UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

TESIS

IMPLEMENTACIÓN DE UN SISTEMA DE VISIÓN ARTIFICIAL CON TRANSFER LEARNING PARA EL DIAGNÓSTICO DE DEFECTOS DE CALIDAD EN LAS ALCACHOFAS

PRESENTADO POR:
Br. ARON YABAR AGUILAR

PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO INFORMÁTICO Y DE SISTEMAS

ASESOR:
DR. ROBERT WILBERT ALZAMORA PAREDES

CUSCO - PERU
2024

INFORME DE ORIGINALIDAD

(Aprobado por Resolución Nro.CU-303-2020-UNSAAC)

El que suscribe, **Asesor** del trabajo de investigación/tesis titulada: Implementación de un sistema de visión artificial con Transfer Learning para el diagnóstico de defectos de calidad en los alcachofas

presentado por: Aron Yabar Aguilar con DNI Nro.: 72641187 presentado por: ..... con DNI Nro.: ..... para optar el título profesional/grado académico de Ingeniero Informatica y de sistemas

Informo que el trabajo de investigación ha sido sometido a revisión por 01 veces, mediante el Software Antiplagio, conforme al Art. 6° del ***Reglamento para Uso de Sistema Antiplagio de la UNSAAC*** y de la evaluación de originalidad se tiene un porcentaje de 3 **%.**

Evaluación y acciones del reporte de coincidencia para trabajos de investigación conducentes a grado académico o título profesional, tesis

Porcentaje	Evaluación y Acciones	Marque con una (X)
Del 1 al 10%	No se considera plagio.	X
Del 11 al 30 %	Devolver al usuario para las correcciones.	
Mayor a 31%	El responsable de la revisión del documento emite un informe al inmediato jerárquico, quien a su vez eleva el informe a la autoridad académica para que tome las acciones correspondientes. Sin perjuicio de las sanciones administrativas que correspondan de acuerdo a Ley.	

Por tanto, en mi condición de asesor, firmo el presente informe en señal de conformidad y **adjunto** la primera página del reporte del Sistema Antiplagio.

Cusco, 05 de Diciembre de 20.24

.....
Firma

Post firma Robert W. Alzamora Poncel

Nro. de DNI 23966386

ORCID del Asesor 0000-0002-5955-6009.

Se adjunta:

1. Reporte generado por el Sistema Antiplagio.
2. Enlace del Reporte Generado por el Sistema Antiplagio: **oid:** 27259:413117715

ARON YABAR AGUILAR

IMPLEMENTACION DE UN SISTEMA DE VISIÓN ARTIFICIAL CON TRANSFER LEARNING PARA EL DIAGNOSTICO DE DEFE...

Universidad Nacional San Antonio Abad del Cusco

Detalles del documento

Identificador de la entrega
trn:oid:::27259:413117715

Fecha de entrega
5 dic 2024, 8:37 p.m. GMT-5

Fecha de descarga
5 dic 2024, 10:18 p.m. GMT-5

Nombre de archivo
Tesis Aron Yabar - IMPLEMENTACION DE UN SISTEMA DE VISION PARA ALCACHOFAS.pdf

Tamaño de archivo
25.7 MB

100 Páginas

25,675 Palabras

140,682 Caracteres

3% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado
- Texto mencionado
- Coincidencias menores (menos de 10 palabras)
- Trabajos entregados

Exclusiones

- N.° de coincidencias excluidas

Fuentes principales

3%	Fuentes de Internet
0%	Publicaciones
0%	Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.° de alerta de integridad para revisión

Caracteres reemplazados
9 caracteres sospechosos en N.° de páginas
Las letras son intercambiadas por caracteres similares de otro alfabeto.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

Dedicatoria

Dedico este trabajo especialmente a mi familia (padres y hermanos), a mi novia y en especial, a mi hijo que está por llegar a este mundo. Ellos han sido mi principal fuente de motivación para alcanzar mis objetivos. Su apoyo incondicional ha sido fundamental a lo largo de esta etapa, especialmente en los momentos más difíciles, cuando sus palabras de aliento me dieron la fuerza necesaria para seguir adelante.

Aron Yabar Aguilar

Agradecimientos

Quiero comenzar agradeciendo a Dios por la fortaleza que me ha permitido concluir este objetivo. Asimismo, expreso mi profunda gratitud a mi asesor, el Dr. Robert Wilbert Alzamora Paredes, por su invaluable guía y apoyo en la realización de mi investigación. En momentos de incertidumbre, su orientación fue fundamental para avanzar en este proceso. Agradezco también el conocimiento que me brindó durante mi etapa académica y su apoyo constante en esta fase de investigación.

Resumen

La alcachofa es una inflorescencia de alto valor nutricional y gran demanda en el mercado internacional. En el ámbito nacional la comunidad campesina de Markjo, en la región de Cusco, se destaca como uno de los principales productores de este cultivo. Sin embargo, un problema significativo que afecta la alcachofa en su etapa de inflorescencia son los defectos de calidad, que se manifiestan como malformaciones en el interior de la alcachofa, conocido como el corazón de alcachofa.
Actualmente, los diagnósticos de estos defectos no se realizan de manera adecuada debido a la falta de conocimientos de los agricultores en su identificación. Para mejorar este proceso, se implementó un sistema de visión artificial basado en Transfer Learning, que permite a los agricultores identificar de manera más eficiente los defectos de calidad.
La implementación del sistema incluyó una revisión de fuentes bibliográficas, el desarrollo de un conjunto de datos de imágenes y el entrenamiento con modelos preentrenados de Transfer Learning, como Xception, InceptionV3, DenseNet y ResNet50. Tras analizar los resultados, se encontró que ResNet50 alcanzó la mayor exactitud, con un 93.75 %. Finalmente, se creó una aplicación móvil que utiliza este sistema de visión artificial, facilitando la identificación y diagnóstico de los defectos de calidad en las alcachofas, y permitiendo así su uso práctico por parte de los agricultores.

Palabras clave: Inflorescencia de la alcachofas, Defectos de calidad en las alcachofas, Transfer Learning, Xception, InceptionV3, DenseNet, Resnet50.

Abstract

Artichokes are an inflorescence with high nutritional value and great demand in the international market. Nationally, the campesino community of Markjo in the Cusco region stands out as one of the main producers of this crop. However, a significant problem affecting artichokes during their flowering stage is quality defects, which manifest as malformations in the interior of the artichoke, known as the heart of the artichoke.
Currently, the diagnosis of these defects is not performed adequately due to a lack of knowledge among farmers in identifying them. To improve this process, an artificial vision system based on Transfer Learning was implemented, allowing farmers to identify quality defects more efficiently.
The implementation of the system included a review of bibliographic sources, the development of an image dataset, and training with pre-trained Transfer Learning models such as Xception, InceptionV3, DenseNet and ResNet50. After analyzing the results, it was found that ResNet50 achieved the highest accuracy, at 93.75 %. Finally, a mobile application was created that utilizes this artificial vision system, facilitating the identification and diagnosis of quality defects in artichokes and allowing practical use by farmers.

Keywords: Inflorescence,Quality defects in artichokes, Transfer Learning, Xception, InceptionV3, DenseNet, Resnet50.

Introducción

La alcachofa es una inflorescencia de alto valor nutricional a nivel mundial y en Perú, la comunidad campesina de Markjo ubicada en la región de Cusco se destaca como su principal productora. Durante la etapa de inflorescencia, la alcachofa puede desarrollar defectos de calidad debido a diversos factores, como la falta de agua y minerales. Estos defectos afectan directamente al interior de la alcachofa, conocido como el corazón de alcachofa.
La clasificación de las alcachofas se realiza en tres grupos según la severidad del daño en el corazón. El primer grupo, denominado alcachofa de primera, no presenta ningún tipo de daño. El segundo grupo, alcachofa de segunda, muestra un daño leve. Finalmente, el tercer grupo, alcachofa de descarte, presenta un daño crítico. El diagnóstico de defectos de calidad en las alcachofas durante la etapa de inflorescencia es un procedimiento que realizan los agricultores, que consiste en la identificación visual de los defectos en alcachofas extraídas de diferentes plantas, seguido de la toma de medidas correctivas y/o preventivas. En este contexto, el presente trabajo de investigación busca tiene como objetivo implementar un sistema de visión artificial basado en Transfer Learning, para el diagnóstico de defectos de calidad en las alcachofas durante su etapa de inflorescencia. Para alcanzar este objetivo, se seguirá la siguiente estructura:

- **Capítulo 1 Aspectos generales:** En este capítulo se abordan los siguientes temas: la descripción del problema general, los problemas específicos, antecedentes, la justificación del proyecto, el objetivo general, los objetivos específicos y el cronograma de actividades.
- **Capítulo 2 Marco teórico:** Se expone la base teórica vinculada al trabajo de investigación, la cual será crucial para el desarrollo del proyecto.
- **Capítulo 3 Desarrollo del proyecto:** En este capítulo se abordan los siguientes temas: la metodología usada en el proyecto, el proceso de recolección de imágenes, preprocesamiento de imágenes, las estrategias de aumento de datos, los entrenamientos realizados con Transfer Learning, la implementación de la aplicación móvil y por último se detalla el entorno de entrenamiento.
- **Capítulo 4 Resultados, conclusiones y recomendaciones:** En este capítulo se presentan los resultados, conclusiones y recomendaciones obtenidos a partir del entrenamiento con Transfer Learning.

Listado de abreviaturas

- **Alcachofa**: Hortaliza con alto valor nutricional.
- **Batch size**: Numero de muestras.
- **Brácteas**: Es una hoja que nace del pedúnculo que envuelve la flor de la inflorescencia.
- **CNN**: Redes neuronales convolucionales.
- **Corazón de alcachofa**: Parte central de la inflorescencia de la alcachofa.
- **Cynara scolymus**: Familia de plantas de herbáceas.
- **Data augmentation**: Aumento de datos y/o imagenes.
- **Defectos de calidad**: Son malformaciones que afectan a las inflorescencias de la alcachofa.
- **Épocas**: Numero de ciclos a entrenar.
- **Fine Tuning**: Ajuste fino, ajuste de hiperparametros(descongelamiento de las capas superiores).
- **Flutter**: Es un software creado por Google para el desarrollo de aplicaciones multiplataforma.
- **IA**: Inteligencia artificial.
- **Inflorescencia**: Es una flor que esta sobre una rama.
- **Inflorescencia**: Rama portadora de flores.
- **Ph**: Es la medida de alcalinidad de una solución.
- **RGB**: Es una combinación de colores rojo, azul y verde.
- **PDF**: Es un formato de almacenamiento de documentos.
- **Roseta**: Los pétalos de la inflorescencia tienen una disposición radial.
- **Transfer Learning**: Transferencia de aprendizaje.
- **Tersorflow**: Es una biblioteca utilizada para el aprendizaje automático.
- **Tflite**: Es un archivo que comprime y optimiza el modelo de Tersorflow para ejecutar en dispositivos móviles.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Aspectos generales

1.1. El problema

En la actualidad, la alcachofa es una hortaliza de alto valor nutricional y con gran demanda en el mercado internacional. A nivel nacional, la región de Cusco se destaca como el principal productor de alcachofas, específicamente en la comunidad campesina de Markjo, ubicada en la provincia de Anta. Esta comunidad produce, en promedio *7.249 toneladas por año en una extensión de 700 hectáreas*, con alrededor de *20 cosechas anuales*, según información proporcionada por la empresa *AGRICOLA ALSUR CUSCO*(ver apéndice A.1 punto 3), dedicada al acopio y procesamiento de alcachofa.
Los defectos de calidad son malformaciones que afectan a las alcachofas durante su etapa de *inflorescencia*, impactando directamente en su interior, conocido como el *corazón de alcachofa*. Estos defectos pueden ser causados por múltiples factores, como la falta de agua y minerales. Las alcachofas se clasifican en tres grupos:*alcachofa de primera*, No presenta ningún defecto y es de la más alta calidad; *Alcachofa de segunda*, presenta daños leves en el corazón de la alcachofa, los defectos de calidad son *fofa* y *cintura*; *alcachofa de descarte*, tiene un daño crítico en el corazón de la alcachofa, el defecto de calidad es *violácea*.
Actualmente, los diagnósticos de defectos de calidad en las alcachofas durante la etapa de inflorescencia no se realizan de manera adecuada debido a la falta de conocimiento de los agricultores en la identificación de estos defectos. El procedimiento actual implica tomar muestras aleatorias de alcachofas de diferentes plantas y realizar una identificación visual, lo que conlleva un alto margen de error y resulta en un diagnóstico poco eficiente, basado en el criterio del agricultor. No seguir un procedimiento adecuado en el diagnóstico de defectos de calidad puede llevar a que las medidas correctivas y/o preventivas adoptadas por los agricultores no sean efectivas, e incluso agraven el problema más aun, permitiendo que los defectos se propaguen en los cultivos. Esto provoca que las cosechas no alcancen la máxima calidad y en casos cuando hay un alto índice de alcachofas de descarte, estas son rechazadas en el momento de la venta, destinándose únicamente al consumo animal. Todo lo descrito en el problema general se basa en el apéndice A.1(puntos 1 y 2), documento proporcionado por la empresa *AGRICOLA ALSUR CUSCO*.
A pesar de que existen numerosos trabajos de investigación sobre la clasificación de imágenes en hortalizas mediante visión artificial y el uso de inteligencia artificial, aún no se ha desarrollado una solución específica para diagnosticar la calidad de las alcachofas en su etapa de inflorescencia, ni se cuenta con un dataset basado en imágenes de los defectos de calidad en esta etapa.

1.1.1. Problema general

El diagnóstico de los defectos de calidad en las alcachofas durante la etapa de inflorescencia no se realiza de manera adecuada, lo que ocasiona su propagación en todo el cultivo y como consecuencia la calidad de las alcachofas cosechadas no es la adecuada.

1.1.2. Problemas específicos

1. La ausencia de un dataset basado en imágenes de defectos de calidad en las alcachofas dificulta la implementación de sistemas de diagnóstico automatizado.
2. La falta de modelos de inteligencia artificial (IA) basados en Transfer Learning para la identificación de defectos de calidad en las alcachofas durante la etapa de inflorescencia.
3. La ausencia de una herramienta de usuario final que asista a los agricultores en la identificación de defectos de calidad en las alcachofas, dificulta la obtención de un diagnóstico adecuado. Esta situación contribuye a la propagación de dichos defectos en los cultivos.

1.2. Antecedentes

Detección de enfermedades de la fresa mediante una red neuronal convolucional

Objetivo

Aplicar y comparar modelos avanzados de redes neuronales convolucionales (CNN), específicamente ResNet50, VGG-16 y GoogLeNet, para la detección automatizada y precisa de enfermedades en fresas, como mancha de la hoja, moho gris y oídio, en las variedades "Taoyuan No. 1ᴢ "Xiang-Shui"de Miaoli, Taiwán Jia-Rong Xiao (2021).

Conclusiones

- El modelo ResNet50 aplicado a la detección de enfermedades en fresas mostró una alta exactitud, alcanzando un 98.06 % en el conjunto de datos original y un 99.60 % en el conjunto de datos con características, siendo la mejor exactitud lograda en el periodo de 20 épocas. Jia-Rong Xiao (2021).
- La técnica basada en redes neuronales convolucionales (CNN) proporciona una solución sencilla y efectiva para la detección automatizada de enfermedades en fresas, lo que puede facilitar la identificación temprana de patologías. Jia-Rong Xiao (2021).
- Este enfoque tiene un gran potencial para ser utilizado en la agricultura, ayudando a los agricultores a detectar enfermedades de manera más rápida y precisa, lo que podría mejorar el manejo de los cultivos y reducir las pérdidas en la producción.Jia-Rong Xiao (2021)

- El uso de conjuntos de datos tanto originales como con características demostró ser efectivo, lo que sugiere que la preparación adecuada de los datos es clave para mejorar la exactitud de los modelos de detección.Jia-Rong Xiao (2021)

Comentarios

Este trabajo de Este trabajo de investigación proporciona una base teórica esencial para el estudio en curso, destacando la relevancia de aplicar Transfer Learning. Los modelos de Transfer Learning evaluados fueron Resnet50, VGG-16 y GoogLeNe siendo Resnet 50 el modelo con mayor exactitud con una exactitud de 98.06 %. Los hiperparámetros utilizados fueron los siguientes: 70 % de los datos para entrenamiento, 20 % para validación y 10 % de testeo; una capa de entrada de tamaño (224, 224, 3); optimizadores Adam; entrenamiento durante 20 épocas; una tasa de aprendizaje de 0.0001 y un tamaño de lote de 32.

Un método de aprendizaje profundo en conjunto para la clasificación de cerezas

Objetivo

Clasificar diferentes especies de cerezas utilizando métodos de aprendizaje en conjunto (ensemble learning), mediante el uso de modelos de aprendizaje profundo entrenados con un conjunto de datos de imágenes de siete especies de cerezas cultivadas en la región de Isparta, Turquía Kayaalp1 (2024).

Conclusiones

- El estudio demostró que el método de transferencia de aprendizaje fue efectivo para clasificar especies de cereza, alcanzando una exactitud del 99.57 % con el modelo DenseNet Kayaalp1 (2024).
- Al utilizar el conjunto de datos incrementado, el modelo DenseNet alcanzó el mejor rendimiento con una exactitud del 99.57 %, mientras que el modelo InceptionV3 tuvo el peor rendimiento con un 94.81 % Kayaalp1 (2024).
- Este estudio es uno de los primeros en aplicar transferencia de aprendizaje para clasificar especies de cereza y además, ha contribuido con un nuevo conjunto de datos a la literatura existente. Kayaalp1 (2024).

Comentarios

Este trabajo de Este trabajo de investigación proporciona una base teórica esencial para el estudio en curso, destacando la relevancia de aplicar Transfer Learning. Los modelos de Transfer Learning evaluados fueron DenseNet, NASNet, VGG19, MobileNet, ResNet152 y InceptionV3 siendo DenseNet el modelo con mayor exactitud con una exactitud de 99.57 %. Los hiperparámetros utilizados fueron los siguientes: el 80 % de los datos para entrenamiento y el 20 % para prueba; una capa de entrada de tamaño (224, 224, 3); el optimizador Adam; entrenamiento durante 20 épocas; una tasa de aprendizaje de $1e-5$ y un tamaño de lote de 32.

Reconocimiento de enfermedades de la zanahoria mediante un enfoque de aprendizaje profundo para la agricultura sostenible

Objetivo

proponer un método eficiente para la identificación y clasificación de enfermedades en zanahorias utilizando un enfoque de aprendizaje profundo, específicamente redes neuronales convolucionales (CNN), con el fin de ayudar a los agricultores a detectar y monitorear enfermedades en los cultivos de zanahorias de manera precisa y efectiva Naimur Rashid Methun (2021).

Conclusiones

- El modelo Inception v3 demostró ser altamente efectivo para identificar y clasificar cinco enfermedades principales de las zanahorias, alcanzando una exactitud del 97.4 %. Naimur Rashid Methun (2021).

- La metodología propuesta ayudará a los agricultores a identificar enfermedades de las zanahorias y tomar acciones en tiempo real, contribuyendo a la reducción de pérdidas económicas y promoviendo la agricultura sostenible Naimur Rashid Methun (2021).

Comentarios

Este trabajo de Este trabajo de investigación proporciona una base teórica esencial para el estudio en curso, destacando la relevancia de aplicar Transfer Learning. Los modelos de Transfer Learning evaluados fueron InceptionV3, MobilNet, VGG19 y VGG16 siendo InceptionV3 el modelo con mayor exactitud con una exactitud de 97.4 %. Los hiperparámetros utilizados fueron los siguientes: 70 % de los datos para entrenamiento, 20 % para validación y 10 % de testeo; una capa de entrada de tamaño (300, 300, 3); optimizadores Adam; entrenamiento durante 100 épocas; una tasa de aprendizaje de 0.0001 y un tamaño de lote de 32.

Reconocimiento de enfermedades del arroz utilizando transferencia de aprendizaje

Objetivo

El objetivo general del estudio es desarrollar un sistema de reconocimiento de enfermedades del arroz utilizando la transferencia de aprendizaje con redes neuronales convolucionales (CNN), con el fin de detectar de manera temprana y precisa enfermedades en las plantas de arroz, mejorando así la productividad y reduciendo las pérdidas económicas en la producción agrícola Ahmad Rofiqul Muslikh (2023).

Conclusiones

- El modelo Xception, en el contexto de transferencia de aprendizaje, demostró ser eficaz para reconocer enfermedades en plantas de arroz basándose en imágenes de hojas, logrando una exactitud de entrenamiento de 0.93 y una exactitud de validación de 90 %. Ahmad Rofiqul Muslikh (2023).
- Utilizando un conjunto de datos relativamente pequeño, Xception alcanzó una exactitud de entrenamiento de 93 % y una exactitud de validación de 90 %, lo que muestra un rendimiento satisfactorio. Ahmad Rofiqul Muslikh (2023).
- Este estudio respalda el uso del modelo Xception para la detección eficaz y eficiente de enfermedades del arroz, con aplicaciones potenciales en la gestión agrícola y la seguridad alimentaria en Indonesia Ahmad Rofiqul Muslikh (2023).

Comentarios

Este trabajo de Este trabajo de investigación proporciona una base teórica esencial para el estudio en curso, destacando la relevancia de aplicar Transfer Learning. Los modelos de Transfer Learning evaluados fueron Xception, EfficientNetV2, MobileNetV2 y VGG16 siendo Xception el modelo con mayor exactitud con una exactitud de 90 %. Los hiperparámetros utilizados fueron los siguientes: 70 % de los datos para entrenamiento, 20 % para validación y 10 % de testeo; una capa de entrada de tamaño (224, 224, 3); optimizadores Adam; entrenamiento durante 10 épocas; una tasa de aprendizaje de 0.0001 y un tamaño de lote de 32.

1.3. Justificación y motivación

1.3.1. Conveniencia

La implementación de un sistema de visión artificial basado en Transfer Learning para el diagnóstico de defectos de calidad en alcachofas durante la etapa de inflorescencia. Se fundamenta en la necesidad de los agricultores de la comunidad campesina de Markjo de contar con una herramienta que les permita identificar correctamente los diferentes tipos de defectos de calidad, facilitando así un diagnóstico más preciso.
Actualmente, estos diagnósticos no se realizan de manera adecuada, ya que muchos agricultores carecen de los conocimientos necesarios para identificar los defectos de calidad. Como resultado, los defectos de calidad se expanden sin control en los cultivos de alcachofa. La solución propuesta consiste en implementar un sistema de visión artificial con Transfer Learning que permita identificar con exactitud los defectos de calidad en las alcachofas. Esto no solo facilitará un diagnóstico más eficiente, sino que también permitirá a los agricultores tomar acciones correctivas y/o preventivas para gestionar adecuadamente sus cultivos.

1.3.2. Relevancia

La implementación de esta solución no solo beneficiará a los agricultores de alcachofa de la comunidad de Markjo, sino que también abrirá nuevas líneas de investigación para optimizar aún más la producción de alcachofas. Además, incentivará a otras comunidades del Perú a considerar el cultivo de esta hortaliza, que posee un alto valor económico en el mercado. Actualmente, la falta de conocimiento en el cultivo y diagnóstico limita su adopción, por lo que esta iniciativa puede ser un catalizador para su expansión.

1.3.3. Implicancias prácticas

La implicación práctica de esta iniciativa radica en profundizar en el uso de Transfer Learning aplicado a la clasificación de imágenes. Esto permitirá lograr una mayor exactitud en la clasificación de las imágenes a estudiar, al tiempo que se minimizará el uso de recursos computacionales, dado que los modelos ya han sido entrenados previamente con diversos conjuntos de datos. Además, se empleará el ajuste fino (Fine Tuning) para optimizar aún más la exactitud del modelo entrenado con Transfer Learning.

1.4. Objectivo general

Implementar un sistema de visión artificial basado en Transfer Learning para el diagnóstico de los defectos de calidad en las alcachofas durante la etapa de inflorescencia.

1.5. Objetivos específicos

1. Recopilar y generar un dataset de imágenes de alta calidad que representen diversos tipos de defectos de calidad presentes en las alcachofas durante su etapa de inflorescencia, con el fin de facilitar la implementación de un sistema de diagnóstico automatizado.

2. Desarrollar modelos de inteligencia artificial (IA) basados en Transfer Learning para la identificación de defectos de calidad en las alcachofas durante su etapa de inflorescencia.

3. Implementar una herramienta de usuario final que asista a los agricultores en la identificación de defectos de calidad en las alcachofas, con el fin de facilitar un diagnóstico adecuado.

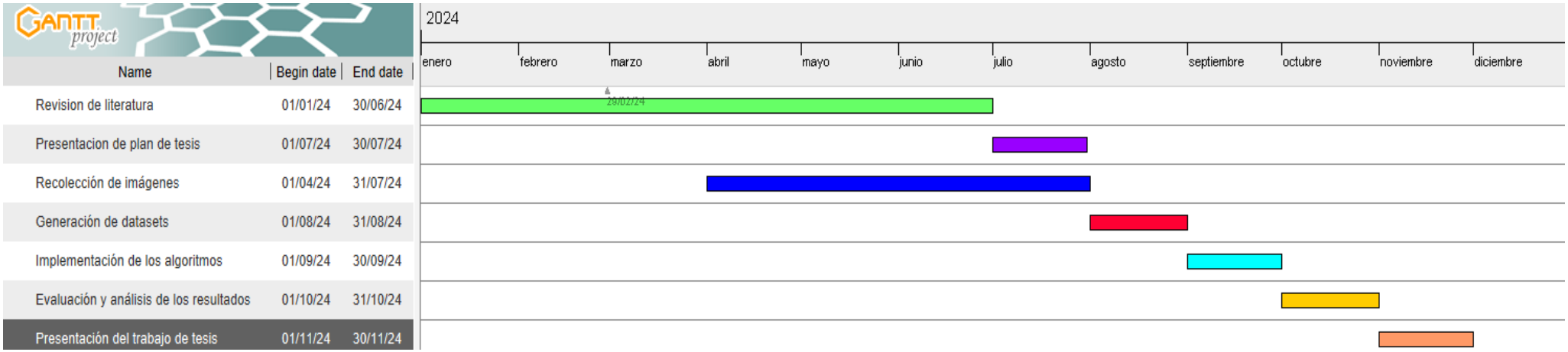
1.6. Cronograma de actividades

Tabla 1.1: Cronograma de actividades.

Actividades	Duración (Semanas)	Fecha Inicio	Fecha Fin
1. Revisión de la literatura	25	01/01/2024	30/06/2024
2. Presentación del plan de tesis	4	01/07/2024	31/07/2024
3. Recolección de imágenes	17	01/04/2024	31/07/2024
4. Generación de datasets	4	01/08/2024	31/08/2024
5. Implementación de los algoritmos	4	01/09/2024	30/09/2024
6. Evaluación y análisis de los resultados	4	01/10/2024	31/10/2024
7. Presentación del trabajo de tesis	4	01/11/2024	30/11/2024

Fuente: Elaboración propia.

Figura 1.1: Diagrama de Gantt del cronograma de actividades.



Name	Begin date	End date
Revision de literatura	01/01/24	30/06/24
Presentacion de plan de tesis	01/07/24	30/07/24
Recolección de imágenes	01/04/24	31/07/24
Generación de datasets	01/08/24	31/08/24
Implementación de los algoritmos	01/09/24	30/09/24
Evaluación y análisis de los resultados	01/10/24	31/10/24
Presentación del trabajo de tesis	01/11/24	30/11/24

Fuente: Elaboración propia.

∞

Capítulo 2

Marco teórico

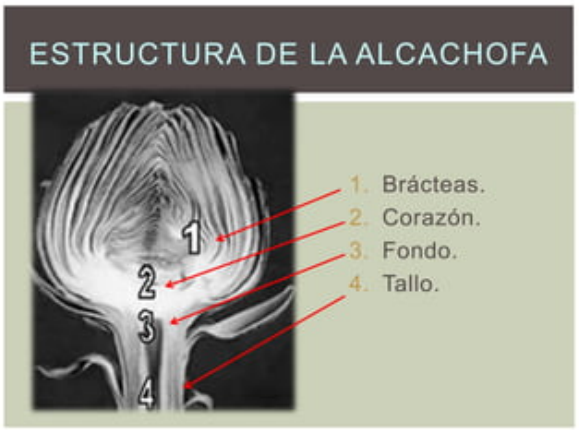
2.1. Alcachofa

Las alcachofas son inflorescencias de la planta del género *Cynara scolymus*, caracterizadas por su conformación de brácteas y un receptáculo carnoso (Moreno, 2023). Esta inflorescencia es muy beneficiosa desde el punto de vista nutricional y crece en las regiones de la sierra y la costa del Perú.
La alcachofa se compone de cuatro zonas:

1. **Brácteas**: Esta es la zona exterior de la alcachofa, con forma de roseta.
2. **Corazón**: Es la parte principal de la alcachofa, donde se concentra la parte comestible, rica en nutrientes.
3. **Fondo**: Esta zona conecta el capítulo con el tallo de la planta.
4. **Tallo**: Es la parte que sostiene la inflorescencia.

En la figura 2.1 se muestran las diferentes zonas de la alcachofa.

Figura 2.1: Estructura de la alcachofa



Fuente: (Maravi, 2014)

2.2. Defectos de calidad de la alcachofa

Los defectos de calidad en las alcachofas son malformaciones que se producen durante la etapa de inflorescencia, específicamente en el área central conocida como el corazón de alcachofa. Estos defectos pueden surgir por múltiples factores, como la falta de agua y minerales. Las alcachofas se clasifican en tres grupos según el tipo de defectos de calidad que presentan: *alcachofa de primera*, *alcachofa de segunda* y *alcachofa de descarte*. Cada uno de estos grupos está asociado con distintos tipos de defectos de calidad.

2.2.1. Alcachofa de primera

Esta variedad de alcachofa se considera de la más alta calidad debido a que su interior es compacto desde la base hasta las brácteas, las cuales son firmes y están en óptimas condiciones, sin defectos que puedan afectar al corazón de la alcachofa. En la figura 2.2 (marcada con un rectángulo rojo), se puede observar claramente la zona de brácteas compactas, libres de defectos de calidad. Para mantener esta variedad de alcachofa, es fundamental asegurar un riego adecuado, un equilibrio mineral apropiado y una fertilización constante.

Figura 2.2: Alcachofa de primera

Fuente: Elaboración propia.

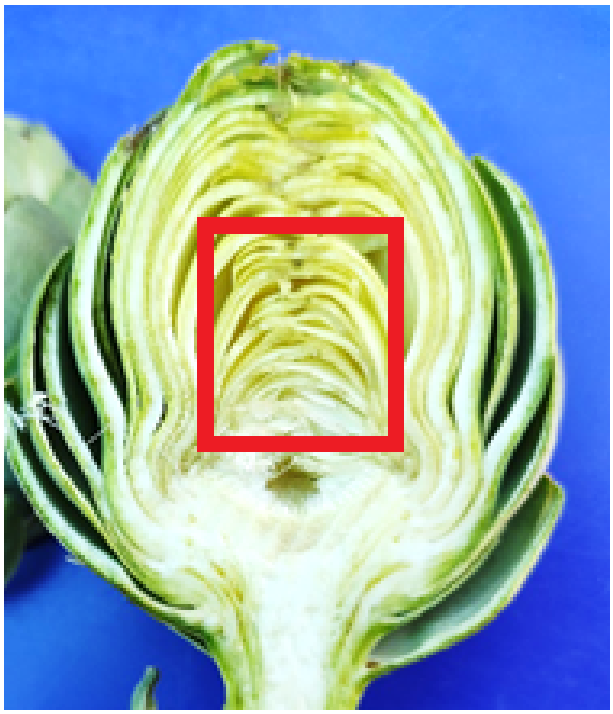
2.2.2. Alcachofa de segunda

En este grupo de alcachofa se considera de segunda porque tienen defectos calidad leves que no afectan críticamente al corazón de la alcachofa, entre los principales se tiene Fofa y cintura.

2.2.2.1. Fofa

La fofa es un defecto de calidad que ocurre cuando las brácteas interiores de la alcachofa no están adecuadamente compactas, lo que expone el corazón de la planta a factores adversos como condiciones climáticas desfavorables,plagas etc. En la figura 2.3 (marcada con un rectángulo rojo), se puede observar con mayor claridad la zona afectada por este defecto, donde la falta de compactación entre brácteas interiores es evidente. Este problema se origina principalmente por un exceso de humedad, insuficiencia de agua, deficiencia de minerales y falta de fertilización durante la etapa de inflorescencia.

Figura 2.3: Defecto de calidad Fofa

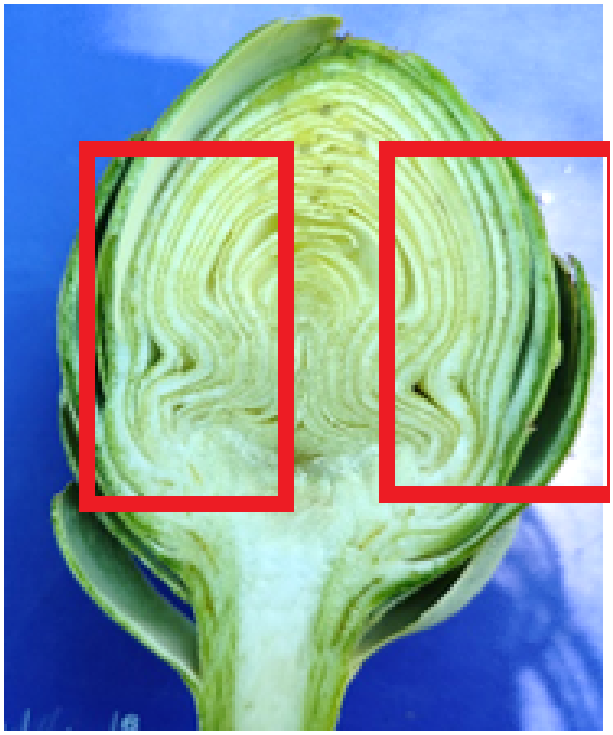


Fuente: Elaboración propia.

2.2.2.2. Cintura

La cintura es un defecto de calidad que se presenta cuando las brácteas medianas exteriores ejercen presión sobre el corazón de la alcachofa, adoptando una forma de ocho que lo afecta levemente. Esto provoca que el corazón de la alcachofa tienda a volverse fibroso, lo cual indica un envejecimiento prematuro de la planta. En la figura 2.4 (marcada con un rectángulo rojo), se puede observar claramente la zona afectada, donde las brácteas medianas forman una cintura que afecta al corazón de la alcachofa. Este defecto se desarrolla principalmente por una considerable falta de agua, deficiencia de minerales y ausencia de fertilización durante la etapa de inflorescencia.

Figura 2.4: Defecto de calidad Cintura Leve.



Fuente: Elaboración propia.

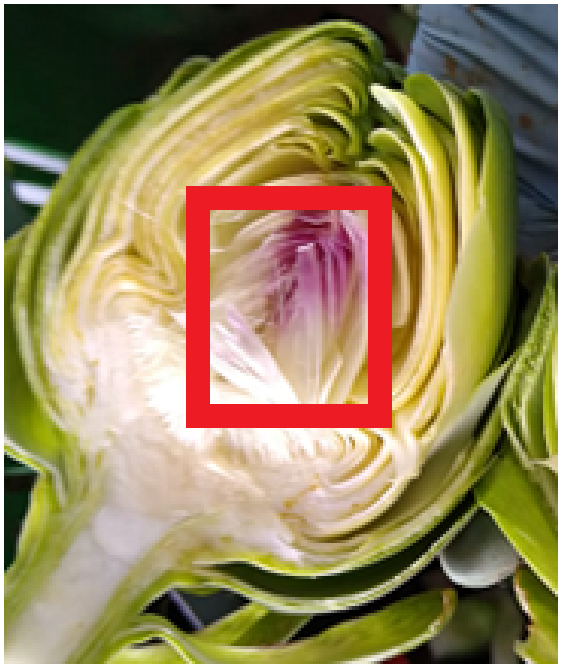
2.2.3. Alcachofa de descarte

Este grupo de alcachofas se clasifica como de descarte debido a un daño crítico en el corazón de la planta. Estas alcachofas están destinadas exclusivamente al consumo animal o al desecho. La razón principal para esta decisión es el alto costo que conlleva su procesamiento para la exportación, siendo el defecto de calidad más importante la violácea.

2.2.3.1. Violácea

La violácea es un defecto de calidad que se presenta cuando el corazón de la alcachofa inicia su etapa de florecimiento, lo que indica un envejecimiento crítico y prematuro de la planta. En esta fase, el corazón tiende a volverse duro, adquiriendo un color lila y una vellosidad abundante. En la figura 2.5 (marcada con un rectángulo rojo), se puede observar claramente la zona afectada, donde el corazón de la alcachofa presenta un color lila y una textura fibrosa. Este defecto se desarrolla principalmente por una falta crítica de agua, deficiencia de minerales y ausencia de fertilización durante la etapa de inflorescencia.

Figura 2.5: Defecto de calidad Violácea.



Fuente: Elaboración propia.

2.3. Agroquímicos

Los agroquímicos son fundamentales para el cultivo de alcachofas, ya que ayudan a corregir los defectos de calidad que surgen durante la etapa de inflorescencia. A continuación, se detallan los agroquímicos utilizados:

2.3.1. Actifer potasio

Actifer Potasio es un producto que se utiliza en la agricultura para mejorar la salud y el desarrollo de las plantas. Este fertilizante potásico es conocido por sus beneficios en la regulación de funciones metabólicas esenciales en las plantas, como la fotosíntesis y la formación de azúcares, almidones y proteínas.

Beneficios de actifer Potasio:

1. **Estimulación del crecimiento:** Favorece el desarrollo de raíces y la formación de brotes, lo que se traduce en un crecimiento más vigoroso.
2. **Mejora en la calidad de los frutos:** Contribuye a un mejor color, tamaño y sabor, además de aumentar la resistencia a enfermedades.
3. **Regulación de la transpiración:** Ayuda a las plantas a manejar mejor el agua, aumentando su resistencia al estrés hídrico.
4. **Balance metabólico:** Facilita la relación entre la respiración y la fotosíntesis, optimizando el uso de nutrientes.
5. **Aumento de la producción:** En general, se traduce en un incremento en el rendimiento de los cultivos.

2.3.2. Actifer fósforo

Actifer Fósforo es un fertilizante diseñado para aportar fósforo a las plantas, esencial para su crecimiento y desarrollo. Aquí te dejo un resumen de sus principales beneficios y funciones:

Beneficios de actifer fósforo:

1. **Desarrollo de raíces:** Promueve un sistema radicular más fuerte y profundo, lo que mejora la absorción de agua y nutrientes.
2. **Florecimiento y fructificación:** El fósforo es crucial para la formación de flores y frutos, ayudando a mejorar la calidad y cantidad de la producción.
3. **Fotosíntesis:** Participa en la síntesis de compuestos energéticos (como el ATP), facilitando la fotosíntesis y el crecimiento general de la planta.
4. **Resistencia a enfermedades:** Mejora la salud general de las plantas, lo que las hace más resistentes a enfermedades y condiciones adversas.

5. **Eficiencia en el uso de nutrientes:** Aumenta la eficiencia en la absorción de otros nutrientes, como el nitrógeno y el potasio.

2.3.3. Nitrato de calcio

El nitrato de calcio es un fertilizante granular que se disuelve por completo en agua, proporcionando una fuente eficaz de calcio y nitrógeno para las plantas. El calcio es un nutriente que mejora la calidad de los frutos y extiende la vida útil de los productos. Además, el nitrógeno (NO3) presente en el nitrato de calcio se absorbe fácilmente por las plantas, lo que a su vez mejora la eficacia en la absorción del calcio (Plaza, 2024).

Beneficios del nitrato de calcio:

1. **Estimulación del crecimiento:** El nitrógeno favorece el crecimiento vegetativo, mientras que el calcio contribuye a la formación de paredes celulares y tejidos (Plaza, 2024).

2. **Prevención de deficiencias:** Ayuda a prevenir problemas como la pudrición apical en tomates y pimientos, que está relacionada con la falta de calcio (Plaza, 2024).

3. **Mejora la calidad del fruto:** El calcio mejora la textura y la calidad de los frutos, aumentando su vida útil (Plaza, 2024).

4. **Regulación del pH del suelo:** Al ser un fertilizante neutro, ayuda a mantener un equilibrio en el pH del suelo (Plaza, 2024).

5. **Efecto positivo en la salud de las plantas:** Fortalece las células, haciendo que las plantas sean más resistentes a enfermedades y condiciones ambientales adversas (Plaza, 2024).

2.3.4. Alga sealand

Alga sealand es un bioestimulante orgánico elaborado a partir de algas marinas y extractos vegetales. Su formulación incluye ácidos orgánicos, carbohidratos, proteínas hidrolizadas y aminoácidos, que proporcionan a las plantas las condiciones óptimas para lograr su máximo rendimiento (Supplier, 2024).

Beneficios de alga sealand:

1. **Estimulación del crecimiento:** Fomenta un desarrollo más vigoroso de las plantas, favoreciendo la formación de raíces y brotes (Supplier, 2024).

2. **Mejora de la resistencia:** Ayuda a las plantas a soportar el estrés ambiental, como sequías y cambios de temperatura (Supplier, 2024).

3. **Aumento de la calidad de los cultivos:** Mejora la calidad y el tamaño de los frutos, además de incrementar su contenido nutricional (Supplier, 2024).

4. **Estimulación de microorganismos:** Favorece la actividad microbiana en el suelo, mejorando la salud del suelo y la absorción de nutrientes (Supplier, 2024).

5. **Efecto positivo en la fotosíntesis:** Puede aumentar la eficacia de la fotosíntesis, contribuyendo a un crecimiento más saludable (Supplier, 2024).

2.3.5. Boro/calcio sealand

Se utiliza comúnmente para mejorar la nutrición de las plantas y puede tener varios beneficios.

Beneficios del boro/calcio sealand:

1. **Mejora de la absorción de nutrientes:** El boro es esencial para la formación de polen y la fertilidad de las flores, mientras que el calcio es crucial para el desarrollo celular.

2. **Aumento de la resistencia:** El calcio fortalece las paredes celulares de las plantas, lo que puede aumentar su resistencia a enfermedades y estrés ambiental.

3. **Prevención de deficiencias:** La aplicación de boro y calcio puede ayudar a prevenir deficiencias en los cultivos, que pueden llevar a problemas como la pudrición apical en tomates y pimientos.

4. **Mejora de la calidad del fruto:** Estos nutrientes pueden contribuir a mejorar la calidad de los frutos, tanto en tamaño como en sabor y textura.

5. **Estimulación del crecimiento:** Ambos nutrientes promueven un crecimiento más equilibrado y saludable de las plantas.

2.3.6. Cobre sealand

El sealand Cobre es un agroquímico que contiene cobre como micronutriente. Este tipo de producto se utiliza en la agricultura para mejorar la salud de las plantas y la calidad del suelo. Algunos de sus beneficios incluyen:

Beneficios del Cobre sealand:

1. **Prevención de deficiencias:** El cobre es esencial para varias funciones biológicas en las plantas, como la fotosíntesis y la respiración celular. Su aplicación ayuda a prevenir deficiencias que pueden afectar el crecimiento.

2. **Mejora de la resistencia a enfermedades:** El cobre tiene propiedades antifúngicas y antibacterianas, lo que puede ayudar a proteger a las plantas de ciertos patógenos.

3. **Desarrollo de raíces:** Contribuye al desarrollo adecuado del sistema radicular, lo que mejora la absorción de nutrientes y agua.

4. **Aumento de la calidad de los cultivos:** Puede influir positivamente en el sabor, la textura y la apariencia de los frutos.

5. **Interacción con otros nutrientes:** Ayuda en la absorción y utilización de otros micronutrientes, mejorando así la salud general de las plantas.

2.3.7. Sulfa 87 LS

El sulfa 87 LS es un producto agroquímico que generalmente se utiliza como fungicida y bactericida en la agricultura. Aquí tienes algunas características y beneficios típicos de este tipo de producto:

Beneficios del Sulfa 87 LS:

- **Composición:** Suele contener ingredientes activos que ayudan a controlar enfermedades fúngicas y bacterianas en cultivos.
- **Eficacia:** Puede ser efectivo contra una variedad de patógenos, ayudando a prevenir y tratar enfermedades comunes en plantas.
- **Forma de aplicación:** Generalmente se aplica en forma de spray, lo que permite una distribución uniforme sobre las plantas afectadas.
- **Compatibilidad:** A menudo puede ser mezclado con otros agroquímicos, aunque siempre es recomendable realizar pruebas de compatibilidad.
- **Manejo de resistencia:** El uso de fungicidas como Sulfa 87 LS en un programa de manejo integrado de plagas puede ayudar a prevenir la resistencia de los patógenos.

2.3.8. Rooty sealand

El rooty sealand es un agroquímico que se utiliza para mejorar el desarrollo de las raíces en las plantas. Generalmente, estos productos están diseñados para aumentar la salud y el crecimiento del sistema radicular, lo que puede tener varios beneficios:

Beneficios del rooty sealand:

1. **Mejora de la absorción de nutrientes:** Un sistema radicular fuerte permite a las plantas absorber más eficientemente agua y nutrientes del suelo.
2. **Estimulación del crecimiento:** Promueve un crecimiento más rápido y vigoroso de las plantas, lo que puede resultar en una mejor producción.
3. **Resistencia al estrés:** Las raíces sanas ayudan a las plantas a resistir mejor condiciones adversas, como sequía o suelos pobres.
4. **Aumento de la calidad de los cultivos:** Un desarrollo radicular adecuado puede mejorar la calidad y el tamaño de los frutos.
5. **Compatibilidad:** Puede ser utilizado en combinación con otros fertilizantes y agroquímicos para maximizar los beneficios.

2.3.9. Benfalar

El Benfalar es un agroquímico que se utiliza como fungicida y bactericida en la agricultura. Aquí te detallo algunos de sus aspectos clave:

Beneficios del benfalar:

1. **Composición:** Generalmente contiene ingredientes activos que ayudan a combatir enfermedades fúngicas y bacterianas en diversos cultivos.
2. **Eficacia:** Es efectivo contra una variedad de patógenos, lo que ayuda a proteger los cultivos y mejorar su salud.
3. **Modo de acción:** Actúa interfiriendo con el crecimiento y la reproducción de hongos y bacterias, previniendo así la propagación de enfermedades.
4. **Aplicación:** Se suele aplicar mediante pulverización foliar, y es importante seguir las recomendaciones de dosis y frecuencia.
5. **Manejo de resistencia:** Puede ser parte de un programa de manejo integrado de plagas, ayudando a prevenir la resistencia de los patógenos.

2.3.10. Dogma

El dogma es un agroquímico que se utiliza como herbicida. Aquí tienes algunos puntos clave sobre este producto:

Beneficios del dogma:

1. **Modo de acción:** Actúa inhibiendo el crecimiento de malezas al interferir con procesos específicos en las plantas, lo que ayuda a controlar su proliferación en cultivos.
2. **Espectro de control:** Es efectivo contra diversas especies de malezas, lo que lo hace útil en una variedad de cultivos.
3. **Aplicación:** Generalmente se aplica a través de pulverización, y es importante seguir las recomendaciones de dosis y momento de aplicación para obtener los mejores resultados.
4. **Estrategia de manejo:** Puede ser parte de un programa de manejo integrado de malezas, ayudando a reducir la presión de malezas en los cultivos.
5. **Compatibilidad:** A menudo puede ser mezclado con otros herbicidas o agroquímicos, pero siempre se recomienda hacer pruebas de compatibilidad.

2.3.11. Coragen

El coragen es un insecticida utilizado en la agricultura para el control de plagas en diversos cultivos. Aquí te dejo algunos puntos clave sobre este producto:

Beneficios del coragen:

1. **Composición:** Contiene el ingrediente activo chlorantraniliprole, que actúa sobre el sistema nervioso de los insectos.
2. **Modo de acción:** Es un insecticida de acción sistémica y de contacto, que interfiere en el proceso de contracción muscular de los insectos, lo que resulta en su muerte.
3. **Espectro de control:** Es eficaz contra una amplia gama de plagas, incluyendo orugas, escarabajos y otros insectos chupadores, como los pulgones.
4. **Aplicación:** Se aplica generalmente mediante pulverización foliar y se puede usar en varios cultivos, incluyendo hortalizas, frutas y cultivos de granos.
5. **Ventajas:** Tiene un perfil de seguridad relativamente alto para los polinizadores y otros organismos benéficos cuando se usa correctamente, lo que lo hace una opción popular en la agricultura sostenible.

2.4. Diagnóstico de defectos de calidad en las alcachofas

El diagnóstico de defectos de calidad en las alcachofas durante la etapa de inflorescencia es un procedimiento que consiste en tomar muestras de un cultivo, generalmente de una hectárea. Para este diagnóstico, se recolectan aproximadamente 75 alcachofas, lo que equivale a una media jaba. Este procedimiento debe llevarse a cabo cada cinco días una vez que inicia la etapa de inflorescencia, con el objetivo de detectar los defectos de calidad y mantener un mejor control sobre ellos. En cada diagnóstico, se clasifican los defectos observados y, según los hallazgos, se procede a suministrar agroquímicos o a hacer recomendaciones para mejorar la calidad de la alcachofa. En la siguiente figura 2.6 se ilustra cómo realiza el diagnóstico el agricultor.

Figura 2.6: Diagnóstico de defectos de calidad.

Fuente: Elaboración propia.

2.5. Visión artificial

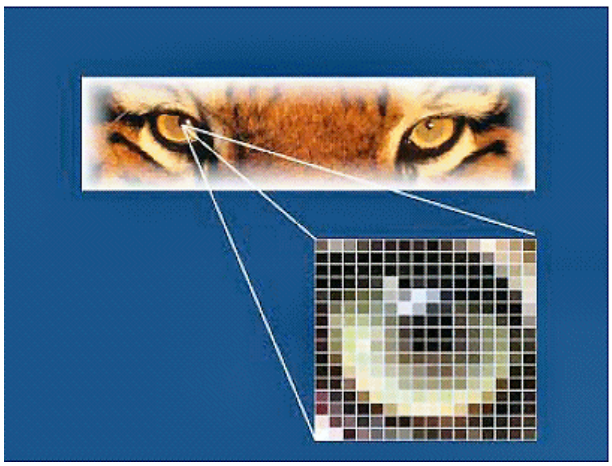
La visión artificial es una disciplina científica que abarca métodos para capturar, procesar y analizar imágenes del mundo real con el propósito de generar información que pueda ser interpretada por una máquina. Se basa en la teoría clásica de matrices aplicada a dimensiones mucho mayores. Su principal objetivo es deducir de manera automática la estructura y las propiedades de un entorno tridimensional a partir de una o varias imágenes bidimensionales (Beatriz Borrella Petisco, 2022).

Hoy en día, la visión artificial abarca no solo la obtención de imágenes, sino también la caracterización e interpretación de los objetos que contienen. Es uno de los elementos sensoriales más importantes para aumentar la autonomía en robótica. Además, los métodos de visión artificial son relativamente económicos en comparación con otras técnicas que ofrecen resultados similares (Beatriz Borrella Petisco, 2022).

2.5.1. Imágen digital

Hoy en día, la imagen digital juega un papel central en un mundo donde cualquier expresión del conocimiento estaría incompleta sin una representación visual que aclare o complemente las ideas presentadas. Las imágenes digitales se obtienen a partir de fotografías electrónicas de escenas o se escanean desde documentos, fotos, manuscritos, textos impresos e ilustraciones. Para crear una imagen digital, se realiza una muestra y se convierte en un mapa de cuadrícula compuesto por puntos o elementos (píxeles). En la imagen proporcionada 2.7, cada píxel recibe un valor tonal; en este caso, 0 representa el negro y 1 el blanco. Al acercarse a una sección de la imagen en la pantalla de un monitor, se pueden observar estos puntos individuales, llamados píxeles. Cuantos más píxeles tenga una imagen, mayor será su resolución. La siguiente figura 2.7 ilustra cómo se representa una imagen digital en un mapa de píxeles (Miguel, 2008).

Figura 2.7: Imágen digital en un mapa de pixeles.



Fuente: (Miguel, 2008).

1. **Tamaño de la imágen digital**

   El tamaño de una imagen se refiere a sus dimensiones en términos de anchura y altura en píxeles. El tamaño del archivo, por otro lado, indica la cantidad de memoria necesaria para almacenar la información de la imagen digitalizada en un medio de almacenamiento, como un CD, DVD, memoria flash, pendrive, entre otros. Cuando se menciona la resolución de una imagen, se puede decir que tiene, por ejemplo, 1,3 megapíxeles o 5 megapíxeles. Si se quiere especificar el tamaño del archivo en términos de espacio en disco, se indica, por ejemplo, que la imagen ocupa 256 kilobytes, 1 megabyte, 2 megabytes, etc. Toda la información almacenada en una computadora o en otros dispositivos similares se guarda en unidades básicas conocidas como "bytes". Este término es universal en computación y representa un grupo de 8 bits, que son valores binarios de ceros (0) y unos (1) (Miguel, 2008).

2. **Bits por pixel y profundidad de color de una imágen digital** Dependiendo de la cantidad de niveles de color que se deseen mostrar en una imagen, cada píxel ocupará una cantidad específica de bytes o bits. Por ejemplo, en una imagen en blanco y negro, como se muestra en la Figura 2.8, solo se necesitan dos valores de color, por lo que cada píxel ocupa 1 bit. En la mayoría de las cámaras y otros dispositivos de captura de imágenes, cada píxel utiliza 24 bits, o 3 bytes, lo que permite representar hasta 16 millones de combinaciones de colores. En la siguiente figura 2.8 se presentan los bits por píxel y la profundidad de color de una imagen digital (Miguel, 2008).

Figura 2.8: Bits por pixel y profundidad de color de una imágen digital.

Bits por píxel	Número máximo de colores
1	2
4	16
8	256
16	32,768 o 65,536 (depende del formato)
24	16,777,216
32	16,777,216

Fuente: (Miguel, 2008).

2.5.1.1. Tipos de formatos de imagen digital

Las imágenes digitales han sido una parte integral de nuestro mundo durante décadas, apareciendo en dispositivos como teléfonos móviles, ordenadores, tabletas e incluso en las antiguas pantallas de televisión. Al adentrarse en el campo de la imagen, ya sea en diseño gráfico, fotografía o edición de video, se empieza a trabajar con diversos formatos y tipos de imágenes digitales. A menudo, estos formatos pueden parecer más nombres de archivos encriptados que simples códigos para identificar diferentes tipos de imágene (Perdomo, 2023).

1. **JPEG/ JPG (Joint Photographic Experts Group)** Es un formato ampliamente conocido y utilizado por la mayoría de los usuarios. Se emplea para almacenar fotografías y otras imágenes con tonos continuos. Gracias a su sistema de compresión eficiente, reduce el tamaño de los archivos. A diferencia del formato GIF, JPEG puede almacenar una gran cantidad de colores (en el espacio de color RGB) sin generar archivos de gran tamaño. Además, los navegadores modernos son capaces de reconocer y mostrar este formato con total fidelidad (Perdomo, 2023).

2. **GIF (Graphics Interchange Format)** El formato GIF es ideal para imágenes con tonos no continuos o áreas amplias de un mismo color. Es uno de los pocos formatos que permite la creación de animaciones, ya que puede ejecutar diferentes fotogramas de manera secuencial. Además, su diseño de compresión está optimizado para reducir el tiempo de transferencia de datos a través de líneas telefónicas. Por estas razones, el formato GIF sigue siendo uno de los más efectivos para ciertos tipos de imágenes digitales en la actualidad (Perdomo, 2023).

3. **PNG (Portable Network Graphics)** Este formato tiene ventajas respecto a los otros formatos más comunes como el JPEG y el GIF. Tiene gran parte de las ventajas de un GIF y de un JPEG. Por ejemplo, permite altos niveles de compresión, además de utilizar la técnica de la indexación para crear colores transparentes, semitransparencias o transparencias degradadas. Su única limitación es que no podemos crear ficheros animados (Perdomo, 2023).

4. **WBMP (Wireless Application Protocol Bitmap Format)** Es un formato monocromo diseñado para la web, aunque es uno de los menos utilizados. Su utilidad radica en que convierte una imagen en una trama en blanco y negro, con opciones para seleccionar entre tres tipos diferentes de tramas. Es adecuado para iconos muy pequeños en un solo tono, aunque los formatos estándar para estos propósitos suelen ser GIF o PNG (Perdomo, 2023).

5. **RAW (crudo)** Las imágenes RAW se capturan con cámaras profesionales y se utilizan principalmente para tomas controladas, como entrevistas, moda, publicidad y paisajismo. La ventaja de las imágenes RAW es que conservan una amplia gama de tonos intermedios, lo que permite un ajuste preciso de los tonos, la iluminación, el balance de blancos y otros aspectos en programas como Adobe Lightroom, Photoshop con Camera Raw, entre otros. Esto facilita alcanzar un acabado preciso y profesional en la edición de imágene (Perdomo, 2023).

6. **PSD (Power Spectral Density)** Este es el formato nativo de Adobe Photoshop, diseñado principalmente para la manipulación de imágenes en lugar de para su uso en publicaciones digitales. Ofrece importantes ventajas para la edición, ya que permite conservar las capas y otros elementos de la manipulación de la imagen. Esto facilita la edición y el ajuste continuo del archivo sin perder detalles o alteraciones previas (Perdomo, 2023).

7. **TIFF (Tagged Image File Format)** Este formato es ampliamente utilizado en la industria gráfica debido a su excelente calidad de imagen e impresión. Anteriormente, era muy popular en el ámbito editorial por su capacidad para conservar los colores originales con gran precisión. Aunque puede mantener las capas si se activa la opción correspondiente al guardar el archivo, su uso está disminuyendo gradualmente en favor del formato JPG, que es preferido por su rapidez en el proceso de edición (Perdomo, 2023).

8. **PDF (Portable Document Format)** Es el formato de documento más estandarizado a nivel mundial, ya que es compatible con todos los dispositivos. En un archivo PDF convergen imágenes (que a menudo se comprimen en JPG), texto que puede ser leído y seleccionado, y formas vectoriales. Actualmente, el PDF es considerado el estándar de excelencia para documentos, utilizado tanto en sitios web y archivos descargables como en la preparación de archivos para impresión (Perdomo, 2023).

9. **EPS (Encapsulated PostScript)** Es un formato creado antes que el PDF, con el mismo propósito de conservar imágenes, vectores y texto originales en un archivo. Sin embargo, su peso es generalmente mayor que el del PDF y requiere las tipografías originales para una visualización adecuada. Debido a estos requisitos, se utiliza principalmente para la impresión en imprenta (Perdomo, 2023).

2.5.2. Procesamiento de imágenes digitales

2.5.2.1. Etapas del procesamiento de imágenes digitales

Las etapas de procesamiento comienzan con la captura de la imagen y su preprocesamiento. Cuando el análisis se centra en el procesamiento de secuencias de imágenes continuas, como en el caso de un video, las técnicas de enfoque y nitidez en la captura adquieren una mayor importancia para los métodos de aprendizaje en inteligencia artificial. A continuación, se presentan las etapas de procesamiento de imágenes (Gutiérrez, 2017):

1. **Binarización**: Antes de procesar una imagen, generalmente se identifican regiones que puedan ser consideradas como picos o patrones en la secuencia de píxeles, para clasificarlas como parte de un mismo patrón o región. Una de las técnicas de preparación utilizadas es la binarización, que implica convertir la imagen a una escala de grises, donde cada píxel recibe un valor entre 0 y 255. En la siguiente figura 2.9 se presenta la binarización en escala de grises (Gutiérrez, 2017).

Figura 2.9: Binarización en escala de grises.

Fuente: (Gutiérrez, 2017).

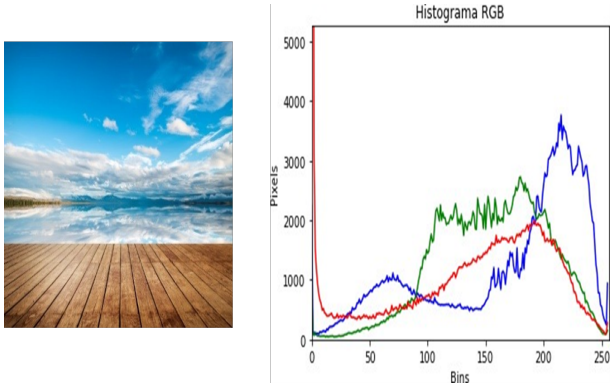
2. **Difuminado**: Una de las técnicas de procesamiento es el blurring o difuminado, que consiste en promediar el valor de un píxel con los valores de intensidad de los píxeles vecinos. Esta combinación de intensidades produce una imagen borrosa. Aunque en las fotografías se busca la mayor nitidez posible, el difuminado resulta útil en el procesamiento de imágenes para tareas como la delimitación de regiones y la detección de bordes. Entre los distintos métodos de blurring se encuentra el método gaussiano, que calcula la media ponderada de los píxeles cercanos a un píxel central, otorgando mayor peso a los píxeles más próximos al centro. En la siguiente figura 2.10 se presenta el difuminando de imágenes (Gutiérrez, 2017).

Figura 2.10: Difuminación.

Fuente: (Gutiérrez, 2017).

3. **Histogramas de gradientes**: Un histograma ilustra la distribución de las intensidades de los píxeles (ya sea en color o en escala de grises) en una imagen. Este gráfico muestra el nivel de intensidad (valor de píxel) del color. En un espacio de color RGB, los valores de los píxeles varían entre 0 y 255. Al trazar un histograma con el eje X dividido en 256 bins, se cuenta el número de veces que cada valor de píxel ocurre. Por otro lado, si se utilizan solo dos segmentos (divididos equitativamente), se contaría cuántos píxeles tienen valores en el rango [0, 128] o en el rango [128, 255]. Analizar el histograma de una imagen proporciona una visión general sobre el contraste, el brillo y la distribución de intensidades. En la siguiente figura se presenta el histograma de gradientes (Gutiérrez, 2017).

Figura 2.11: Histogramas de gradientes.



Fuente: (Gutiérrez, 2017).

2.6. Aprendizaje de máquina (machine learning)

El aprendizaje de máquina (AM) es una rama de la ciencia de la computación que se enfoca en el desarrollo y la aplicación de técnicas y algoritmos capaces de .ªprenderz mejorar su rendimiento a partir de la experiencia y la adaptación a condiciones cambiantes con el tiempo. Esto permite que las máquinas aumenten su capacidad para inferir y extraer nuevos conocimientos de grandes volúmenes de datos (Edian F. Franco, 2023).

Uno de los principales objetivos del aprendizaje de máquina es permitir que los computadoras aprendan y mejoren continuamente. Para lograr esto, se emplean diversas técnicas y metodologías estadísticas, matemáticas y lógicas que facilitan el manejo e interpretación de grandes cantidades de datos. El aprendizaje de máquina ha ganado relevancia en el campo de la biotecnología debido a su capacidad para adaptarse a diferentes tipos de análisis y datos, así como a la amplia variedad de experimentos que se pueden realizar utilizando estos algoritmos (Edian F. Franco, 2023).

Estas técnicas se pueden clasificar en tres tipos: su- pervisados, no supervisados y semi-supervisados, siendo que cada uno tiene una función diferente según el tipo de datos con el que se trabaja y el objetivo de la investigación o análisis que se está realizando (Edian F. Franco, 2023).

2.6.1. Aprendizaje de máquinas supervisadas

Las técnicas de aprendizaje supervisado, también conocidas como técnicas predictivas, se basan en algoritmos que requieren conocimientos previos, es decir, datos etiquetados, para ser entrenados usando un conjunto de entrenamiento. Una vez entrenados, estos algoritmos pueden predecir la etiqueta de nuevos datos de entrada que no tienen etiqueta, utilizando un conjunto de prueba. Un ejemplo de este tipo de clasificación es la predicción de biomarcadores específicos de tejidos a partir de datos de expresión génica (Edian F. Franco, 2023).

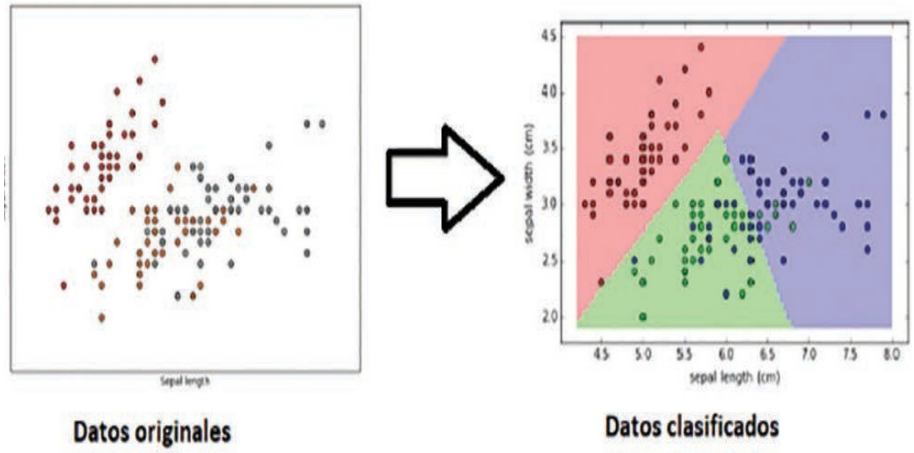
Dentro de estas técnicas se encuentran varios tipos de algoritmos, entre ellos (Edian F. Franco, 2023):

1. **Máquinas de Vectores de Soporte (SVM)**: Construyen líneas de separación en un espacio multidimensional para distinguir entre dos clases de objetos (Edian F. Franco, 2023).
2. **Árboles de Decisión**: Generan reglas de división en los conjuntos de datos, organizándolos en forma de árbol para predecir el valor de una variable basada en otras variables de entrada (Edian F. Franco, 2023).
3. **Algoritmos Basados en Instancias**: Clasifican nuevos valores de entrada basándose en la proximidad de estos valores con los datos utilizados para entrenar el modelo (Edian F. Franco, 2023).

Dentro de estas técnicas se encuentran diferentes tipos de algoritmos como son: las máquinas de vectores de soporte (SVM), las cuales construyen líneas de separaciones para distinguir entre dos objetos diferentes dentro de un espacio multidimensional; los árboles de decisiones, que crea reglas de división dentro de los conjuntos de datos y las organiza en forma de árbol, para de este modo poder predecir el valor de una variable partiendo de otras variables de entradas; otro tipo de algoritmos son los basados en instancias o de aprendizaje vago, que clasifican un nuevo valor de entrada con base en la proximidad de este valor con los valores que fueron utilizados para el entrenamiento de los modelos (Edian F. Franco, 2023).

Otra técnica de aprendizaje automático, clasificada como supervisada, es el algoritmo de regresión. A diferencia de los algoritmos de clasificación, que predicen categorías discretas, los algoritmos de regresión utilizan datos etiquetados para predecir valores continuos. Dentro de los métodos de regresión, se encuentran los algoritmos de regresión lineal, los cuales permiten identificar y modelar la relación de dependencia entre una variable dependiente y una o más variables independientes. En la siguiente figura 2.12 se presenta el modelo de clasificación aplicado a un conjunto de datos (Edian F. Franco, 2023).

Figura 2.12: Modelo de clasificación aplicado a un conjunto de datos.



Fuente: (Edian F. Franco, 2023).

2.6.2. Aprendizaje de máquinas no supervisadas

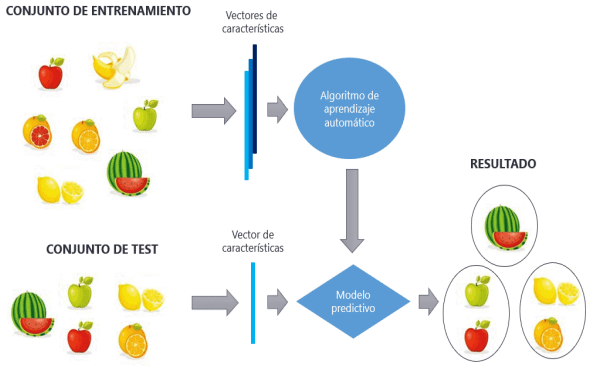
Este tipo de aprendizaje se basa en algoritmos que no requieren conocimiento previo de los datos ni datos etiquetados. También conocido como aprendizaje descriptivo, permite descubrir patrones y conocimientos a partir de las características intrínsecas de los datos. Estos algoritmos son útiles para identificar grupos de genes con patrones de expresión similares en datos de expresión génica bajo diferentes condiciones de estrés ambiental. Dentro de este enfoque, se incluyen diversas metodologías como las técnicas de agrupamiento (clustering), el análisis de componentes principales, la selección de variables y las reglas de asociación (Edian F. Franco, 2023).

Las técnicas de agrupamiento se utilizan para descubrir la estructura de los datos mediante el análisis de la información inherente, enfocándose en maximizar la similitud entre los miembros de un mismo grupo (intra-clusters) y la diferencia entre miembros de grupos distintos (ver Figura 3). Estas técnicas permiten identificar grupos con funciones genéticas similares y detectar agrupaciones de genes o proteínas con homología, lo cual es crucial para el diseño de vacunas y el desarrollo de fármacos. Además, estos algoritmos han contribuido a la mejora de cultivos y prácticas agrícolas (Edian F. Franco, 2023).

El análisis de componentes principales (PCA) es una técnica empleada para describir y reducir la dimensionalidad de los conjuntos de datos, conservando la mayor parte de la variabilidad presente en los datos. PCA facilita la identificación de tendencias, patrones, similitudes y agrupamientos en los datos, permitiendo su evaluación de manera visual. Esta técnica se utiliza principalmente para reducir la cantidad de datos necesarios para el análisis, al tiempo que mantiene la representatividad del conjunto, lo que mejora el rendimiento computacional (Edian F. Franco, 2023).

La selección de variables es el proceso de elegir los atributos o variables más relevantes para la construcción de modelos y análisis. Esta técnica ayuda a reducir la cantidad de datos en un conjunto, lo que simplifica la interpretación de los modelos, mejora el rendimiento de los algoritmos y acorta el tiempo de análisis. Es especialmente útil en conjuntos de datos con un gran número de variables, como en los metagenomas, donde las dimensiones de los datos son considerablemente grandes. En la siguiente figura 2.13 se presenta el modelo de aprendizaje no supervisado (Edian F. Franco, 2023).

Figura 2.13: Modelo de aprendizaje no supervisado.

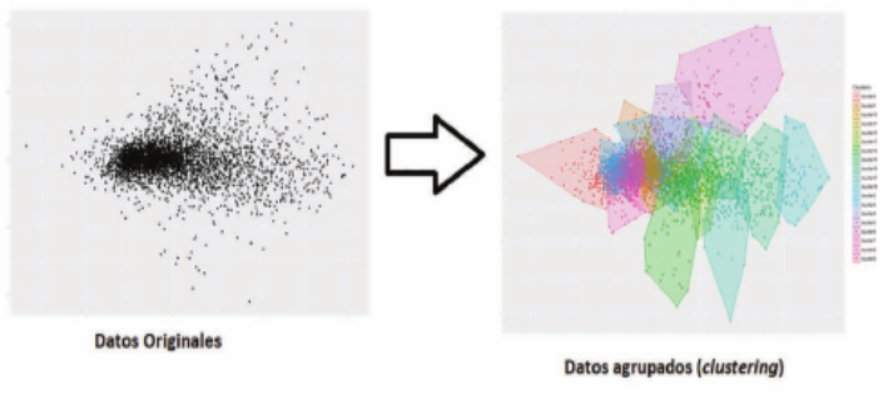


Fuente: (Edian F. Franco, 2023).

2.6.3. Aprendizaje de máquina semi-supervisado y ensamblado

Las técnicas de este tipo de aprendizaje de máquina combinan elementos de los métodos supervisados y no supervisados. En estos algoritmos, se utiliza un conjunto de datos que incluye tanto datos etiquetados como no etiquetados para construir modelos que permiten la descripción y predicción de los datos. Estas metodologías se aplican, por ejemplo, en el entrenamiento de sistemas para la predicción de genes y en la mejora de los procesos de anotación automática de nuevos genomas. En la siguiente figura 2.14 se presenta el modelo de semi-supervisado y ensamblado (Edian F. Franco, 2023).

Figura 2.14: Ejemplo de un algoritmo de agrupamiento (clustesring) aplicados a datos de expresión de RNA-seq.



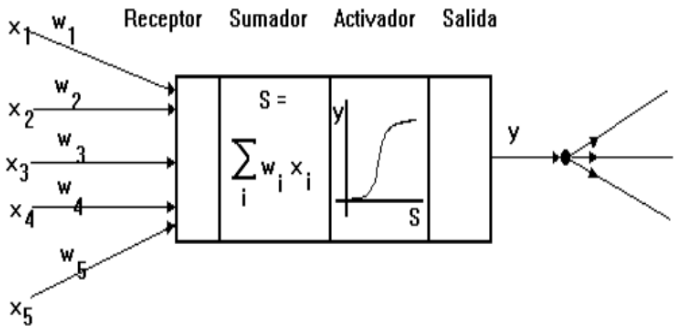
Fuente: (Edian F. Franco, 2023).

Las técnicas de aprendizaje ensamblado combinan diversos algoritmos y métodos independientes en un único modelo con el objetivo de mejorar la precisión de las predicciones. Estas metodologías se basan en la idea de que cada técnica puede tener debilidades o sesgos específicos para identificar patrones o manejar ciertos tipos de datos. Al integrar diferentes algoritmos, se pueden crear modelos predictivos más robustos y precisos en comparación con el uso de un solo algoritmo (Edian F. Franco, 2023).

2.6.4. Redes neuronales artificiales

Una Red Neuronal Artificial (RNA) es un modelo matemático basado en el funcionamiento biológico de las neuronas y en la estructura del cerebro. Estas redes pueden considerarse sistemas inteligentes que realizan tareas de manera diferente a las computadoras tradicionales. Aunque las computadoras modernas son muy rápidas en procesar información, enfrentan desafíos significativos en tareas complejas como el reconocimiento y clasificación de patrones, que el cerebro humano maneja con aparente facilidad (por ejemplo, identificar un rostro familiar en una multitud). En la siguiente figura 2.16 se presenta el esquema de una neuronal artificial (Torres, 2021).

Figura 2.15: Esquema de una neuronal artificial.



Fuente: (Rosano, 2021).

1. **El elemento receptor**.- Es el elemento que recibe una o varias señales de entrada W_i que generalmente provienen de otras neuronas y son atenuadas o amplificadas según un factor de peso W_i, es el conector o sinapsis. Estos pesos W_i determinan la conectividad entre la neurona de origen y la neurona de destino (Rosano, 2021).

2. **El elemento sumador**.- Es el elemento que efectúa la suma algebraica ponderada de las señales de entrada en una neurona es el sumador o unidad de suma. Esta etapa combina las señales de entrada multiplicadas por sus respectivos pesos usando la siguiente expresión (Rosano, 2021):

$$S = \sum_{\iota=1}^{n} W_i X_i \tag{2.1}$$

3. **El elemento de función activadora**.- Es el elemento que aplica una función no lineal de umbral a la salida del sumador es el nodo de activación de la neurona. Esta función no lineal, que puede ser una función escalón o una curva logística (como la función sigmoide), decide si la neurona se activa y produce una salida en función de la suma ponderada de las entradas (Rosano, 2021).

4. **El elemento de salida**.- Es el elemento que produce la señal que constituye la salida de la neurona es el axón. En las Redes Neuronales Artificiales, este modelo neuronal se utiliza ampliamente, aunque con variaciones en el tipo de función de activación empleada para determinar cómo se procesa y se transforma la señal antes de producir la salida final (Rosano, 2021).

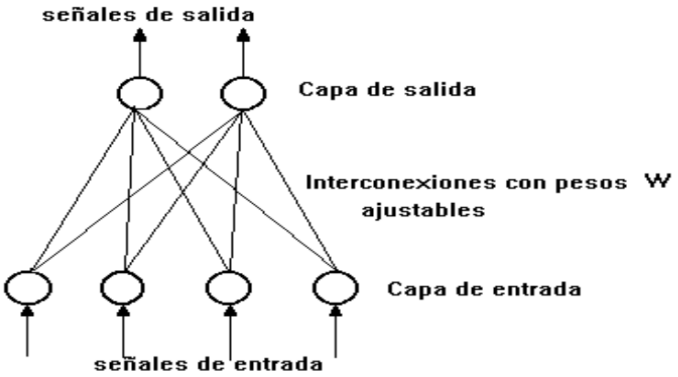
2.6.4.1. Modelos de redes neuronales artificiales

1. **Perceptrón**.- El Perceptrón, creado por Rosenblatt en 1958, es una neurona artificial que incluye varios componentes: entradas, un sumador, un activador y una salida. Las señales de entrada llegan a la neurona a través de conexiones, cada una con un peso asociado W_i. El sumador realiza una suma ponderada de las entradas. Luego, el activador utiliza una función escalón de umbral: si la suma ponderada es mayor o igual a un valor umbral U, la neurona emite una salida de Y de tal manera que:

$$y = 1 \quad SI \quad S > U$$
$$y = 0 \quad SI \quad S < U$$

La red neuronal más básica construida con perceptrones consta de dos capas: una capa de entrada, donde cada neurona simplemente transmite su entrada sin alterarla, y una capa de salida compuesta por perceptrones que están completamente conectados a la capa de entrada mediante líneas de comunicación con pesos ajustables. Así, cada neurona de la capa de entrada se conecta a cada neurona de la capa de salida a través de estas líneas con pesos que pueden modificarse. La regla de aprendizaje del perceptrón ajusta estos pesos para maximizar la probabilidad de obtener la salida deseada para un conjunto específico de entradas. En la siguiente figura 2.16 se presenta el perceptrón de dos capas (Rosano, 2021).

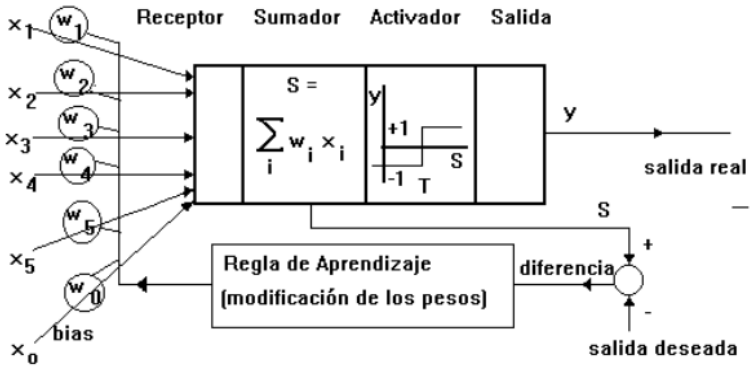
Figura 2.16: Perceptrón de dos capas.



Fuente: (Rosano, 2021).

2. **La Adaline**.- La Adaline es un tipo de neurona creado por Bernard Widrow y Marcian E. Hoff en 1960. Su nombre original fue adaptive linear neuron, fue más tarde cambiado a adaptive linear element cuando los modelos neuronales perdieron popularidad. Aunque es similar a un perceptrón en que calcula la suma ponderada de sus entradas, presenta algunas diferencias clave (Rosano, 2021). La Adaline es similar a un perceptrón en que también calcula la suma ponderada de sus entradas, pero presenta las siguientes diferencias. En la siguiente figura 2.17 se presenta el esquema de aprendizaje en un Adaline (Rosano, 2021):

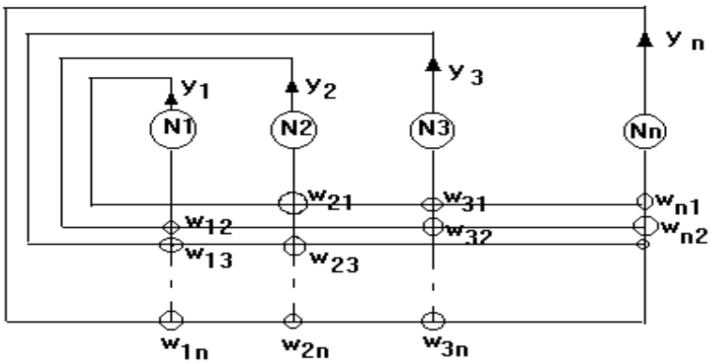
Figura 2.17: Esquema de aprendizaje en un Adaline.



Fuente: (Rosano, 2021).

3. **Redes de Hopfield**.- En 1982, John Hopfield presentó un nuevo tipo de redes neuronales que se conocieron posteriormente como Redes de Hopfield (Hopfield 1982). Una red de Hopfield básica consta de una sola capa de neuronas, donde todas están interconectadas mediante arcos ponderados bidireccionales. Los pesos de estas conexiones pueden ser tanto positivos como negativos. Esta estructura de conexiones bidireccionales convierte a la red en una red retroalimentada o recursiva (ver 2.18) (Rosano, 2021).
   Cada neurona en la red tiene una salida que puede ser 0 o 1. La salida es 1 si la suma ponderada d; de lo contrario, la salida es 0. En cada momento, toda la red se encuentra en un estado determinado por un vector de estados, donde cada elemento del vector corresponde al estado de una de las n neuronas. Estos estados de la red también se pueden visualizar como los vértices de un hipercubo de n dimensiones, donde n representa el número de neuronas. En la siguiente figura 2.18 se presenta el esquema de una red hopfield (Rosano, 2021).

Figura 2.18: Esquema de una red hopfield.

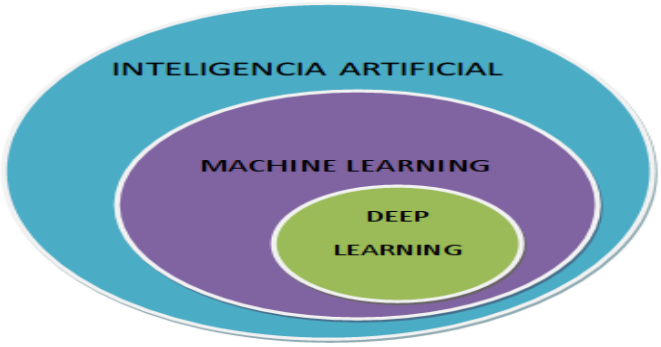


Fuente: (Rosano, 2021).

2.6.5. Deep learning

El aprendizaje profundo, o Deep Learning, es una subdisciplina del aprendizaje automático centrada en la construcción y entrenamiento de redes neuronales artificiales profundas. Estas redes, que están compuestas por múltiples capas, tienen la capacidad de aprender automáticamente a extraer características y realizar tareas complejas, como el reconocimiento de imágenes, el procesamiento del lenguaje natural y la toma de decisiones (by ezertis, 2021).
El entrenamiento de redes neuronales profundas comienza con la inicialización de los pesos y sesgos de la red. A continuación, se presentan a la red ejemplos de entrenamiento que consisten en pares de datos de entrada y sus salidas esperadas. La red realiza una propagación hacia adelante, en la que los datos pasan a través de las diferentes capas para calcular las salidas de las neuronas. Estas salidas se comparan con las salidas esperadas para evaluar el error. Luego, se aplica el algoritmo de retropropagación, que ajusta los pesos y sesgos de la red para reducir el error. Este algoritmo calcula el error retrocediendo a través de las capas y evaluando cómo cada neurona contribuye al error total. Los pesos y sesgos se actualizan utilizando métodos de optimización, como el descenso del gradiente. Este proceso se repite con múltiples ejemplos de entrenamiento, permitiendo que la red ajuste gradualmente sus pesos y sesgos, mejorando así su capacidad de generalización (by ezertis, 2021).
El Deep Learning ha demostrado ser extremadamente eficaz en diversas aplicaciones. En el reconocimiento de imágenes, ha superado el rendimiento humano en tareas como la clasificación y detección de objetos, así como en la segmentación semántica. En el procesamiento del lenguaje natural, ha revolucionado áreas como la traducción automática, el análisis de sentimientos, la generación de texto y el procesamiento del habla. Además, en el reconocimiento de voz, ha contribuido significativamente al desarrollo de asistentes virtuales y sistemas de transcripción con alta precisión. En el campo de la conducción autónoma, el Deep Learning ha mostrado su capacidad para interpretar el entorno, detectar obstáculos y tomar decisiones en tiempo real, impulsando el avance hacia vehículos autónomos más seguros y confiables. En la siguiente figura 2.19 se presenta la estructura del Deep Leearning (by ezertis, 2021).

Figura 2.19: Inteligencia artificial, Machine Learning y Deep Leearning.



Fuente: (by ezertis, 2021).

2.6.5.1. Enfoques y características en el uso de técnicas de aprendizaje profundo

1. **Adaptativos**: Los algoritmos de aprendizaje profundo (DL) son altamente sensibles a los cambios en la información durante el proceso de inferencia. Una vez que se diseñan las redes neuronales, deben ser entrenadas utilizando un subconjunto de datos. Este entrenamiento no se realiza de una sola vez, sino que se lleva a cabo en múltiples fases o iteraciones. En cada iteración, los algoritmos de DL tienen la capacidad de evolucionar, permitiéndoles identificar patrones y descubrir el conocimiento subyacente en los datos (Lagares, 2023).

2. **Escalables**:software independiente. En este contexto, estos algoritmos pueden funcionar como un servicio que proporciona capacidades predictivas o de inferencia para ser utilizadas por otros sistemas informáticos. Este enfoque se conoce como Software como Servicio (SaaS). Hoy en día, los algoritmos de DL, respaldados por una escalabilidad extensa, juegan un papel crucial en el éxito empresarial, especialmente en el manejo de casos de Big Data. Estas soluciones SaaS ofrecen características avanzadas para gestionar grandes volúmenes de datos, mantener alta velocidad de procesamiento y manejar la variedad de datos, facilitando así su integración y aplicación en diversos escenarios empresariales (Lagares, 2023).

3. **Iterativos**: Los algoritmos de aprendizaje profundo buscan encontrar una solución para un problema específico mediante un proceso que incluye entrenamiento y ajuste de hiperparámetros. La robustez de estas técnicas se debe a su flexibilidad y capacidad para manejar variaciones del problema una vez entrenadas. Es importante destacar que una red neuronal no necesita ser adaptada a cada posible escenario en el que se utilizará. Sin embargo, la calidad de los resultados que ofrece frente a casos nuevos dependerá en gran medida de la fase de entrenamiento y de la calidad de los datos de entrada utilizados (Lagares, 2023).

4. **Contextuales**: Cuando utilizamos algoritmos de aprendizaje profundo (Deep Learning), es fundamental comprender el contexto de los problemas que buscan resolver, así como los parámetros que utilizan y los resultados que generan. Las soluciones proporcionadas por una red neuronal deben ajustarse a cada caso específico. De manera similar, es importante reconocer que las respuestas a un problema pueden cambiar con el tiempo, según el entorno en el que se encuentren (Lagares, 2023). Por ejemplo, si se emplea una red neuronal para pronosticar el clima o para hacer predicciones en el mercado bursátil, la red debe manejar y adaptarse a una gran cantidad de datos de entrada que están en constante cambio. Además, las redes neuronales tienen la capacidad de extraer conocimientos tanto de datos estructurados como no estructurados (Lagares, 2023).

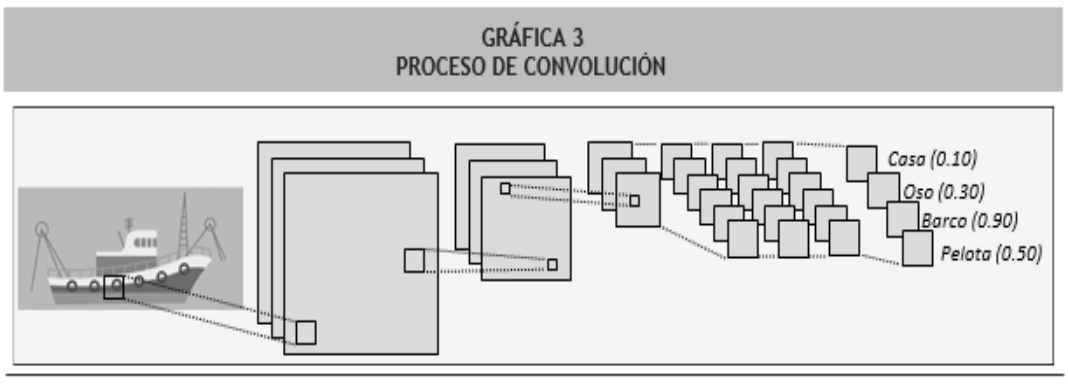
2.7. Redes neuronales convolucionales (CNN)

El término convolución se refiere a la aplicación de una función matemática sobre otra función. En el contexto de redes neuronales, este concepto se utiliza para identificar patrones en los datos, especialmente en las redes neuronales convolucionales (CNN), que están diseñadas para el procesamiento de imágenes y visión por computadora. La efectividad de las CNN se debe a la combinación de múltiples capas junto con el concepto matemático de convolución. En las CNN, cada capa tiene el propósito de identificar formas u objetos en las imágenes a través de la convolución realizada por la capa anterior. A diferencia de las redes neuronales profundas, las redes convolucionales presentan una conexión más reducida entre neuronas. En cambio, cada capa se enfoca en aplicar la convolución y pasar el procesamiento a la siguiente capa para extraer características relevantes de las imágenes (Lagares, 2023).

Para procesar imágenes, primero se convierten en matrices bidimensionales de bytes que representan sus píxeles. Además, existe una tercera dimensión, que corresponde a los canales de la imagen. Por ejemplo, las imágenes en blanco y negro tienen 2 canales, mientras que las imágenes en color RGB tienen 3 canales. Cada canal se representa con una matriz numérica que contiene la información específica de ese canal. Así, una imagen se transforma en un conjunto de matrices que se superponen según el número de canales. Cuando se aplica la convolución, la imagen se divide en áreas pequeñas, y cada una se analiza de manera matricial. El objetivo es reducir la dimensión de las matrices aplicando operaciones matemáticas que resuman la información contenida en esas áreas o matrices. El filtro convolucional recorre todas las matrices de la imagen, generando una nueva matriz que representa la imagen procesada. Esta técnica ayuda a reducir el número de píxeles, mejorando así el procesamiento y permitiendo la identificación de información relevante en regiones específicas de la imagen. Los valores del filtro y su desplazamiento son parámetros ajustables dentro de la red neuronal. (Lagares, 2023).

Este tipo de redes es ideal para asignar importancia a diferentes objetos en imágenes y facilitar su identificación. La combinación de capas convolucionales permite detectar variaciones en las imágenes, identificar objetos y aplicar filtros para corregir desenfoques. En la siguiente figura 2.20 se muestra el proceso de convolución (Lagares, 2023).

Figura 2.20: Proceso de convolución.



Fuente: (Lagares, 2023).

2.7.1. Capas de las redes neuronales convolucionales

2.7.1.1. Capa de entrada

La entrada para una ConvNet (red neuronal convolucional) consiste en un conjunto de imágenes, cada una de las cuales tiene HH filas, WW columnas y DD canales de color. La cantidad de canales depende del tipo de codificación de la imagen (por ejemplo, RGB tiene 3 canales: rojo, verde y azul). En general, se recomienda disponer de un gran volumen de datos de entrenamiento. Si esto no es posible, se puede recurrir a un proceso de data augmentation (aumento de datos). Este proceso implica aplicar hasta 21 transformaciones diferentes, como recortes, ajustes de brillo, desenfoques, variaciones de saturación y otros efectos a las imágenes de entrada para aumentar la diversidad del conjunto de datos.
En resumen, un conjunto de datos es la colección de información con la que trabajará la red (Hernández, 2021). Si se comienza con un único conjunto de datos, este debe ser dividido en:

- **Set de entrenamiento**: Es un conjunto de muestras utilizado para entrenar el modelo, durante el cual se ajustan los parámetros del mismo. Este proceso de entrenamiento permite que el modelo aprenda a partir de los datos y mejore su capacidad para hacer predicciones o clasificaciones (Hernández, 2021).
- **Set de validación**: Es un conjunto de muestras utilizado para ajustar los parámetros del clasificador. Este proceso de afinación permite que el clasificador optimice su rendimiento en función de los datos de entrenamiento (Hernández, 2021).
- **Set de pruebas**: Es un conjunto de muestras utilizado exclusivamente para evaluar el rendimiento de un clasificador que ya ha sido entrenado. Este conjunto de datos permite verificar cómo se comporta el clasificador en datos no vistos durante el proceso de entrenamiento (Hernández, 2021).

Partiendo de los tipos de conjuntos de datos mencionados anteriormente, se realiza una división del conjunto de entrada. Por ejemplo, el 70 % se destina al conjunto de entrenamiento, mientras que el 30 % restante se utiliza para el conjunto de pruebas. La razón para esta separación es evitar evaluar la calidad del modelo con los mismos datos que se usaron para entrenarlo. Este proceso busca estimar el error de generalización del modelo y prevenir problemas como el sobreajuste (overfitting) o el subajuste (underfitting) (Hernández, 2021).

Limpieza de set de imágenes
La creación de un conjunto de datos a partir de múltiples fuentes puede presentar desafíos para desarrollar un buen modelo, debido a la presencia de muestras que no corresponden a las clases definidas para el entrenamiento. Estas muestras irrelevantes, conocidas como ruido, pueden afectar negativamente el rendimiento del modelo. Por ejemplo, si se tiene un conjunto de datos con cuatro clases de imágenes que representan bosques en diferentes estaciones del año (invierno, verano, otoño y primavera) y se incluyen imágenes de automóviles o casas en la categoría de verano, estas imágenes no pertenecen a la clase correcta y constituyen un claro caso de ruido. Entrenar un modelo con un conjunto de datos que contiene ruido puede resultar en la generación de un modelo inexacto (Hernández, 2021).

Para detectar el ruido en las clases de un conjunto de datos, se pueden utilizar varios métodos. Uno de los enfoques más simples consiste en representar gráficamente el conjunto de datos y observar si hay partes que no cumplen con los parámetros de las clases definidas. Una vez identificadas las áreas que no se ajustan a los patrones esperados, se pueden eliminar las muestras problemáticas. Luego, se debe verificar si la eliminación ha sido efectiva y adecuada (Hernández, 2021).

Formato de entrada del set de datos
En las redes neuronales convolucionales (CNN), los formatos de entrada suelen ser BxHxWxD o BxDxHxW, donde B representa el número de imágenes en el lote, H es la altura, W es el ancho y D es la profundidad o el número de canales de cada imagen. Un formato comúnmente utilizado en conjuntos de datos de imágenes es el del conjunto MNIST. Este conjunto de datos se organiza en dos archivos: el primero contiene el tensor de imágenes con el formato BxHxWxD, mientras que el segundo archivo incluye las etiquetas correspondientes a cada imagen en el tensor (Hernández, 2021).

Por ejemplo, en un conjunto de datos de autos, si la primera imagen es de un auto, se guardará en el tensor de imágenes como (1xHxWxD), y su etiqueta en el archivo de etiquetas será (1xT). Para la segunda imagen, se almacenará como (2xHxWxD) con su etiqueta en (2xT). De manera general, para la i-ésima imagen en el tensor con el formato (Bii x HxWxD), su etiqueta se encontrará en el tensor de etiquetas como (ii x T) (Hernández, 2021).

2.7.1.2. Capa de convolución

La operación de convolución aplica un filtro (o kernel) a una imagen para extraer ciertas características. Por ejemplo, si un filtro está diseñado para detectar bordes, la imagen resultante de la convolución mostrará esos bordes. La idea principal de la convolución es aplicar un patrón a la imagen para identificar y extraer características o patrones específicos. Para ilustrar esto matemáticamente, consideremos una imagen simple de 6×6 píxeles y un filtro de 3×3 píxeles con una orientación vertical. El filtro, que suele ser más pequeño que la imagen, se desplaza por la imagen realizando operaciones en cada posición. En cada iteración, el filtro se coloca sobre una porción de la imagen y se multiplican punto a punto los valores del filtro por los píxeles correspondientes de la imagen. Luego, se suman estos productos para obtener el valor del píxel en la imagen de salida. Durante la primera iteración, se multiplica cada coeficiente del filtro por los valores de los píxeles en la porción de la imagen bajo el filtro y se suman estos productos para obtener el primer píxel en la imagen resultante. En la segunda iteración, el filtro se desplaza una posición a la derecha y se realiza el mismo cálculo para obtener el siguiente píxel de salida. Este proceso continúa, desplazando el filtro por toda la imagen. Cuando el filtro llega al extremo derecho, se mueve una posición hacia abajo y comienza de nuevo desde el lado izquierdo, repitiendo el proceso hasta que ha recorrido toda la imagen. Finalmente, se obtiene la imagen resultante, que luego se pasa a la capa de reducción (Hernández, 2021).

2.7.1.3. Capa de Agrupado

La capa de reducción, que es la segunda etapa en una CNN, se encarga de disminuir el flujo de datos procedente de las capas de convolución, extrayendo las características más importantes de la imagen convolucionada. Esta capa ayuda a reducir la carga computacional de la red, facilitando un entrenamiento más eficiente y una mejor segmentación de las características esenciales al eliminar el ruido de la imagen. Sin embargo, un uso excesivo del max pooling puede no solo eliminar el ruido, sino también suprimir características importantes, por lo que es crucial no abusar de esta capa. El objetivo final de estas capas es reducir el tamaño de la imagen mientras se conservan las características principales (las mayores activaciones). Como resultado, la imagen final, después de aplicar la convolución y la reducción, será más pequeña y contendrá las características principales, resultando en una imagen más limpia desde el punto de vista computacional en comparación con la imagen original. Por ejemplo, si partimos de una imagen de 4x4, después de la convolución y la reducción, obtendremos una imagen de 2x2 (Hernández, 2021).

2.7.1.4. Capa completamente conectada o fully connected

Estas capas calculan la puntuación para cada clase basada en las características extraídas por las capas de convolución y max pooling, relacionando todas las características previamente extraídas para producir una clasificación final. No hay reglas estrictas sobre el número de capas totalmente conectadas (fully connected) que se deben usar en este bloque final de la CNN; sin embargo, la literatura sugiere el uso de entre 2 y 4 capas, como se observa en redes como LeNet, VGG Net y la influyente AlexNet. Cada neurona en una capa totalmente conectada aplica una función no lineal, y la elección de esta función depende de la tarea específica que realiza la CNN. Sin embargo, dado que las capas de clasificación son computacionalmente intensivas, en los últimos años se han propuesto enfoques alternativos, como la capa de agrupación promedio global (global average pooling) y la capa de agrupación promedio (average pooling), que ayudan a reducir significativamente la carga computacional (Hernández, 2021).

2.7.2. Optimizadores en redes neuronales profundas

La optimización es una rama de las matemáticas que busca encontrar la "mejor" solución según un criterio cuantitativo específico. En la última década, la optimización matemática de procesos descritos por ecuaciones diferenciales parciales ha avanzado significativamente, siendo aplicada en diversas áreas como la ciencia, la ingeniería, las matemáticas, la economía y el comercio. La teoría de la optimización ofrece algoritmos para abordar problemas bien definidos y también proporciona un análisis de estos algoritmos. Un problema típico de optimización consiste en una función objetivo que se desea minimizar o maximizar, sujeto a ciertas restricciones. En el ámbito del aprendizaje automático, especialmente en redes neuronales, los algoritmos de optimización tienen el propósito de minimizar una función objetivo (comúnmente conocida como función de pérdida o costo), que representa la discrepancia entre los valores predichos y los esperados.
Seguidamente se listan los optimizadores a usar en el trabajo de investigación.

- **Adam (Kingma y Ba, 2014)** es un algoritmo para optimizar funciones objetivo estocásticas que se basa en gradientes de primer orden y utiliza estimaciones adaptativas de momentos de orden inferior. Este algoritmo se ha convertido en una de las herramientas de optimización más utilizadas por los profesionales del aprendizaje automático. La combinación del primer momento, normalizado por el segundo momento, determina la dirección de la actualización.
  La regla de actualización del optimizador Adam:

$$\theta_{n+1} = \theta_n - \frac{\alpha}{\sqrt{\upsilon_n + \epsilon}} m_n \tag{2.2}$$

- **RMSprop** es una técnica de optimización basada en gradientes utilizada para entrenar redes neuronales, propuesta por Geoffrey Hinton, uno de los pioneros en el campo de la retropropagación. Los gradientes de funciones complejas, como los de las redes neuronales, tienden a desvanecerse o a explotar a medida que los datos se propagan (relacionado con el problema de los gradientes que desaparecen). RMSprop fue diseñado como un enfoque estocástico para el aprendizaje en minilotes.

  Este método aborda el problema mencionando al emplear un promedio móvil de los gradientes al cuadrado, lo que permite normalizar el gradiente. Esta normalización ayuda a equilibrar el tamaño del paso (momentum), reduciendo el paso para gradientes grandes para evitar explosiones y aumentándolo para gradientes pequeños para prevenir su desvanecimiento.
  En resumen, RMSprop utiliza una tasa de aprendizaje adaptativa, lo que significa que esta tasa varía a lo largo del tiempo en lugar de ser un hiperparámetro fijo.
  La regla de actualización del optimizador RMSprop:

$$\begin{aligned}
\vartheta_{dw} &= \beta.\nu_{dw} + (1-\beta).dw^2 \\
\vartheta_{db} &= \beta.\nu_{dw} + (1-\beta).db^2 \\
W &= W - \alpha.\frac{dw}{\sqrt{\nu_{dw}} + \epsilon} \\
b &= b - \alpha.\frac{db}{\sqrt{\nu_{db}} + \epsilon}
\end{aligned}$$

2.7.3. Evaluación de modelos

Tanto los modelos heurísticos, estadísticos como los basados en principios físicos son simplificaciones y aproximaciones de una realidad que es mucho más compleja y contiene aspectos que aún no comprendemos completamente. Por ello, es esencial evaluar en qué medida estamos realmente representando esa realidad. Para ello, primero debemos establecer las siguientes definiciones (Edier, 2022):

- **Exactitud (accuracy)**:Se refiere al grado en el que una medida o valor mapeado, o una clase en un mapa, se aproxima a su valor verdadero o clase real en el terreno (Edier, 2022).

- **Error**: La diferencia entre el valor mapeado o la clase y el valor o clase real (Edier, 2022).

- **Precisión**: Se refiere al grado en el cual medidas repetidas bajo condiciones constantes producen resultados consistentes (Edier, 2022).
- **Incertidumbre (uncertainty)**: El grado en el que las características reales del terreno pueden ser representadas de manera precisa en un mapa (Edier, 2022).

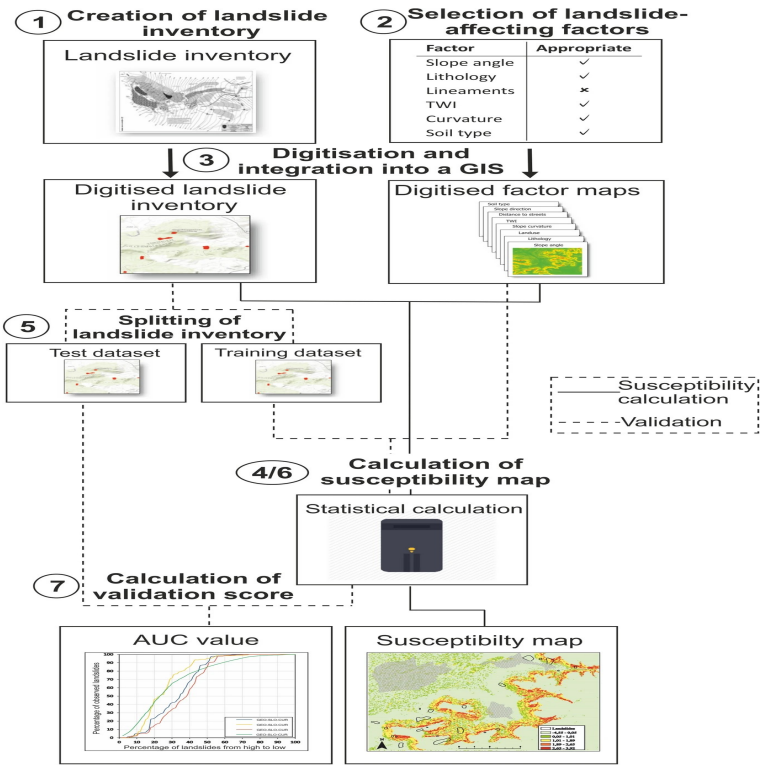
Los términos objetividad y subjetividad se utilizan para indicar si los diferentes pasos en la evaluación de un grado de amenaza son verificables y reproducibles por otros investigadores, o si dependen del juicio personal del investigador. Existen dos tipos de incertidumbre asociadas a los modelos. La incertidumbre aleatoria se refiere a la variabilidad que podemos conocer y que está implícita en muchas de las variables que usamos en los modelos. Esta incertidumbre puede ser integrada en nuestro análisis y es la que intentamos evaluar. Por otro lado, la incertidumbre epistémica está relacionada con el desconocimiento o las limitaciones que tenemos para describir los fenómenos físicos que estamos modelando. Dado que se refiere a aspectos que aún no conocemos, no puede ser evaluada o incorporada directamente en los modelos (Edier, 2022).

2.7.3.1. Técnicas de evaluación de modelos

1. **Validación cruzada**.- Como se ha indicado, la evaluación de un modelo debe abarcar dos aspectos: el desempeño y la capacidad de predicción. La evaluación del desempeño es relativamente directa y se refiere a cómo se comporta el modelo en función de los datos utilizados para su entrenamiento. Por otro lado, la capacidad de predicción es más compleja, ya que solo la aparición de nuevos eventos con el tiempo puede validar la efectividad del modelo en predecir situaciones futuras. Aunque la capacidad de predicción es crucial, la evaluación del desempeño también es esencial al construir el modelo. Ambas evaluaciones son necesarias para asegurar que el modelo no solo se ajuste bien a los datos actuales, sino que también pueda generalizar y predecir con precisión en escenarios futuros (Edier, 2022).

   Para llevar a cabo estas evaluaciones, se utilizan diversas estrategias de partición de datos. Una técnica comúnmente empleada es la validación cruzada, un método de remuestreo que divide los datos en diferentes subconjuntos para entrenar y evaluar un modelo en múltiples iteraciones. Esta técnica puede aplicarse de dos formas: espacialmente, donde se utiliza una región específica para entrenar el modelo y otra región, distinta pero dentro del mismo área de estudio, para evaluar su desempeño; o temporalmente, en la que se usan datos de un periodo de tiempo para entrenar el modelo y datos de un periodo posterior para evaluar su capacidad de predicción. En el caso de los modelos de susceptibilidad a movimientos en masa, se prefiere la partición espacial. Sin embargo, esta partición no se realiza generalmente por áreas específicas debido a las posibles diferencias en los mecanismos que afectan el proceso en distintas regiones. En cambio, se opta por una partición aleatoria en toda el área de estudio para evitar errores que podrían surgir si el modelo se basa en un mecanismo que no es representativo del área de evaluación. En la siguiente figura 2.21 se muestra el procedimiento para la validación cruzada (Edier, 2022).

Figura 2.21: Procedimiento para la validación cruzada.



Fuente: (Edier, 2022).

2. **Matriz de confusión**.- La matriz de confusión, también conocida como matriz de contingencia o matriz de error. Esta matriz es una tabla cruzada de frecuencias utilizada para una variable categórica binaria, y se emplea para comparar los resultados predichos por el modelo con los valores reales o medidos de la variable dependiente, en este caso, los movimientos en masa. En términos simples, la matriz de confusión compara las predicciones del modelo con los datos reales, proporcionando una base para calcular métricas de desempeño como la precisión, recuperación y otros indicadores clave (Edier, 2022).

   A partir de la matriz de confusión se derivan varias métricas clave:

   Verdaderos positivos (VP): Son los casos en los que el modelo identifica correctamente las celdas como inestables y estas coinciden con eventos registrados en el inventario de movimientos en masa. Los TP representan aciertos del modelo (Edier, 2022).

   Verdaderos negativos (VN): Son las celdas que el modelo clasifica correctamente como estables y que efectivamente no contienen eventos según el inventario de movimientos en masa. Los TN también son aciertos del modelo (Edier, 2022).

   Falsos positivos (FP): También conocidos como errores tipo I, son los casos en los que el modelo clasifica erróneamente las celdas como inestables, pero el inventario de movimientos en masa no muestra eventos en esas celdas. Estos representan errores del modelo (Edier, 2022).

   Falsos negativos (FN): También llamados errores tipo II, son las celdas que el modelo clasifica como estables, pero el inventario de movimientos en masa indica que sí hay eventos. Estos también son errores del modelo (Edier, 2022).

Estas métricas se utilizan para evaluar el desempeño del modelo en cuanto a su capacidad para identificar correctamente las condiciones de inestabilidad o estabilidad en el contexto de los movimientos en masa. En la siguiente figura 2.23 se muestra la matriz de confusión (Edier, 2022).

Figura 2.22: Matriz de confusión.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Fuente: (Edier, 2022).

3. **Curva ROC**.- Las curvas ROC (Receiver Operating Characteristic) son herramientas que ayudan a evaluar el desempeño de un modelo de clasificación binaria al mostrar cómo cambia la tasa de verdaderos positivos frente a la tasa de falsos positivos para diferentes umbrales de decisión. Estas curvas permiten elegir un umbral que balancee el número de falsos positivos y falsos negativos, de acuerdo con los objetivos específicos del problema.

   Un buen modelo de clasificación tendrá una curva ROC que se sitúe más cerca de la esquina superior izquierda del gráfico, indicando una alta tasa de verdaderos positivos y una baja tasa de falsos positivos. El área bajo la curva ROC (AUC) ofrece una medida global de la calidad del modelo, donde un valor de 0.5 sugiere un rendimiento aleatorio, y un valor de 1.0 indica un desempeño perfecto. Así, las curvas ROC te permiten seleccionar el umbral que minimiza el tipo de error que prefieras según las necesidades de tu aplicación (Silva, 2021).

   a) **Tasa de verdaderos positivos (TPR)**: También conocida como sensibilidad o recall, representa la proporción de verdaderos positivos que el modelo ha identificado correctamente (Silva, 2021). Se calcula como:

$$TPR = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Negativos} \tag{2.3}$$

   b) **Tasa de falsos positivos (FPR)**: Representa la proporción de negativos que el modelo ha clasificado incorrectamente como positivos (Silva, 2021). Se calcula como:

$$FPR = \frac{Falsos\ Positivos}{Falsos\ Positivos + Verdaderos\ Negativos} \tag{2.4}$$

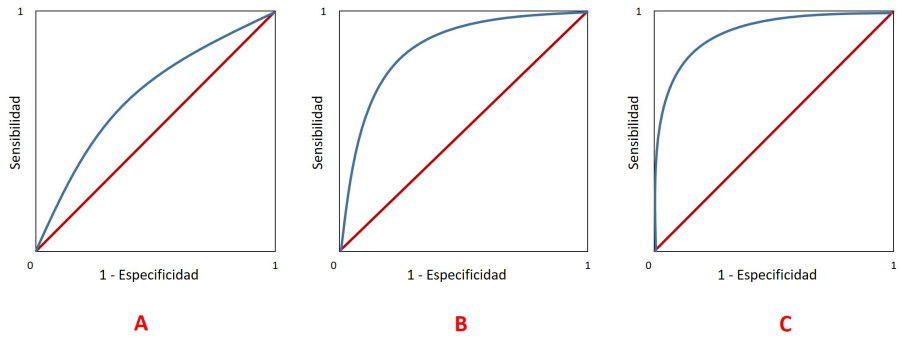
c) **Interpretación de la curva**: La curva ROC muestra cómo cambian la TPR y la FPR con diferentes umbrales de clasificación. Una curva más alta y a la izquierda indica un mejor rendimiento del modelo, ya que logra una alta TPR mientras mantiene una baja FPR (Silva, 2021).

Las curvas ROC pueden diferenciarse por su grado de convexidad de la siguiente manera:

1) **Poca convexidad (A)**: Si la curva ROC tiene poca convexidad, esto significa que el área entre la diagonal de referencia (línea de no discriminación) y la curva es pequeña. Esto sugiere que el modelo tiene un rendimiento deficiente, ya que no mejora significativamente en la clasificación en comparación con un clasificador aleatorio (Silva, 2021).
2) **Área mayor entre la diagonal y la curva (B)**: Un mayor área entre la diagonal y la curva indica un mejor desempeño del modelo de clasificación. Esto significa que el modelo es más efectivo en distinguir entre las clases positivas y negativas (Silva, 2021).
3) **Curva óptima (C)**: La mejor curva ROC es aquella que tiene un área más grande y muestra un crecimiento rápido al principio, seguido de una inclinación casi horizontal en los valores altos de sensibilidad. Esta forma indica que el modelo clasifica correctamente la mayoría de los casos positivos con pocos falsos positivos, permitiendo así una selección más eficiente del umbral de decisión (Silva, 2021).

En la siguiente figura 2.23 se muestra las curvas ROC con distintos grados de convexidad

Figura 2.23: Curvas ROC con distintos grados de convexidad.

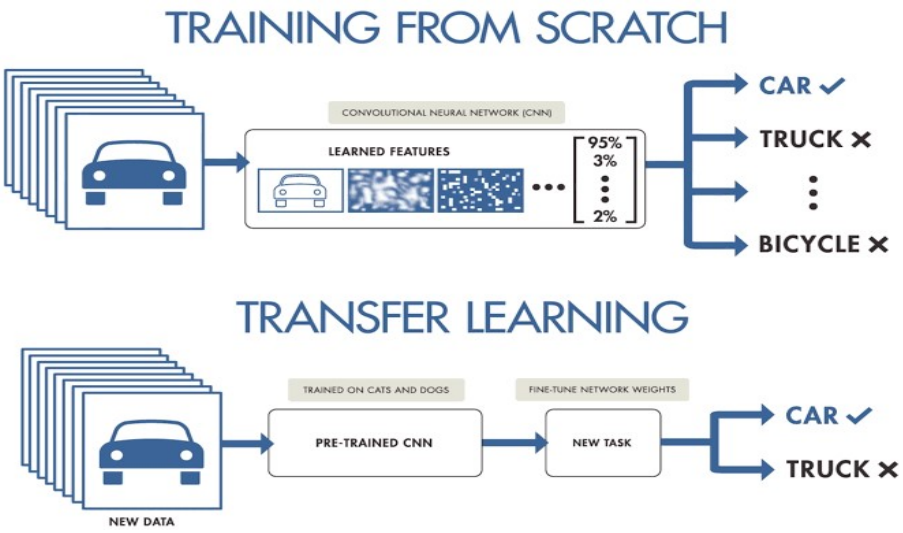


Fuente: (Silva, 2021).

2.7.4. Transferencia de aprendizaje

El aprendizaje por transferencia es un concepto clave en los proyectos, ya que permite evitar la tarea de construir una red neuronal desde cero al aprovechar modelos pre-entrenados y adaptarlos a nuestra tarea específica. Este proceso implica seleccionar un modelo previamente entrenado para una tarea similar, como los modelos que clasifican imágenes en diferentes categorías de ImageNet. Se congelan los valores de los pesos de las capas del modelo original y se reemplazan las capas superiores responsables de la clasificación por nuevas capas ajustadas a nuestra tarea particular, en la siguiente imagen se muestra la estructura de la transferencia de aprendizaje. En la siguiente figura 2.24 se muestra la estructura de la transferencia de aprendizaje (González, 2023).

Figura 2.24: Estructura de la transferencia de aprendizaje.



Fuente: (González, 2023).

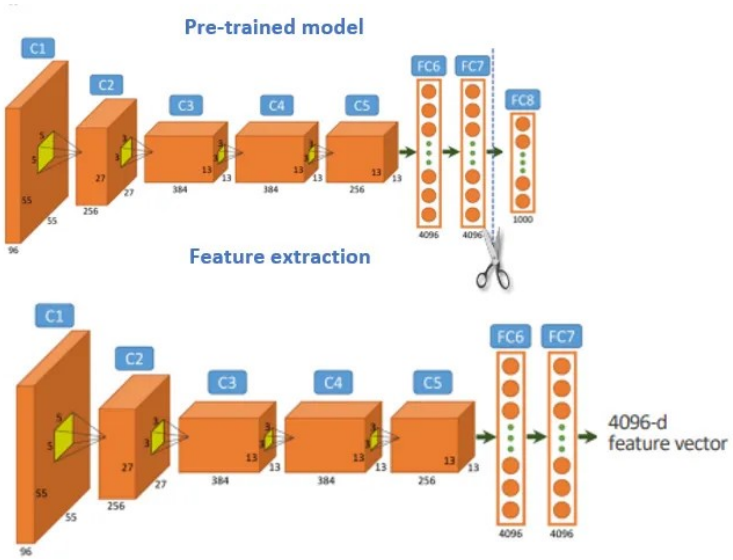
2.7.4.1. Técnicas de aprendizaje por transferencia

La extracción de características y el ajuste fino son técnicas utilizadas para aprovechar el conocimiento de un modelo previamente entrenado en una tarea de origen extensa para mejorar el rendimiento en una tarea de destino más específica. La extracción de características implica utilizar las representaciones aprendidas por el modelo pre-entrenado para extraer características relevantes de los datos, mientras que el ajuste fino ajusta los pesos del modelo en función de los datos de la tarea de destino para adaptar el modelo a las nuevas necesidades (Bosco, 2024).

- **Extracción de características feature extraction**.- Utiliza las representaciones aprendidas por la red previamente entrenada para extraer características significativas de nuevas muestras en una tarea específica. En este enfoque, se puede agregar un nuevo clasificador que se entrenará desde cero sobre el modelo preentrenado, permitiendo reutilizar el mapa de características ya aprendido. Específicamente, los parámetros del modelo preentrenado se mantienen congelados durante el entrenamiento para la tarea de destino, lo que evita la necesidad de volver a entrenar todo el modelo (Bosco, 2024).

De este modo, la red base, que ya ha aprendido características útiles, proporciona una base sólida para adaptarse a diferentes tareas específicas. En la siguiente figura 2.25 se muestra la extracción de características (Bosco, 2024).

Figura 2.25: Extracción de características.



Fuente: (Bosco, 2024).

- **Ajuste fino o fine-tuning**.- En el ajuste fino, los parámetros del modelo pre-entrenado se ajustan (o afinan) utilizando el conjunto de datos de la tarea de destino, junto con las nuevas capas añadidas. Este proceso implica descongelar algunas de las capas del modelo base que estaban previamente congeladas y entrenar conjuntamente tanto las nuevas capas de clasificación como las últimas capas del modelo base. De esta manera, se permite que no solo las capas adicionales se desarrollen, sino también que algunas de las capas anteriores del modelo pre-entrenado se ajusten para adaptarse mejor a la tarea específica, en la siguiente imagen se muestra la estructura del ajuste fino o fine-tuning. En la siguiente figura 2.26 se muestra la estructura del ajuste fino (Bosco, 2024).

Figura 2.26: Estructura del ajuste fino.

Fuente: (Bosco, 2024).

El ajuste fino se utiliza comúnmente cuando el conjunto de datos de la tarea de destino es relativamente grande y similar al conjunto de datos con el que se entrenó el modelo previamente. Sin embargo, en conjuntos de datos pequeños, puede ser propenso al sobreajuste, especialmente si la nueva tarea difiere significativamente de la tarea original para la que se entrenó el modelo. Para mitigar este problema, se pueden aplicar técnicas de regularización, como el abandono (dropout) y la detención temprana. Además, el ajuste fino generalmente requiere más recursos computacionales y un tiempo de entrenamiento más largo en comparación con la extracción de características, ya que todo el modelo se entrena y el proceso de retropropagación debe actualizar una mayor cantidad de parámetros (Bosco, 2024).

Como se muestra en la figura 2.24, una ventaja significativa es que estamos utilizando un modelo que ha aprendido a extraer características a partir de un conjunto de datos amplio. Esto nos permite obtener un modelo con una gran capacidad de generalización, incluso cuando trabajamos con un conjunto de datos pequeño como el nuestro (González, 2023).

2.7.5. El aumento de datos o data augmentation

La mejor manera de mejorar la capacidad de generalización de un modelo de aprendizaje automático es entrenarlo con una mayor cantidad de datos. Sin embargo, en la práctica, la disponibilidad de datos puede ser limitada. Una solución a este problema es generar datos sintéticos y añadirlos al conjunto de datos de entrenamiento. Crear datos sintéticos es relativamente sencillo para algunas tareas de aprendizaje automático, especialmente cuando se trata de problemas de clasificación. En estas tareas, un clasificador toma una entrada x de múltiples dimensiones y la asigna a una categoría única.Esto implica que la principal tarea de un clasificador es ser robusto frente a una amplia gama de transformaciones. Podemos generar nuevos pares (x, y) de manera sencilla transformando las entradas x en nuestro conjunto de entrenamiento. Sin embargo, este enfoque no es tan aplicable a muchas otras tareas. Por ejemplo, es complicado crear datos sintéticos para una tarea de estimación de densidad, a menos que ya hayamos resuelto previamente el problema de estimación de densidad (Calvo, 2024).
El aumento del conjunto de datos ha demostrado ser especialmente eficaz para un problema de clasificación particular: el reconocimiento de objetos. Las imágenes tienen una alta dimensionalidad y una gran variabilidad, y muchos de estos efectos de variabilidad se pueden simular de manera efectiva. Por ejemplo, realizar translaciones de unos pocos píxeles en diferentes direcciones sobre las imágenes de entrenamiento puede mejorar significativamente la capacidad de generalización del modelo. La generación de datos artificiales implica crear conjuntos de datos sintéticos que imiten los datos reales que los modelos de visión por computadora deben analizar y comprender. Estos datos sintéticos se producen mediante técnicas avanzadas, como la generación de imágenes con modelos generativos o la manipulación y transformación de imágenes existentes (Calvo, 2024).

2.7.5.1. Técnicas de aumento de datos en imágenes

En esta sección, presentamos algunas técnicas de aumento básicas pero altamente efectivas que son ampliamente utilizadas (Calvo, 2024).

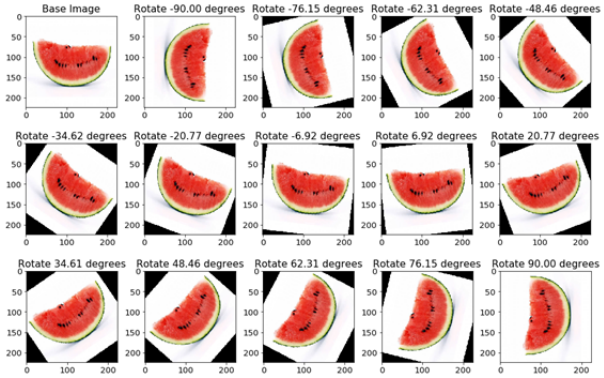
- **Voltear**.- Las imágenes pueden ser volteadas horizontal o verticalmente. Algunos marcos no ofrecen opciones para giros verticales, pero un giro vertical es equivalente a rotar la imagen 180 grados y luego realizar un giro horizontal. A continuación, se presentan ejemplos de imágenes que han sido volteadas. En la siguiente figura 2.27 se muestra una imagen volteada (Calvo, 2024).

Figura 2.27: Imagen volteada.

Fuente: (Calvo, 2024).

- **Rotar**.- Un aspecto importante a considerar con respecto a esta operación es que las dimensiones de la imagen pueden no mantenerse constantes después de la rotación. Si la imagen es cuadrada, girarla a ángulos rectos mantendrá el tamaño original. En el caso de una imagen rectangular, una rotación de 180 grados conservará el tamaño. Sin embargo, rotar la imagen en ángulos más pequeños puede modificar el tamaño final de la imagen. En la siguiente sección, abordaremos cómo manejar este problema, en la siguiente figura 2.28 se muestra una imagen rotada (Calvo, 2024).

Figura 2.28: Imagen con rotación.



Fuente: (Calvo, 2024).

- **Escalado**.- La imagen puede ser escalada hacia afuera o hacia adentro. Al escalar hacia afuera, la imagen final será más grande que la original. En muchos casos, se recorta una sección de la imagen ampliada para que coincida con el tamaño de la imagen original. En la siguiente sección, nos enfocaremos en el escalado hacia adentro, que reduce el tamaño de la imagen y requiere hacer suposiciones sobre el contenido que queda fuera de los límites.

A continuación, se presentan ejemplos de imágenes escaladas, en la siguiente figura 2.29 se muestra una imagen escaldada. (Calvo, 2024).

Figura 2.29: Imagen con escalado.

Fuente: (Calvo, 2024).

- **Recortar**.- A diferencia del escalado, donde la imagen completa se ajusta al nuevo tamaño, el recorte aleatorio implica seleccionar aleatoriamente una sección de la imagen original y luego redimensionarla para que coincida con el tamaño de la imagen original. Este método es conocido como recorte aleatorio. A continuación, se presentan ejemplos de recortes aleatorios. Al observar con atención, es posible notar las diferencias entre este enfoque y el escalado, en la siguiente figura 2.30 se muestra una imagen recortada. (Calvo, 2024).

Figura 2.30: Imagen con recorte.

Fuente: (Calvo, 2024).

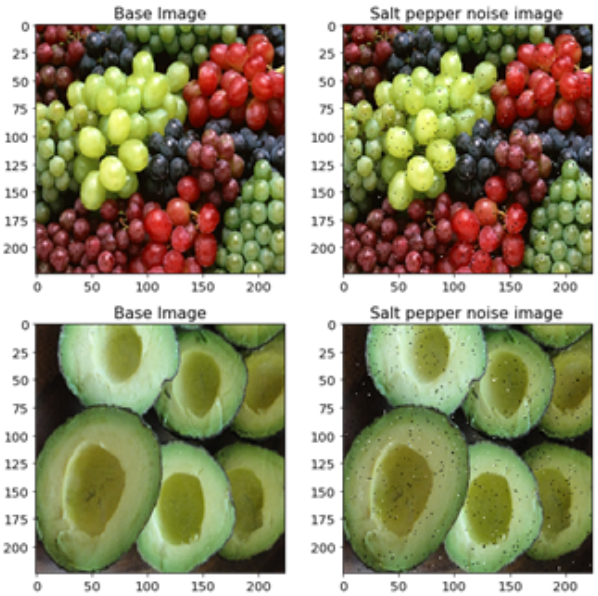
- **Traslación**.- La traslación simplemente consiste en desplazar la imagen a lo largo de las direcciones X o Y (o ambas). En el ejemplo siguiente, se asume que la imagen tiene un fondo negro más allá de sus bordes, permitiendo una traslación adecuada. Este método de aumento es especialmente útil, ya que la mayoría de los objetos pueden aparecer en casi cualquier parte de la imagen. Esto obliga a la red neuronal convolucional a buscar en todas las regiones de la imagen, en la siguiente figura 2.31 se muestra una imagen con traslacion. (Calvo, 2024).

Figura 2.31: Imagen con traslación.

Fuente: (Calvo, 2024).

- **Ruido Gausiano**.- El ajuste excesivo suele ocurrir cuando la red neuronal intenta aprender características de alta frecuencia, que son patrones que se repiten con frecuencia pero que pueden no ser útiles. El ruido gaussiano, que tiene una media de cero, contiene puntos de datos en todas las frecuencias, lo que distorsiona las características de alta frecuencia. Esto también afecta a los componentes de baja frecuencia (generalmente, los datos esperados), aunque la red neuronal puede aprender a ignorar estas distorsiones. Añadir una cantidad controlada de ruido puede mejorar la capacidad de aprendizaje del modelo. Una forma más suave de esto es el ruido de sal y pimienta, que consiste en introducir píxeles aleatorios en blanco y negro en la imagen. Aunque tiene un efecto similar al del ruido gaussiano, el ruido de sal y pimienta tiende a causar una menor distorsión de la información, en la siguiente figura 2.32 se muestra una imagen con ruido gausiano (Calvo, 2024).

Figura 2.32: Imagen de ruido gausiano.



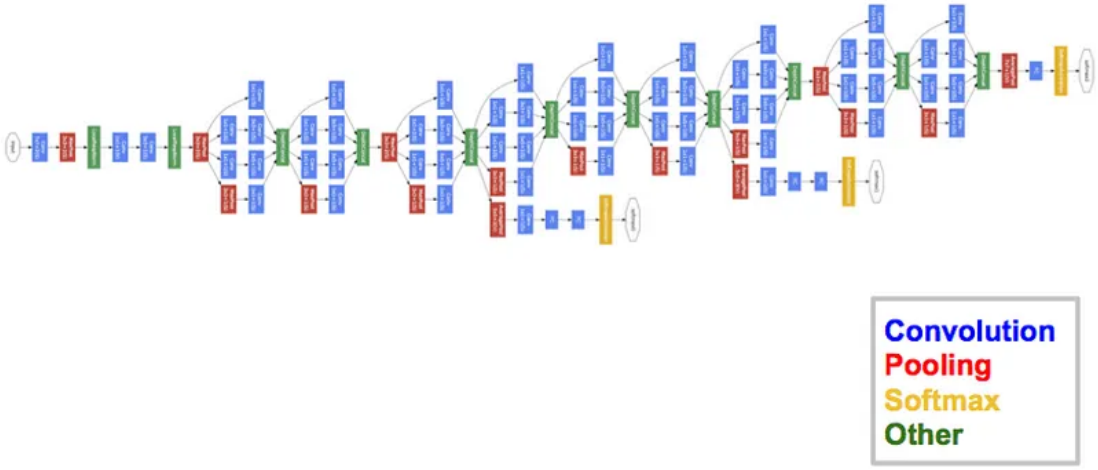
Fuente: (Calvo, 2024).

2.7.6. Arquitecturas de neuronales convolucionales

2.7.6.1. InceptionV3.

InceptionV3, la tercera versión de la red neuronal convolucional Inception desarrollada por Google, incorpora varias mejoras clave. Estas incluyen el uso del optimizador RMSProp, convoluciones factorizadas de 7 × 7, la aplicación de Batch Normalization en los clasificadores auxiliares y la técnica de suavizado de etiquetas. Las convoluciones factorizadas se destacan por reducir la cantidad de parámetros, manteniendo la eficiencia de la red (Ali Hasan Md. Linkon, 2017). En la siguiente figura 2.33 se muestra la arquitectura InceptionV3.

Figura 2.33: Arquitectura InceptionV3.



Fuente: (Siddharth, 2017).

2.7.6.2. VGGNet.

La arquitectura VGGNet es una red neuronal convolucional profunda y estándar con múltiples capas. El número de capas de VGG-16 o VGG-19 constan de 16 y 19 capas convolucionales respectivamente. Las innovaciones en los modelos de reconocimiento de objetos se basan en la arquitectura VGGNet, siendo este una de las arquitecturas de reconocimiento de imágenes más populares. En la siguiente figura 2.34 se muestra la arquitectura VGGNET (Siddharth, 2017).

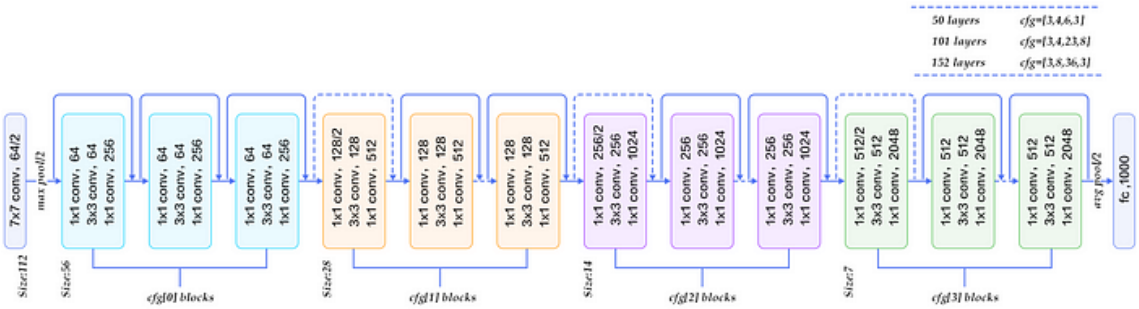
Figura 2.34: Arquitectura VGGNET.

Fuente: (Siddharth, 2017).

2.7.6.3. ResNet.

La arquitectura ResNet es una red directa de 34 capas que se inspiró en la VGG-19. ResNet añadió conexiones directas, transformando la arquitectura en una red residual. Estas conexiones directas se muestran en la figura 2.35 y han hecho de ResNet una de las arquitecturas de reconocimiento de imágenes más populares. Puede manejar tareas y conjuntos de datos más allá de ImageNet. En la siguiente figura 2.35 se muestra la arquitectura ResNet.

Figura 2.35: Arquitectura ResNet.

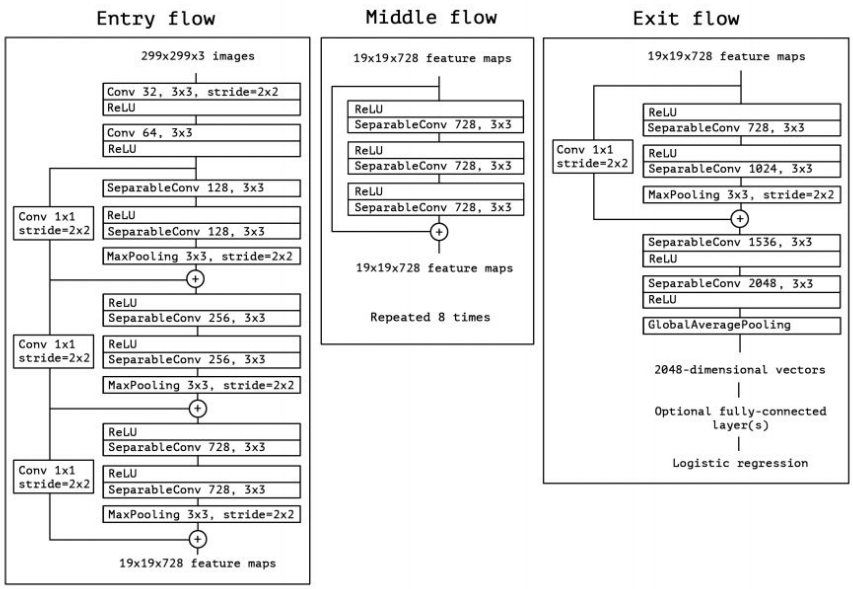


Fuente: (Siddharth, 2017).

2.7.6.4. Xception

Xception es una arquitectura de red neuronal convolucional profunda que utiliza convoluciones separables en profundidad. Fue desarrollada por Google, que reinterpretó los módulos Inception en redes neuronales convolucionales como un avance entre las convoluciones estándar y las convoluciones separables en profundidad (que combinan una convolución en profundidad con una convolución puntual). En este contexto, las convoluciones separables en profundidad pueden verse como una forma avanzada de los módulos Inception, con un número máximo de torres. Esta observación llevó a los investigadores a crear una nueva arquitectura de red neuronal convolucional profunda, basada en Inception, en la que los módulos Inception tradicionales fueron sustituidos por convoluciones separables en profundidad.(Fabien, 2017).
Los datos ingresan primero en el flujo de entrada, luego pasan por el flujo intermedio, que se repite ocho veces, y finalmente llegan al flujo de salida. Es importante notar que cada capa de convolución y convolución separable está seguida de una normalización por lotes. La arquitectura Xception ha superado a VGG-16, ResNet e Inception V3 en la mayoría de los desafíos tradicionales de clasificación, en la siguiente figura 2.36 se muestra la arquitectura Xception .(Fabien, 2017).

Figura 2.36: Arquitectura Xception.

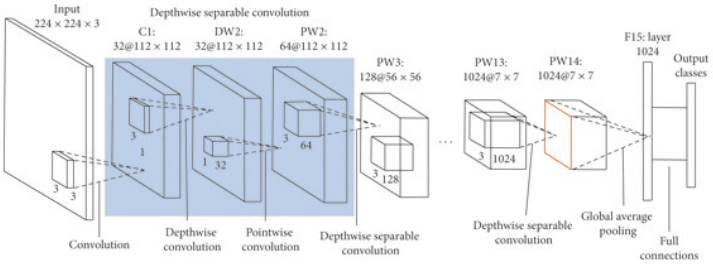


Fuente: (Fabien, 2017).

2.7.6.5. MobileNet

MobileNet es una arquitectura diseñada para ser eficiente, utilizando convoluciones separables en profundidad para construir redes neuronales convolucionales que sean tanto profundas como ligeras, ideales para aplicaciones móviles e integradas. Su diseño se basa en la utilización de filtros separables en profundidad, que se dividen en dos tipos: filtros de convolución en profundidad y filtros de convolución puntual. Los filtros de convolución en profundidad realizan una convolución de manera independiente en cada canal de entrada, mientras que los filtros de convolución puntual combinan las salidas de estas convoluciones en profundidad a través de convoluciones 1x1, logrando una convolución lineal en la siguiente figura 2.37 se muestra la arquitectura MobileNet (Wei Wang, 2020).

Figura 2.37: Arquitectura MobileNet.

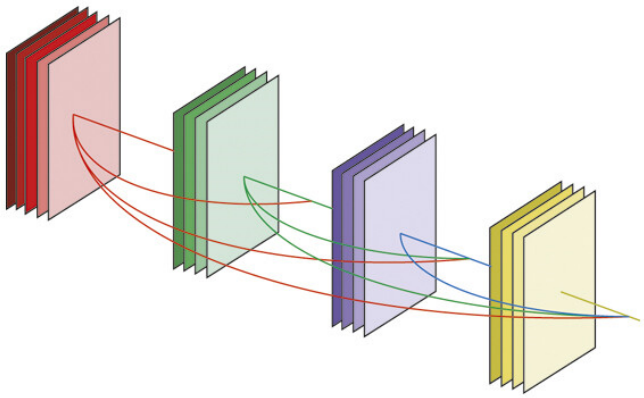


Fuente: (Wei Wang, 2020).

2.7.6.6. DenseNet

DenseNet utiliza bloques densos como unidades básicas en su estructura. Cada bloque denso está formado por 4 capas interconectadas de manera densa, con una tasa de crecimiento de 4. En esta configuración, cada capa recibe los mapas de características generados por las capas previas como entrada. A diferencia de la unidad residual en ResNet, que suma los mapas de características de las capas anteriores en una sola capa, el bloque denso transmite los mapas de características a todas las capas siguientes, incrementando la dimensión de los mapas de características en lugar de simplemente sumar los valores de los píxeles, en la siguiente figura 2.38 se muestra la arquitectura DenseNet (Wei Wang, 2020).

Figura 2.38: Arquitectura DenseNet



Fuente: (Wei Wang, 2020).

Capítulo 3

Desarrollo del proyecto

3.1. Metodología

La metodología de investigación se basará en el enfoque cuantitativo de Hernández-Sampieri (2018), dado que se busca medir la precisión y eficacia de los modelos preentrenados del Transfer Learning. Este estudio se clasifica como investigación aplicada, centrada en el desarrollo e implementación de una solución tecnológica.
Para el desarrollo de la aplicación móvil, se aplicó la metodología Scrum, debido a que su enfoque iterativo facilita una rápida adaptación de los modelos y permite una integración continua con la aplicación.
Para el desarrollo del siguiente proyecto se siguieron los siguientes pasos:

1. **Revisión de bibliografía**. La revisión bibliográfica se llevará a cabo a partir de artículos científicos que abordan la clasificación de imágenes mediante Transfer Learning. Para establecer la base teórica de cada defecto de calidad, se consultará a diversos profesionales expertos en la zona, incluyendo técnicos de campo y evaluadores de calidad de la empresa *AGRICOLA ALSUR CUSCO*, quienes aportarán su experiencia en el diagnóstico de defectos de calidad de la alcachofa en la etapa de inflorescencia.

2. **Recolección de imágenes**. En esta fase, se procederá a la recolección de imágenes utilizando una cámara fotográfica profesional. Cada alcachofa con defectos de calidad será colocada sobre una superficie plana y blanca para minimizar el ruido en la imagen. Las fotografías se tomaron en Arequipa y Cusco. El conjunto de datos se dividió en tres grupos: 70 % para el entrenamiento, 20 % para la validación y 10 % para la prueba.

3. **Preprocesamiento de imágenes**. En esta fase, se procederá a limpiar el conjunto de datos, eliminando el ruido generado durante la recolección, como los problemas de iluminación (exceso o falta de luz) y la presencia de agentes externos, como tierra, humedad e insectos.

4. **Entrenamiento de modelos del Transfer Learning**. En esta fase, se llevará a cabo el entrenamiento de los principales modelos preentrenados de Transfer Learning utilizados para la clasificación de imágenes. Los modelos a entrenar son DenseNet, InceptionV3, Xception y ResNet 50.

5. **Implementación de la aplicación móvil**. En esta fase, se implementó la aplicación móvil utilizando el lenguaje de programación Dart y el framework Flutter. La aplicación interactuará con el modelo que ofrece la mejor precisión en la clasificación de defectos de calidad, proporcionando así un diagnóstico preciso.

3.2. Recolección de imágenes

Para implementar un sistema de visión artificial, es esencial recolectar imágenes que muestren los diferentes tipos de defectos de calidad en las alcachofas durante su etapa de inflorescencia. Estos defectos se calsificaron en las siguientes categorías: Primera, Fofa, Cintura y Violácea. Las imágenes se capturan sobre una superficie blanca y plana, utilizando diferentes ángulos de la cámara para facilitar la identificación de los defectos de calidad. En las siguientes figuras B.15 se presentan ejemplos de las imágenes obtenidas.
La recolección de imágenes se llevó a cabo en las ciudades de Arequipa (Pedregal) y Cusco (comunidad campesina de Markjo) entre los meses de abril y julio. Los equipos empleados para la captura de imágenes se detallan en la siguiente tabla 3.1. En la figura 3.2(a) se muestra un campo de cultivo de alcachofas durante su etapa de inflorescencia. Durante este periodo, se recorrieron diversos campos donde los agricultores realizaban sus diagnósticos, tanto de manera motorizada como a pie. La figura 3.2(b) ilustra el proceso de captura de imágenes en el campo.
Para la clasificación de las imágenes recolectadas, se contó con la colaboración del encargado del área de calidad de la empresa AGRICOLA ALSUR CUSCO, quien también proporcionó los datos técnicos necesarios para el diagnóstico de los defectos de calidad. Se decidió recolectar un total de 2,240 imágenes, distribuidas equitativamente entre las categorías de defectos, con 560 imágenes por cada tipo. Estas imágenes se dividirán en conjuntos de entrenamiento (70 %), validación (20 %) y prueba (10 %).

Tabla 3.1: Equipos y condiciones para la captura de imágenes.

Equipo de Captura	**Características del Equipo**	**Condiciones de Captura**	**Observaciones**
Canon EOS 2000D	Píxeles reales: 24,7 MP Resolución máxima: 6000 x 4000 Tipo de sensor: CMOS	Exterior, luz natural y una superficie blanca	La superficie blanca esta sobre un tablero para un facil uso
Smartphone Xiaomi POCO F5	Píxeles: 64 MP Resolución máxima: 2400 x 1080 Tipo de sensor: Sony IMX686 de 64 MP.	Exterior, luz natural y una superficie blanca	La superficie blanca esta sobre un tablero para un facil uso

Fuente: Elaboración propia.

Tabla 3.2: tabla de distribución de imágenes.

Tipo de Conjunto	**Cantidad de Imágenes**	**Porcentaje de imágenes**
Entrenamiento	1568	70 %
Validación	448	20 %
Testeo	224	10 %
Total	**2240**	**100 %**

Fuente: Elaboración propia.

Figura 3.1: Toma de imágenes de defectos de calidad en los cultivos alcachofa.

(a) Cultivo de alcachofa. (b) Toma de imágenes.

Fuente: Elaboración propia.

Figura 3.2: Imágenes de defectos de calidad en las alcachofas en etapa de inflorescencia.

(a) Primera. (b) Fofa. (c) Cintura. (d) Violácea.

Fuente: Elaboración propia.

3.3. Preprocesamiento de imágenes

El preprocesamiento de imágenes consiste en aplicar técnicas que mejoran la calidad de las imágenes de entrada, lo que facilita una clasificación más precisa en redes neuronales convolucionales. Durante la captura de datos, se observó que las imágenes de alcachofas con defectos de calidad presentaban ruido, como tierra, suciedad y humedad, lo que generaba confusión en la red neuronal durante el entrenamiento y reducía tanto la claridad como el contraste de las imágenes. Para mitigar estas deficiencias, se emplean diversas técnicas de preprocesamiento, tales como la ecualización de histograma, el filtro gaussiano y el filtrado de mediana.

3.3.0.1. Ecualización de histograma

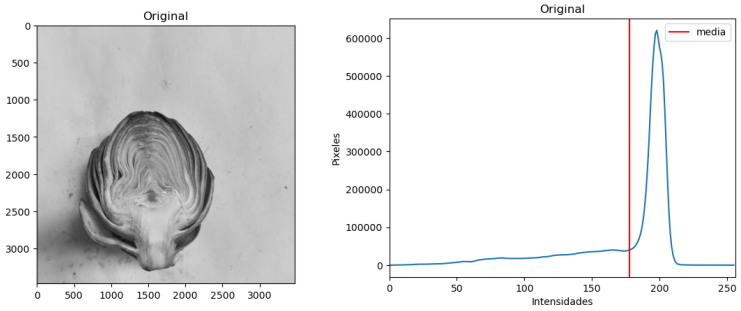
La ecualización de histograma es una técnica esencial para mejorar el contraste en imágenes, especialmente cuando las condiciones de iluminación, como el sol o las sombras, afectan negativamente la calidad visual Roberto M. Dyke (2023). En el caso de las alcachofas, este procesamiento permite resaltar de manera más efectiva los defectos superficiales, como se ilustra en la Figura 3.3. La imagen original muestra una distribución desigual de los niveles de intensidad, mientras que la imagen ecualizada presenta una distribución más uniforme, facilitando la detección de imperfecciones.

Algorithm 1 Ecualización de histograma

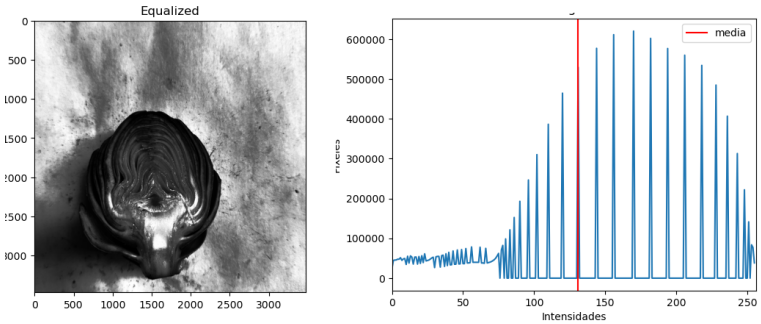
1: **Entrada:** Ruta de la imagen **ruta_img**
2: **Salida:** Imágen procesada
3: $img \leftarrow$ LeerImagen(ruta_img)
4: $img\_gray \leftarrow$ ConvertirAEscalaDeGrises(img)
5: $img\_histeq \leftarrow$ EcualizarHistograma($img\_gray[:, :, 0]$)
6: $img\_histeq\_ \leftarrow$ RedimensionarImagen($img\_histeq, (224, 224)$)
7: **Retornar:** $img\_histeq\_$

Fuente: Elaboración propia.

Figura 3.3: Ecualización de histograma.



(a) Imagen Original.



(b) Imagen con ecualización del histograma.

Fuente: Elaboración propia.

3.3.0.2. Filtrado de mediana

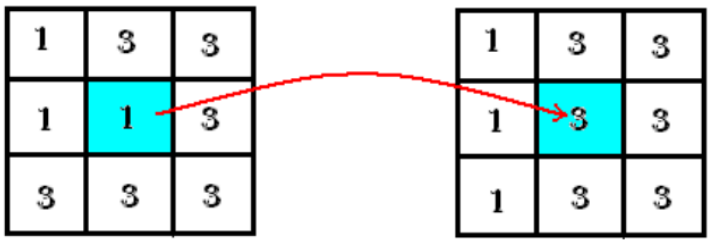
El filtro de la mediana tiene como objetivo reducir el empañamiento de los bordes, reemplazando el píxel en análisis por la mediana de los niveles de intensidad de gris de los píxeles vecinos más cercanos. Para ello, se define un área alrededor del píxel, se ordenan los niveles de gris de menor a mayor y se selecciona el valor mediano de esta muestra Fredes Hubert (2023).

Algorithm 2 Proceso de filtrado de mediana

1: **Entrada:** Ruta de la imagen **ruta_img**
2: **Salida:** Imágen redimensionada y procesada
3: imagen_redimen ← Redimensionar **ruta_img** a 224×224
4: imagen_procesada ← Aplicar filtro mediana de tamaño 3×3 a **imagen_redimen**
5: img_mediana ← Redimensionar **imagen_procesada** a 224×224
6: **Retornar:** img_mediana

Fuente: Elaboración propia.

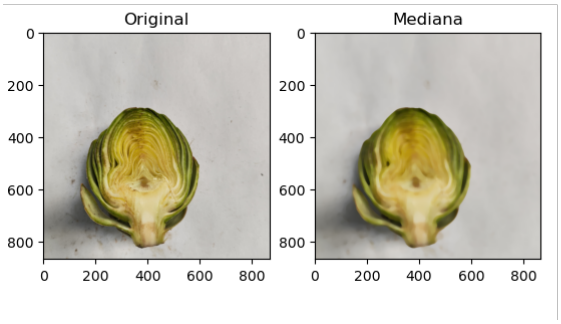
Figura 3.4: Filtrado de mediana.



Fuente: Elaboración propia.

El filtrado de mediana es esencial para eliminar el ruido impulsivo de una imagen, preservando los bordes más destacados. Estos bordes, a su vez, pueden ser realzados posteriormente con filtros de alta frecuencia, como se sugiere en Fredes Hubert (2023). La Figura 3.4 ilustra este proceso: la imagen original (izquierda) presenta un ruido significativo, mientras que la imagen filtrada (derecha) muestra una reducción notable en la variación de intensidad de los píxeles.

Figura 3.5: Filtrado de mediana.



Fuente: Elaboración propia.

3.3.0.3. Filtro gaussiano

El filtro gaussiano es un tipo de filtro pasa-bajo cuyo diseño se basa en la función de distribución gaussiana. Este filtro utiliza los valores de dicha función para calcular sus coeficientes y es ampliamente empleado en el procesamiento de imágenes y señales. Su principal función es suavizar los datos, reduciendo el ruido, mientras mantiene los bordes con mayor eficacia que otros filtros Fredes Hubert (2023).
El filtro gaussiano se utiliza principalmente para suavizar imágenes afectadas por ruido. La máscara más comúnmente empleada en este filtro, que sigue una distribución gaussiana, tiene la forma de una matriz cuyos valores se calculan a partir de la función de distribución gaussiana Fredes Hubert (2023). La estructura del kernel del filtro gaussiano se presenta en la siguiente fórmula 3.1.

$$W = \frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.1}$$

El kernel gaussiano, también conocido como máscara gaussiana, es una matriz de 3x3 que se utiliza para el suavizado de imágenes. Esta máscara se caracteriza por asignar mayores pesos al píxel central y a los píxeles adyacentes cercanos, mientras que los píxeles más distantes reciben un peso menor. La disposición de los pesos sigue la forma de la función gaussiana, con el valor más alto en el píxel central y valores decrecientes a medida que nos alejamos del centro Fredes Hubert (2023). En la figura 3.6, a la izquierda, se presenta la imagen original, que contiene ruido de fondo. Al aplicar el filtro gaussiano, como se muestra en la figura 3.6 a la derecha, utilizando una máscara de 5x5, se observa una reducción significativa del ruido y una mejora en la calidad de la imagen.

Algorithm 3 Aplicar filtro Gaussiano a una imagen

1: **Entrada:** imagen_original
2: **Salida:** Imágenes procesadas guardadas en el directorio adecuado
3: tamaño $\leftarrow (3, 3)$
4: coeficientes $\leftarrow$ [1, 2, 1, 2, 4, 2, 1, 2, 1]
5: imagen_filtrada $\leftarrow$ Aplicar filtro gaussiano con kernel tamaño y coeficientes a imagen_original
6: imagen_suavizado $\leftarrow$ Redimensionar imagen_filtrada a 224×224
7: **Guardar:** imagen_suavizado en 'img_preprocesado/filtro_gauss.JPG'
8: **Retornar:** imagen_suavizado

Figura 3.6: Filtrado Gaussiano.

Fuente: Elaboración propia.

3.4. Aumento de datos

Para aumentar la diversidad del conjunto de datos y reducir el riesgo de sobreajuste, se implementó una técnica de aumento de datos basada en transformaciones geométricas. Se comenzó con la creación de un modelo simple, el cual se integrará posteriormente al modelo principal. En el algoritmo 4 se muestran dos capas que se emplearán para las transformaciones geométricas como parte del aumento de datos: **RandomFlip('horizontal_and_vertical')**, que voltea aleatoriamente las imágenes tanto en dirección horizontal como vertical y **RandomRotation(0.2)**, que rota aleatoriamente las imágenes entre -72 y 72 grados.

Algorithm 4 Estructura del modelo secuencial para el aumento de datos.

1: **Inicializar el modelo secuencial:**
2: `model = Sequential()`
3: **Añadir una capa de flip aleatorio horizontal y vertical:**
4: `model.add(RandomFlip('horizontal_and_vertical'))`
5: **Añadir una capa de rotación aleatoria con un valor máximo de 0.2:**
6: `model.add(RandomRotation(0.2))`

Fuente: Elaboración propia.

El algoritmo 4 describe con mayor detalle el proceso de generación de imágenes según las capas definidas en el algoritmo 5. Primero, el algoritmo recibe un conjunto de datos de entrenamiento o validación. Luego, las imágenes del conjunto de datos son iteradas y se les aplican las transformaciones geométricas correspondientes. Estas transformaciones se integran a las imágenes originales y finalmente, se retorna el conjunto de datos aumentado con las transformaciones geométricas aplicadas.

Algorithm 5 Aumento de datos.

Require: Conjunto de datos: $Dataset$
Ensure: Conjunto de datos aumentado: D_{aug}
1: **for** $i \leftarrow 1$ to $DatasetDataNumber$ **do**
2: $Data1 \leftarrow \text{HorizontalVerticalFlip}(Dataset)$
3: $Data2 \leftarrow \text{RandomRotation}(Dataset, 0.2)$
4: $D_{aug} \leftarrow \text{Add}(Dataset, Data1, Data2)$
5: **end for**
6: **return** D_{aug}

Fuente: Elaboración propia.

En la siguiente figura 3.7 se presenta un ejemplo del resultado de la aplicación del aumento de datos en una imagen, donde se pueden observar las transformaciones geométricas aplicadas.

Figura 3.7: Aumento de datos.

Fuente: Elaboración propia.

3.5. Entrenamiento con transfer learning

3.5.1. Definición de modelos para el entrenamiento

En la tabla 3.3 se presentan los modelos seleccionados para el entrenamiento, definidos a partir de un análisis de antecedentes previos. En este análisis se identificaron los modelos que alcanzaron la mayor exactitud en cada caso estudiado. A continuación, se detallan los modelos con mejor desempeño, los cuales serán utilizados en nuestro proyecto de investigación.

Tabla 3.3: Tabla de modelos para el entrenamiento según antecedentes.

	Estructura	Ventajas	Selección
InceptionV3	Red de 48 capas que combina convoluciones estándar con técnicas innovadoras	Eficiencia en el uso de parámetros, reduce el tiempo de entrenamiento.	Fue seleccionado para tareas con limitaciones de recursos computacionales, donde se busca eficiencia tanto en términos de parámetros como de procesamiento. Además, porque en el trabajo de investigación (clasificación de enfermedades de la zanahoria) de Naimur Rashid Methun (2021) alcanzó una exactitud del **98 %**, lo que resulta altamente relevante y comparable con nuestro caso de estudio.
DenseNet	Capas densas (cada capa conecta con todas las anteriores)	Mayor reutilización de características, mejora el flujo de gradientes	Fue seleccionado debido a su capacidad para mejorar la eficiencia del modelo mediante la reutilización de características a lo largo de la red. Además, porque en el trabajo de investigación (clasificación de cerezas) de Kayaalp1 (2024) alcanzó una exactitud del **99.57 %**, lo que resulta altamente relevante y comparable con nuestro caso de estudio.
Resnet 50	50 capas(residuos o "skip connections").	Evita el problema de desvanecimiento de gradientes, permite redes más profundas.	Fue seleccionado debido a su capacidad para entrenar redes extremadamente profundas y precisas, especialmente en tareas de clasificación o detección avanzada. Además, porque en el trabajo de investigación (detección de enfermedades de las fresas) de Jia-Rong Xiao (2021) alcanzó una exactitud del **99.57 %**, lo que resulta altamente relevante y comparable con nuestro caso de estudio.
Xception.	Basada en separables de convolución profunda.	Xception es más eficiente en términos de parámetros y velocidad de procesamiento.	Fue seleccionado debido a su capacidad para realizar tareas avanzadas, buscando un modelo eficiente en términos de parámetros, pero con un alto rendimiento. Además, poque en el trabajo de investigación (reconocimiento de enfermedades del arroz) de Ahmad Rofiqul Muslikh (2023) alcanzó una exactitud del **90 %**, lo cual resulta altamente relevante y comparable con nuestro caso de estudio.

Fuente: Elaboración propia.

3.5.2. Definición de hiperparametros

Los hiperparámetros empleados para el entrenamiento con y sin ajuste fino de los modelos de Transfer Learning se basan en los antecedentes descritos en el capítulo 3, con la excepción de la tasa de aprendizaje, que en este trabajo de investigación se ha ajustado a un valor más bajo. A continuación, en la tabla 3.4, se detallan los hiperparámetros utilizados.

Tabla 3.4: Tabla de hiperparametros.

Hiperparametros	ResNet 50	Xception	InceptioV3	DenseNet
Epocas	20	20	20	20
Batch size	32	32	32	32
Capas adicionales	2	2	2	2
Numero neuronas de la capa 1	1024	1024	1024	1024
Numero neuronas de la capa 2	512	512	512	512
Función de activación en las capas intermedias	ReLu	ReLu	ReLu	ReLu
Tasa de aprendizaje	0.00001	0.00001	0.00001	0.00001
Optimizador	Adam	Adam	Adam	Adam
Hiperparametros con ajuste fino				
Epocas	20	20	20	20
Descongelamiento de capas(a partir de la capa)	100	100	260	650
Tasa de aprendizaje con ajuste fino	0.000001	0.00001	0.000001	0.000001
Optimizador con ajuste fino	RMSprop	RMSprop	RMSprop	RMSprop

Fuente: Elaboración propia.

3.5.3. Procedimientos para el entrenamiento con Transfer Learning

Los procedimientos que se describen a continuación explican cómo se construyeron nuevos modelos basados en modelos preentrenados de Transfer Learning, específicamente para la clasificación de imágenes de defectos de calidad en alcachofas.
A continuación, se detallan los pasos seguidos para la construcción de estos modelos:

3.5.3.1. Seleccionar una arquitectura base

En este procedimiento, se selecciona una arquitectura preentrenada disponible en diversas bibliotecas, como Keras, PyTorch, Matlab, entre otras. Estas arquitecturas, compuestas por múltiples capas, han sido entrenadas en conjuntos de datos de gran escala, como ImageNet. El algoritmo 6 detalla cómo configurar los parámetros del modelo preentrenado antes de su uso. A continuación, se presenta una lista detallada de los parámetros a configurar.

1. **Tamaño de entrada de la imagen(input_shape)**: Se define el tamaño de las imágenes que el modelo procesará, especificando la altura, el ancho y el número de canales. Para este trabajo de investigación, se considerará un tamaño de (224, 224, 3).

2. **Capas superiores(include_top)**: Se omiten las últimas capas del modelo preentrenado, ya que están altamente especializadas para la tarea original.

3. **Pesos preentrenados(weights)**: Se aprovechan los pesos preentrenados en ImageNet, lo que implica que el modelo ya ha aprendido a extraer características visuales de bajo nivel (como bordes y texturas) a partir de millones de imágenes. Al utilizar este conocimiento previo, se acelera considerablemente el proceso de entrenamiento y se optimiza el rendimiento del modelo.

4. **Congelamiento de capas(modelo_base.trainable)**: En este procedimiento, se congelan las capas del modelo preentrenado para preservar su conocimiento previo, evitando que sus pesos se modifiquen. Esto previene el sobreajuste, adapta el modelo a la nueva tarea y reduce el costo computacional.

Algorithm 6 Configuración de parámetros para un modelo preentrenado.

Definir la forma de la imagen:
  `IMG_SHAPE = (224, 224, 3)`
Cargar el modelo preentrenado:
  `Resnet 50, InceptionV3, Xception y DenseNet`
Configurar el modelo con los siguientes parámetros:
  `input_shape = IMG_SHAPE`
  `include_top = False` # No incluir la capa de salida
  `weights = 'imagenet'` # Usar los pesos preentrenados de ImageNet
Asignar el modelo cargado a la variable:
  `modelo_base`
Congelar las capas del modelo estableciendo
  `modelo_base.trainable = False`

Fuente: Elaboración propia.

3.5.3.2. Construcción de capas adicionales

En este procedimiento, se construyen e integran nuevas capas al modelo, adaptándolas específicamente a nuestro caso de estudio. El algoritmo 7 detalla paso a paso cómo se lleva a cabo la construcción e integración de las diferentes capas, finalizando con la creación del modelo final, el cual estará listo para ser entrenado.

Algorithm 7 Construcción del modelo.

Entrada: `IMG_SHAPE = (224,224,3)`, `NUM_CLASSES = 4`
Inicializar: `modelo_base` (modelo preentrenado)
Paso 1: Definir la capa de entrada con forma `IMG_SHAPE`
  `entrada = Input(shape=IMG_SHAPE)`
Paso 2: Aplicar aumento de datos
  `x = data_augmentation(entrada)`
Paso 3: Preprocesar las imágenes de entrada
  `x = preprocess_input(x)`
Paso 4: Pasar las imágenes por el modelo preentrenado sin entrenar sus pesos
  `x = modelo_base(x, training=False)`
Paso 5: Aplicar global average pooling para reducir la dimensionalidad
  `x = GlobalAveragePooling2D()(x)`
Paso 6: Aplanar las características para la capa densa
  `x = Flatten()(x)`
Paso 7: Añadir una capa densa con 1024 neuronas y activación ReLU
  `x = Dense(1024, activation='relu')(x)`
Paso 8: Aplicar dropout con una tasa del 30 % para prevenir sobreajuste
  `x = Dropout(0.3)(x)`
Paso 9: Añadir una capa densa con 512 neuronas y activación ReLU
  `x = Dense(512, activation='relu')(x)`
Paso 10: Aplicar dropout con una tasa del 30 % para prevenir sobreajuste
  `x = Dropout(0.3)(x)`
Paso 11: Añadir una capa de salida con `NUM_CLASSES` neuronas
  `salida = Dense(NUM_CLASSES)(x)`
Paso 12: Crear el modelo final
  `modelo = Model(entrada, salida)`
Retornar: `modelo`

Fuente: Elaboración propia.

3.5.3.3. Entrenamiento el modelo

En este procedimiento, se lleva a cabo el primer entrenamiento utilizando los hiperparámetros establecidos en la tabla 3.4, sin realizar ajuste fino. El algoritmo 8 describe la estructura de compilación y el proceso de entrenamiento del modelo. Los parámetros de entrada considerados son: la tasa de aprendizaje, conjunto de entrenamiento y conjunto de validación. Al finalizar, el entrenamiento devuelve el historial de ejecución, que incluye las métricas y los valores de la función de pérdida a lo largo de las épocas.

Algorithm 8 Compilación y entrenamiento del modelo sin ajuste fino.

Entrada: `train_dataset`, `validation_dataset`, `inicial_epochs = 20`, `tasa_aprendizaje_base = 0.00001`
Inicializar: `modelo` (modelo previamente definido)
Paso 1: Compilar el modelo con el optimizador Adam, función de pérdida, tasa de aprendizaje y métricas

```
modelo.compilar(optimizador=Adam(tasa_aprendizaje=tasa_aprendizaje_base),
  pérdida = SparseCategoricalCrossentropy,
  metrica = accuracy)
```

Paso 2: Entrenar el modelo con los datos de entrenamiento y validación

```
history = modelo.entrenar(train_dataset,
  epocas=inicial_epochs,
  validation_data=validation_dataset)
```

Retornar `history` (historial del entrenamiento)

Fuente: Elaboración propia.

3.5.3.4. Entrenamiento del modelo con ajuste fino

En este procedimiento, el modelo previamente entrenado sin ajuste fino se reentrena, descongelando únicamente las capas superiores, a partir de la capa 100. El optimizador utilizado es RMSprop, tasa de aprendizaje se reduce en un factor de 10 y la función de pérdida se mantiene constante. Para este entrenamiento, se emplean los hiperparámetros establecidos en la tabla 3.4, específicamente diseñados para el ajuste fino. A continuación, en el algoritmo 9, se detalla el procedimiento paso a paso para el entrenamiento con ajuste fino.

Algorithm 9 Compilación y entrenamiento del modelo con ajuste fino.

Entrada: `train_dataset`, `validation_dataset`, `total_epochs = 20 + inicial_epochs`, `tasa_aprendizaje_base = 0.000001`, `fine_tune_epochs = 20`, `nroCapas = (100:Resnet50, 100:Xception, 260:Inception, 650: DenseNet)`
Paso 1: Descongelar todas las capas del modelo

```
modelo_base.trainable = True
```

Paso 2: Definir desde que capa se descongelara

```
fine_tune_at = nroCapas
```

Paso 3: Congelar las capas iniciales dejando las capas superiores descongeladas
for layer in modelo_base.layers[0: fine_tune_at] **do**

```
layer.trainable = False
```

end for
Paso 4: Compilar el modelo con el optimizador RMSprop, función de pérdida, tasa de aprendizaje divido entre 10 y métricas

```
modelo.compilar(pérdida = SparseCategoricalCrossentropy,
optimizador = RMSprop(tasa_aprendizaje_base/10), metrica = accuracy)
```

Paso 5: Entrenar el modelo con los datos de entrenamiento y validación desde la epoca 21.

```
history_fine =  modelo.entrenar(train_dataset, epocas=total_epochs,
initial_epoch=history.epoch[-1], validation_data=validation_dataset)
```

Retornar `history_fine` (historial del entrenamiento)

Fuente: Elaboración propia.

3.5.3.5. Guardar modelo entrenado en formato tflite

Una vez finalizado el entrenamiento y la evaluación del modelo, el modelo con mayor precisión se guarda en un formato comprimido que conserva la precisión alcanzada durante el entrenamiento. El archivo generado está diseñado para su integración en aplicaciones móviles y es esencial para la implementación de la aplicación destinada al diagnóstico de defectos de calidad en alcachofas. En el algoritmo 10 se describe el procedimiento para generar y almacenar el modelo en formato TFLITE.

Algorithm 10 Guardar y convertir el modelo a formato TFLITE

1: **Entrada:** `modelo_entrenado`
2: **Paso 1:** Definir directorio de guardado de modelo H5
3: `file_model = "../resnet50.h5"`
4: **Paso 2:** Cargar el modelo en formato H5
5: `load_model(modelo_entrenado, file_model)`
6: **Paso 3:** Convertir el modelo a TFLITE:
7: `converter = TFLiteConverter(modelo_entrenado)`

3.6. Implementación de la aplicación móvil

Para facilitar su uso en el campo, el sistema de visión artificial se implementará como una aplicación móvil multiplataforma desarrollada con Flutter. Esta aplicación utilizará un modelo entrenado de alta precisión, optimizado en formato TFLITE (ver sección 3.5.3.5), lo que permitirá realizar la clasificacion en tiempo real sobre las imágenes capturadas por la cámara del dispositivo móvil. Durante el proceso, se mostrará el porcentaje de precisión, tal como se ilustra en la figura 3.8.
Al finalizar el diagnóstico, la aplicación generará un resumen con recomendaciones personalizadas e información técnica relevante, adaptadas al tamaño del cultivo, como se muestra en la figura 3.9. Gracias a la portabilidad de Flutter y la eficiencia de TFLITE, la aplicación funcionará de manera fluida en dispositivos Android e iOS, ofreciendo una solución versátil, escalable y eficiente para la inspección y diagnóstico en el cultivo de alcachofas.

3.6.1. Información técnica del diagnóstico

El sistema está diseñado para generar recomendaciones agrícolas personalizadas a través del análisis de imágenes de alcachofas. Este diagnóstico se basa, generalmente, en el análisis de un promedio de **75 imágenes por hectárea de cultivo**. Nuestro sistema tiene la capacidad de adaptarse a diferentes tamaños de terreno, ajustando las dosis de agroquímicos de acuerdo con las dimensiones específicas de cada área cultivada. A partir del diagnóstico, se determinarán las dosis óptimas de agroquímicos necesarias. En la tabla 2 se especifican los agroquímicos recomendados según los distintos tipos de defectos de calidad, con las dosis indicadas para una hectárea de cultivo. Sin embargo, estas dosis serán ajustadas en función del tamaño real del terreno analizado.

Tabla 3.5: Tabla de información técnica.

	Primera	Fofa	Cintura	Violácea
ACTIFER POTASIO	1000 ML en 200 LITROS de agua por hectárea			
ACTIFER FOSFORO	1000 ML en 200 LITROS de agua por hectárea			
NITRATO DE CALCIO	260 KG por hectárea			
ALGA SEALAND		1000 ML en 200 LITROS de agua por hectárea		
CALCIO/BORO SEALAND		1000 ML en 200 LITROS de agua por hectárea		
SEALAND COBRE		1000 ML en 200 LITROS de agua por hectárea		
ROOTY SEALAND			1000 ML en 200 LITROS de agua por hectárea	
SULFA 87 LS			1000 ML en 200 LITROS de agua por hectárea	
BENLAFAR				200 GR en 200 LITROS de agua por hectárea
DOGMA				200 ML en 200 LITROS de agua por hectárea
CORAGEN				200 ML en 200 LITROS de agua por hectárea

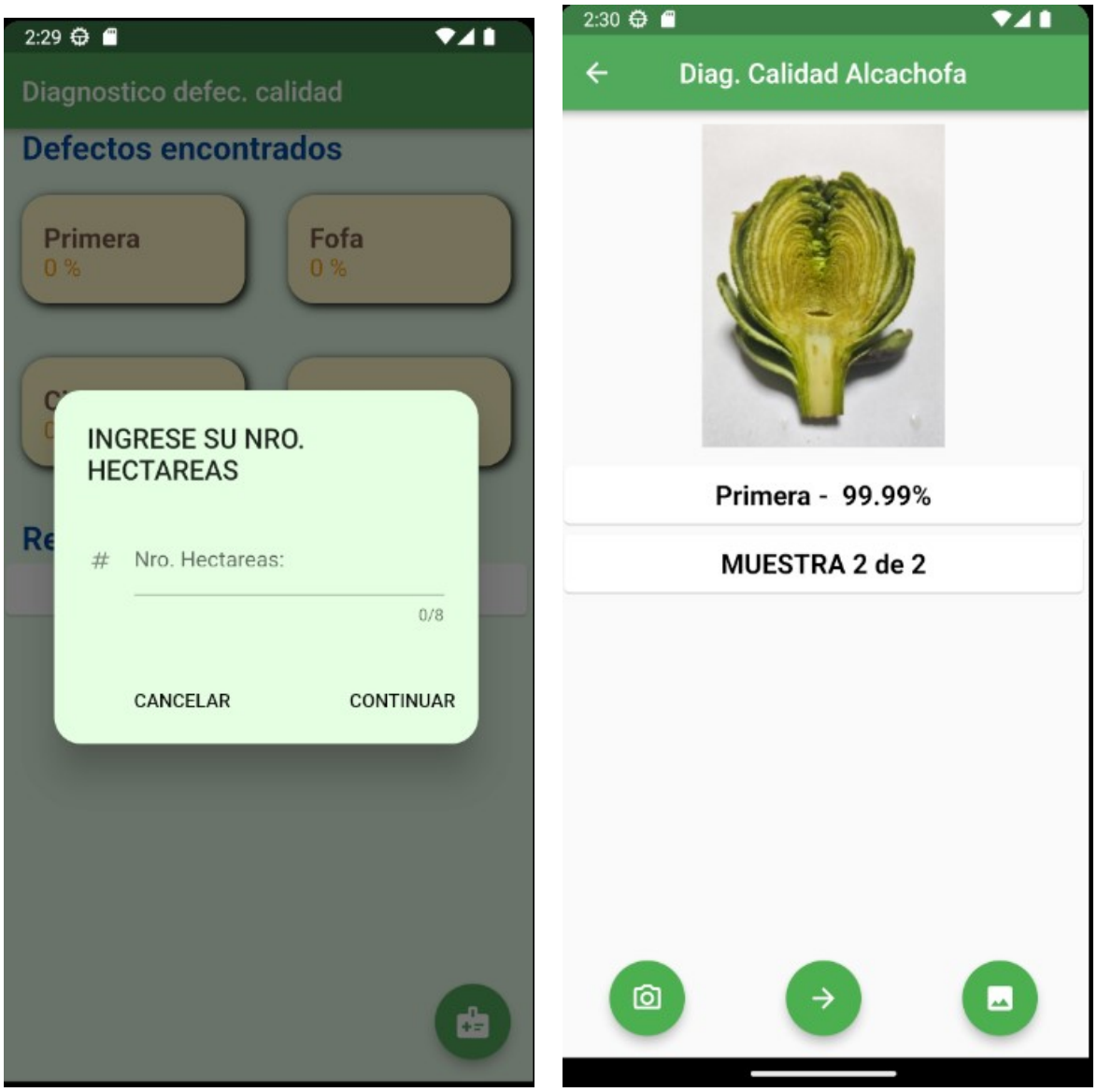
3.6.2. Funcionamiento del sistema de visión artificial

El funcionamiento del sistema de visión artificial, implementado como una aplicación móvil, se divide en dos etapas principales: la captura de imágenes y el diagnóstico final.

3.6.2.1. Toma y clasificación de imágenes

En esta etapa, las imágenes de las alcachofas se capturan utilizando la galería o la cámara en tiempo real del dispositivo móvil. Antes de iniciar, se debe ingresar el tamaño del cultivo, lo que permitirá calcular el número adecuado de muestras en función de las dimensiones del terreno.Para garantizar una óptima iluminación y contraste, las fotografías deben tomarse sobre un fondo blanco, lo que facilita significativamente la clasificación de los defectos de calidad. La figura 3.8 ilustra el proceso de captura de imágenes y la clasificación de los defectos en las alcachofas.

Figura 3.8: Toma y clasificación de imágenes en la aplicación móvil..



(a) Ingreso nro. hectáreas. (b) Comienzo de clasificación.

Fuente: Elaboración propia.

3.6.2.2. Diagnóstico y información técnica

En esta etapa, se genera un informe detallado y un diagnóstico basado en los resultados obtenidos de la clasificación de las imágenes. El informe incluye información técnica derivada de los datos presentados en la tabla 2, ajustando las dosis recomendadas de agroquímicos según el tamaño del cultivo. Esto proporciona al agricultor información técnica relevante y personalizada. La figura 3.9 ilustra el proceso de generación del informe y el diagnóstico de los defectos de calidad en las alcachofas.

Figura 3.9: Diagnóstico y información técnica.

Fuente: Elaboración propia.

3.7. Entorno de entrenamiento y pruebas

- **Hardware para el entrenamiento**:
  Estación de trabajo: ASUS TUF GAMMING FX504
  Procesador: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz
  Memoria RAM: 12.0 GB
  Sistema Operativo: Windows 11 de 64 bits
  Video: Nvidia Getforce 1050 de 4 GB.

- **Software para el entrenamiento**:
  Lenguaje de programación: Python 3.7.3
  Framework: Tensorflow, Keras
  Plataforma: Jupyter, Visual Studio Code

- **Hardware para pruebas**:
  Estación de trabajo: : SAMSUNG A20
  Procesador: Exynos 7884 de ocho núcleos
  Memoria RAM: 4.0 GB
  Sistema Operativo: Android

- **Software para pruebas**:
  Lenguaje de programación: Dart
  Framework: Flutter
  Plataforma: Visual Studio Code

Capítulo 4

Resultados

En esta sección se analizarán y discutirán los resultados obtenidos del entrenamiento de los modelos preentrenados Xception, InceptionV3, DenseNet y ResNet50, seleccionados en función de los antecedentes descritos y su destacado rendimiento en la clasificación de imágenes. Estos modelos se emplearán para el proceso de entrenamiento, utilizando los hiperparámetros detallados en la tabla 3.4.

4.1. Evaluación del modelo Xception

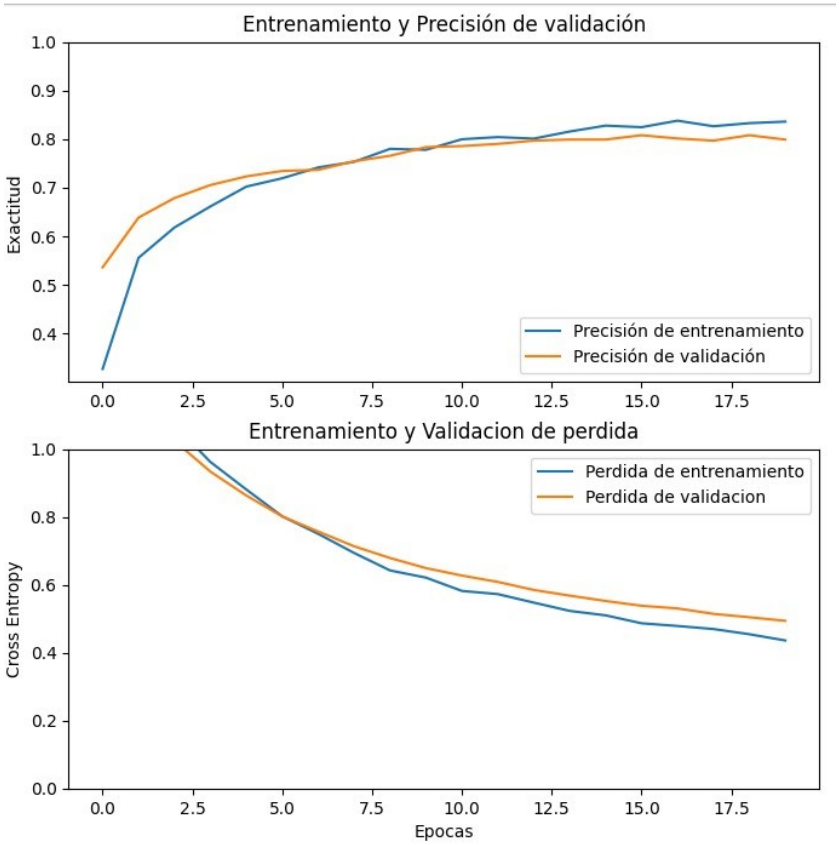
4.1.0.1. Resultados del entrenamiento

1. **Entrenamiento del modelo Xception sin ajuste fino**.- En la siguiente figura 4.1 se presentan los resultados de exactitud y pérdida del modelo Xception entrenado sin ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** Inicialmente, la exactitud en el conjunto de entrenamiento es de 0.32, mientras que en el conjunto de validación es de 0.53. A partir de la iteración 8, ambas métricas convergen hacia un valor cercano a 0.75, estabilizándose hasta la iteración 20. Al finalizar la fase de evaluación, el modelo alcanza una exactitud final del 85.26 %.

   b) **Gráfica de pérdida (parte inferior):** Inicialmente, la función de pérdida en el conjunto de entrenamiento es de 1.35, mientras que en el conjunto de validación es de 1.25. A lo largo del entrenamiento, ambas métricas presentan una disminución progresiva. Sin embargo, a partir de la iteración 7, las curvas de pérdida convergieron y se estabilizaron, manteniendo una tendencia decreciente.

Figura 4.1: Gráfica de perdida y exactitud sin ajuste fino del modelo entrenado Xception.



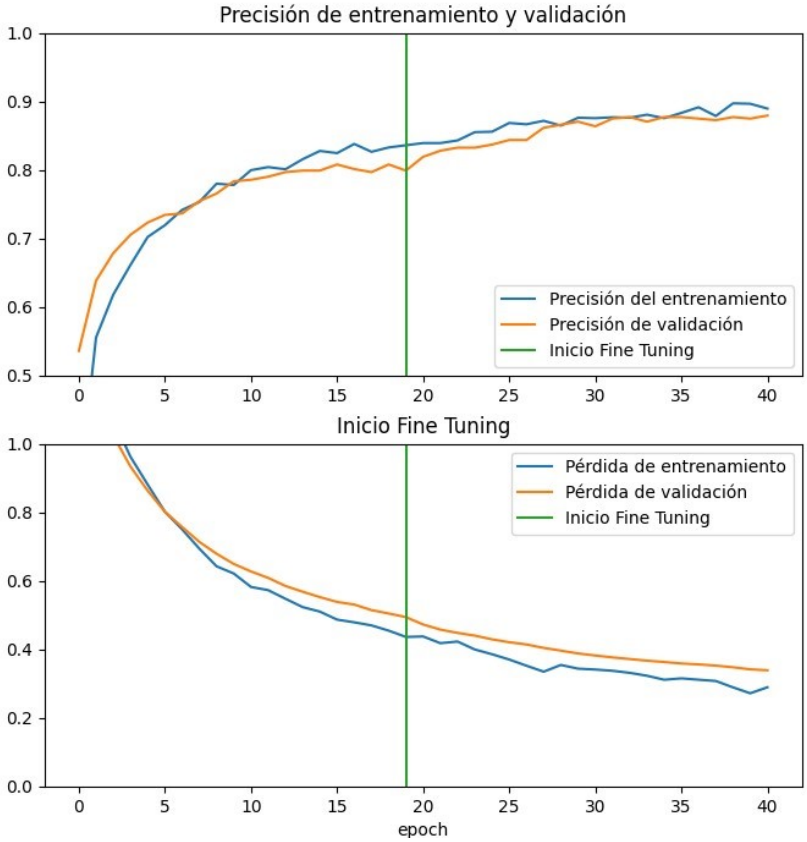
Fuente: Elaboración propia.

2. **Entrenamiento del modelo Xception con ajuste fino**.- En la siguiente figura 4.2 se presentan los resultados de exactitud y pérdida del modelo Xception entrenado con ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** A partir de la iteración 21 (línea verde) se realizó ajuste fino al modelo, reduciendo la tasa de aprendizaje y descongelando las capas superiores. El reentrenamiento inició con valores iniciales de exactitud de 0.83 en el conjunto de entrenamiento y 0.81 en el conjunto de validación. Se observó una tendencia más ascendente en ambas métricas y estabilizándose hasta la iteración 40. En la fase de testeo, el modelo alcanzó una exactitud final del 87.5 %.

   b) **Gráfica de pérdida (parte inferior):** Posterior al ajuste fino, se reinició el entrenamiento. La pérdida inicial fue de 0.43 en el conjunto de entrenamiento y de 0.47 en el conjunto de validación. Durante el reentrenamiento, ambas métricas presentaron una disminución más progresiva, tendiendo a cero.

Figura 4.2: Gráfica de perdida y exactitud con ajuste fino del modelo entrenado Xception.



Fuente: Elaboración propia.

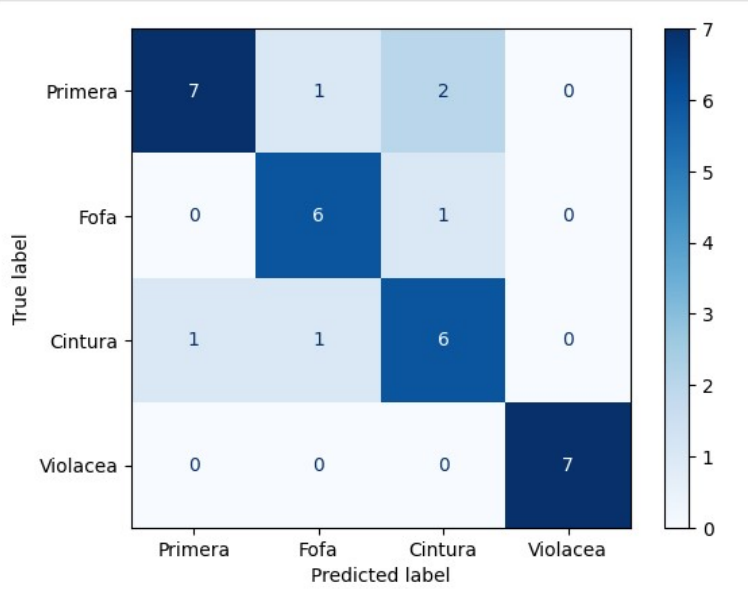
4.1.0.2. Matriz de confusión

Figura 4.3: Métricas del modelo entrenado Xception.

	precision	recall	f1-score	support
Primera	0.88	0.70	0.78	10
Fofa	0.75	0.86	0.80	7
Cintura	0.67	0.75	0.71	8
Violacea	1.00	1.00	1.00	7

Fuente: Elaboración propia.

Figura 4.4: Matriz de confusión del modelo entrenado Xception.



Fuente: Elaboración propia.

En la figura 4.3 se presentan las métricas del modelo, que demuestran una excelente capacidad para clasificar los defectos de calidad fofa y violácea (F1-Score de 0.80 y 1.00, respectivamente). Sin embargo, se observó un ligero descenso en la precisión al clasificar los defectos cintura y primera (F1-Score de 0.71 y 0.78, respectivamente). La matriz de confusión (figura 4.4) respalda esta tendencia, detallando lo siguiente:

1. **Columna primera**: De las muestras predichas como *primera*, 7 fueron clasificadas correctamente, mientras que 1 fue clasificada incorrectamente como *cintura*.

2. **Columna fofa:**: De las muestras predichas como *fofa*, 6 fueron clasificadas correctamente, 1 fue erróneamente clasificada como *cintura*, 1 fue erróneamente clasificada como *primera*.

3. **Columna cintura**: De las muestras predichas como *cintura*, 6 fueron clasificadas correctamente, 2 fue clasificada incorrectamente como *primera* y 1 como *fofa*.

4. **Columna violácea**: Todas las muestras predichas como *violácea* fueron clasificadas correctamente.

4.2. Evaluación del modelo InceptionV3

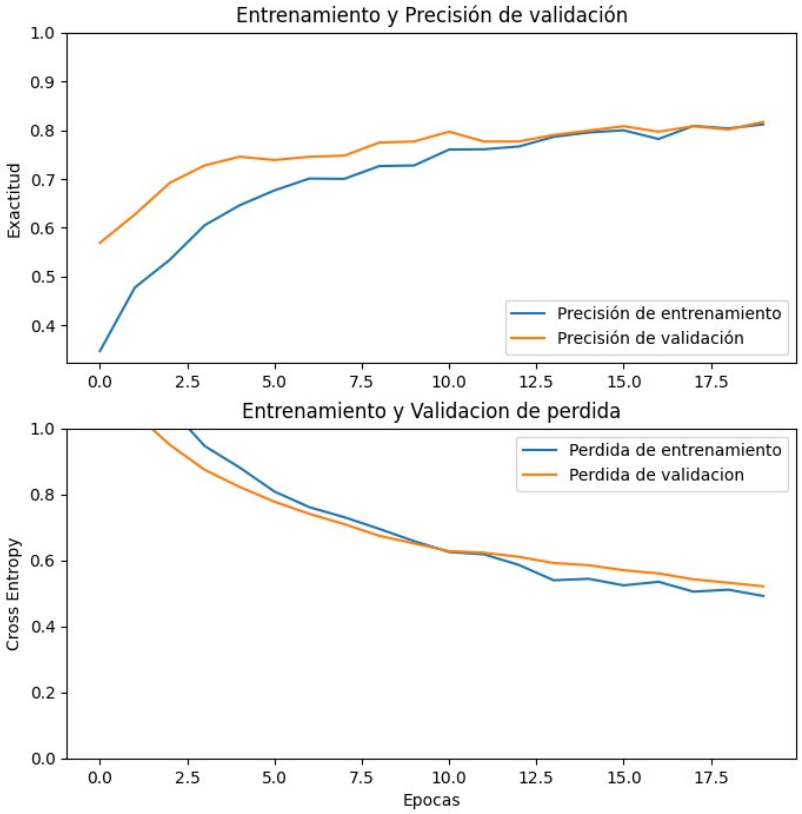
4.2.0.1. Resultados del entrenamiento

1. **Entrenamiento del modelo InceptionV3 sin ajuste fino**.- En la siguiente figura 4.5 se presentan los resultados de exactitud y pérdida del modelo InceptionV3 entrenado sin ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** Inicialmente, la exactitud en el conjunto de entrenamiento es de 0.34, mientras que en el conjunto de validación es de 0.56. A partir de la iteración 16, ambas métricas convergen hacia un valor cercano a 0.79, estabilizándose hasta la iteración 20. Al finalizar la fase de evaluación, el modelo alcanza una exactitud final del 83.92 %.

   b) **Gráfica de pérdida (parte inferior):** Inicialmente, la función de pérdida en el conjunto de entrenamiento es de 1.39, mientras que en el conjunto de validación es de 1.17. A lo largo del entrenamiento, ambas métricas presentan una disminución progresiva. Sin embargo, a partir de la iteración 15, las curvas de pérdida convergieron y se estabilizaron, manteniendo una tendencia decreciente.

Figura 4.5: Gráfica de perdida y exactitud sin ajuste fino del modelo entrenado InceptionV3.



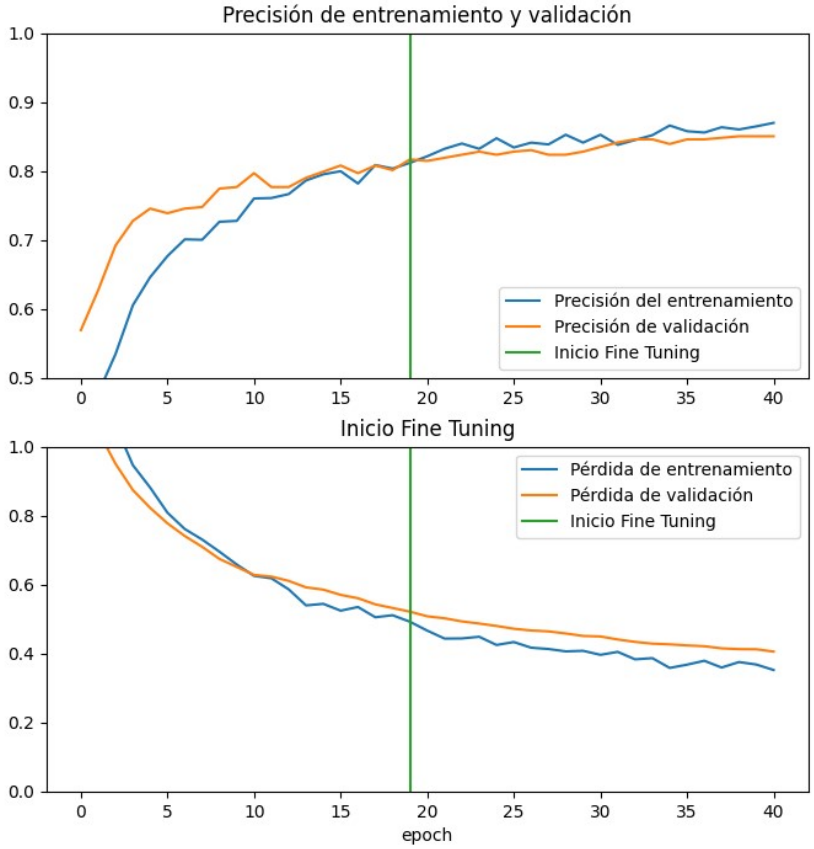
Fuente: Elaboración propia.

2. **Entrenamiento del modelo InceptionV3 con ajuste fino**.- En la siguiente figura 4.6 se presentan los resultados de exactitud y pérdida del modelo InceptionV3 entrenado con ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** A partir de la iteración 21 (línea verde) se realizó ajuste fino al modelo, reduciendo la tasa de aprendizaje y descongelando las capas superiores. El reentrenamiento inició con valores iniciales de exactitud de 0.82 en el conjunto de entrenamiento y 0.81 en el conjunto de validación. Se observó una tendencia más ascendente en ambas métricas y estabilizándose hasta la iteración 40. En la fase de testeo, el modelo alcanzó una exactitud final del 85.26 %.

   b) **Gráfica de pérdida (parte inferior):** Posterior al ajuste fino, se reinició el entrenamiento. La pérdida inicial fue de 0.46 en el conjunto de entrenamiento y de 0.50 en el conjunto de validación. Durante el reentrenamiento, ambas métricas presentaron una disminución más progresiva, tendiendo a cero.

Figura 4.6: Gráfica de perdida y exactitud con ajuste fino del modelo entrenado InceptionV3.



Fuente: Elaboración propia.

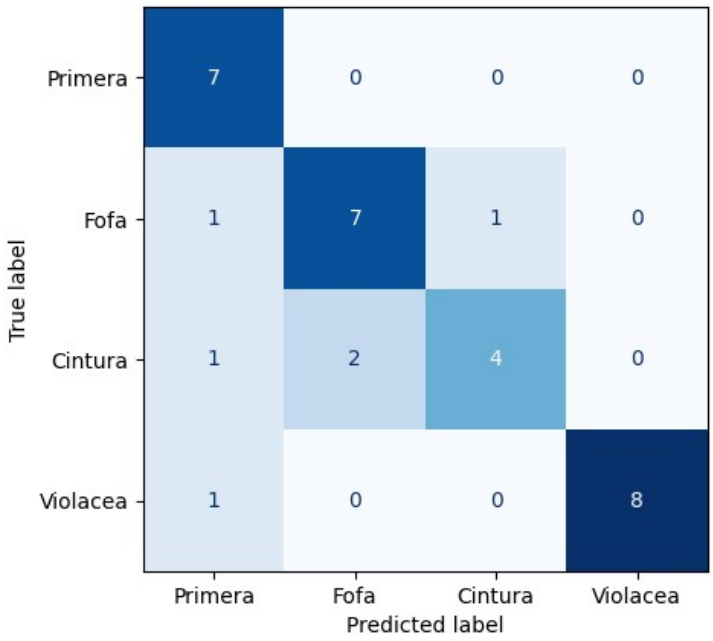
4.2.0.2. Matriz de confusión

Figura 4.7: Métricas del modelo entrenado InceptionV3.

	precision	recall	f1-score	support
Primera	0.70	1.00	0.82	7
Fofa	0.78	0.78	0.78	9
Cintura	0.80	0.57	0.67	7
Violacea	1.00	0.89	0.94	9
accuracy			0.81	32
macro avg	0.82	0.81	0.80	32
weighted avg	0.83	0.81	0.81	32

Fuente: Elaboración propia.

Figura 4.8: Matriz de confusión del modelo entrenado InceptionV3.



Fuente: Elaboración propia.

En la figura 4.7 se presentan las métricas del modelo, que demuestran una excelente capacidad para clasificar los defectos de calidad violácea y primera (F1-Score de 0.94 y 0.82, respectivamente). Sin embargo, se observó un ligero descenso en la exactitud al clasificar los defectos cintura y fofa (F1-Score de 0.67 y 0.78, respectivamente). La matriz de confusión (figura 4.8) confirma esta tendencia, mostrando lo siguiente:

1. **Columna cintura:** De las muestras predichas como *cintura*, 4 fueron clasificadas correctamente y 1 fueron clasificadas incorrectamente como *fofa*.
2. **Columna fofa:** De las muestras predichas como *fofa*, 7 fueron clasificadas correctamente y 2 fueron clasificadas incorrectamente como *cintura*.
3. **Columna primera:** De las muestras predichas como *primera*, 7 fueron clasificadas correctamente, 1 fueron clasificadas incorrectamente como *fofa*, 1 fueron clasificadas incorrectamente como *cintura* y 1 fueron clasificadas incorrectamente como *violácea*,

4. **Columna violácea:** Todas las muestras predichas como *violácea* fueron clasificadas correctamente.

4.3. Evaluación del modelo Resnet 50

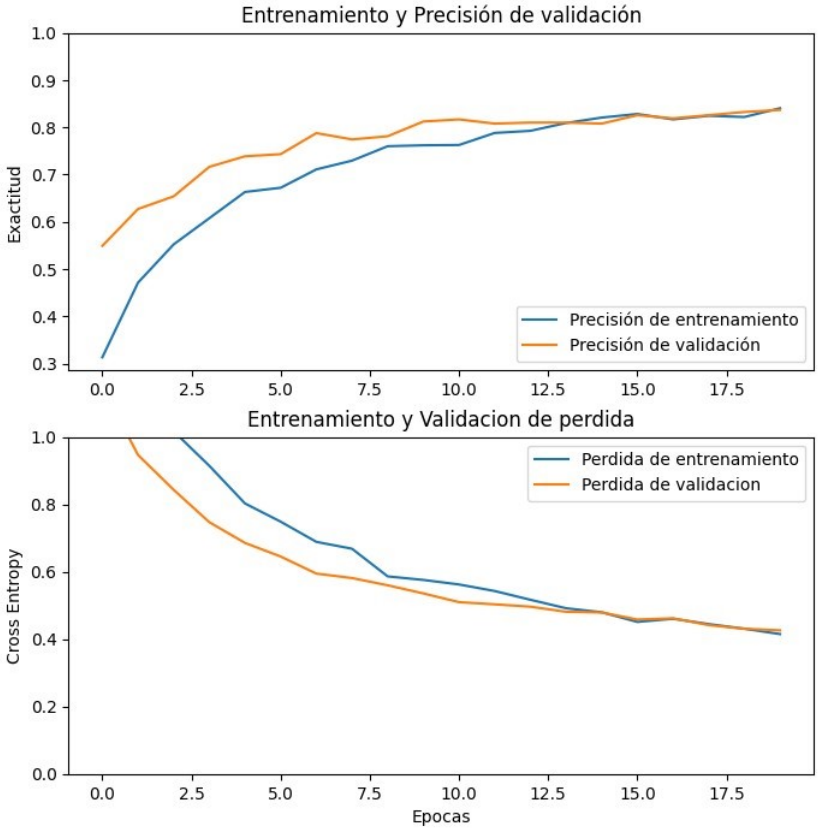
4.3.0.1. Resultados del entrenamiento

1. **Entrenamiento del modelo Resnet 50 sin ajuste fino**.- En la siguiente figura 4.9 se presentan los resultados de exactitud y pérdida del modelo ResNet 50 entrenado sin ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** Inicialmente, la exactitud en el conjunto de entrenamiento es de 0.31, mientras que en el conjunto de validación es de 0.54. A partir de la iteración 14, ambas métricas convergen hacia un valor cercano a 0.8, estabilizándose hasta la iteración 20. Al finalizar la fase de evaluación, el modelo alcanza una exactitud final del 88.83 %.

   b) **Gráfica de pérdida (parte inferior):** Inicialmente, la función de pérdida en el conjunto de entrenamiento es de 1.51, mientras que en el conjunto de validación es de 1.15. A lo largo del entrenamiento, ambas métricas presentan una disminución progresiva. Sin embargo, a partir de la iteración 16, las curvas de pérdida convergieron y se estabilizaron, manteniendo una tendencia decreciente.

Figura 4.9: Gráfica de perdida y exactitud sin ajuste fino del modelo entrenado Resnet 50.



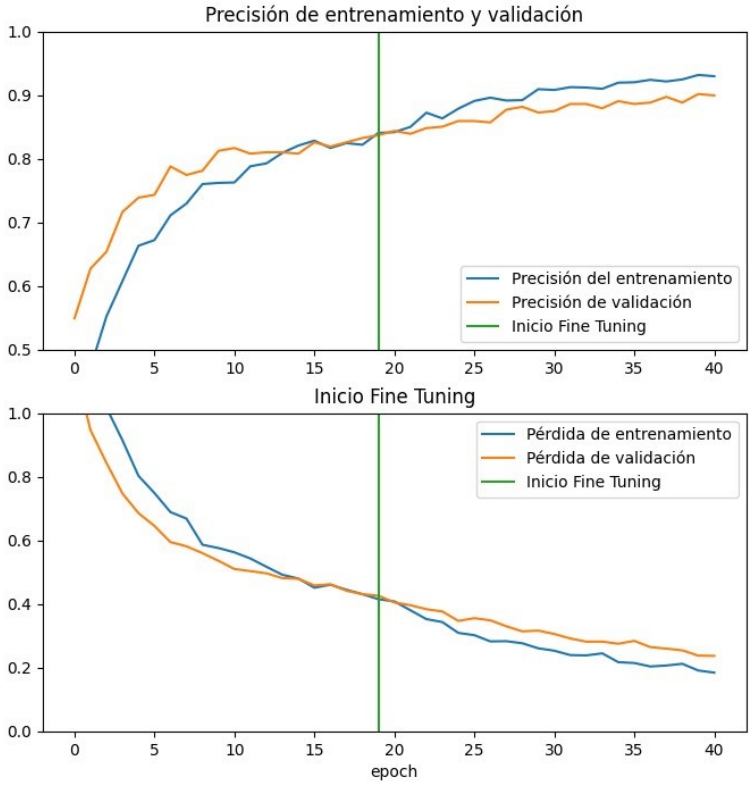
Fuente: Elaboración propia.

2. **Entrenamiento del modelo Resnet 50 con ajuste fino**.- En la siguiente figura 4.10 se presentan los resultados de exactitud y pérdida del modelo ResNet 50 entrenado con ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** A partir de la iteración 21 (línea verde) se realizó ajuste fino al modelo, reduciendo la tasa de aprendizaje y descongelando las capas superiores. El reentrenamiento inició con valores iniciales de exactitud de 0.84 en el conjunto de entrenamiento y 0.84 en el conjunto de validación. Se observó una tendencia más ascendente en ambas métricas y estabilizándose hasta la iteración 40. En la fase de testeo, el modelo alcanzó una exactitud final del 93.75 %.

   b) **Gráfica de pérdida (parte inferior):** Posterior al ajuste fino, se reinició el entrenamiento. La pérdida inicial fue de 0.40 en el conjunto de entrenamiento y de 0.40 en el conjunto de validación. Durante el reentrenamiento, ambas métricas presentaron una disminución más progresiva, tendiendo a cero.

Figura 4.10: Gráfica de perdida y exactitud con ajuste fino del modelo entrenado Resnet 50.



Fuente: Elaboración propia.

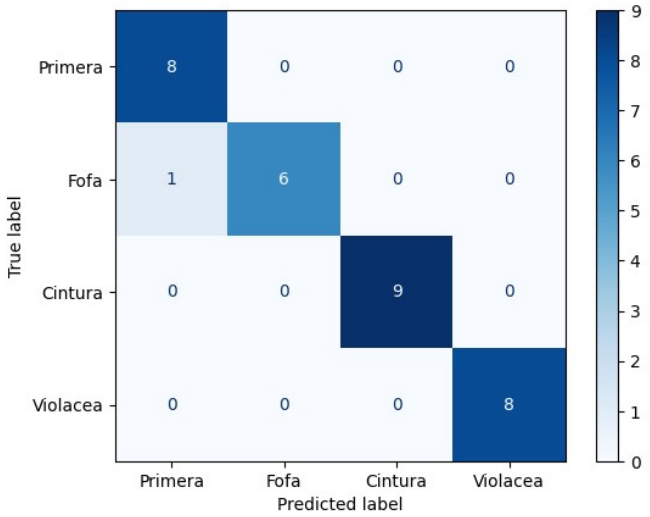
4.3.0.2. Matriz de confusión

Figura 4.11: Métricas del modelo entrenado Resnet 50.

	precision	recall	f1-score	support
Primera	0.89	1.00	0.94	8
Fofa	1.00	0.86	0.92	7
Cintura	1.00	1.00	1.00	9
Violacea	1.00	1.00	1.00	8
accuracy			0.97	32
macro avg	0.97	0.96	0.97	32
weighted avg	0.97	0.97	0.97	32

Fuente: Elaboración propia.

Figura 4.12: Matriz de confusión del modelo entrenado Resnet 50.



Fuente: Elaboración propia.

En la figura 4.11 se presentan las métricas del modelo, que demuestran una excelente capacidad para clasificar los defectos de calidad cintura y violácea, alcanzando un F1-Score perfecto de 1.00. Sin embargo, se observó un ligero descenso en la precisión al clasificar los defectos primera y fofa (F1-Score de 0.94 y 0.92, respectivamente). La matriz de confusión (4.12) confirma esta tendencia, mostrando lo siguiente:

1. **Columna primera:** De las muestras predichas como *fofa*, 8 fueron clasificadas correctamente y 1 fue clasificada incorrectamente como *fofa*.

2. **Columna fofa:** De las muestras predichas como *fofa*, todas fueron clasificadas correctamente.

3. **Columna cintura:** De las muestras predichas como *cintura*, todas fueron clasificadas correctamente.

4. **Columna violácea:** Todas las muestras predichas como *violácea* fueron clasificadas correctamente.

4.4. Evaluación del modelo DenseNet

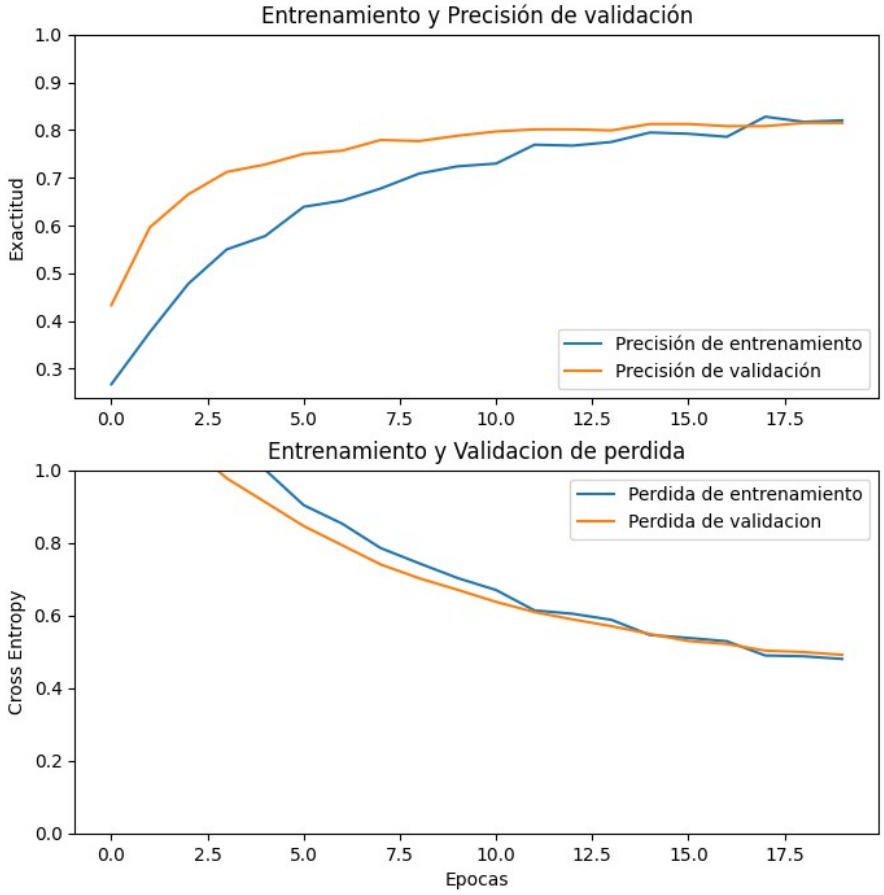
4.4.0.1. Resultados del entrenamiento

1. **Entrenamiento del modelo DenseNet sin ajuste fino**.- En la siguiente figura 4.13 se presentan los resultados de exactitud y pérdida del modelo InceptionV3 entrenado sin ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** Inicialmente, la exactitud en el conjunto de entrenamiento es de 0.26, mientras que en el conjunto de validación es de 0.43. A partir de la iteración 19, ambas métricas convergen hacia un valor cercano a 0.81, estabilizándose hasta la iteración 20. Al finalizar la fase de evaluación, el modelo alcanza una exactitud final del 85.26 %.

   b) **Gráfica de pérdida (parte inferior):** Inicialmente, la función de pérdida en el conjunto de entrenamiento es de 1.56, mientras que en el conjunto de validación es de 1.30. A lo largo del entrenamiento, ambas métricas presentan una disminución progresiva. Sin embargo, a partir de la iteración 15, las curvas de pérdida convergieron y se estabilizaron, manteniendo una tendencia decreciente.

Figura 4.13: Gráfica de perdida y exactitud sin ajuste fino del modelo entrenado DenseNet.



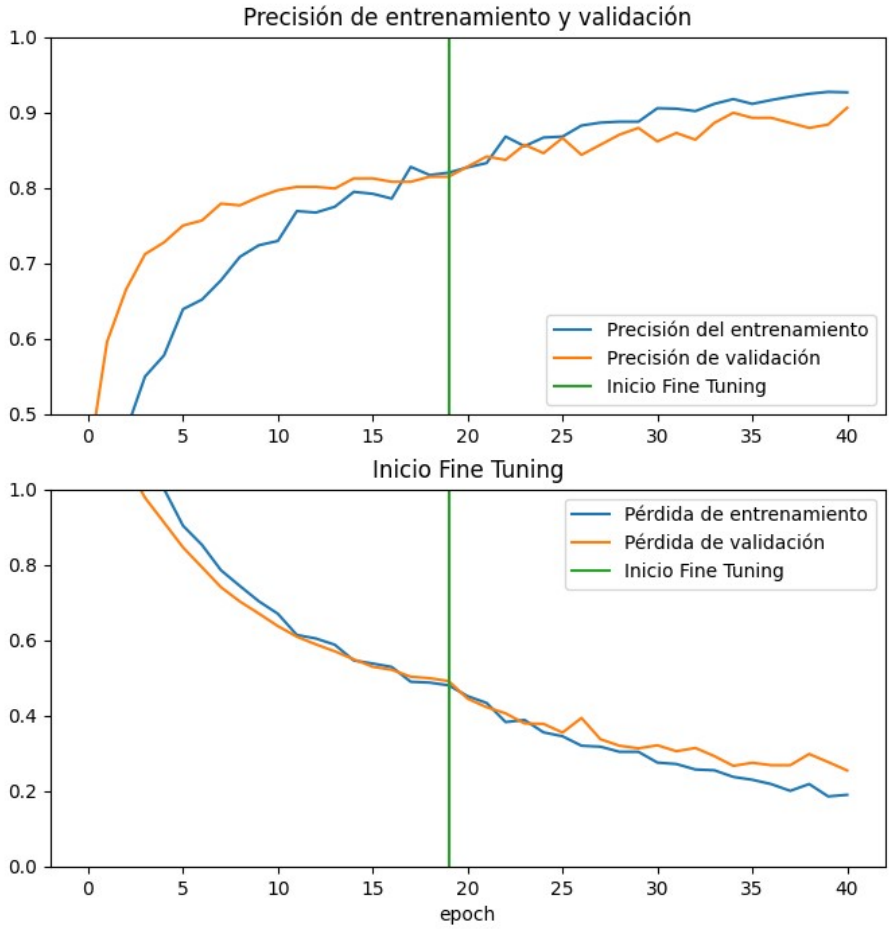
Fuente: Elaboración propia.

2. **Entrenamiento del modelo DensetNet con ajuste fino**.- En la siguiente figura 4.14 se presentan los resultados de exactitud y pérdida del modelo DenseNet entrenado con ajuste fino, distribuidos en gráficos en la parte superior e inferior respectivamente.

   a) **Gráfica de exactitud (parte superior):** A partir de la iteración 21 (línea verde) se realizó ajuste fino al modelo, reduciendo la tasa de aprendizaje y descongelando las capas superiores. El reentrenamiento inició con valores iniciales de exactitud de 0.82 en el conjunto de entrenamiento y 0.82 en el conjunto de validación. Se observó una tendencia más ascendente en ambas métricas y estabilizándose hasta la iteración 40. En la fase de testeo, el modelo alcanzó una exactitud final del 86.6 %.

   b) **Gráfica de pérdida (parte inferior):** Posterior al ajuste fino, se reinició el entrenamiento. La pérdida inicial fue de 0.45 en el conjunto de entrenamiento y de 0.44 en el conjunto de validación. Durante el reentrenamiento, ambas métricas presentaron una disminución más progresiva, tendiendo a cero.

Figura 4.14: Gráfica de perdida y exactitud con ajuste fino del modelo entrenado DenseNet.



Fuente: Elaboración propia.

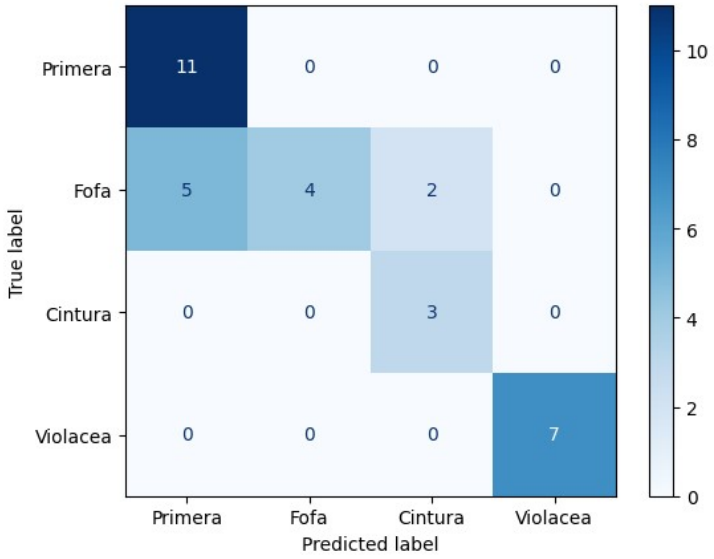
4.4.0.2. Matriz de confusión

Figura 4.15: Métricas del modelo entrenado DenseNet.

	precision	recall	f1-score	support
Primera	0.69	1.00	0.81	11
Fofa	1.00	0.36	0.53	11
Cintura	0.60	1.00	0.75	3
Violacea	1.00	1.00	1.00	7
accuracy			0.78	32
macro avg	0.82	0.84	0.77	32
weighted avg	0.86	0.78	0.75	32

Fuente: Elaboración propia.

Figura 4.16: Matriz de confusión del modelo entrenado DenseNet.



Fuente: Elaboración propia.

En la figura 4.15 se presentan las métricas del modelo, que demuestran una excelente capacidad para clasificar los defectos de calidad violácea y primera (F1-Score de 1.00 y 0.81, respectivamente). Sin embargo, se observó un ligero descenso en la precisión al clasificar los defectos cintura y fofa (F1-Score de 0.75 y 0.53, respectivamente). La matriz de confusión (figura 4.16) confirma esta tendencia, mostrando lo siguiente:

1. **Columna cintura:** De las muestras predichas como *cintura*, 3 fueron clasificadas correctamente y 2 fueron clasificadas incorrectamente como *Fofa*.
2. **Columna fofa:** De las muestras predichas como *fofa*, 4 fueron clasificadas correctamente.
3. **Columna primera:** De las muestras predichas como *fofa*, 11 fueron clasificadas correctamente y 5 fueron clasificadas incorrectamente como *Fofa*.
4. **Columna violácea:** Todas las muestras predichas como *violácea* fueron clasificadas correctamente.

4.5. Comparación de resultados de los modelos

La Tabla 4.1 presenta los resultados de exactitud de los modelos entrenados, ordenados de mayor a menor. Destaca el modelo ResNet 50, que obtuvo la mayor exactitud en la clasificación de defectos de calidad en las alcachofas.

Tabla 4.1: Tabla de comparación de resultados del entrenamiento de modelos.

Modelos	**Exatitud**
Resnet 50	**93.75 %**
Xception	87.5 %
DenseNet	86.60 %
InceptionV3	85.26 %

Fuente: Elaboración propia.

Al analizar la tabla 4.2 con los puntajes F1 de cada modelo entrenado, se observa un desempeño variable según la categoría. Aunque ResNet50 es el modelo con el mejor rendimiento en general, la efectividad de los modelos varía significativamente según el tipo de defecto. El defecto de calidad *violácea* es el más fácil de clasificar, ya que presenta un puntaje F1 alto en todos los modelos. En cambio, las categorías *primera*, *cintura* y *fofa* presentan mayores desafíos en su clasificación, con puntajes F1 ligeramente bajos, especialmente para InceptionV3, DenseNet y Xception. Sin embargo, en el caso de ResNet 50, aunque se observa una ligera disminución en el puntaje F1 en estas categorías, el nivel de precisión sigue siendo alto.

Tabla 4.2: Tabla score F1 de los modelos entrenados.

Categorias		**Score F1**			
		InceptionV3	**DenseNet**	**Xception**	**Resnet 50**
0	**Primera**	0.78	0.81	0.78	**0.94**
1	**Fofa**	0.80	0.53	0.80	**0.92**
2	**Cintura**	0.71	0.75	0.71	**1.00**
3	**Violacea**	1.00	0.82	1.00	**1.00**

Fuente: Elaboración propia.

En conclusión, tras analizar la tabla 4.2, los puntajes F1 ofrecen una visión general del desempeño de los diferentes modelos en la tarea de clasificación. ResNet 50 se destaca como el modelo más óptimo para la clasificación de defectos de calidad en las alcachofas, mostrando un rendimiento superior en todas las categorías de defectos (*primera*, *fofa*, *cintura* y *violácea*).

Conclusiones

1. Se implementó con éxito un sistema de visión artificial basado en Transfer Learning para mejorar el diagnóstico de defectos de calidad en las alcachofas durante la etapa de inflorescencia. Se entrenaron modelos utilizando Transfer Learning, como InceptionV3, DenseNet, Xception y ResNet 50. Los resultados obtenidos tras el ajuste fino de cada modelo fueron los siguientes: 85.26 %, 86.60 %, 87.5 % y 93.75 %, respectivamente. De estos, ResNet 50 alcanzó la mayor exactitud (93.75 %) y demostró un excelente puntaje F1 en la clasificación de todas las categorías (Primera: 0.94, Fofa: 0.92, Cintura: 1.00, y Violácea: 1.00).

2. La creación de un conjunto de datos de alta calidad y equilibrado de imágenes de alcachofas con diferentes tipos de defectos de calidad ha sido fundamental para el éxito de este proyecto. Este dataset ha permitido entrenar de manera efectiva modelos del Transfer Learning, lo que a su vez ha posibilitado la implementación de un sistema de diagnóstico automatizado capaz de identificar con precisión los defectos de calidad en las alcachofas durante la etapa de inflorescencia.

3. Se logro desarrollar modelos de inteligencia artificial (IA) basados en Transfer Learning, como Xception, InceptionV3, DenseNet y ResNet 50, con un bajo costo computacional. De estos, ResNet 50 alcanzó la mayor exactitud (93.75 %) en la clasificación de defectos de calidad en las alcachofas. Este logro fue posible gracias a la aplicación de técnicas avanzadas de aprendizaje profundo, como el aumento artificial de datos y el ajuste fino (Fine Tuning).

4. Se logró implementar una herramienta para el usuario final(agricultores) en forma de aplicación móvil, que interactúa con el modelo de mayor exactitud (ResNet 50). Este modelo se optimizó en una versión compacta y ligera (TFLITE), compatible para la integración en aplicaciones móviles. La aplicación estará disponible para todos los agricultores, permitiéndoles identificar y diagnosticar de manera más precisa los defectos de calidad en sus cultivos de alcachofa durante la etapa de inflorescencia. Esta solución facilita el acceso a tecnología avanzada, contribuyendo significativamente a mejorar la eficiencia en el manejo y cuidado de los cultivos de alcachofa.

Recomendaciones

1. Se recomienda considerar otros modelos basados en Transfer Learning de libre acceso, como VGG16, VGG19, LeNet, entre otros, para su entrenamiento y evaluación en la clasificación de defectos de calidad en las alcachofas. Estos modelos podrían ofrecer diferentes perspectivas y resultados.

2. Se recomienda explorar el uso de técnicas de segmentación para las categorías *primera* y *fofa*, debido a que presentan un puntaje F1 relativamente bajo (0.94 y 0.92, respectivamente). El objetivo es aislar las regiones de interés de las alcachofas, lo que permitiría resaltar las características distintivas de cada defecto.

3. Se recomienda realizar un monitoreo continuo a largo plazo del sistema de visión artificial, con el fin de evaluar su desempeño de manera constante y detectar posibles variaciones en su precisión al identificar defectos de calidad en las alcachofas durante la etapa de inflorescencia. Esto permitirá aplicar mejoras de forma continua, asegurando que el sistema mantenga su eficacia y adaptabilidad a lo largo del tiempo.

4. Se recomienda construir un conjunto de datos de pruebas que incluya diversos entornos y condiciones, como superficies deterioradas, fondos de diferentes colores, entre otros, en lugar de limitarse a superficies blancas. Esto permitirá evaluar de manera más robusta la capacidad del modelo para identificar defectos de calidad en las alcachofas durante la etapa de inflorescencia.

5. Se recomienda explorar el uso y reentrenamiento de los modelos con imágenes multiespectrales e infrarrojas de los distintos tipos de defectos de calidad en las alcachofas. Estas imágenes permitirán obtener información adicional más allá del espectro visible, lo que facilitará la detección y resalte de defectos, mejorando así la exactitud de los modelos en la clasificación de defectos de calidad.

Apéndices

Apéndice A

Documentos sustentatorios del proyecto

A.1. Documento de veracidad de la información

Figura A.1: Documento de veracidad de la información brindada por la empresa AGRICOLA ALSUR CUSCO.

PRODUCTOS VEGETALES

AGRÍCOLA ALSUR CUSCO

PRODUCTOS VEGETALES NATURALES

CONSTANCIA DE VERACIDAD DE LA INFORMACION

Mediante el presente documento, la empresa **AGRÍCOLA ALSUR CUSCO S.A.C.** certifica la veracidad de la información presentada en el trabajo de investigación del Sr. (o Sra.) **Aron Yabar Aguilar**, cuyos detalles se exponen a continuación:

1. Actualmente, durante el proceso de acopio de alcachofas, se observa un alto índice de descarte debido a defectos de calidad. Esto se debe a un manejo inadecuado de los defectos en los cultivos por parte de los agricultores, quienes, en muchos casos, realizan el diagnóstico de manera tardía o no logran identificar con precisión los defectos. Como resultado, no aplican los agroquímicos en las dosis correctas recomendadas para cada etapa del acopio.
2. Actualmente, la empresa no brinda soporte presencial en los cultivos para el diagnóstico de defectos de calidad en las alcachofas, ofreciendo únicamente una guía de manejo del cultivo durante el acopio. Esto se debe a diversos factores, siendo uno de los principales la falta de vías de comunicación para acceder a los cultivos.
3. La extensión total de los cultivos de alcachofa en promedio es de 700 hectáreas, distribuidas entre varios agricultores que proveen a la empresa, con una cosecha anual promedio de 7,249 toneladas.

Se expide el duplicado a solicitud del interesado, para los fines que estime conveniente el interesado.

Anta, 06 de enero del 2025.

AGRICOLA ALSUR CUSCO S.A.C
Liliana Sánchez Cabel
ENCARGADA DE AC

AGRÍCOLA ALSUR CUSCO S.A.C., URB. TRES REYES S/N, C.C. MARKJO, DISTRITO Y PROVINCIA DE ANTA - CUSCO
TELEFAX: 051-054-438631.

Fuente: Elaboración propia.

Apéndice B

Estructura del sistema de visión artificial

B.1. Preprocesamiento de imágenes

B.1.1. Ecualización de histograma

En la figura B.1 se muestra el proceso de ecualización de histogramas aplicado a las imágenes.

Figura B.1: Ecualización de histograma.

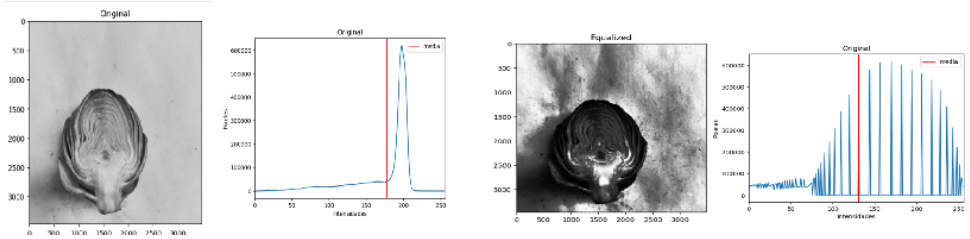
```python
#https://programacionpython80889555.wordpress.com/2023/03/29/ecualizacion-de-histogramas-en-python-con-opencv/
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

```python
def show_hist(title,img):
    media_valores=np.mean(img)#media de las intensidades
    hist = cv2.calcHist([img],[0],None,[256],[0,256])
    plt.figure()
    plt.title(title)
    plt.xlabel('Intensidades')
    plt.ylabel('Pixeles')
    plt.plot(hist)
    plt.axvline(x=media_valores,color="r",label="media")
    plt.legend()
    plt.xlim([0,256])
    plt.show()
```

```python
#cargar imagen y convertirla a esacala de grises
image=cv2.imread("D:/tesis/fotos-aqp/cintura/IMG_20231018_141742.jpg")
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
```

```python
#mostrar Imagen
plt.imshow(cv2.cvtColor(gray,cv2.COLOR_GRAY2RGB))
plt.title('Original')
plt.show()

#mostrar histograma
show_hist('Original',gray)
```



Fuente: Elaboración propia.

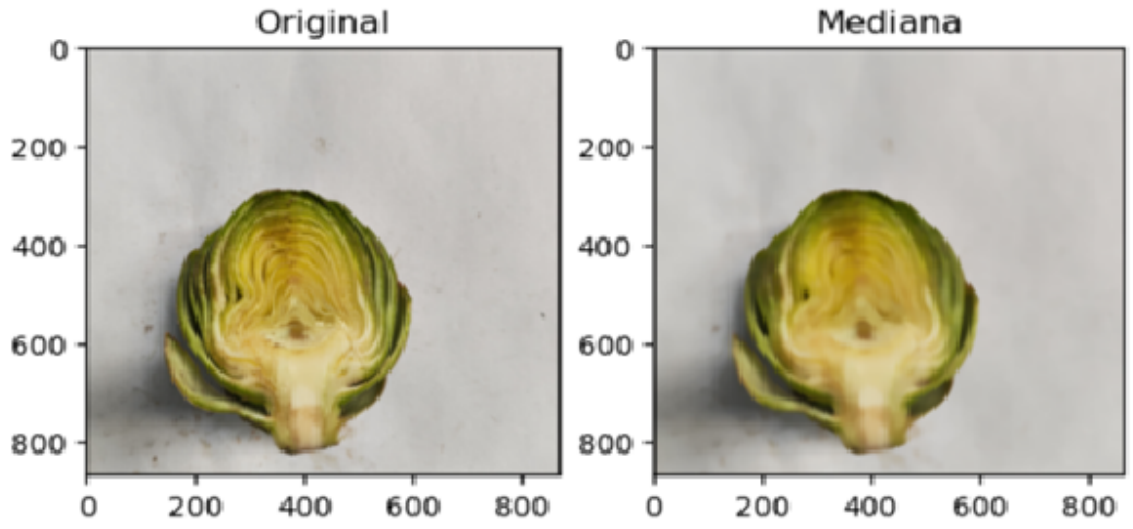
B.1.2. Filtrado de mediana

En la figura B.2 se muestra el proceso de ecualización de histogramas aplicado a las imágenes.

Figura B.2: Filtro mediana.

```
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('D:/tesis/fotos-aqp/dudas/IMG_20231019_142711.jpg')
imgRGB=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
height,width,_=imgRGB.shape
scale=1/4
width=int(width*scale)
height=int(height*scale)
imgRGB= cv2.resize(imgRGB,(width,height))
imgFilter=cv2.medianBlur(imgRGB,15)

plt.figure()
plt.subplot(121)
plt.imshow(imgRGB)
plt.title('Original')
plt.subplot(122)
plt.imshow(imgFilter)
plt.title('Mediana')
plt.show()
```



Fuente: Elaboración propia.

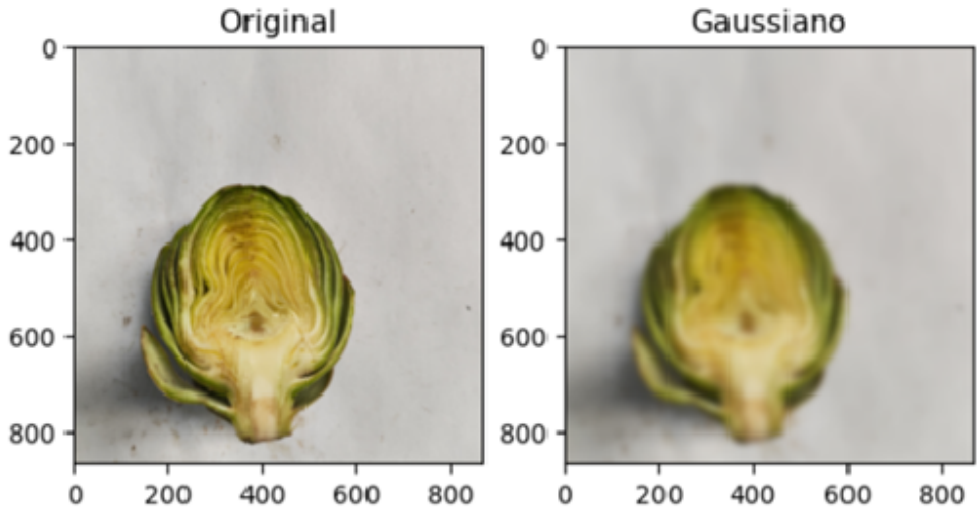
B.1.3. Filtro gaussiano

En la figura B.3 se muestra el proceso de ecualización de histogramas aplicado a las imágenes.

Figura B.3: Filtro gaussiano.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('D:/tesis/fotos-aqp/dudas/IMG_20231019_142711.jpg')
imgRGB=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
height,width,_=imgRGB.shape
scale=1/4
width=int(width*scale)
height=int(height*scale)
imgRGB= cv2.resize(imgRGB,(width,height))
imgFilter=cv2.GaussianBlur(imgRGB,(3,3),0)

plt.figure()
plt.subplot(121)
plt.imshow(imgRGB)
plt.title('Original')
plt.subplot(122)
plt.imshow(imgFilter)
plt.title('Gaussiano')
plt.show()
```



Fuente: Elaboración propia.

B.2. Cargar dataset de defectos de calidad

En la figura B.4 se muestra cómo se realizó la carga del dataset para el entrenamiento posterior, dividiéndolo en conjuntos de entrenamiento, validación y prueba.

Figura B.4: Cargar dataset.

```
PATH = 'D:/CarpetaAron/FotosCuscoDepuracion/FotosPrubeaV2/'


train_direccion = os.path.join(PATH, 'train')
test_direccion = os.path.join(PATH, 'test')
validation_direccion = os.path.join(PATH, 'validation')

#definir los tamaños de imagenes, nro de lotes
BATCH_SIZE_TRAIN = 32
BATCH_SIZE_TEST = 32
IMG_SIZE = (224, 224)

# Extraer los sets de entrenamiento, validación y prueba
train_dataset = tf.keras.utils.image_dataset_from_directory(train_direccion,
                                                            shuffle=True,
                                                            batch_size=BATCH_SIZE_TRAIN,
                                                            image_size=IMG_SIZE)

test_dataset = tf.keras.utils.image_dataset_from_directory(test_direccion,
                                                           shuffle=True,
                                                           batch_size=BATCH_SIZE_TEST,
                                                           image_size=IMG_SIZE)
validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_direccion,
                                                           shuffle=True,
                                                           batch_size=BATCH_SIZE_TRAIN,
                                                           image_size=IMG_SIZE)
#nro de clases
NUM_CLASSES = len(train_dataset.class_names)
```

```
Found 1568 files belonging to 4 classes.
Found 224 files belonging to 4 classes.
Found 448 files belonging to 4 classes.
```

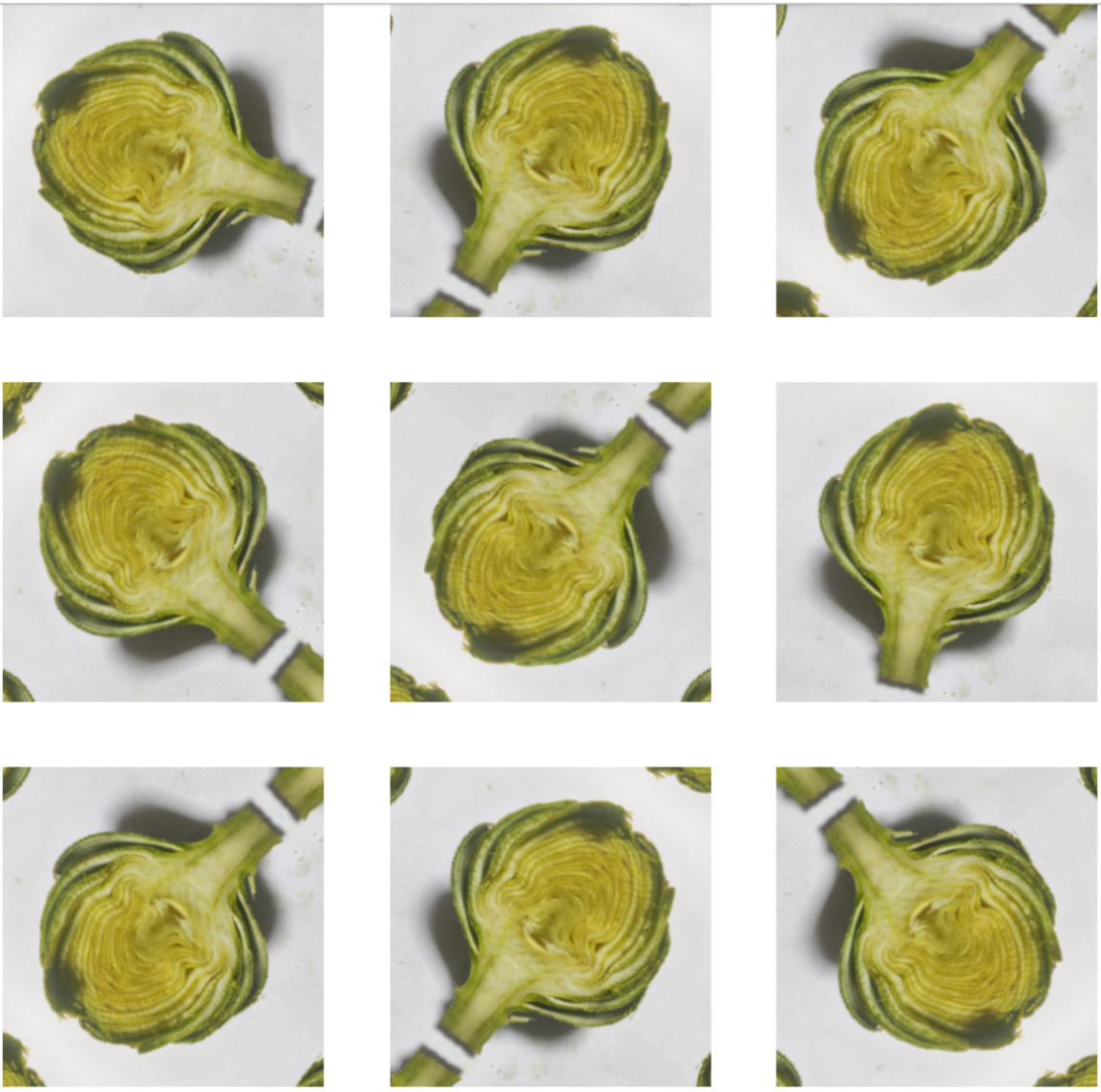
Fuente: Elaboración propia.

B.3. Definir el modelo para el aumento de datos

En la figura B.5 se muestra cómo se realizó el aumento de datos mediante un red neuronal secuencial en cual posteriormente sera parte de la capa del modelo entrenado.

Figura B.5: Modelo para el aumento de datos.

```
[6]: #definir un modelo para la genracion de data adicional para el entrenamiento
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(0.2),
])
```



Fuente: Elaboración propia.

B.4. Adición de capas

En la figura B.6 se muestra cómo se añadieron capas adicionales y se integró el modelo secuencial de aumento de datos al modelo preentrenado de ResNet50.

Figura B.6: Adición de capas.

```
# Capa de entrada
entrada = tf.keras.Input(shape=IMG_SHAPE)
# Conectar entrada con el data_augmetation
x = data_augmentation(entrada)
#prepocesar las imagenes segun el requerimiento del modelo
x = preprocess_input(x)
# Conectar entrada al modelo pre-entrenado: x = entrada + modelo_base
x = modelo_base(x, training=False)
# 1.-Promediar todos los valores según el último eje
# x = entrada + modelo_base + GlobalAveragePooling2D
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Flatten(name='Aplanar1')(x)
x = tf.keras.layers.Dense(1024, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
#x = tf.keras.layers.Flatten(name='Aplanar2')(x)
x = tf.keras.layers.Dense(512, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
# 4. Capa "Dense" con 4 neuronas de salida
salida = tf.keras.layers.Dense(NUM_CLASSES)(x)
# Y conectar "entrada" y "salida" para crear el modelo
modelo = tf.keras.Model(entrada, salida)
```

```
tasa_aprendizaje_base = 0.00001
modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=tasa_aprendizaje_base),
               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])
```

Fuente: Elaboración propia.

B.5. Entrenamiento sin ajuste fino

En la figura B.7 se muestra cómo se realizó el primer entrenamiento del modelo preentrenado ResNet-50 sin ajuste fino.

Figura B.7: Entrenamiento sin ajuste fino.

```
history = modelo.fit(train_dataset,
                     epochs=inicial_epochs,
                     validation_data=validation_dataset)
```

```
Epoch 1/20
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
49/49 [==============================] - 35s 586ms/step - loss: 1.5180 - accuracy: 0.3131 - val_loss: 1.1500 - val_accuracy: 0.5491
Epoch 2/20
49/49 [==============================] - 34s 592ms/step - loss: 1.2109 - accuracy: 0.4713 - val_loss: 0.9469 - val_accuracy: 0.6272
Epoch 3/20
49/49 [==============================] - 35s 628ms/step - loss: 1.0174 - accuracy: 0.5523 - val_loss: 0.8435 - val_accuracy: 0.6540
Epoch 4/20
49/49 [==============================] - 36s 649ms/step - loss: 0.9146 - accuracy: 0.6078 - val_loss: 0.7473 - val_accuracy: 0.7165
Epoch 5/20
49/49 [==============================] - 34s 607ms/step - loss: 0.8029 - accuracy: 0.6633 - val_loss: 0.6858 - val_accuracy: 0.7388
Epoch 6/20
49/49 [==============================] - 36s 643ms/step - loss: 0.7489 - accuracy: 0.6722 - val_loss: 0.6455 - val_accuracy: 0.7433
Epoch 7/20
49/49 [==============================] - 37s 644ms/step - loss: 0.6890 - accuracy: 0.7111 - val_loss: 0.5947 - val_accuracy: 0.7879
Epoch 8/20
49/49 [==============================] - 34s 596ms/step - loss: 0.6686 - accuracy: 0.7296 - val_loss: 0.5815 - val_accuracy: 0.7746
Epoch 9/20
49/49 [==============================] - 37s 663ms/step - loss: 0.5864 - accuracy: 0.7602 - val_loss: 0.5600 - val_accuracy: 0.7812
Epoch 10/20
49/49 [==============================] - 37s 660ms/step - loss: 0.5759 - accuracy: 0.7621 - val_loss: 0.5358 - val_accuracy: 0.8125
Epoch 11/20
49/49 [==============================] - 34s 601ms/step - loss: 0.5624 - accuracy: 0.7628 - val_loss: 0.5100 - val_accuracy: 0.8170
Epoch 12/20
49/49 [==============================] - 34s 603ms/step - loss: 0.5430 - accuracy: 0.7883 - val_loss: 0.5034 - val_accuracy: 0.8080
Epoch 13/20
49/49 [==============================] - 37s 648ms/step - loss: 0.5170 - accuracy: 0.7927 - val loss: 0.4966 - val accuracy: 0.8103
```

Fuente: Elaboración propia.

B.6. Entrenamientos con ajuste fino

En las siguientes figuras se presentan los diferentes tipos de experimentos realizados con ajuste fino. En el recuadro resaltado en rojo se indica a partir de qué capas se descongelaron para el entrenamiento con ajuste fino. A continuación, se muestran las figuras de los experimentos realizados, donde la figura B.8 muestra el entrenamiento óptimo obtenido, y las figuras B.9, B.10, B.11 y B.12 corresponden a los experimentos secundarios.

Figura B.8: Experimento óptimo con ajuste fino.

```
#  ver cuántas capas hay en el modelo base.
print("Número de capas en el modelo base: ", len(modelo_base.layers))
```

```
Número de capas en el modelo base:  175
```

```
# Afinar desde esta capa en adelante
fine_tune_at = 100

# Congela todas las capas antes de la capa `fine_tune_at
for layer in modelo_base.layers[:fine_tune_at]:
  layer.trainable = False
```

```
modelo.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               optimizer = tf.keras.optimizers.Adam(tasa_aprendizaje_base/10),
               metrics=['accuracy'])
```

```
history_fine = modelo.fit(train_dataset,
                          epochs=total_epochs,
                          initial_epoch=history.epoch[-1],
                          validation_data=validation_dataset)
```

```
Epoch 20/40
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
49/49 [==============================] - 42s 646ms/step - loss: 0.4082 - accuracy: 0.8418 - val_loss: 0.4049 - val_accuracy: 0.8438
Epoch 21/40
49/49 [==============================] - 34s 610ms/step - loss: 0.3804 - accuracy: 0.8501 - val_loss: 0.3962 - val_accuracy: 0.8393
Epoch 22/40
49/49 [==============================] - 36s 629ms/step - loss: 0.3526 - accuracy: 0.8724 - val_loss: 0.3837 - val_accuracy: 0.8482
Epoch 23/40
49/49 [==============================] - 34s 612ms/step - loss: 0.3432 - accuracy: 0.8635 - val_loss: 0.3765 - val_accuracy: 0.8504
Epoch 24/40
49/49 [==============================] - 35s 614ms/step - loss: 0.3090 - accuracy: 0.8788 - val_loss: 0.3469 - val_accuracy: 0.8594
Epoch 25/40
49/49 [==============================] - 35s 612ms/step - loss: 0.3020 - accuracy: 0.8909 - val_loss: 0.3554 - val_accuracy: 0.8594
Epoch 26/40
49/49 [==============================] - 36s 630ms/step - loss: 0.2825 - accuracy: 0.8960 - val_loss: 0.3487 - val_accuracy: 0.8571
Epoch 27/40
49/49 [==============================] - 36s 647ms/step - loss: 0.2831 - accuracy: 0.8916 - val_loss: 0.3300 - val_accuracy: 0.8772
Epoch 28/40
49/49 [==============================] - 37s 674ms/step - loss: 0.2766 - accuracy: 0.8922 - val_loss: 0.3140 - val_accuracy: 0.8817
Epoch 29/40
49/49 [==============================] - 36s 639ms/step - loss: 0.2606 - accuracy: 0.9094 - val_loss: 0.3163 - val_accuracy: 0.8728
Epoch 30/40
49/49 [==============================] - 37s 649ms/step - loss: 0.2531 - accuracy: 0.9082 - val_loss: 0.3057 - val_accuracy: 0.8750
Epoch 31/40
49/49 [==============================] - 36s 646ms/step - loss: 0.2394 - accuracy: 0.9126 - val_loss: 0.2916 - val_accuracy: 0.8862
Epoch 32/40
49/49 [==============================] - 40s 691ms/step - loss: 0.2384 - accuracy: 0.9120 - val_loss: 0.2815 - val_accuracy: 0.8862
Epoch 33/40
49/49 [==============================] - 39s 684ms/step - loss: 0.2448 - accuracy: 0.9101 - val_loss: 0.2818 - val_accuracy: 0.8795
Epoch 34/40
49/49 [==============================] - 40s 706ms/step - loss: 0.2171 - accuracy: 0.9196 - val_loss: 0.2751 - val_accuracy: 0.8906
Epoch 35/40
49/49 [==============================] - 38s 659ms/step - loss: 0.2145 - accuracy: 0.9203 - val_loss: 0.2840 - val_accuracy: 0.8862
Epoch 36/40
49/49 [==============================] - 36s 639ms/step - loss: 0.2036 - accuracy: 0.9241 - val_loss: 0.2646 - val_accuracy: 0.8884
Epoch 37/40
49/49 [==============================] - 36s 640ms/step - loss: 0.2065 - accuracy: 0.9216 - val_loss: 0.2597 - val_accuracy: 0.8973
Epoch 38/40
49/49 [==============================] - 37s 646ms/step - loss: 0.2121 - accuracy: 0.9247 - val_loss: 0.2544 - val_accuracy: 0.8884
Epoch 39/40
49/49 [==============================] - 35s 628ms/step - loss: 0.1910 - accuracy: 0.9318 - val_loss: 0.2377 - val_accuracy: 0.9018
Epoch 40/40
49/49 [==============================] - 37s 648ms/step - loss: 0.1842 - accuracy: 0.9298 - val_loss: 0.2369 - val_accuracy: 0.8996
```

Fuente: Elaboración propia.

Figura B.9: Experimento con ajuste fino.

```
#  ver cuántas capas hay en el modelo base.
print("Número de capas en el modelo base: ", len(modelo_base.layers))
```

```
Número de capas en el modelo base:  175
```

```
# Afinar desde esta capa en adelante
fine_tune_at = 120

# Congela todas las capas antes de la capa `fine_tune_at
for layer in modelo_base.layers[:fine_tune_at]:
  layer.trainable = False
```

```
modelo.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               optimizer = tf.keras.optimizers.Adam(tasa_aprendizaje_base/10),
               metrics=['accuracy'])
```

```
history_fine = modelo.fit(train_dataset,
                          epochs=total_epochs,
                          initial_epoch=history.epoch[-1],
                          validation_data=validation_dataset)
```

```
Epoch 20/40
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
49/49 [==============================] - 49s 703ms/step - loss: 0.4575 - accuracy: 0.8157 - val_loss: 0.4723 - val_accuracy: 0.8192
Epoch 21/40
49/49 [==============================] - 40s 725ms/step - loss: 0.4413 - accuracy: 0.8163 - val_loss: 0.4379 - val_accuracy: 0.8304
Epoch 22/40
49/49 [==============================] - 39s 696ms/step - loss: 0.4070 - accuracy: 0.8335 - val_loss: 0.4133 - val_accuracy: 0.8348
Epoch 23/40
49/49 [==============================] - 38s 674ms/step - loss: 0.3743 - accuracy: 0.8616 - val_loss: 0.3816 - val_accuracy: 0.8460
Epoch 24/40
49/49 [==============================] - 39s 685ms/step - loss: 0.3557 - accuracy: 0.8610 - val_loss: 0.3666 - val_accuracy: 0.8549
Epoch 25/40
49/49 [==============================] - 38s 669ms/step - loss: 0.3603 - accuracy: 0.8629 - val_loss: 0.3639 - val_accuracy: 0.8683
Epoch 26/40
49/49 [==============================] - 38s 680ms/step - loss: 0.3348 - accuracy: 0.8699 - val_loss: 0.3482 - val_accuracy: 0.8728
Epoch 27/40
49/49 [==============================] - 37s 657ms/step - loss: 0.3436 - accuracy: 0.8705 - val_loss: 0.3296 - val_accuracy: 0.8795
Epoch 28/40
49/49 [==============================] - 36s 641ms/step - loss: 0.2985 - accuracy: 0.8807 - val_loss: 0.3224 - val_accuracy: 0.8817
Epoch 29/40
49/49 [==============================] - 38s 664ms/step - loss: 0.2966 - accuracy: 0.8871 - val_loss: 0.3097 - val_accuracy: 0.8862
Epoch 30/40
49/49 [==============================] - 38s 674ms/step - loss: 0.2651 - accuracy: 0.9011 - val_loss: 0.3096 - val_accuracy: 0.8906
Epoch 31/40
49/49 [==============================] - 39s 675ms/step - loss: 0.2610 - accuracy: 0.9069 - val_loss: 0.2953 - val_accuracy: 0.8929
Epoch 32/40
49/49 [==============================] - 37s 651ms/step - loss: 0.2490 - accuracy: 0.9075 - val_loss: 0.2856 - val_accuracy: 0.8884
Epoch 33/40
49/49 [==============================] - 37s 651ms/step - loss: 0.2474 - accuracy: 0.9094 - val_loss: 0.2766 - val_accuracy: 0.8929
Epoch 34/40
49/49 [==============================] - 40s 710ms/step - loss: 0.2488 - accuracy: 0.9094 - val_loss: 0.2717 - val_accuracy: 0.8929
Epoch 35/40
49/49 [==============================] - 38s 660ms/step - loss: 0.2402 - accuracy: 0.9088 - val_loss: 0.2745 - val_accuracy: 0.8951
Epoch 36/40
49/49 [==============================] - 39s 691ms/step - loss: 0.2238 - accuracy: 0.9171 - val_loss: 0.2701 - val_accuracy: 0.9040
Epoch 37/40
49/49 [==============================] - 36s 634ms/step - loss: 0.2071 - accuracy: 0.9228 - val_loss: 0.2572 - val_accuracy: 0.8996
Epoch 38/40
49/49 [==============================] - 36s 641ms/step - loss: 0.1990 - accuracy: 0.9241 - val_loss: 0.2608 - val_accuracy: 0.8951
Epoch 39/40
49/49 [==============================] - 40s 715ms/step - loss: 0.2075 - accuracy: 0.9190 - val_loss: 0.2531 - val_accuracy: 0.8973
Epoch 40/40
49/49 [==============================] - 38s 679ms/step - loss: 0.1982 - accuracy: 0.9311 - val_loss: 0.2469 - val_accuracy: 0.9040
```

Fuente: Elaboración propia.

Figura B.10: Experimento con ajuste fino.

```
#  ver cuántas capas hay en el modelo base.
print("Número de capas en el modelo base: ", len(modelo_base.layers))
```

```
Número de capas en el modelo base:  175
```

```
# Afinar desde esta capa en adelante
fine_tune_at = 150

# Congela todas las capas antes de la capa `fine_tune_at`
for layer in modelo_base.layers[:fine_tune_at]:
  layer.trainable = False
```

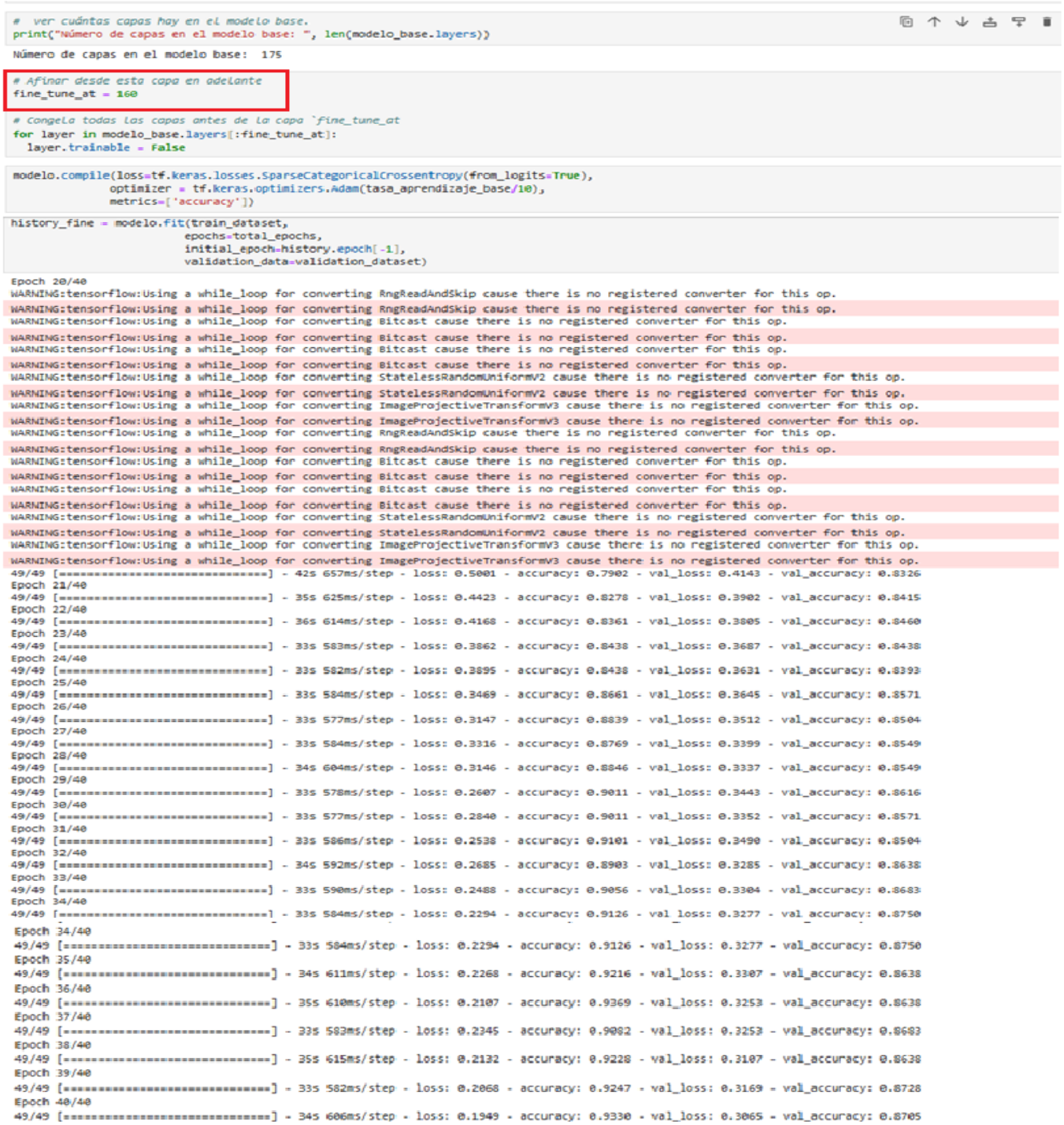
```
modelo.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.Adam(tasa_aprendizaje_base/10),
              metrics=['accuracy'])
```

```
history_fine = modelo.fit(train_dataset,
                          epochs=total_epochs,
                          initial_epoch=history.epoch[-1],
                          validation_data=validation_dataset)
```

```
Epoch 20/40
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
49/49 [==============================] - 42s 657ms/step - loss: 0.5001 - accuracy: 0.7902 - val_loss: 0.4143 - val_accuracy: 0.8326
Epoch 21/40
49/49 [==============================] - 35s 625ms/step - loss: 0.4423 - accuracy: 0.8278 - val_loss: 0.3902 - val_accuracy: 0.8415
Epoch 22/40
49/49 [==============================] - 36s 614ms/step - loss: 0.4168 - accuracy: 0.8361 - val_loss: 0.3805 - val_accuracy: 0.8460
Epoch 23/40
49/49 [==============================] - 33s 583ms/step - loss: 0.3862 - accuracy: 0.8438 - val_loss: 0.3687 - val_accuracy: 0.8438
Epoch 24/40
49/49 [==============================] - 33s 582ms/step - loss: 0.3895 - accuracy: 0.8438 - val_loss: 0.3631 - val_accuracy: 0.8393
Epoch 25/40
49/49 [==============================] - 33s 584ms/step - loss: 0.3469 - accuracy: 0.8661 - val_loss: 0.3645 - val_accuracy: 0.8571
Epoch 26/40
49/49 [==============================] - 33s 577ms/step - loss: 0.3147 - accuracy: 0.8839 - val_loss: 0.3512 - val_accuracy: 0.8504
Epoch 27/40
49/49 [==============================] - 33s 584ms/step - loss: 0.3316 - accuracy: 0.8769 - val_loss: 0.3399 - val_accuracy: 0.8549
Epoch 28/40
49/49 [==============================] - 34s 604ms/step - loss: 0.3146 - accuracy: 0.8846 - val_loss: 0.3337 - val_accuracy: 0.8549
Epoch 29/40
49/49 [==============================] - 33s 578ms/step - loss: 0.2607 - accuracy: 0.9011 - val_loss: 0.3443 - val_accuracy: 0.8616
Epoch 30/40
49/49 [==============================] - 33s 577ms/step - loss: 0.2840 - accuracy: 0.9011 - val_loss: 0.3352 - val_accuracy: 0.8571
Epoch 31/40
49/49 [==============================] - 33s 586ms/step - loss: 0.2538 - accuracy: 0.9101 - val_loss: 0.3490 - val_accuracy: 0.8504
Epoch 32/40
49/49 [==============================] - 34s 592ms/step - loss: 0.2685 - accuracy: 0.8903 - val_loss: 0.3285 - val_accuracy: 0.8638
Epoch 33/40
49/49 [==============================] - 33s 590ms/step - loss: 0.2488 - accuracy: 0.9056 - val_loss: 0.3304 - val_accuracy: 0.8683
Epoch 34/40
49/49 [==============================] - 33s 584ms/step - loss: 0.2294 - accuracy: 0.9126 - val_loss: 0.3277 - val_accuracy: 0.8750
Epoch 35/40
49/49 [==============================] - 34s 611ms/step - loss: 0.2268 - accuracy: 0.9216 - val_loss: 0.3307 - val_accuracy: 0.8638
Epoch 36/40
49/49 [==============================] - 35s 610ms/step - loss: 0.2107 - accuracy: 0.9369 - val_loss: 0.3253 - val_accuracy: 0.8638
Epoch 37/40
49/49 [==============================] - 33s 583ms/step - loss: 0.2345 - accuracy: 0.9082 - val_loss: 0.3253 - val_accuracy: 0.8683
Epoch 38/40
49/49 [==============================] - 35s 615ms/step - loss: 0.2132 - accuracy: 0.9228 - val_loss: 0.3107 - val_accuracy: 0.8638
Epoch 39/40
49/49 [==============================] - 33s 582ms/step - loss: 0.2068 - accuracy: 0.9247 - val_loss: 0.3169 - val_accuracy: 0.8728
Epoch 40/40
49/49 [==============================] - 34s 606ms/step - loss: 0.1949 - accuracy: 0.9330 - val_loss: 0.3065 - val_accuracy: 0.8705
```

Fuente: Elaboración propia.

Figura B.11: Experimento con ajuste fino.



```
# ver cuántas capas hay en el modelo base.
print("Número de capas en el modelo base: ", len(modelo_base.layers))
```

```
Número de capas en el modelo base:  175
```

```
# Afinar desde esta capa en adelante
fine_tune_at = 160

# Congela todas las capas antes de la capa `fine_tune_at`
for layer in modelo_base.layers[:fine_tune_at]:
  layer.trainable = False
```

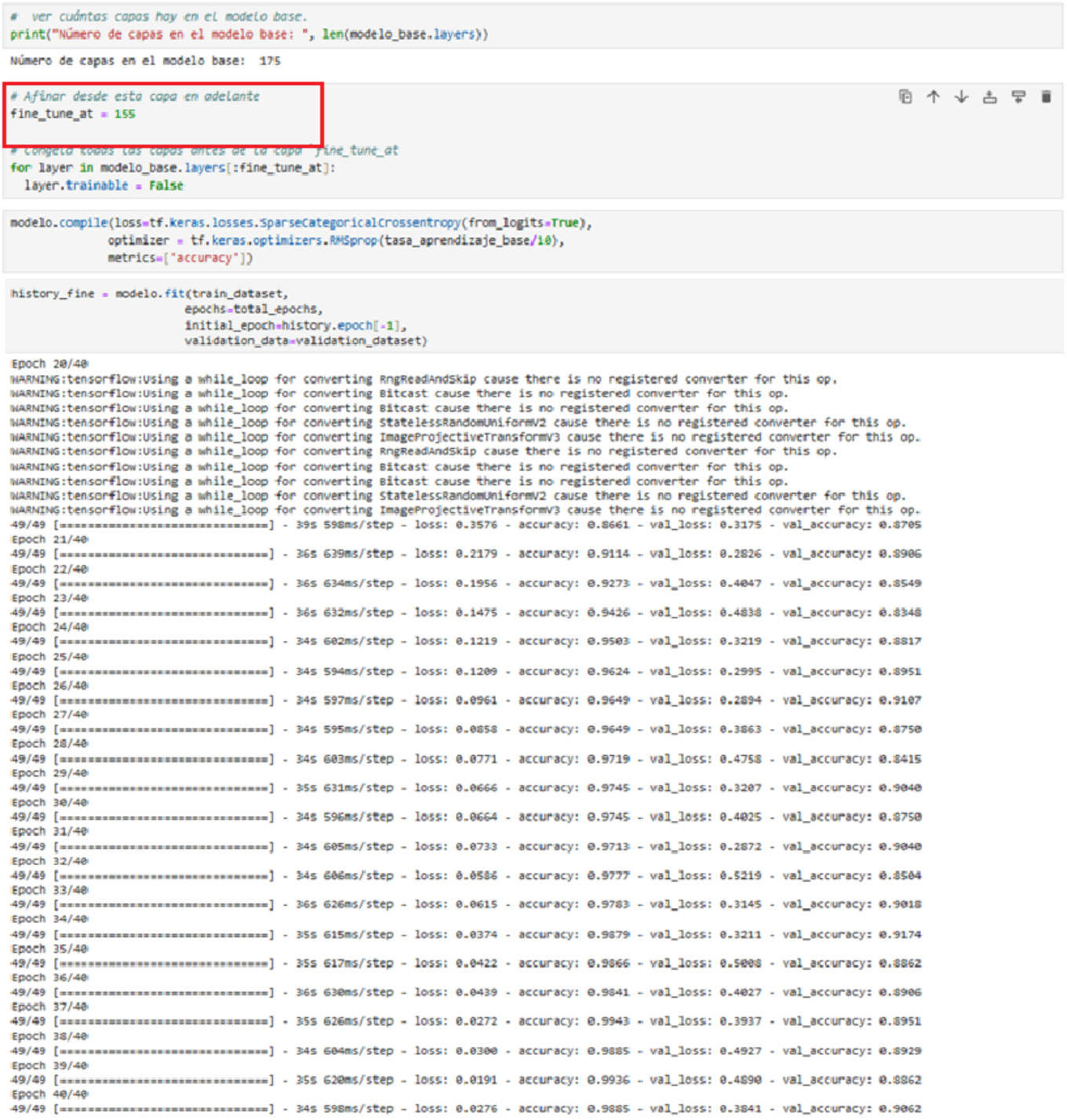
```
modelo.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               optimizer = tf.keras.optimizers.Adam(tasa_aprendizaje_base/10),
               metrics=['accuracy'])
```

```
history_fine = modelo.fit(train_dataset,
                          epochs=total_epochs,
                          initial_epoch=history.epoch[-1],
                          validation_data=validation_dataset)
```

```
Epoch 20/40
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
49/49 [==============================] - 42s 657ms/step - loss: 0.5001 - accuracy: 0.7902 - val_loss: 0.4143 - val_accuracy: 0.8326
Epoch 21/40
49/49 [==============================] - 35s 625ms/step - loss: 0.4423 - accuracy: 0.8278 - val_loss: 0.3902 - val_accuracy: 0.8415
Epoch 22/40
49/49 [==============================] - 36s 614ms/step - loss: 0.4168 - accuracy: 0.8361 - val_loss: 0.3805 - val_accuracy: 0.8460
Epoch 23/40
49/49 [==============================] - 33s 583ms/step - loss: 0.3862 - accuracy: 0.8438 - val_loss: 0.3687 - val_accuracy: 0.8438
Epoch 24/40
49/49 [==============================] - 33s 582ms/step - loss: 0.3895 - accuracy: 0.8438 - val_loss: 0.3631 - val_accuracy: 0.8393
Epoch 25/40
49/49 [==============================] - 33s 584ms/step - loss: 0.3469 - accuracy: 0.8661 - val_loss: 0.3645 - val_accuracy: 0.8571
Epoch 26/40
49/49 [==============================] - 33s 577ms/step - loss: 0.3147 - accuracy: 0.8839 - val_loss: 0.3512 - val_accuracy: 0.8504
Epoch 27/40
49/49 [==============================] - 33s 584ms/step - loss: 0.3316 - accuracy: 0.8769 - val_loss: 0.3399 - val_accuracy: 0.8549
Epoch 28/40
49/49 [==============================] - 34s 604ms/step - loss: 0.3146 - accuracy: 0.8846 - val_loss: 0.3337 - val_accuracy: 0.8549
Epoch 29/40
49/49 [==============================] - 33s 578ms/step - loss: 0.2607 - accuracy: 0.9011 - val_loss: 0.3443 - val_accuracy: 0.8616
Epoch 30/40
49/49 [==============================] - 33s 577ms/step - loss: 0.2840 - accuracy: 0.9011 - val_loss: 0.3352 - val_accuracy: 0.8571
Epoch 31/40
49/49 [==============================] - 33s 586ms/step - loss: 0.2538 - accuracy: 0.9101 - val_loss: 0.3490 - val_accuracy: 0.8504
Epoch 32/40
49/49 [==============================] - 34s 592ms/step - loss: 0.2685 - accuracy: 0.8903 - val_loss: 0.3285 - val_accuracy: 0.8638
Epoch 33/40
49/49 [==============================] - 33s 590ms/step - loss: 0.2488 - accuracy: 0.9056 - val_loss: 0.3304 - val_accuracy: 0.8683
Epoch 34/40
49/49 [==============================] - 33s 584ms/step - loss: 0.2294 - accuracy: 0.9126 - val_loss: 0.3277 - val_accuracy: 0.8750
Epoch 34/40
49/49 [==============================] - 33s 584ms/step - loss: 0.2294 - accuracy: 0.9126 - val_loss: 0.3277 - val_accuracy: 0.8750
Epoch 35/40
49/49 [==============================] - 34s 611ms/step - loss: 0.2268 - accuracy: 0.9216 - val_loss: 0.3307 - val_accuracy: 0.8638
Epoch 36/40
49/49 [==============================] - 35s 610ms/step - loss: 0.2107 - accuracy: 0.9369 - val_loss: 0.3253 - val_accuracy: 0.8638
Epoch 37/40
49/49 [==============================] - 33s 583ms/step - loss: 0.2345 - accuracy: 0.9082 - val_loss: 0.3253 - val_accuracy: 0.8683
Epoch 38/40
49/49 [==============================] - 35s 615ms/step - loss: 0.2132 - accuracy: 0.9228 - val_loss: 0.3107 - val_accuracy: 0.8638
Epoch 39/40
49/49 [==============================] - 33s 582ms/step - loss: 0.2068 - accuracy: 0.9247 - val_loss: 0.3169 - val_accuracy: 0.8728
Epoch 40/40
49/49 [==============================] - 34s 606ms/step - loss: 0.1949 - accuracy: 0.9330 - val_loss: 0.3065 - val_accuracy: 0.8705
```

Fuente: Elaboración propia.

Figura B.12: Experimento con ajuste fino.



```
#  ver cuántas capas hay en el modelo base.
print("Número de capas en el modelo base: ", len(modelo_base.layers))
```

```
Número de capas en el modelo base:  175
```

```
# Afinar desde esta capa en adelante
fine_tune_at = 155

# Congele todas las capas antes de la capa `fine_tune_at`
for layer in modelo_base.layers[:fine_tune_at]:
  layer.trainable = False
```

```
modelo.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(tasa_aprendizaje_base/10),
              metrics=["accuracy"])
```

```
history_fine = modelo.fit(train_dataset,
                          epochs=total_epochs,
                          initial_epoch=history.epoch[-1],
                          validation_data=validation_dataset)
```

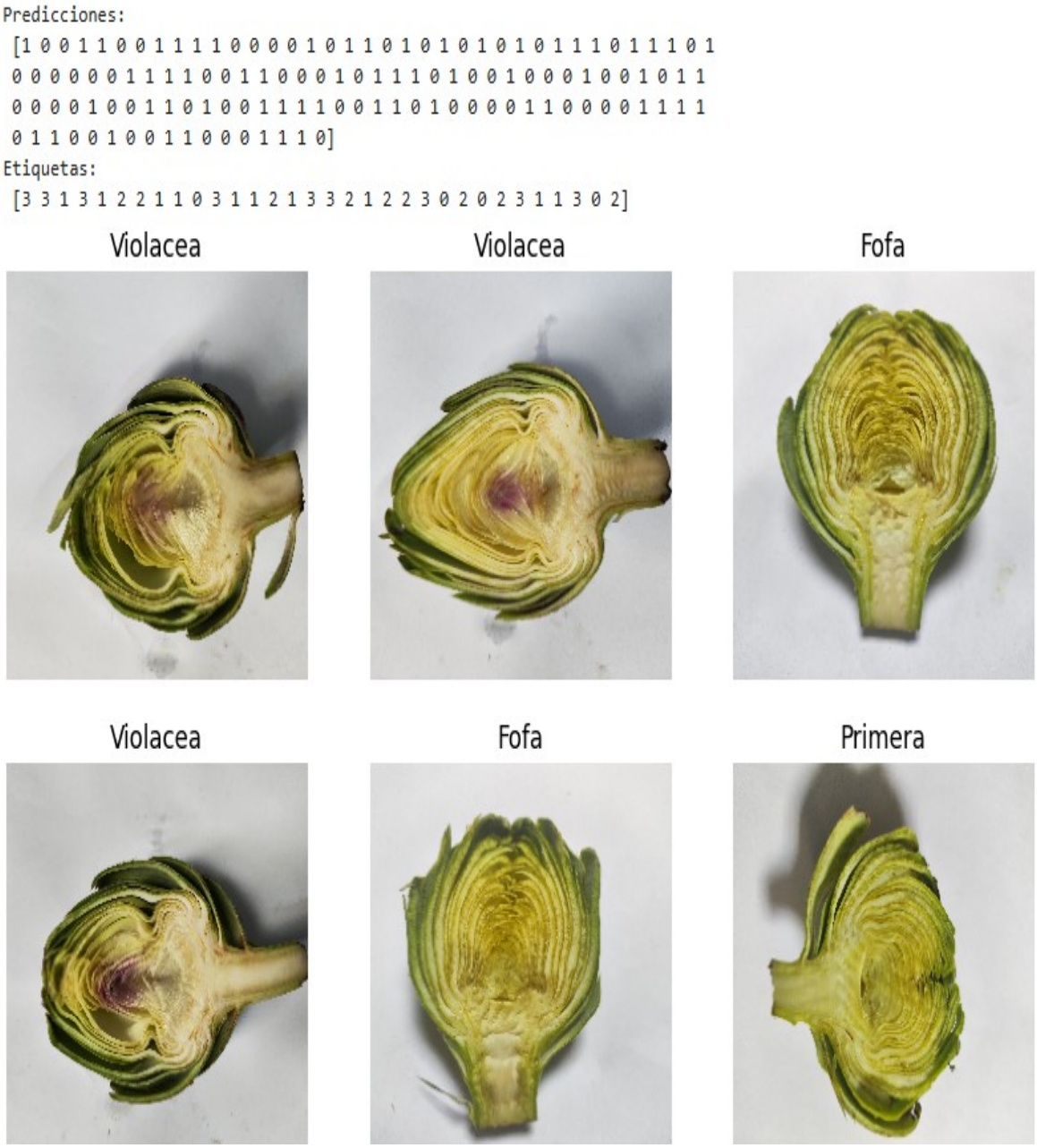
```
Epoch 20/40
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
49/49 [==============================] - 39s 598ms/step - loss: 0.3576 - accuracy: 0.8661 - val_loss: 0.3175 - val_accuracy: 0.8705
Epoch 21/40
49/49 [==============================] - 36s 639ms/step - loss: 0.2179 - accuracy: 0.9114 - val_loss: 0.2826 - val_accuracy: 0.8906
Epoch 22/40
49/49 [==============================] - 36s 634ms/step - loss: 0.1956 - accuracy: 0.9273 - val_loss: 0.4047 - val_accuracy: 0.8549
Epoch 23/40
49/49 [==============================] - 36s 632ms/step - loss: 0.1475 - accuracy: 0.9426 - val_loss: 0.4838 - val_accuracy: 0.8348
Epoch 24/40
49/49 [==============================] - 34s 602ms/step - loss: 0.1219 - accuracy: 0.9503 - val_loss: 0.3219 - val_accuracy: 0.8817
Epoch 25/40
49/49 [==============================] - 34s 594ms/step - loss: 0.1209 - accuracy: 0.9624 - val_loss: 0.2995 - val_accuracy: 0.8951
Epoch 26/40
49/49 [==============================] - 34s 597ms/step - loss: 0.0961 - accuracy: 0.9649 - val_loss: 0.2894 - val_accuracy: 0.9107
Epoch 27/40
49/49 [==============================] - 34s 595ms/step - loss: 0.0858 - accuracy: 0.9649 - val_loss: 0.3863 - val_accuracy: 0.8750
Epoch 28/40
49/49 [==============================] - 34s 603ms/step - loss: 0.0771 - accuracy: 0.9719 - val_loss: 0.4758 - val_accuracy: 0.8415
Epoch 29/40
49/49 [==============================] - 35s 631ms/step - loss: 0.0666 - accuracy: 0.9745 - val_loss: 0.3207 - val_accuracy: 0.9040
Epoch 30/40
49/49 [==============================] - 34s 596ms/step - loss: 0.0664 - accuracy: 0.9745 - val_loss: 0.4025 - val_accuracy: 0.8750
Epoch 31/40
49/49 [==============================] - 34s 605ms/step - loss: 0.0733 - accuracy: 0.9713 - val_loss: 0.2872 - val_accuracy: 0.9040
Epoch 32/40
49/49 [==============================] - 34s 606ms/step - loss: 0.0586 - accuracy: 0.9777 - val_loss: 0.5219 - val_accuracy: 0.8504
Epoch 33/40
49/49 [==============================] - 36s 626ms/step - loss: 0.0615 - accuracy: 0.9783 - val_loss: 0.3145 - val_accuracy: 0.9018
Epoch 34/40
49/49 [==============================] - 35s 615ms/step - loss: 0.0374 - accuracy: 0.9879 - val_loss: 0.3211 - val_accuracy: 0.9174
Epoch 35/40
49/49 [==============================] - 35s 617ms/step - loss: 0.0422 - accuracy: 0.9866 - val_loss: 0.5008 - val_accuracy: 0.8862
Epoch 36/40
49/49 [==============================] - 36s 630ms/step - loss: 0.0439 - accuracy: 0.9841 - val_loss: 0.4027 - val_accuracy: 0.8906
Epoch 37/40
49/49 [==============================] - 35s 626ms/step - loss: 0.0272 - accuracy: 0.9943 - val_loss: 0.3937 - val_accuracy: 0.8951
Epoch 38/40
49/49 [==============================] - 34s 604ms/step - loss: 0.0300 - accuracy: 0.9885 - val_loss: 0.4927 - val_accuracy: 0.8929
Epoch 39/40
49/49 [==============================] - 35s 620ms/step - loss: 0.0191 - accuracy: 0.9936 - val_loss: 0.4890 - val_accuracy: 0.8862
Epoch 40/40
49/49 [==============================] - 34s 598ms/step - loss: 0.0276 - accuracy: 0.9885 - val_loss: 0.3841 - val_accuracy: 0.9062
```

Fuente: Elaboración propia.

B.7. Testeo del modelo

En la siguiente figura B.13 se muestra el proceso de prueba del modelo entrenado.

Figura B.13: Testeo del modelo.



Fuente: Elaboración propia.

B.8. Generación del archivo Tflite

En la siguiente figura B.14 se muestra cómo se genera el archivo TFLite para poder integrar el modelo entrenado en aplicaciones móviles.

Figura B.14: Generación de archivo Tflite.

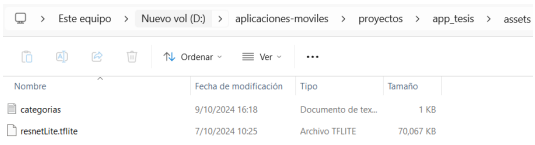
```
keras_file_model = "D:/CarpetaAron/resnet50.h5"
tf.keras.models.save_model(modelo , keras_file_model)
```

```
keras_file_tflite = "D:/CarpetaAron/resnetLite.tflite"
converter = tf.lite.TFLiteConverter.from_keras_model(modelo)
tflite_model = converter.convert()
open(keras_file_tflite, "wb").write(tflite_model)
```

Fuente: Propia

B.9. Interacción Tflite con Flutter

Figura B.15: Sistema de visión artificial para el diagnóstico de defectos de calidad en las alcachofas en etapa de inflorescencia.



(a) Agregar archivos Tflite al proyecto Flutter.

```
Future loadModel() async {
  Tflite.close();
  final dynamic res;

  res = await Tflite.loadModel(
    model: "assets/resnetLite.tflite",
    //model: "assets/mobilnetLite.tflite",
    labels: "assets/categorias.txt",
    numThreads: 1,
  isAsset: true,
  useGpuDelegate: false
  );
}
```

(b) Cargar el archivo Tflite para la predicción de imágenes..

Apéndice C

Repositorio del proyecto

Para el proyecto de investigación se crearon varios directorios como: preprocesamiento de imágenes, entrenamientos de modelos y el código del sistema de visión artificial el cual fue desarrollado en Flutter. Todos estos archivos están almacenados en GitLab.

URL: `https://gitlab.com/aronquis123/proy-defectos-calidad.git`

Bibliografía

Ahmad Rofiqul Muslikh, Rosal Ignatius Moses Setiadi. 2023. *Reconocimiento de enfermedades del arroz utilizando transferencia de aprendizaje.* https://pdfs.semanticscholar.org/27dd/9e6a64e3a7c31a7b2b33c538f9f6b5bbd876.pdf.

Ali Hasan Md. Linkon, Md. Mahir Labib, Tarik Hasan Mozammal Hossain. 2017. *Deep learning in prostate cancer diagnosis and Gleason grading in histopathology images: An extensive study.* https://www.sciencedirect.com/science/article/pii/S2352914821000721.

Beatriz Borrella Petisco. 2022. *Introduction yo Computer.* https://docta.ucm.es/rest/api/core/bitstreams/ddebb1a6-8474-4a20-91cb-78247481577f/content.

Bosco, Alessandro. 2024. *Transfer Learning: Feature Extraction and Fine tuning.* https://medium.com/data-reply-it-datatech/transfer-learning-feature-extraction-and-fine-tuning-db7d82767992.

by ezertis, Autentica. 2021. *Inteligencia Artificial.* https://ahorasomos.izertis.com/autentia/wp-content/uploads/2023/06/IA.pdf.

Calvo, Jorge. 2024. *El Aumento sintético de datos.* https://europeanvalley.es/noticias/el-aumento-sintetico-de-datos/.

Edian F. Franco, Rommel Thiago Jucá Ramos. 2023. *Técnicas de Segmentación en Procesamiento Digital de Imágenes.* https://www.researchgate.net/publication/338026011_Aprendizaje_de_maquina_y_aprendizaje_profundo_en_biotecnologia_aplicaciones_impactos_y_desafios.

Edier, Aristizábal. 2022. *Evaluación del modelo.* https://edieraristizabal.github.io/Libro_cartoGeotecnia/11_evaluacion.html.

Fabien, Maël. 2017. *Modelo XCEPTION y convoluciones separables en profundidad.* https://maelfabien.github.io/deeplearning/xception/#.

Fredes Hubert, Enrique Marañón. 2023. *The Visual Computer.* https://www.researchgate.net/publication/304625636_OPTIMIZACION_DE_LOS_FILTROS_MEDIANA-GAUSSIANO_PARA_UNA_MEJOR_CONVERGENCIA_DEL_SNAKE_EN_LA_SEGMENTACION_DE_IMAGENES_MEDICAS.

González, Nicolás Hernández. 2023. *Análisis de rendimiento de modelos basados en aprendizaje por transferencia con TensorFlow y TensorRT.* https://riull.ull.es/xmlui/bitstream/handle/915/32397/Analisis%20de%20rendimiento%20de%20modelos%20basados%20en%20aprendizaje%20por%20transferencia%20con%20TensorFlow%20y%20TensorRT.pdf?sequence=1&isAllowed=y.

Gutiérrez, José Antonio Taquía. 2017. *El procesamiento de imágenes y su potencial aplicación en empresas con estrategia digital*. https://dialnet.unirioja.es/descarga/articulo/6230450.pdf.

Hernández, Federico Lubinus Badillo César Andrés Rueda. 2021. *Redes neuronales convolucionales: un modelo de Deep Learning en imágenes diagnósticas*. https://rcr.acronline.org/index.php/rcr/article/view/161.

Hernández-Sampieri, Roberto. 2018. Metodología de la investigación.

Jia-Rong Xiao, Pei-Che Chung y Hung-Yi. 2021. *Detection of Strawberry Diseases Using a Convolutional Neural Network*. https://pmc.ncbi.nlm.nih.gov/articles/PMC7823414/.

Kayaalp1, Kiyas. 2024. *A deep ensemble learning method for cherry classification*. https://link.springer.com/article/10.1007/s00217-024-04490-3.

Lagares, Jose Antonio. 2023. *aprendizaje profundo: una nueva vía para convertir el dato en conocimiento*. https://www.mintur.gob.es/Publicaciones/Publicacionesperiodicas/EconomiaIndustrial/RevistaEconomiaIndustrial/423/LAGARES,%20DI%CC%81AZ-DI%CC%81AZ%20Y%20BARRANCO%20GONZALEZ.pdf.

Maravi, Stefanny. 2014. La Alcachofa.

Miguel, Miranda. 2008. La imagen digital.

Moreno, Bernat. 2023. La imagen digital.

Naimur Rashid Methun, Rumana Yasmin. 2021. *Reconocimiento de enfermedades de la zanahoria mediante un enfoque de aprendizaje profundo para la agricultura sostenible*. https://thesai.org/Downloads/Volume12No9/Paper_81-Carrot_Disease_Recognition_using_Deep_Learning_Approach.pdf.

Perdomo, Susana. 2023. *Tipos de formatos de imagen digital*. https://www.deustoformacion.com/blog/diseno-produccion-audiovisual/tipos-formatos-imagen-digital.

Plaza, Agro. 2024. *NITRATO DE CALCIO SOLUBLE x 25KG*. https://www.agroplaza.pe/producto/nitrato-de-calcio/.

Roberto M. Dyke, Kai Hormann. 2023. *The Visual Computer*. https://link.springer.com/article/10.1007/s00371-022-02723-8.

Rosano, Felipe Lara. 2021. *Fundamentos de Redes Neuronales Artificiales*. https://conceptos.sociales.unam.mx/conceptos_final/598trabajo.pdf.

Siddharth, Das. 2017. *Arquitecturas CNN: LeNet, AlexNet, VGG, GoogLeNet, ResNet*. https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

Silva, Jenifer María Campos. 2021. *Evaluación del modelo*. https://www.debatesiesa.com/una-herramienta-para-afinar-criterios-de-decision/.

Supplier, Organic Agriculture. 2024. *Producto Agrícola ALGA Sealand*. https://oasupplier.com/agricultura/producto-agricola-alga-sealand/.

Torres, Claudio Javier Tablada – German Ariel. 2021. *Redes Neuronales Artificiales*. https://www.famaf.unc.edu.ar/~revm/digital24-3/redes.pdf.

Wei Wang, Yutao Li. 2020. *A Novel Image Classification Approach via Dense-MobileNet Models*. https://onlinelibrary.wiley.com/doi/10.1155/2020/7602384.