoca {epoch + 1}/{Config.NUM_EPOCHS}] LR: {current_lr:.2e}")
190
191         epoch_stats = {'epoch': epoch + 1}
192
193         for phase in ['train', 'val']:
194             if phase == 'train':
195                 model.train()
196             else:
197                 model.eval()
198
199             running_loss = 0.0
200             running_corrects = 0
201
202             # Barra de progreso
203             pbar = tqdm(loaders[phase], desc=f" {phase.upper()}", file=sys.stdout)
204
205             for inputs, labels in pbar:
206                 inputs = inputs.to(Config.DEVICE, non_blocking=True)
207                 labels = labels.to(Config.DEVICE, non_blocking=True).float().unsqueeze(1)
208
209                 optimizer.zero_grad(set_to_none=True)
210
211                 with torch.set_grad_enabled(phase == 'train'):
212                     # AMP Context Manager
213                     with torch.amp.autocast('cuda', enabled=(Config.DEVICE.type == 'cuda')):
214                         outputs = model(inputs)
215                         loss = criterion(outputs, labels)
216

```

Figura 60. código de entrenamiento convnext.

```

185 def train_engine(model, loaders, criterion, optimizer, scheduler): 1 usage
222     with torch.set_grad_enabled(phase == 'train'):
228         if phase == 'train':
229             scaler.scale(loss).backward()
230             scaler.step(optimizer)
231             scaler.update()
232
233             running_loss += loss.item() * inputs.size(0)
234             preds = (torch.sigmoid(outputs) > 0.5).float()
235             running_corrects += torch.sum(preds == labels.data)
236
237             # Actualizar barra
238             pbar.set_postfix(loss=loss.item())
239
240             epoch_loss = running_loss / len(loaders[phase].dataset)
241             epoch_acc = running_corrects.double() / len(loaders[phase].dataset)
242
243             epoch_stats[f'{phase}_loss'] = epoch_loss
244             epoch_stats[f'{phase}_acc'] = epoch_acc.item()
245
246             print(f" -> {phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")
247
248             if phase == 'val':
249                 if epoch_loss < best_loss:
250                     best_loss = epoch_loss
251                     best_weights = copy.deepcopy(model.state_dict())
252                     logger.info(f" [RECORD] Nueva mejor pérdida: {best_loss:.4f}")
253
254                 early_stopping(epoch_loss)
255
256             # Limpieza de VRAM tras validación
257             if phase == 'val' and torch.cuda.is_available():

```

Figura 61. código de entrenamiento convnext.

```

# =====
# 4. EVALUACIÓN Y GUARDADO DE ARTEFACTOS
# =====
def evaluate_and_save(model, loaders, class_names, crit_idx): 1 usage
    logger.info("Iniciando fase de evaluación y optimización...")
    model.eval()

    # 1. Threshold Moving (Encontrar el umbral perfecto)
    y_true, y_probs = [], []
    with torch.no_grad():
        for inputs, labels in loaders['val']:
            outputs = model(inputs.to(Config.DEVICE))
            y_probs.extend(torch.sigmoid(outputs).cpu().numpy().flatten())
            y_true.extend(labels.numpy().flatten())

    thresholds = np.arange(0.1, 0.96, 0.01)
    best_f1, best_t = 0.0, 0.5
    for t in thresholds:
        # Calculamos F1 considerando la clase critica
        score = f1_score(y_true, (np.array(y_probs) >= t).astype(int), pos_label=crit_idx)
        if score > best_f1: best_f1, best_t = score, t

    logger.info(f"Umbral Optimizado: {best_t:.3f} (F1-Score: {best_f1:.4f})")

    # 2. Test Final con el Umbral Optimizado
    y_test, y_pred = [], []
    with torch.no_grad():
        for inputs, labels in tqdm(loaders['test'], desc=" TEST FINAL"):
            outputs = model(inputs.to(Config.DEVICE))
            probs = torch.sigmoid(outputs).cpu().numpy().flatten()

```

Figura 62. código de entrenamiento convnext.

```

337
338
339 ▶ if __name__ == '__main__':
340     try:
341         seed_everything(42)
342         logger.info(f"Iniciando Pipeline. GPU: {torch.cuda.get_device_name(0) if torch.cuda.is_available() else 'CPU'}")
343
344         # 1. Cargar Arquitectura SOTA
345         model = timm.create_model(Config.MODEL_NAME, pretrained=True, num_classes=Config.NUM_CLASSES)
346         model = model.to(Config.DEVICE)
347
348         # 2. Datos
349         loaders, names, crit_idx = get_data_loaders()
350
351         # 3. Entrenamiento
352         criterion = FocalLoss(alpha=Config.ALPHA, gamma=Config.GAMMA)
353         optimizer = optim.AdamW(model.parameters(), lr=Config.LEARNING_RATE, weight_decay=Config.WEIGHT_DECAY)
354         scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=Config.NUM_EPOCHS)
355
356         model = train_engine(model, loaders, criterion, optimizer, scheduler)
357
358         # 4. Evaluación y Guardado
359         evaluate_and_save(model, loaders, names, crit_idx)
360
361         logger.info("ENTRENAMIENTO COMPLETADO.")
362
363     except Exception as e:
364         logger.critical(msg=f"ERROR FATAL: {e}", exc_info=True)
365         sys.exit(1)

```

Figura 63. código de entrenamiento convnext.