DIAGNÓSTICO DE ENFERMEDADES EN LA HOJA DEL TOMATE MEDIANTE UN SISTEMA EMBEBIDO USANDO INTELIGENCIA ARTIFICIAL E IMÁGENES.

CAMILO ANDRES SANTOS ORTIZ
HAROLD HERNANDO RODRÍGUEZ RODRÍGUEZ
WILMER ANDRES GUERRERO RANGEL

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

DIAGNÓSTICO DE ENFERMEDADES EN LA HOJA DEL TOMATE MEDIANTE UN SISTEMA EMBEBIDO USANDO INTELIGENCIA ARTIFICIAL E IMÁGENES.

CAMILO ANDRES SANTOS ORTIZ
HAROLD HERNANDO RODRÍGUEZ RODRÍGUEZ
WILMER ANDRES GUERRERO RANGEL

Trabajo de Grado para optar al título de
Ingeniero Electrónico

Director
Jaime Guillermo Barreo Pérez
Magíster en potencia eléctrica

Codirector
Carlos Alberto Flórez Arias
Magíster en Ingeniería Mecánica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

DEDICATORIA

A nuestros padres queremos expresarles nuestra profunda gratitud por su amor incondicional, su apoyo inquebrantable y su confianza en nosotros. Desde nuestra infancia hasta nuestro último semestre de universidad, siempre han estado a nuestro lado, alentándonos y guiándonos en todo momento. Sin su amor y apoyo, nunca hubiéramos llegado hasta aquí. Esta tesis está dedicada a ustedes, nuestros queridos padres, por ser nuestros pilares y por enseñarnos a nunca rendirnos.

A nuestros maestros y profesores, les agradecemos por su paciencia, dedicación y sabiduría. Gracias por compartir con nosotros su conocimiento y experiencia, por motivarnos a ser mejores cada día y por ayudarnos a desarrollar nuestra pasión por la electrónica. Esta tesis está dedicada a todos ustedes, como muestra de nuestro agradecimiento por su compromiso con nuestra educación y por su invaluable contribución a nuestra formación académica.

A la Universidad Industrial de Santander, agradecemos por brindarnos la oportunidad de estudiar en una de las mejores universidades de Colombia. Gracias por su excelencia académica, sus instalaciones de primer nivel y por el compromiso de su facultad en formar profesionales capaces y comprometidos con la sociedad. Esta tesis está dedicada a nuestra alma mater, como una muestra de nuestro orgullo y gratitud por haber sido parte de ella.

AGRADECIMIENTOS

Antes de empezar, queremos dar gracias a Dios por su amor y su gracia infinita. Él es la fuente de toda sabiduría y guía, estamos agradecidos por su presencia en nuestras vidas y por haber hecho posible la culminación de esta tesis de grado en Ingeniería Electrónica.

A nuestros padres, gracias por ser nuestra inspiración y por apoyarnos en todo momento. Agradecemos su amor incondicional, paciencia, dedicación y enseñanzas, que nos han permitido llegar hasta donde estamos hoy. Gracias por brindarnos su apoyo emocional y financiero, y por creer en nosotros incluso en los momentos más difíciles. Sin su amor y guía, no habríamos logrado este gran sueño. Los amamos mucho.

Agradecemos a nuestros profesores, especialmente a Carlos Flórez y Jaime Barrero por compartir sus conocimientos y experiencias con nosotros y por motivarnos a ser mejores cada día. A nuestros compañeros, gracias por ser una fuente constante de inspiración, apoyo y colaboración en este largo y emocionante camino. Los extrañaremos mucho y siempre tendremos un lugar especial en nuestros corazones para todos ustedes.

A la Universidad Industrial de Santander, gracias por brindarnos una educación de alta calidad. Agradecemos especialmente a los grupos de investigación CEMOS y DicBot, por brindarnos la oportunidad de trabajar en proyectos innovadores y por permitirnos aprender de los mejores en el campo de la electrónica.

Finalmente, queremos agradecer a todos los agricultores, al ingeniero agrónomo

Jairo Rueda y a todos aquellos que, de una u otra forma, han hecho posible la culminación de este proyecto. Gracias por su apoyo, colaboración y enseñanzas. Este logro es de todos nosotros y estamos agradecidos por haber tenido la oportunidad de compartirlo con ustedes.

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

TÍTULO: DIAGNÓSTICO DE ENFERMEDADES EN LA HOJA DEL TOMATE MEDIANTE UN SISTEMA EMBEBIDO USANDO INTELIGENCIA ARTIFICIAL E IMÁGENES. *

AUTOR: CAMILO ANDRES SANTOS ORTIZ, HAROLD HERNANDO RODRÍGUEZ RODRÍGUEZ, WILMER ANDRES GUERRERO RANGEL **

PALABRAS CLAVE: HOJA DE TOMATE, REDES NEURONALES PROFUNDAS, PYTHON, IMÁGENES, ENFERMEDADES EN PLANTAS.

DESCRIPCIÓN:

Dentro del sector agrícola, el cultivo de tomates es uno de los más relevantes a nivel global, debido a su alta demanda y la diversidad de usos que presenta. No obstante, estos cultivos pueden verse perjudicados por enfermedades y plagas, que conlleva consecuencias irreversibles en la producción si no se detectan a tiempo. Por ello, se plantea la necesidad de desarrollar un sistema especializado en inteligencia artificial, que pueda identificar enfermedades en las hojas de tomate mediante el uso de imágenes. Este sistema es diseñado para ser utilizado tanto por agricultores con experiencia cultivando tomate, como por ingenieros agrónomos, con el objetivo de realizar diagnósticos tempranos y asegurar la salud del cultivo. Asimismo, su implementación podría tener fines educativos, permitiendo el estudio de las enfermedades que afectan al tomate y se ven manifestadas en las hojas, apoyando el seguimiento de los cultivos. Para llevar a cabo este proyecto, se utilizó una base de datos proporcionada por la comunidad *Kaggle* y se visitaron diferentes invernaderos para capturar imágenes de hojas sanas y enfermas de los cultivos de tomate. Se emplearon herramientas de software libre y código abierto para cada etapa del proyecto, entre ellas Google Colaboratory, con el fin de entrenar una inteligencia artificial de jerarquía simplificada y tres modelos de *Transfer Learning*. Finalmente, el modelo resultante fue implementado en dos sistemas embebidos, NVIDIA Jetson Nano de 4GB y Raspberry pi 4B, y se realizó un análisis comparativo de los resultados obtenidos para validar su rendimiento en ambas tarjetas de desarrollo.

---

* Trabajo de grado

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director:Jaime Guillermo Barreo Pérez, Magíster en potencia eléctrica. Codirector:Carlos Alberto Flórez Arias, Magíster en Ingeniería Mecánica

ABSTRACT

TITLE: TOMATO LEAF DISEASE DIAGNOSIS BY AN EMBEDDED SYSTEM USING ARTIFICIAL INTELLIGENCE AND IMAGING. *

AUTHOR: CAMILO ANDRES SANTOS ORTIZ, HAROLD HERNANDO RODRÍGUEZ RODRÍGUEZ, WILMER ANDRES GUERRERO RANGEL **

KEYWORDS: TOMATO LEAF, DEEP NEURAL NETWORKS, PYTHON, IMAGES, PLANT DISEASES.

DESCRIPTION:

Within the agricultural sector, tomato cultivation is one of the most relevant globally, due to its high demand and the diversity of uses it presents. However, these crops can be harmed by diseases and pests, which can have irreversible consequences on production if not detected in time. Therefore, there is a need to develop a specialized artificial intelligence system that can identify diseases in tomato leaves using images. This system would be aimed at both experienced farmers and agricultural engineers, with the aim of making early diagnoses and ensuring the health of the crop. Likewise, its implementation could have educational purposes, allowing the study of diseases that affect tomato leaves and supporting crop monitoring. To carry out this project, a database provided by the Kaggle community was used and different greenhouses were visited to capture images of healthy and diseased tomato leaves. Free and open-source software tools were used for each stage of the project, including Google Colaboratory, in order to train a simplified hierarchy artificial intelligence and three Transfer Learning models. Finally, the resulting model was implemented in two embedded systems, NVIDIA Jetson Nano 4GB and Raspberry Pi 4B, and a comparative analysis of the results obtained was carried out to validate its performance on both development boards.

---

* Bachelor Thesis

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director:Jaime Guillermo Barreo Pérez, Magíster en potencia eléctrica. Codirector:Carlos Alberto Flórez Arias, Magíster en Ingeniería Mecánica

INTRODUCCIÓN

El cultivo del tomate es uno de los más importantes en la agricultura a nivel mundial debido a su alta demanda y sus diferentes usos en la cocina[1]. Esta hortaliza es especialmente consumida por su alto contenido en antioxidantes, los cuales tienen efectos beneficiosos para la salud cardiovascular. Los agricultores han desarrollado diversas técnicas para maximizar la productividad y la calidad de esta hortaliza. Sin embargo, los cultivos de tomate también pueden verse afectados por diversas enfermedades y plagas, las cuales suelen ser diagnosticadas por medio de una inspección visual basada en los síntomas y apariencia de las plantas. En este sentido, se sugiere que los análisis visuales sean realizados por un ingeniero agrónomo o un agricultor con basta experiencia en el tema, ya que un diagnóstico equivocado puede dar lugar a pérdidas irreparables en el cultivo[2] y desaprovechar toda la producción.

La inteligencia artificial es una disciplina en constante crecimiento que ha transformado muchos procesos de la industria, ofreciendo una amplia variedad de aplicaciones que incluyen reconocimiento de objetos por medio de la visión artificial, el reconocimiento de voz, la comprensión del lenguaje natural, la traducción automática, entre otras[3].

---

1 Franco Alirio Vallejo Cabrera. *Mejoramiento genético y producción de tomate en Colombia*. Colombia: Universidad Nacional de Colombia, nov. de 1999.

2 González Frayre et al. "*Estudio y comparativa de algoritmos de detección de objetos con redes neuronales artificiales convolucionales para la detección de enfermedades en hojas*". En: (2019). Publisher: Asociación Mexicana de la Industria Automotriz, pág. 1. DOI: https://doi.org/10.48779/p41k-fx69.

3 Daniel Alexis Pérez-Aguilar, Redy Henry Risco-Ramos y Luis Casaverde-Pacherrez. "Transfer learning en la clasificación binaria de imágenes térmicas". En: *Ingenius* 26 (29 de jun. de 2021), págs. 71-86. DOI: 10.17163/ings.n26.2021.07.

El problema a resolver en el marco del presente proyecto de investigación, se encamina hacia la necesidad de desarrollar un sistema robusto capaz de diagnosticar enfermedades en la hoja del tomate usando imágenes e inteligencia artificial, facilitando la detección temprana de estas. Las hojas al igual que el tallo son los primeros en evidenciar los síntomas, por lo tanto, detectar la enfermedad en la hoja garantiza un diagnóstico temprano de la misma. El sistema puede dar solución a necesidades reales, como brindar apoyo a un ingeniero agrónomo o un agricultor experto para que haga el diagnostico o realizar el monitoreo de un cultivo, incluso puede ser usado con fines académicos, permitiendo a las próximas generaciones utilizar nuevas tecnologías y aprender sobre las enfermedades de la hoja del tomate.

Para el desarrollo del proyecto se obtuvo información a través de una base de datos proporcionada por la comunidad *Kaggle*, la cual fue utilizada para los primeros entrenamientos del modelo de inteligencia artificial, además, se visitaron diversos invernaderos ubicados en los municipios de Betulia, la Mesa de los Santos y Zapatoca. Durante estas visitas se tomaron múltiples imágenes de hojas sanas y enfermas de los cultivos de tomate, las cuales fueron utilizadas para ampliar y mejorar la base de datos.

Con el propósito de realizar el proyecto para facilitar el acceso a posteriores trabajos con fines académicos, se optó por el uso de software libre, por ello, se desarrolló dentro del sistema operativo Ubuntu usando herramientas de código abierto para cada una de las etapas del proyecto. Se diseñó un modelo en 3D con la herramienta FreeCAD que protege el sistema y sus periféricos. Utilizando la herramienta de Google, Google Colaboratory, se entrenó una inteligencia artificial de jerarquía simplificada y tres modelos de *Transfer learning*. Finalmente se desplegó en dos sistemas embebidos, NVIDIA Jetson Nano de 4GB y Raspberry pi 4B con ayuda de una interfaz de usuario. Para validar el desempeño del sistema en ambas tarjetas de desarrollo, se llevo a cabo un análisis comparativo de los resultados.

1. OBJETIVOS

Objetivo general

- Diseñar e implementar un algoritmo de aprendizaje profundo que permita identificar enfermedades en la hoja de la planta del tomate desplegado en un sistema embebido.

Objetivos específicos

- Obtener una base de datos con imágenes de hojas sanas y hojas con enfermedades en la planta del tomate, que permita entrenar el algoritmo de aprendizaje profundo.
- Diseñar un algoritmo de aprendizaje profundo con representación jerárquica reducida, que sea capaz de reconocer al menos 3 enfermedades diferentes en la hoja de la planta del tomate.
- Implementar un algoritmo de aprendizaje profundo en mínimo dos tarjetas de desarrollo para realizar una comparación de consumo de potencia eléctrica.
- Realizar pruebas de campo para comprobar el desempeño en condiciones de iluminación natural del algoritmo de aprendizaje profundo, desplegado en el sistema embebido, montado en una carcasa.

2. ESTADO DEL ARTE.

En esta sección se exploran las ultimas tendencias en el desarrollo de sistemas de inteligencia artificial para la agricultura. Se encuentran algoritmos de aprendizaje profundo con jerarquía reducida para la detección de enfermedades en plantas, con ayuda de *Transfer learning* se busca mejorar su desempeño, también el despliegue de algunos algoritmos en sistemas para comprobar su rendimiento, finalizando con métodos alternativos para la detección de enfermedades.

Los investigadores Maryum, et al.[4] desarrollaron una arquitectura de aprendizaje profundo EfficientNet, para clasificar imágenes de enfermedades de la hoja de yuca de forma rápida y precisa. Además, se utiliza el modelo de segmentación semántica U-Net para segmentar con precisión las hojas de las imágenes. Los resultados muestran un rendimiento razonable con una tasa de precisión del 81,48 % y el 89,09 % en los conjuntos de datos originales y segmentados, respectivamente. En el artículo propuesto por Geetharamani y Arun Pandian[5] se presenta un modelo de redes neuronales convolucionales de jerarquía reducida con un total de 9 capas, que alcanza una precisión media del 96,46 % en la clasificación de las imágenes de hojas de plantas del conjunto de pruebas. El modelo propuesto por los autores puede clasificar eficazmente 38 clases distintas de plantas sanas y enfermas utilizando imágenes de hojas proporcionadas por la base de datos de *Plant village*. Además,

---

[4] Alina Maryum, Muhammad Usman Akram y Anum Abdul Salam. “Cassava Leaf Disease Classification using Deep Neural Networks”. En: *2021 IEEE 18th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*. 2021, págs. 32-37. DOI: 10.1109/HONET53078.2021.9615488.

[5] Geetharamani G. y Arun Pandian J. “Identification of plant leaf diseases using a nine-layer deep convolutional neural network”. En: *Computers & Electrical Engineering* 76 (1 de jun. de 2019), págs. 323-338. DOI: 10.1016/j.compeleceng.2019.04.011.

se comparó el modelo propuesto con modelos SVM, *logistic regression, decision tree* y K-NN, y los resultados mostraron que el modelo propuesto es superior a todos los modelos mencionados. También se discute la importancia del aumento de datos para mejorar el rendimiento del modelo en la clasificación de imágenes de hojas de plantas.

Un estudio realizado por Yunong Tian, et al.[6] propone un modelo mejorado de detección de manzanas en huertos utilizando el método DenseNet en combinación con el modelo YOLO-V3, este modelo puede detectar manzanas en diferentes etapas de crecimiento y en condiciones de iluminación fluctuante y fondos complejos, además, los resultados experimentales muestran que el modelo propuesto supera al modelo YOLO-V3 original y al modelo Faster R-CNN con VGG16 net. Además, el modelo propuesto también puede detectar manzanas superpuestas y ocultas en tiempo real. Por otra parte los autores YU-Dong Zhang, et al. desarrollaron un método para la clasificación de frutas mediante una red convolucional de 13 Capas[7]. El algoritmo logro una precisión general del 94,94 %. Adicionalmente, gracias al aumento de datos, el equipo pudo expandir significativamente la cantidad de datos para el entrenamiento, pasando de 1800 a 63000 datos. En cuanto al rendimiento, el análisis de tiempo demostró que la GPU puede acelerar significativamente el proceso de entrenamiento en un factor de 177 y un factor de 175 para los datos de prueba.

En algunos casos es necesario desarrollar arquitecturas que se adapten al sistema en el que se va a desplegar. En este sentido, Ruchi Gajjar, et al. desarrollaron una nueva arquitectura de Red Neuronal Convolucional para la identificación de en-

---

[6] Yunong Tian et al. “Apple detection during different growth stages in orchards using the improved YOLO-V3 model”. en. En: *Computers and Electronics in Agriculture* 157 (feb. de 2019), págs. 417-426. DOI: 10.1016/j.compag.2019.01.012.

[7] Yu-Dong Zhang et al. “Image based fruit category classification by 13-layer deep convolutional neural network and data augmentation”. en. En: *Multimedia Tools and Applications* 78.3 (feb. de 2019), págs. 3613-3632. DOI: 10.1007/s11042-017-5243-3.

fermedades en 20 tipos diferentes de plantas. La arquitectura propuesta alcanzó un rendimiento del 96,88 %[8] y se desplegó en una Tarjeta Nvidia Jetson TX1. La validación en campo del sistema ha demostrado que es robusto y capaz de funcionar en variadas condiciones, incluyendo diferentes niveles de iluminación, tamaños, orientaciones y condiciones de fondo, lo que sugiere su potencial para ser una herramienta útil en la prevención y control de enfermedades en los cultivos.

En el trabajo de Alvaro Fuentes, et al. se propone un sistema de detección robusto basado en aprendizaje profundo para la identificación de enfermedades y plagas en plantas de tomate [9], los autores consideraron tres arquitecturas de detección, las cuales son Faster R-CNN, SSD y R-FCN, además usaron dos extractores de características para determinar el modelo más adecuado, en su trabajo se propone un método de aumento de datos para mejorar el rendimiento del modelo. El conjunto de datos de enfermedades y plagas de tomate usado, incluye diferentes variaciones para probar el modelo y se encontró que el modelo propuesto es capaz de reconocer nueve categorías diferentes de enfermedades y plagas con una precisión satisfactoria. Otras arquitecturas usadas con mayor frecuencias, son las propuestas por los investigadores Shaha Manali y Pawar Meenakshi[10] especificamente: AlexNet, VGG16 y VGG19 en la tarea de clasificación de imágenes de dos bases de datos diferentes: GHIM10K y CalTech256. Los autores probaron estas arquitecturas

---

[8] Ruchi Gajjar et al. “Real-time detection and identification of plant leaf diseases using convolutional neural networks on an embedded platform”. En: *The Visual Computer* 38.8 (ago. de 2022), págs. 2923-2938. DOI: 10.1007/s00371-021-02164-9.

[9] Alvaro Fuentes et al. “A Robust Deep Learning Based Detector for Real Time Tomato Plant Diseases and Pests Recognition”. en. En: *Sensors* 17.9 (sep. de 2017). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, pág. 2022. DOI: 10.3390/s17092022.

[10] Manali Shaha y Meenakshi Pawar. “Transfer Learning for Image Classification”. En: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA). Mar. de 2018, págs. 656-660. DOI: 10.1109/ICECA.2018.8474802.

en ambas bases de datos y compararon su desempeño utilizando tres medidas de rendimiento: recall, precision y F1score, donde la arquitectura VGG19 tuvo el mejor rendimiento en ambas bases de datos.

Los autores Yanchao Zhangm, et al. seleccionaron la red YOLOv4-tiny como base para su investigación y le realizaron modificaciones para satisfacer la demanda de detección de fresas en tiempo real. Al reducir gradualmente la complejidad de la red, se reducen los parámetros y se mejora la velocidad de detección[11]. RTSD-Net se propuso a partir de la red neuronal ligera YOLOv4-tiny, donde la velocidad se mejora significativamente, aunque se produjo una ligera perdida de precisión. Este modelo genera grandes ventajas para la computación, permitiendo el control en tiempo real de un robot de cosecha en un futuro. Finalmente Sindhuja Sankaran, et al. en su estudio describen técnicas no invasivas para detectar enfermedades en plantas, incluyendo espectroscopia e imágenes, y la detección basada en el perfilado de compuestos orgánicos volátiles[12], sus métodos demuestran una buena capacidad de detección, pero enfrentan desafíos en la optimización y automatización en condiciones de campo. El estudio de estas técnicas podrían integrarse en vehículos agrícolas autónomos para una detección de enfermedades de plantas de forma fiable y en tiempo real.

---

[11] Yanchao Zhang et al. “Real-time strawberry detection using deep neural networks on embedded system (rtsd-net): An edge AI application”. En: *Computers and Electronics in Agriculture* 192 (ene. de 2022), pág. 106586. DOI: 10.1016/j.compag.2021.106586.

[12] Anil A. Bharate y M. S. Shirdhonkar. “A review on plant disease detection using image processing”. En: *2017 Internationa Conference on Intelligent Sustainable Systems (ICISS)*. Dic. de 2017, págs. 103-109. DOI: 10.1109/ISS1.2017.8389326.

2.1. LA PLANTA DE TOMATE.

El tomate es una de las hortalizas más producidas a nivel mundial, así lo asegura la organización de las naciones unidas para la alimentación y la agricultura[13]. El tomate es rico en vitaminas, minerales y antioxidantes, lo que lo convierte en un alimento muy beneficioso para la salud.

Las enfermedades en las plantas de tomate son causadas por patógenos, hospederos y el ambiente[14]. Los patógenos que atacan a las plantas de tomate pueden ser hongos, bacterias, virus, nematodos y otros organismos que se propagan en el ambiente[15]. Los invernaderos, por su diseño, crean un ambiente propicio para la propagación de enfermedades, debido a que si alguna planta se encuentra infectada, la alta densidad de plantas y la limitada circulación de aire, generan que todo el cultivo se infecte de forma acelerada. Los cultivos en invernaderos suelen ser sometidos a un estrés constante, tanto físico como biológico, debido a las altas demandas de producción y a la necesidad de aplicar tecnología para maximizar la productividad. En consecuencia, la prevención y el control de enfermedades en invernaderos son fundamentales para mantener la salud de las plantas y garantizar una producción sostenible y de calidad.

El cultivo de tomate de invernadero ha sido seleccionado para este proyecto, debido a que las enfermedades que afectan a esta planta no suelen generar perforaciones en las hojas, lo que resulta beneficioso para el análisis y diagnostico preciso de las enfermedades. En la Figura 1 se encuentra una muestra de la hoja sana del tomate.

---

[13] Centro de Estudios para el Desarrollo Rural Sustentable y la Soberania Alimentaria. *Análisis de la producción y consumo de hortalizas*. 2020.

[14] Roberto Bernal. *ENFERMEDADES DE TOMATE (Lycopersicum esculentum Mill.) EN INVERNADERO EN LAS ZONAS DE SALTO Y BELLA UNION*. 2010.

[15] Dominique Blancard. *Enfermedades del tomate*. Mundi-Prensa Libros, 1 de ene. de 2011. 682 págs.

Figura 1. Hoja sana obtenida: (a) Visita técnica, (b) Plant Village

(a) (b)

2.1.1. Enfermedades.

Tizón temprano. Es una enfermedad fúngica causada por el hongo *Alternaria Solani Sorauer* [16]. Esta enfermedad puede manifestarse varias veces en el mismo cultivo debido a que el hongo puede sobrevivir en todo el ambiente de la plantación y propagarse mediante plantines infectados, semillas, viento y agua.

Los primeros signos de la enfermedad suelen manifestarse en las hojas, donde se produce la muerte de los tejidos, lo que puede provocar su estrangulación. Las hojas afectadas presentan manchas circulares de color marrón con un borde amarillo, las cuales se distinguen por tener anillos concéntricos oscuros y una textura polvorienta, se evidencia en la Figura 2. Conforme la enfermedad avanza, estas manchas pueden propagarse a los tallos y frutos de la planta, lo que puede afectar la calidad de los frutos y hacerlos poco saludables para el consumo humano.

---

[16] Nercy Sita Ricardo Paz, Angel Gustavo Polanco Aballe y Sonia Reyes Gómez. “Comportamiento del tizón temprano del tomate (Alternaria solani) en las condiciones del municipio de Holguín, Cuba”. En: (2013-8).

Figura 2. Tizón temprano obtenido : (a) Visita técnica, (b) Plant Village

(a) (b)

Tizón tardío. Esta enfermedad del tomate es causada por el hongo *Phytophthora Infestans*, un *Oomyceto* que puede atacar el cultivo en las diversas etapas del crecimiento de la planta. Este patógeno afecta a las hojas, tallos y frutos de la planta del tomate, propagándose fácilmente a través de las esporas transportadas por el viento, el agua e insectos. La enfermedad suele manifestarse en periodos de alta humedad ambiental y temperaturas que oscilan entre los 17 y 22 ℃.
Los síntomas iniciales se manifiestan en hojas con áreas necrosadas rodeadas de un fieltro blanco [17], como lo muestra la Figura 3 las cuales pueden expandirse rápidamente y cubrir toda la superficie foliar. En estados avanzados de la enfermedad, las hojas tienden a marchitarse y secarse por completo.

Virus del Mosaico. El genero *Tobamovirus* es el grupo de virus al cual pertenece el virus del mosaico del tomate [18]. Los síntomas tempranos son la aparición de manchas amarillas y moteadas en las hojas, que pueden expandirse y unirse para formar un patrón de mosaico característico, los foliolos se deforman apareciendo

---

[17] Ceferino Flores, Sebastian Buono y Sergio Giorgini. *Guía de consulta Enfermedades de Tomate*. Argentina: Instituto Nacional de Tecnologia Agropecoria Yuto, 10 de nov. de 2012. 138 págs.

[18] Ana Aguado Martinez et al. *EL VIRUS DEL MOSAICO DEL TOMATE*. Ene. de 2014.

Figura 3. Tizón tardío obtenido : (a) Visita técnica, (b) Plant Village

(a) (b)

rizados, abarquillados o con aspecto filiforme. Ver Figura 4 La infección afecta directamente el crecimiento de la planta, disminuyendo el tamaño y la cantidad de frutos, ocasionando una baja producción. La severidad de los síntomas pueden verse afectados por varios factores, como la variedad de la cepa viral, la intensidad de luz, temperatura, edad de la planta y la disponibilidad de nutrientes esenciales como el nitrógeno en el suelo.

Figura 4. Virus del mosaico obtenido : (a) Visita técnica, (b) Plant Village

(a) (b)

2.2. INTELIGENCIA ARTIFICIAL.

La inteligencia artificial es un campo de investigación que comenzó en 1950 y tiene como objetivo principal la automatización de tareas que requieren razonamiento,

a menudo realizadas por seres humanos[19]. Estas tareas incluyen la capacidad de aprender de los errores, adaptarse a nuevas situaciones, comprender el lenguaje natural y la toma de decisiones. Además, la inteligencia artificial se divide en diferentes ramas, como el aprendizaje automático, el cual a su vez incluye el aprendizaje profundo. Estos algoritmos utilizan técnicas matemáticas y estadísticas para analizar datos y mejorar la capacidad de la máquina para realizar tareas específicas.

2.2.1. Aprendizaje automático. El aprendizaje automático se define como la rama de la inteligencia artificial, que se enfoca en el diseño y desarrollo de algoritmos que permiten a los computadores aprender de los datos[20], sin necesidad de ser explícitamente programadas para cada tarea. Se puede usar aprendizaje automático, cuando el problema a resolver es complejo, tiene reglas extensas y no entendibles, además, si se tiene a disposición datos para resolverlo.

Los tipos de aprendizaje dentro del *machine learning* son: el supervisado, no supervisado y por refuerzo[20]. En el aprendizaje supervisado se proporciona al algoritmo un conjunto de datos de entrenamiento, que incluyen tanto las características de entrada, como las correspondientes soluciones deseadas, llamadas etiquetas. Por otro lado, en el aprendizaje no supervisado, el algoritmo no cuenta con etiquetas, por lo que debe identificar patrones y estructuras en los datos por sí mismo. Finalmente el aprendizaje por refuerzo no cuenta con ningún tipo de dato, por ende, existe la necesidad de indicarle un *agente*, un *ambiente* y *decisiones*, para que el algoritmo escoja la mejor opción cada momento.

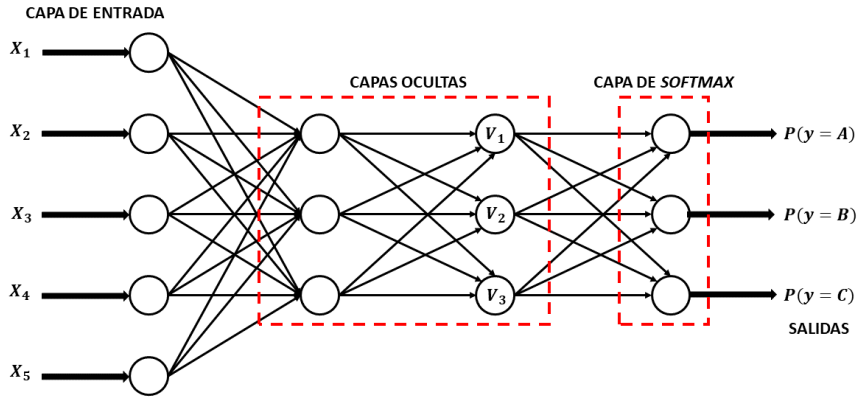
---

[19] François Chollet. *Deep Learning with Python*. 2.ª ed. NY: Manning Publications Co, 2021. 504 págs.

[20] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3.ª ed. O'Reilly Media, Inc, 2022. 1196 págs.

Redes neuronales. Las redes neuronales son un algoritmo de aprendizaje automático inspirado en el funcionamiento del cerebro humano, porque consiste en una serie de unidades interconectadas llamadas neuronas, que procesan y transmiten información[21]. Dichas neuronas se componen de algoritmos tradicionales del aprendizaje automático, como la regresión lineal y la regresión logística. Un ejemplo de red neuronal se puede ver en la Figura 5.

Figura 5. Red neuronal.



Perceptrón El perceptrón es una red neuronal simple de una sola neurona compuesto de una capa de entrada y un nodo de salida, se utiliza en la clasificación binaria de datos[22] ver Figura 6.

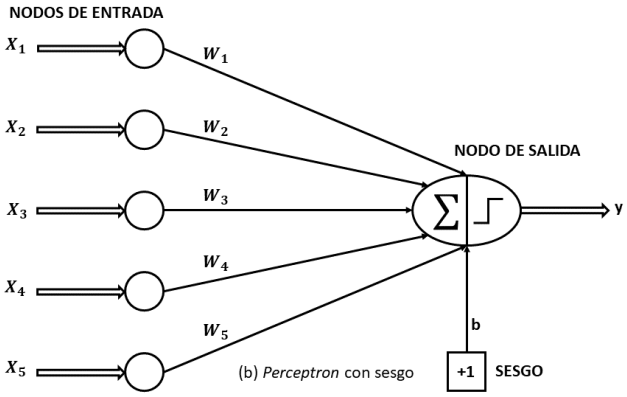
En su capa de entrada, se encuentran n características $\bar{X}$ =[x1,..xn] que se multiplican por un peso específico $\bar{W}$=[w1,...wn], generando una sumatoria de estos

---

[21] Charu C. Aggarwal. *Neural Networks and Deep Learning*. 1.ª ed. NY: Springer International Publishing AG, 2018. 512 págs.

[22] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." En: *Psychological Review* 65 (1958). Place: US Publisher: American Psychological Association, págs. 386-408. DOI: 10.1037/h0042519.

Figura 6. Perceptron.



productos. En algunos casos, la predicción tiene un factor invariante, lo que requiere la suma de un sesgo que permita captar este factor. Finalmente, el valor resultante se pasa por una función de activación, que en el caso del perceptrón es la función de *Heaviside* o la función escalón, que permite que la salida tenga un valor de uno o de cero, como lo muestra la ecuación 1.

$$\hat{y} = \gamma(\bar{W} * \bar{X} + b) = \gamma(\sum_{i=1}^{n} w_i x_i + b) \tag{1}$$

la finalidad del perceptrón es minimizar el error de predicción y esto se logra encontrando el valor mínimo de la función que describe el error, también llamado como función de perdida, la cual se describe en la ecuación 2.

$$\nabla L = \sum_{(\bar{X},y)\in n} (y - \hat{y})\bar{X} \tag{2}$$

Este proceso se realiza actualizando el vector de pesos $\bar{W}$ mediante la regla de aprendizaje del perceptrón descrita en la ecuación 3.

$$\bar{W} = \bar{W} + \alpha(y - \hat{y})\bar{X} \tag{3}$$

Backpropagation

El algoritmo de *Backpropagation* es una técnica muy popular en el campo de redes neuronales debido a que nos permite entrenar un modelo. Es un algoritmo para calcular los gradientes a partir de la regla de la cadena, para luego actualizar los pesos[23]. El proceso del algoritmo comienza con el *forward*, el cual consiste en propagar los datos de entrada a través de las capas ejecutando la ecuación 1, esta salida se convierte en la entrada de la siguiente capa y el proceso se repite hasta llegar a la capa de salida. al igual que el perceptrón, el objetivo es minimizar el error de predicción, por ende, se escoge una función de perdida de acuerdo al problema planteado.

El error se propaga hacia atrás a través de la red neuronal para ajustar los pesos, conocido como el proceso de *backward*. En el proceso de *backward*, se calcula el gradiente de la función de pérdida con respecto a los pesos en cada capa de la red neuronal utilizando la regla de la cadena. Luego, se actualizan los pesos en cada capa de la red neuronal utilizando el algoritmo del descenso del gradiente, que nos permite encontrar el mínimo global de la función de pérdida[20]. Este proceso se repite varias veces hasta que se alcanza un valor de pérdida mínimo o se cumple un criterio de parada predefinido.

El algoritmo del descenso del gradiente consiste en iniciar en un punto aleatorio de la función y calcular el gradiente en ese punto, el siguiente punto lo hallamos con la ecuación 4 .

---

[23] Kevin P. Murphy. *Probabilistic Machine Learning An Introduction*. 1.ª ed. London: The MIT Press Cambridge, Massachusetts, 2022. 856 págs.

$$x_{i+1} = xi - \alpha \frac{df}{dx} \tag{4}$$

donde α se denomina como la tasa de aprendizaje o *learning rate* que indica a qué velocidad se desciende por el gradiente. La elección adecuada de α es crucial ya que un valor demasiado pequeño puede resultar en una convergencia lenta mientras que un valor demasiado grande puede hacer que el algoritmo diverja.

El descenso del gradiente es uno de los algoritmos básicos para la actualización de pesos, sin embargo, existen otros algoritmos eficientes para realizar esa tarea como el *SGD* [19]. El algortimo que permite la velocidad de convergencia es llamado optimizador[20], dentro de los optimizadores encontramos: *momentum, Nesterov accelerated gradient, AdaGrad, RMSProp* y *Adam* y sus variantes.

El optimizador *momentum* consiste en acumular una fracción del gradiente anterior para evadir las partes planas de la curva de la función de perdida, de igual forma, el optimizador *Nesterov Accelerated Gradient* es una variante de *momentum* porque utiliza la pendiente del gradiente calculado en la dirección actual para determinar el siguiente punto. Por otro lado, el optimizador *AdaGrad* adapta la tasa de aprendizaje de cada parámetro en función de la frecuencia con la que se actualiza, similar a la idea del optimizador *RMSProp* que utiliza una tasa de aprendizaje diferente para cada parámetro en función de la magnitud de los gradientes anteriores. Finalmente el optimizador *Adam* combina las ideas de *momentum* y *RMSProp*, sin embargo, no significa que *Adam* sea el mejor optimizador, todo depende del problema que se quiera resolver.

Funciones de activación. El uso de diferentes funciones de activación nos permite simular algoritmos de aprendizaje automático, como la regresión lineal, regresión logística, entre otros[21]. Es importante que además de escoger una función de activación adecuada para el problema a resolver, se haga una correcta inicialización

de pesos $\bar{W}$, ya que se pueden presentar problemas al calcular los gradientes tales como: el desvanecimiento del gradiente, el cual consiste en que los gradientes se vuelven muy pequeños, y la explosión del gradiente que consiste en gradientes demasiado grandes[20]. Por ende, es importante realizar una correcta inicialización de pesos dependiendo de la función de activación que se use, como lo muestra la Tabla 1.

Tabla 1. Inicialización de pesos para cada función de activación.

Inicialización	**Función de activación**
Glorot	Ninguna, Tanh, Sigmoide, Softmax
He	Relu y sus variantes
LeCun	Selu

2.2.2. Redes neuronales profundas. Las redes neuronales profundas o *DNN* es un campo dentro de las redes neuronales compuestas por decenas o millones de capas de procesamiento de datos, lo que permite que los modelos puedan aprender características y patrones cada vez más complejos y abstractos[19].

Las neuronas realizan predicciones basadas en operaciones lineales, este comportamiento es utilizado en las capas densamente conectadas. Sin embargo, estas no son adecuadas para el trabajo de visión por computadora. En su lugar, se utilizan las capas convolucionales, que se basan en la operación matemática de convolución. La principal diferencia entre ambas radica en que una capa densamente conectada aprende de patrones globales, mientras que una capa convolucional aprende de patrones locales[19].

Las capas convolucionales son esenciales en la tarea de procesamiento de imágenes y visión por computadora, ya que son capaces de aprender patrones sin importar la ubicación de estos, además, de reconocer bordes, texturas en las primeras capas y características especificas en capas más profundas.

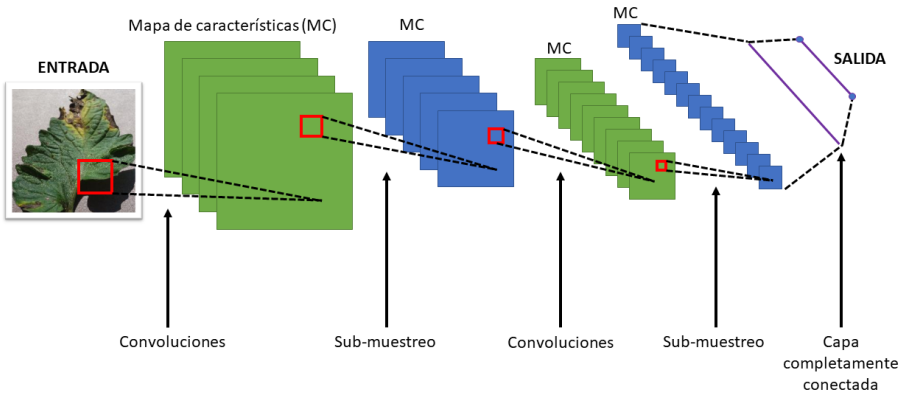
Las capas convolucionales al trabajar con imagenes, operan entre tensores de 3 dimensiones: alto de la imagen, ancho de la imagen y su número de canales, en caso de que sea una imagen a color es 3, es decir, RGB y seria 1 si es una imagen en escala de grises. Existen dos parámetros en estas capas, los cuales son el tamaño del filtro, que suele ser 3x3 o 5x5, que permiten realizar un mapa de características de la entrada, y la profundidad del mapa de características los cuales son el número de neuronas calculadas por cada convolución[19]. En algunos casos no es posible profundizar en la imagen, ya que, por cada convolución se disminuye su dimensión mediante la ecuación 5.

$$Dimension_{out} = \frac{dim_{in} - filtro_{size}}{stride} + 1 \tag{5}$$

La dimensión de salida es igual al factor de la dimensión de entrada menos el tamaño del filtro sobre el *stride* más uno, donde el stride es la distancia entre dos convoluciones consecutivas para crear el mapa de características. Para poder profundizar en una imagen existe una técnica llamada *padding* la cual nos permite agregarle ceros a la imagen, evitando la reducción en la dimensión de salida[21]. Luego de una capa convolucional siempre debe exitir una capa de *Pooling*, ya que esta nos permite realizar un submuestreo y extraer las características mas importantes de la convolución hecha anteriormente. El proceso anteriormente descrito se puede ver en la Figura 7.

2.2.3. Medidas de rendimiento. Una de las mejores formas de observar el rendimiento de un algoritmo de aprendizaje automático para un problema de clasificación, es con la matriz de confusión. La matriz de confusión es una tabla donde cada fila representa una clase verdadera y cada columna representa un clase predicha, por lo cual permite observar el número de veces que se predijo correctamente e incorrectamente cada clase de datos[20].

Figura 7. Extracción de características de una imagen.



Dentro de la matriz de confusión encontramos diferentes resultados los cuales son: los verdaderos positivos (VP), los verdaderos negativos (VN), los falsos positivos (FP) y los falsos negativos(FN). Un verdadero positivo es cuando la predicción acierta una clase positiva. Un verdadero negativo es cuando la predicción acierta una clase negativa. Un falso positivo es cuando se predice incorrectamente una clase positiva. Un falso negativo es cuando se predice incorrectamente una clase negativa.

En muchos casos la matriz de confusión proporciona información general sobre el desempeño de un algoritmo de aprendizaje automático. Por lo tanto, es importante conocer medidas de rendimiento específicas[5], tales como la exactitud (*accuracy*), la precisión positiva (*precision*), la sensibilidad (*recall*) y la medida F1 (*F1 score*)."

Accuracy. La *accuracy* es una medida de rendimiento que representa las predicciones correctas del modelo sobre el total de predicciones y tiene un rango entre 0 y 1. Ver ecuación 6

$$Accuracy = \frac{VP + VN}{N + P} \tag{6}$$

Precision. La *precision* es la medida que representa la cantidad de datos con clases positivas que se predicen correctamente como positivos. Esta medida busca principalmente que no existan falsos positivos y posee un rango entre 0 y 1. Ver ecuación 7

$$Precision = \frac{VP}{VP + FP} \tag{7}$$

Recall. El *Recall* es la medida que representa la cantidad de verdaderos positivos (VP) que se identificaron correctamente. Indica principalmente que no existan falsos negativos y tiene un rango entre 0 y 1. Ver ecuación 8

$$Recall = \frac{VP}{VP + FN} \tag{8}$$

F1 score. El *F1 score* es la medida que representa la cantidad de clases clasificadas correctamente. También se define como la medida que representa el equilibrio entre *precision* y *recall*, su rango se encuentra entre 0 y 1. Ver ecuación 9

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{9}$$

2.2.4. Transferencia de aprendizaje. La transferencia de aprendizaje o *Transfer learning* es una técnica usada en el *machine learning* que utiliza un modelo pre-entrenado en un conjunto de datos para resolver un problema. En lugar de entrenar un modelo desde cero, donde en muchos casos se considera un proceso largo y de requisitos altos de hardware, el modelo pre-entrenado se ajusta a los nuevos datos y se utiliza como punto de partida para la tarea actual[10]. El modelo pre-entrenado generalmente posee diversas capas y ha sido entrenado en grandes conjuntos de

datos, lo que permite que el modelo que se esta entrenando, logre una mejor generalización del error ante nuevos datos, además, usar un modelo pre-entrenado ahorra tiempo y recursos en la etapa de entrenamiento.

2.3. SISTEMAS EMBEBIDOS.

Los sistemas embebidos son dispositivos electrónicos programables diseñados para realizar tareas específicas[24]. Estos sistemas se encuentran en una amplia variedad de productos, desde automóviles y ascensores hasta juguetes y electrodomésticos. Los sistemas embebidos son altamente eficientes en términos de energía y recursos, ya que están diseñados para realizar tareas específicas de manera rápida y eficiente. Gracias a los avances tecnológicos, hoy en día es posible desarrollar soluciones de almacenamiento de estado sólido con capacidades suficientes para la mayoría de sistemas embebidos, lo que los hace ideales para aplicaciones específicas[25]. En resumen, los sistemas embebidos son una parte fundamental de la tecnología moderna y tienen un papel crucial en la automatización de una amplia variedad de sistemas y dispositivos. En este documento se clasifican los sistemas embebidos en tres grupos principales los cuales son; microcontroladores, SBCs y SBCs con GPU.

---

[24] Sol Gonzalez. *¿Qué es un sistema embebido?* Medium. 10 de mayo de 2021. URL: https://ardebytes.medium.com/que-es-un-sistema-embebido-9fd6d3ac94d9 (visitado 11-04-2023).

[25] *Sistemas embebidos: qué son y para qué se utilizan*. InnovaciónDigital360. 16 de dic. de 2022. URL: https://www.innovaciondigital360.com/iot/sistemas-embebidos-que-son-y-para-que-se-utilizan/ (visitado 11-04-2023).

2.3.1. Microcontroladores. Un microcontrolador es un tipo de computadora que se utiliza principalmente para el control de máquinas y sistemas electrónicos [26]. A diferencia de los microprocesadores, los microcontroladores incorporan sistemas de entrada/salida, procesadores y memoria [27]. Estos dispositivos son muy versátiles y se utilizan en una amplia variedad de aplicaciones, desde electrodomésticos hasta automóviles y aviones [28].

Los microcontroladores son especialmente útiles en la tecnología de Internet de las cosas (IoT), donde los dispositivos pueden ser monitoreados y controlados a través de Internet [28]. Para utilizar un microcontrolador, es necesario tener conocimientos básicos sobre su funcionamiento y programación. El proceso implica conectar el microcontrolador a una placa prototipo o breadboard, agregar componentes adicionales según sea necesario y, finalmente, programar el dispositivo utilizando lenguajes como C o Python. Cabe aclarar que los microcontroladores no suelen ser usados para aplicaciones de inteligencia artificial debido a las limitaciones en los recursos de computo; sin embargo, es posible desplegar algoritmos de inteligencia artificial sencillos mediante modelos matemáticos iterativos.

2.3.2. Single board computer(SBC). Una SBC, por sus siglas en inglés single board computer es un tipo de computadora embebida en una sola tarjeta, que contiene componentes como procesador, memoria RAM, almacenamiento, puertos de entrada/salida y una fuente de alimentación. Debido a que estas placas tienen un bajo costo y un tamaño compacto, son una opción popular para proyectos de elec-

---

[26] Fernando Valdes y Ramón Pallás Areny. *Microcontroladores Fundamentos y Aplicaciones con PIC*. España: Marcombo, 28 de feb. de 2007. 346 págs.

[27] Juan Carlos Vesga. *Microcontroladores motorolafreescale*. Mexico: Alpha Editorial, jun. de 2008.

[28] *Uso básico del microcontrolador para medición y control*. es. Oct. de 2022.

trónica, robótica e inteligencia artificial [29]. A diferencia de los microcontroladores las SBCs tienen un mejor desempeño en cuanto al despliegue de inteligencias artificiales, esto se debe a que tienen una mayor cantidad de recursos de computo y un sistema operativo basado en linux, que permite que la instalación de librerías y dependencias necesarias para el despliegue de una inteligencia artificial sea mas sencillo.

Existen una gran cantidad de características y especificaciones técnicas que ofrecen los SBCs. Algunos modelos se enfocan en aplicaciones específicas, tal como la Raspberry Pi Zero W que está diseñada para proyectos de IoT o la BeagleBone Black que se adapta a aplicaciones industriales. Asimismo, hay otros modelos con características más avanzadas, tales como conectividad inalámbrica Wi-Fi o Bluetooth incorporado, puertos Ethernet Gigabit o soporte para pantallas táctiles. Además, los SBCs pueden correr diversos sistemas operativos, como Linux, Android o Windows IoT Core.

2.3.3. SBC con GPU. Un SBC con GPU es una placa que integra una unidad de procesamiento gráfico (GPU), la cual se utiliza para acelerar el procesamiento de gráficos y mejorar el rendimiento general del sistema. Estas placas son más pequeñas y económicas en comparación con las computadoras de escritorio, pero pueden manejar tareas intensivas en gráficos. Uno de los SBCs con GPU más conocido es el Nvidia Jetson Nano.

Los SBC con GPU son útiles en aplicaciones que requieren gran capacidad de procesamiento gráfico, como la visión por computadora y en general la inteligencia artificial. Su tamaño compacto los convierte en una opción ideal para proyectos embe-

---

[29] Gavin Phillips. *The Best Single-Board Computers for Chrome OS and Android*. 22 de ene. de 2020. URL: https://www.makeuseof.com/tag/best-single-board-computers/ (visitado 11-04-2023).

bidos o portátiles. Además, su eficiencia energética los hace ideales para proyectos alimentados por batería o energía solar.

Los SBC con GPU a diferencia de los SCB sin unidad de procesamiento gráfico, tienen un mejor desempeño en cuanto al despliegue de modelos de inteligencia artificial, ya que al poseer una unidad de procesamiento gráfico es posible realizar una mayor cantidad de procesos en menor tiempo, lo cual se traduce a un menor tiempo de inferencia.

2.3.4. Comparación. A continuación se presenta la Tabla 2 que muestra la comparativa entre microcontroladores, SBC (Single Board Computers) y SBC con GPU (Graphics Processing Unit):

Tabla 2. Comparativa entre microcontroladores, SBC y SBC con GPU.

Características	**Microcontroladores**	**SBC**	**SBC con GPU**
Procesamiento de datos	Bajo a moderado	Moderado a alto	alto
Núcleos de procesamiento	1 - 2	2 - 8	4 - 16
Velocidad de reloj	Hasta 200 MHz	Hasta 2.5 GHz	Hasta 3.5 GHz
Memoria RAM	Hasta 512 KB	Hasta 16 GB	Hasta 32 GB
Almacenamiento	Hasta 2 MB	Hasta varios TB	Hasta varios TB
Conectividad	Limitada	Amplia variedad de opciones	Amplia variedad de opciones
Consumo de energía	Bajo	Moderado	Alto
Precio	Bajo	Moderado	Moderado a alto
Uso típico	Controladores de dispositivos, sistemas embebidos, sensores	Servidores web, proyectos de IoT, multimedia	Inteligencia artificial, renderización 3D

Como se puede observar en la Tabla 2, los microcontroladores tienen una capacidad de procesamiento de datos menor que los SBC y SBC con GPU, lo que los hace más adecuados para tareas sencillas y específicas, como controladores de dispositivos y sistemas embebidos. Los SBC tienen una capacidad de procesamiento moderada a alta y una amplia variedad de opciones de conectividad, lo que los hace adecuados para proyectos de IoT, servidores web y multimedia. Los SBC con GPU tienen una capacidad de procesamiento mucho mayor que los SBC y son ideales para apli-

caciones de inteligencia artificial y renderización 3D. Sin embargo, su consumo de energía y precio pueden ser más altos que los de los microcontroladores y SBC convencionales.

2.4. PROGRAMAS PARA EL DISEÑO 3D DE CÓDIGO ABIERTO.

El software de código abierto, son programas cuyo código fuente está disponible para su acceso y modificación libremente por cualquier persona. Esto permite que los usuarios colaboraren con otros desarrolladores para mejorarlo, dependiendo así de revisiones de distintas personas o comunidades para su aprobación, suele ser más económico y flexible.

Además, el uso de software libre en instituciones educativas de nivel superior también fomenta la creatividad y la innovación. Los estudiantes pueden aprender a programar y a modificar el código fuente de aplicaciones ya existentes, lo que les permite desarrollar nuevas funcionalidades y soluciones a problemas específicos. Esto no solo es una oportunidad para que los estudiantes desarrollen habilidades técnicas valiosas, sino que también les enseña a trabajar en equipo y a colaborar con otros desarrolladores.

CAD.

Diseño Asistido por Computador o sus siglas en inglés “CAD”, involucra el uso de la tecnología de computación, reemplazando los procedimientos manuales en el diseño. Los programas de software CAD permiten observar, diseñar, crear y simular de forma que el usuario tenga más claro el funcionamiento de su diseño[30].

---

[30] Martinez Sarmiento Edgar Alexis. “DESARROLLO DE UN SOFTWARE PARA EL DISEÑO ASISTIDO DE EJES SOMETIDOS A CARGAS ESTÁTICAS Y DINÁMICAS”. Tesis doct. Quito: Escuela Politécnica Nacional, 2016. 179 págs.

BLENDER.

Es un Software de código abierto, desarrollado para la creación de piezas 3D, el cual permite el modelado, montaje, animación, simulación, renderizado, composición y seguimiento de movimiento. Ofreciendo a los usuarios usar comandos de Python para la personalización y creación de nuevas herramientas. Es un software multiplataforma el cual corre en sistemas operativos como Windows, Linux y MAC os, el cual se adapta a todo tipo de personas beneficiándose de sus actualizaciones y desarrollo constante.[31]

OPENSCAD.

Es un software de código abierto, destinado a la creación de modelos 3D, adecuados para usuarios profesionales que vean favorable la programación, debido a que el modelado es basado en scripts, los cuales utilizan su propio lenguaje, excluyendo de esta manera el uso de un mouse. Proporciona 2 técnicas de modelado; la primera está basada en la geometría solidad (CSG) y como segunda tecnica encontramos la extracción de bocetos 2D, teniendo la posibilidad de leer distintos parámetros de diseño de archivos DXF. Disponible para sistemas operativos como Windows, Linux, MAC os.

FREECAD.

Freecad es un software de modelado 3D paramétrico, completamente gratuito y de código abierto con la licencia *General Public Licence (GPL2+)*. Esta diseñado principalmente para el uso en ingeniería y diseño de productos. Permite la creación de bocetos en 2D que posteriormente se transformara en modelos 3D. El boceto permite crear distintos tipos de formas geométricas en la interfaz, controlando la precisión

---

[31] *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. 2023. URL: https://www.blender.org/ (visitado 20-03-2023).

de las dimensiones, posiciones, formas, etc. Además, permite el acceso al historial del modelado. Es un software disponible en la mayoría de los sistemas operativos, multiplataforma Windows, Mac, Linux, siendo compatible con distintos formatos tales como: STEP, .IGES, .SVG, .OBJ, entre otros.

2.5. IMPRESIÓN 3D.

La impresión 3D hecha por filamento *(FDM)* por sus siglas en ingles *fused deposition modeling*, se realiza mediante la superposición de capas sucesivas de material permitiendo la creación de los diversos objetos, para este caso se utilizó un filamento PLA de 1.75 mm ya que es el usado por las impresoras 3D en los laboratorios de la escuela de Mecánica, además, este no es toxico y ofrece gran resistencia a la humedad.

Para realizar una correcta impresión 3D es necesario verificar que los diseños no tengan ángulos rectos, los sólidos internos no contengan huecos, una orientación de la pieza que permita soportar cargas y la verificación del tamaño del archivo STL, debido a que este formato construye los cuerpos mediante mallas de triángulos, de forma que entre más triángulos mejor resolución tendrá el objeto que se imprima, pero a su vez entre mayor tamaño de archivo, mayor procesamiento requiere la impresora 3D.

2.5.1. Ácido poli-lactico El ácido Poli-lactico o más conocido como PLA, se consigue por medio de recursos renovables como el almidón de maíz, raíces de tapioca o caña de azúcar. Para este caso, el maíz se muele con el fin de apartar el almidón y mezclarlo con distintos ácidos. Con esta mezcla el almidón rompe todo tipo de azúcar, ocasionando que la fermentación de este produzca el ácido Láctico, conocido

como PLA.[32]

Este material es de gran aceptación para la fabricación de diversas piezas porque es muy fácil de emplear. Se considera un material policristalino que admite una temperatura de fusión de 180 ℃, relativamente bajo en comparación del filamento ABS[33]. Es un material brillante, resistente a la humedades y algunas grasas, presentan barreras de sabor y de olor típicamente a la de polietileno tereftalato.

2.5.2. acrilonitrilo butadieno estireno El Acrilonitrilo Butadieno Estireno conocido como ABS, es un material termoplástico altamente utilizado en la industria, se caracteriza por su excelente resistencia a bajas temperaturas y ser ligero. Su principal uso se encuentra en decoraciones y juguetes famosos como los LEGO[34]. Es un material con propiedades únicas, las cuales permiten realizar la impresión de piezas 3D. Es seleccionado por su alta resistencia a impactos, sus superficies lisas, su capacidad de ser soldado químicamente con acetona y de soportar temperaturas entre los -20 ℃ y los 80 ℃.

---

[32] Lucás C. *Guía completa: el filamento PLA en la impresión 3D*. 3Dnatives. 18 de ago. de 2019. URL: https://www.3dnatives.com/es/guia-filamento-pla-en-la-impresion-3d-190820192/ (visitado 23-01-2023).

[33] Carlos Ruiz. *Guía completa: el filamento PLA en la impresión 3D*. AUROS Colombia. 1 de ene. de 2020. URL: https://www.auros.com.co/guia-completa-filamento-pla-la-impresion-3d/ (visitado 20-03-2023).

[34] Lucás C. *El filamento de ABS en la impresión 3D*. 3Dnatives. 6 de jun. de 2019. URL: https://www.3dnatives.com/es/filamento-de-abs-impresion-3d-06062019/ (visitado 23-01-2023).

3. BASE DE DATOS.

3.1. OBTENCIÓN DE LAS IMÁGENES.

Es importante destacar que una de las mayores complejidades en el desarrollo de sistemas basados en aprendizaje autónomo, es la obtención de una base de datos adecuada para su entrenamiento. En este sentido, la comunidad científica ha logrado desarrollar proyectos de este tipo gracias al uso de bases de datos como la de Plant Village[35]. Esta base de datos se generó de la siguiente manera: primero se infectaron los cultivos con enfermedades comunes y cuando empezaron a mostrar síntomas, técnicos expertos recolectaron hojas afectadas, las cuales fueron colocadas sobre una superficie gris o negra en la misma orientación para finalmente tomar las fotografías. Entre las nueve enfermedades con las que se inocularon los cultivos de tomate, se encuentran el tizón temprano, la mancha foliar, la mancha anillada, el moho en la hoja, la mancha bacteriana, el tizón tardío, el virus de enrollamiento de hoja amarilla, el virus del mosaico y el ácaro; de estas nueve enfermedades se seleccionaron el tizón temprano, el tizón tardío y el virus del mosaico, además de incluir la hoja sana del tomate, ver Figura 8, con un total de 4873 imágenes etiquetadas RGB de 256 píxeles x 256 píxeles con su respectivo nombre de enfermedad, ver en la Tabla 3.

En colaboración con el ingeniero agrónomo Jairo Rueda, se ha logrado ampliar la base de datos con imágenes capturadas en invernaderos afectados por las diferentes enfermedades seleccionadas. Los invernaderos seleccionados se encuentran ubicados en los municipios de Betulia, Zapatoca y la Mesa de los Santos. Sin em-

---

[35] David Hughes, Marcel Salathé et al. “An open access repository of images on plant health to enable the development of mobile disease diagnostics”. En: *arXiv preprint arXiv:1511.08060* (2015).

Figura 8. Clases de la base de datos: (a) Hoja sana, (b) Tizón tardío (c) Tizón temprano, (d) Virus del mosaico

(a) (b) (c) (d)

Tabla 3. Banco de imágenes de Plant Village.

Clase	**Total de imágenes**
Tizón tardío	1909
Tizón temprano	1000
Virus del mosaico	373
Hoja sana	1591

bargo, uno de los mayores desafíos fue identificar las enfermedades sin ninguna otra infección presente. En algunos casos, se observaron hojas que presentaban dos o incluso tres enfermedades simultáneamente, por lo que la base de datos se compone de las hojas más representativas de cada enfermedad.

Durante el desarrollo del proyecto, se llevaron a cabo cuatro visitas técnicas con el objetivo de recolectar hojas infectadas para posteriormente tomar fotografías con condiciones de luz natural y ambiente controlado, utilizando un porta muestras, tal como se muestra en la Figura 9. La base de datos se recopiló en un horario de 2 pm a 6 pm y consta de un total de 1284 imágenes etiquetadas RGB de 256 px x 256 px obtenidas de las diferentes visitas técnicas, como se detalla en la Tabla 4.

La Tabla 5 muestra el total de imágenes obtenidas por medio de las visitas técnicas y la base de datos de Plant Village. Se obtuvieron 2136 imágenes para la enfermedad Tizón tardío, 1283 imágenes para Tizón temprano, 811 imágenes para el virus del mosaico y 1927 imágenes para hojas sanas.

Figura 9. Imágenes tomadas: (a) Imagen tomada sin el porta muestras, (b) Imagen tomada con el porta muestras

(a) (b)

Tabla 4. Banco de imágenes de las visitas técnicas.

Clase	Porta muestras	Imágenes	Total de imágenes
Tizón tardío	SI NO	152 75	227
Tizón temprano	SI NO	160 123	283
Virus del mosaico	SI NO	204 234	438
Hoja sana	SI NO	244 92	336

3.2. PROCESAMIENTO DE IMÁGENES.

Una vez obtenida la base de datos de las diferentes enfermedades y hojas sanas, se procedió a dividirla en tres conjuntos de datos diferentes: entrenamiento, validación y prueba. Se decidió realizar esta división en proporciones de 60 %, 20 % y 20 %, respectivamente[20].Se consideró que el conjunto de datos de entrenamiento contara con un porcentaje considerable de las imágenes totales para permitir una mejor generalización del error al modelo. De esta forma, se obtuvo una cantidad prudente de imágenes para los conjuntos de validación y prueba, lo que permitió intentar evitar un sesgo en la predicción del modelo.

Para llevar a cabo la división de la base de datos en conjuntos de entrenamiento, validación y prueba, se realizó la separación de cada categoría de enfermedad,

Tabla 5. Banco de imágenes obtenido.

Clase	Origen	Imágenes	Total de imágenes
Tizón tardío	Plant Village Visitas técnicas	1909 227	2136
Tizón temprano	Plant Village Visitas técnicas	1000 283	1283
Virus del mosaico	Plant Village Visitas técnicas	373 438	811
Hoja sana	Plant Village Visitas técnicas	1591 336	1927

considerando las imágenes obtenidas tanto de Plant Village como de las visitas técnicas utilizando el porta muestras y con luz natural. Posteriormente, se unieron los tres grupos de imágenes en los respectivos conjuntos de datos, asegurando que se mantenga una distribución equitativa y estratificada de las muestras en cada conjunto de datos.

El proceso se realizó utilizando el código de Division_datos, el cual se encuentra disponible en el Anexo 4 y fue ejecutado en un entorno Ubuntu. Por lo tanto, el conjunto de datos cuenta con 3699 imágenes para entrenamiento, 1225 imágenes para validación y 1232 imágenes para prueba, divididas como lo muestra la Tabla 6.

Tabla 6. División del conjunto de datos.

Clase	Conjunto de datos	Número de imágenes
Tizón tardío	Entrenamiento Validación Prueba	1283 426 427
Tizón temprano	Entrenamiento Validación Prueba	770 256 257
Virus del mosaico	Entrenamiento Validación Prueba	488 160 163
Hoja sana	Entrenamiento Validación Prueba	1158 383 385

Para mejorar el desempeño del modelo de aprendizaje profundo, es esencial enriquecer el conjunto de entrenamiento de la base de datos, mediante la realización

de diversas transformaciones en las imágenes. En este sentido, se empleó el framework TensorFlow y específicamente, el objeto *ImageDataGenerator*, que permitió aplicar transformaciones tales como volteo vertical, volteo horizontal, rango de rotación y rango de zoom, este procesaamiento se realizó usando el codigo Aumento_datos. De esta manera, se generó un conjunto de datos más diverso, lo que favoreció la capacidad del modelo para generalizar patrones y mejorar su predicción[19]. Se aumentó el conjunto de entrenamiento en un factor de dos, en total se cuenta con una Base de datos de 7398 imágenes etiquetadas, evidenciadas en la Tabla 7.

Tabla 7. División del conjunto de datos.

Clase	**Conjunto de datos**	**Número de imágenes**
Tizón tardío	Entrenamiento	2566
	Validación	426
	Prueba	427
Tizón temprano	Entrenamiento	1540
	Validación	256
	Prueba	257
Virus del mosaico	Entrenamiento	976
	Validación	160
	Prueba	163
Hoja sana	Entrenamiento	2316
	Validación	383
	Prueba	385

4. RED NEURONAL IMPLEMENTADA.

La implementación de la red neuronal se llevó a cabo utilizando Google Colab, una plataforma en línea que permite ejecutar código en la nube mediante Jupyter Notebooks. Esta herramienta resulta especialmente útil para proyectos de aprendizaje automático, ya que brinda acceso gratuito, aunque limitado, a recursos informáticos potentes como GPUs y TPUs. Además, se integró fácilmente con Google Drive, donde se almacenaron los conjuntos de datos de entrenamiento, validación y prueba.
Se utilizó el objeto *ImageDataGenerator* de Keras para cargar los conjuntos de datos. En la definición del objeto se especificó la ruta donde se encuentra almacenado el conjunto de datos, el tamaño de las imágenes que es 256 píxeles x 256 píxeles y el tamaño del *batch* que es 32, lo que indica que el modelo recibe grupos de 32 imágenes hasta completar el total de imágenes. Para el conjunto de entrenamiento se mezclaron los datos, mientras que para los conjuntos de validación y prueba no se realizó esta mezcla. Finalmente, se definió el tipo de etiqueta como *categorical* o *one-hot encoding*, lo que convierte cada etiqueta en un vector binario del mismo tamaño que el número total de clases, en el cual todos los elementos son cero excepto uno. Los códigos se encuentran en el Anexo 4.

4.1. ARQUITECTURA DE LA RED NEURONAL.

La red neuronal propuesta es una red neuronal de jerarquía reducida[5], lo que significa que tiene un número limitado de capas en comparación con otras redes neuronales sin comprometer su capacidad de generalización del error. Al reducir el número de capas, se reduce el número de parámetros de la red y se obtiene una red más eficiente y con menor tamaño, lo que la hace adecuada para su uso en dispositivos con recursos limitados. La reducción del número de capas también puede ayudar a

prevenir el *overfitting* o sobre entrenamiento de la red y mejorar su capacidad de generalización, ya que se reduce la complejidad del modelo y se evita la memorización de los datos de entrenamiento.

La capa de entrada se compone de imágenes con dimensiones de 256 píxeles por 256 píxeles y 3 canales RGB. La primera capa después de la entrada es una capa de *BatchNormalization*, que normaliza los datos para facilitar las operaciones en la siguiente capa. La capa oculta es la capa de procesamiento que se compone de tres bloques repetitivos, cada uno de los cuales tiene una capa convolucional con 32 filtros que van disminuyendo por bloque, un tamaño de filtro de 3x3 y sin *padding*. Los pesos se inicializan con *he*. A continuación, se encuentra una capa de *MaxPooling* con un tamaño de filtro de 2 x 2 con *stride* de uno y no tiene *padding*. Posteriormente, se agrega una capa de *BatchNormalization*. luego una capa de una función de activación *relu*, de acuerdo a lo especificado en la Tabla 1.

La siguiente capa es una capa de *Flatten* o acoplamiento que se utiliza para aplanar los datos de la capa anterior y pasarlos a una capa densa de 128 neuronas con una función de activación *relu*. Los pesos de esta capa se inicializan con *he*. A continuación, se agrega una capa de *BatchNormalization* para normalizar los datos y una capa de *Dropout* del 20 % para reducir el *overfitting*.

Finalmente, se encuentra la capa de salida que consiste en una capa densa de 4 neuronas, que representan las 4 clases del modelo, con una función de activación *Softmax*, los pesos de esta capa se inicializan con *glorot*. El número total de parámetros fue de novecientos mil, distribuidos como lo muestra la Tabla 8. El código Inteligencia_Artificial muestra de forma detalla la red neuronal de jerarquía reducida.

Tabla 8. Parámetros de la red neuronal propuesta.

Parámetros	**Número de parámetros**
Total parámetros	929672
Parámetros entrenables	929298
Parámetros no entrenables	374

4.2. CALLBACKS.

Los *callbacks* son una herramienta muy útil en el aprendizaje automático, ya que permiten a los desarrolladores personalizar el proceso de entrenamiento de un modelo de acuerdo a sus necesidades específicas. Considerados como métodos de regularización del modelo para evitar el *overfitting*, existen diferentes *callbacks* que nos permite usar Keras, en este proyecto se usaron *EarlyStopping* y *ReduceLROPlateau*.

En Keras, podemos implementar esta técnica utilizando el objeto de *EarlyStopping*. Al definir este objeto, debemos especificar la variable que se va a monitorear, que en este caso es la función de pérdida de la validación. Además, podemos establecer el número de épocas que el modelo esperará antes de detener el entrenamiento si no se produce mejora en la función de pérdida. En este proyecto, se estableció un valor de 5 épocas antes de detener el entrenamiento, despúes de diversas pruebas realizadas. También se puede configurar el objeto para que restaure los pesos del modelo a la mejor época durante el entrenamiento. De esta manera, se asegura que el modelo final tenga el mejor desempeño posible en datos no vistos durante el entrenamiento.

El *ReduceLROPlateau* también es una técnica de regularización que puede mejorar el rendimiento del modelo en el entrenamiento al reducir el *learning rate*, cuando la variable que se monitorea deja de mejorar. En el objeto de Keras, es necesario definir la variable que se va a monitorear, que en este caso es la función de pérdida de la validación. Además, se debe especificar el factor de reducción de la tasa de aprendizaje, que es 0.2, y el número de épocas que esperará antes de reducir la tasa de aprendizaje, que son 2 épocas. Cabe aclarar que este número de épocas fue definido mediante prueba y error.

4.3. ENTRENAMIENTO.

Antes de iniciar el entrenamiento del modelo, es necesario compilarlo mediante el método *compile* de Keras. Este método permite definir el optimizador, la función de pérdida y la métrica a seguir durante el entrenamiento. En este proyecto se utilizó el optimizador *Adam* con un *learning rate* de 0.001. La función de pérdida elegida fue *categorical crossentropy*, que se utiliza cuando las etiquetas están codificadas en formato *one-hot encoding*, como en este caso. Además, se escogió la métrica de exactitud *accuracy*, que es la medida estándar para evaluar la eficacia de un modelo de clasificación.

Una vez que se han definido la arquitectura del modelo y se ha compilado, se procede a realizar el entrenamiento con el método *fit* de Keras. En este método se debe especificar el conjunto de entrenamiento, el tamaño del *batch*, el conjunto de validación, los *callbacks* a utilizar y el número de épocas que lo define *EarlyStopping*.

4.4. RED MOBILENETV2.

MobileNetV2 es una arquitectura de red neuronal convolucional en forma de “V” para lograr una buena precisión en tareas de clasificación de imágenes y detección de objetos, basada en la idea de utilizar bloques residuales que permiten que la información fluya directamente a través de las capas, en lugar de pasar por capas adicionales de procesamiento, cada bloque residual consta de varias capas convolucionales y de agrupación, seguidas de una conexión de salto residual que permite que la información se salte una o varias capas.

En términos de su estructura, MobileNetV2 consta de 53 capas que contienen bloques residuales, además de capas convolucionales, convoluciones *Depthwise*, capas de funciones de activación *relu*, *BatchNomalization* y *Zero padding*. La combinación de capas convolucionales y convoluciones *Depthwise* permiten la reducción

del número de parámetros y operaciones.

El código Inteligencia_Artificial_MobileNetV2 describe un modelo de MobileNetV2, donde las imágenes de entrada al modelo son de 224 píxeles x 224 píxelesx x 3 canales RGB. La red MobileNetV2 utiliza pesos pre-entrenados a partir de la base de datos de ImageNet, lo que permite al modelo aprovechar el conocimiento previo adquirido en un conjunto de datos muy grande y diverso. Por ende no se entrenaron esas capas alivianando el proceso de entrenamiento. Después de las capas ocultas, sigue una capa *Flatten* y una capa densa de 128 neuronas con función de activación *relu*, que inicializa los pesos con el *he*. Se añade una capa de *BatchNormalization* para normalizar los datos y una capa de *Dropout* con una probabilidad de 0.2 para evitar el overfitting del modelo. Finalmente, se agrega una capa densa con 4 neuronas correspondientes a las clases con una función de activación *Softmax* y una inicialización de pesos *he*.

Las configuraciones de los *callbacks* y los métodos *compile* y *fit* fueron las mismas, con un total de 10 millones de parámetros, las especificaciones las encontramos en la Tabla 9.

Tabla 9. Parámetros de la red neuronal MobileNetV2.

Parámetros	**Número de parámetros**
Total parámetros	10287300
Parámetros entrenables	8029060
Parámetros no entrenables	2258240

4.5. RED RESNET50.

La arquitectura de ResNet50 se basa de la premisa de que el aumento de la profundidad de una red neuronal, debería mejorar su capacidad para aprender y representar características de las imágenes. Para abordar este problema, ResNet50 utiliza una arquitectura de *residual learning* que permite que las capas en la red sal-

ten ciertas capas para formar conexiones residuales. La arquitectura de ResNet50 consta de 50 capas compuestas de bloques residuales, capas de *Zero padding* para aumentar el tamaño de la imagen y capas de *BatchNormalization*.

El tamaño de las imágenes de entrada al modelo es de 224 píxeles x 224 píxeles x 3 canales RGB. En cuanto a la arquitectura del modelo, se utilizan pesos pre-entrenados a partir de la base de datos de ImageNet.

En este caso, se decidió utilizar solo las capas ocultas de la arquitectura ResNet50, y agregarle una capa de *Flatten* seguida de una capa densa de 128 neuronas con función de activación *relu*, que inicializa los pesos con el *he*. Posteriormente, se añade una capa de *BatchNormalization* para normalizar los datos y una capa de *Dropout* con una probabilidad de 0.2 para evitar el *overfitting* del modelo. Finalmente, se agrega una capa densa con 4 neuronas correspondientes a las clases de las diferentes enfermedades, con una función de activación *Softmax* y una inicialización de pesos *he*.

Durante el entrenamiento, se define que solo se entrenarán las últimas capas del modelo ResNet50 y no todo el modelo, ya que los pesos de las capas de convolución ya han sido pre-entrenados. Esto permite que el modelo se adapte a las características específicas de la tarea en cuestión sin perder el conocimiento previo adquirido en la base de datos de ImageNet. Además, se mantuvieron las mismas configuraciones de los *callbacks* y los métodos *compile* y *fit*. El total de parámetros es alrededor de 36 millones, la distribución la encontramos en la Tabla 10. El código Inteligencia_Artificial_ResNet50 describe el modelo de *transfer learning* descrito anteriormente.

Tabla 10. Parámetros de la red neuronal ResNet50.

Parámetros	**Número de parámetros**
Total parámetros	36433929
Parámetros entrenables	12845956
Parámetros no entrenables	23587968

4.6. RED VGG19.

La red VGG19 es una red neuronal convolucional caracterizada por tener una estructura de capas uniforme y una profundidad relativamente grande. Al modelo ingresan imágenes de 224 píxeles x 224 píxeles x 3 canales RGB, donde sus pesos han sido pre-entrenados con las imágenes de ImageNet, lo que lo hace muy efectivo en tareas de clasificación de imágenes.

El código Inteligencia_Artificial_VGG19 muestra la arquitectura de un modelo VGG19, que se compone de 19 capas de bloques repetitivos, compuestos de capas convolucionales con tamaños de filtro 3 x 3 y *stride* de uno, seguidas de capas de *Max-Pooling*. Durante el entrenamiento, los métodos *compile* y *fit* fueron los mismos a los modelos anteriores, al igual que las configuraciones de los *callbacks*, además para facilitar el proceso solo se entrenaron las capas ocultas y se agregaron una capa *Flatten* y una capa densa de 128 neuronas con función de activación *relu*, con pesos incializados con *he*, una capa de *BatchNormalization*, una capa de *Dropout* con una probabilidad de 0.2 y una capa densa con 4 neuronas correspondientes a las clases del problema, con una función de activación *Softmax* y una inicialización de pesos *he*. El total de parámetros son 23 millones, los cuales se distribuyen según la Tabla 11.

Tabla 11. Parámetros de la red neuronal VGG19.

Parámetros	**Número de parámetros**
Total parámetros	23236804
Parámetros entrenables	3212164
Parámetros no entrenables	20024640

5. IMPLEMENTACIÓN DEL MODELO.

Para implementar una red neuronal en los sistemas embebidos seleccionados y con el objetivo de hacer la experiencia del usuario más sencilla y cómoda, se diseñó una interfaz utilizando Python, de manera que fuera compatible con los sistemas operativos instalados en cada una de las tarjetas. Por otro lado, también fue necesario crear un encapsulado que proteja y soporte los dispositivos requeridos para el funcionamiento del sistema de detección. En este capítulo se mencionan todos los aspectos necesarios para llevar a cabo la implementación del modelo.

5.1. DISPOSITIVOS Y PERIFERICOS.

Antes de realizar la implementación fue necesario seleccionar las tarjetas y dispositivos que se ajustaran a los objetivos del proyecto. Gracias a que contamos con el apoyo de los grupos de investigación DicBot y CEMOS, se tuvo acceso a una Jetson Nano de 2GB, una Raspberry PI 4B, una pantalla táctil Raspberry de 7 pulgadas y una *Power Bank*, sin embargo, a lo largo del desarrollo se tuvieron que realizar cambios en el hardware para poder alcanzar los objetivos del proyecto. Los elementos usados en la implementación final son los siguientes:

NVIDIA Jetson Nano de 4 GB: Es una SBC que cuenta con una unidad de procesamiento gráfico (GPU), estas características la hacen una tarjeta idonea para el desarrollo del proyecto, evidenciada en la Figura 10. Inicialmente se realizaron pruebas usando la Jetson Nano de 2GB, pero debido a sus limitaciones de procesamiento y la falta de documentación para esta tarjeta hoy en día, se tuvo que adquirir de forma externa una Jetson Nano de 4GB, la cual tiene más capacidad y sus características nos permiten instalar librerías y dependencias necesarias para el despliegue. Las especificaciones de la tarjeta se muestran en la Tabla 12.

Tabla 12. Especificaciones de la NVIDIA Jetson Nano de 4 GB.

Características	**Especificaciones**
Procesador	NVIDIA Quad-core ARM Cortex-A57X
GPU	NVIDIA Maxwell de 128 núcleos
Memoria RAM	4GB LPDDR4
Puertos	USB 3.0, HDMI, Gigabit Ethernet
Aplicaciones	Robótica, Inteligencia Artificial, IoT
Velocidad de reloj	1.43 GHz
Consumo de potencia	10 W

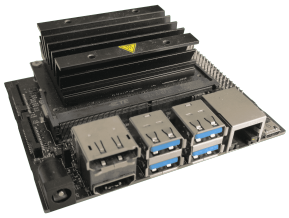
Raspberry PI 4B: Es una placa de desarrollo de bajo costo y tamaño compacto con una gran capacidad de procesamiento, ver Figura 10, también cuenta con conectividad inalámbrica y por cable, además de tener puertos para pantalla y cámaras, cuenta también con puertos GPIO de 40 pines para conexión de periféricos y expansión de funcionalidades. Con su bajo consumo de energía y su flexibilidad para ejecutar múltiples sistemas operativos, la Raspberry Pi 4B fue una de las tarjetas seleccionadas. Las características de esta tarjeta se muestran en la Tabla 13.

Tabla 13. Especificaciones de la Raspberry PI 4B.

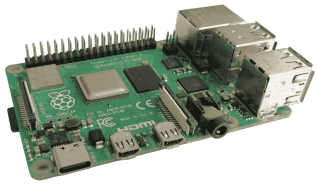
Características	**Especificaciones**
Procesador	Broadcom BCM2711 de cuatro núcleos a 1.5 GHz
Memoria RAM	4 GB
Puertos	Gigabit Ethernet, puertos USB 3.0 y 2.0, Puerto HDMI, Puerto para cámara
GPIO	GPIO de 40 pines para conexión de periféricos y expansión de funcionalidades
Aplicaciones	Robótica, Inteligencia Artificial, IoT
Velocidad de reloj	1.5 GHz
Consumo de potencia	3.5 W a 7.5 W

Pantalla: Se dispuso de una Pantalla táctil capacitiva para Raspberry de 7 pulgadas, ver Figura 11.

Figura 10. Tarjetas de desarrollo: (a) NVIDIA Jetson Nano 4GB, (b) Raspberry PI 4B



(a)



(b)

Cámara: Inicialmente se usó la cámara Raspberry PI v2, pero luego de realizar pruebas, debido a que sus filtros en el lente perturbaron el modelo, se optó por usar una cámara USB de 8 megapixeles que se adquirió de forma externa. Ver Figura 11.

Ventilador: Se utilizó un ventilador de 3 pines a 5V, que permitiera refrijerar la Jetson Nano, debido a que por la alta exigencia en procesamiento se eleva su temperatura. Ver Figura 11.

Figura 11. Periféricos: (a) Pantalla táctil, (b) Cámara, (c) Ventilador

(a)

(b)

(c)

Banco de baterías: Se dispuso de un banco de baterías compuesto por 4 baterías de litio 18650 de 3.7 V a 2600 mA. Puede entregar 5V con una corriente máxima de salida de 5A. Ver Figura 12.

Figura 12. Banco de baterías

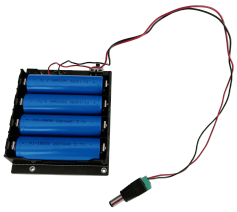
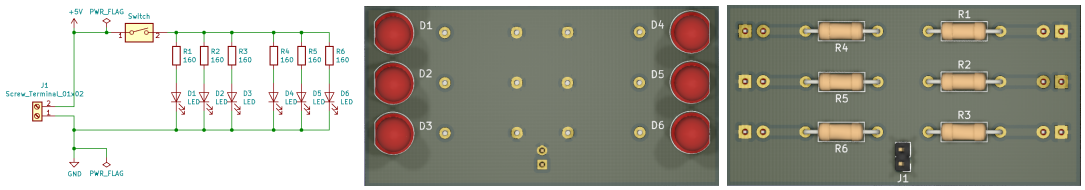


Figura 13. Sistema de iluminación: (a) Esquemático de la PCB, (b) PCB vista frontal, (c) PCB vista trasera



(a) (b) (c)

Sistema de iluminación: Con el fin de garantizar una iluminación controlada dentro del porta muestras, se diseñó una placa de circuito impreso (PCB) por sus siglas en inglés. Dicha PCB es un circuito compuesto por LEDs blancos que son encendidos mediante un interruptor dispuesto en un costado de la carcasa, cabe aclarar que el circuito de iluminación es alimentado usando los pines GPIO de la SBC montada en el sistema.

El circuito está compuesto por seis LEDs de color blanco y seis resistencias de 160 Ω conectadas a 5 V, como se mencionó anteriormente el circuito es encendido mediante un interruptor, ver Figura 13. Los documentos y archivos correspondientes a este sistema de iluminación están contenidos dentro del Anexo 2.

5.2. ALGORITMO DE DESPLIEGUE.

Para el despliegue de un modelo de TensorFlow en un sistema embebido, se debe realizar un proceso detallado y cuidadoso para garantizar la funcionalidad y eficien-

cia del sistema. Para ello se llevaron a cabo los siguientes pasos:

- **Optimización del modelo**:Lo primero es asegurarse de que el modelo de TensorFlow esté optimizado para su implementación en un sistema embebido. Para ello, se deben reducir el tamaño del modelo, eliminar capas y operaciones innecesarias.

- **Selección de Hardware:** Es importante seleccionar una plataforma de hardware adecuada para su implementación. Se deben tener en cuenta las especificaciones de la plataforma, como la memoria, la velocidad de procesamiento y los requisitos de energía. Como se mencionó anteriormente para el desarrollo fueron seleccionadas la NVIDIA Jetson Nano de 4GB y la Raspberry PI 4B.

- **Configuración del entorno de ejecución en la tarjeta:** Se debe configurar el entorno de ejecución de TensorFlow en la plataforma. Esto implica instalar las herramientas, librerías y dependencias necesarias para configurar las variables de entorno apropiadas.

- **Ejecución del modelo:** Una vez instaladas las dependencias, el modelo está listo para ser desplegado, para ello es necesario un *script* que lo ejecute. En este caso se construyó una interfaz de usuario que por medio de la cámara instalada en el sistema, captura la foto que ingresará al modelo, además, dicha interfaz ajusta el tamaño de la imagen capturada al requerido por el modelo desplegado, finalmente dicha interfaz muestra el resultado de la inferencia.

- **Pruebas y ajustes:** Finalmente se realizaron pruebas y ajustes que fueron necesarias para asegurar que el sistema funcione correctamente.

5.2.1. Despliegue en JETSON NANO. Antes de realizar el despliegue del modelo en la tarjeta, es necesario instalar la imagen que contenga el sistema operativo

adecuado para la misma. Para este proyecto se usó una imagen que contiene un sistema operativo basado en linux, con versión de JetPack 4.6.0 que contiene las librerías necesarias para el despliegue del modelo, tales como OpenCV 4.5.3 y Tensorflow 2.4.1. Cabe aclarar que el JetPack es una plataforma de desarrollo integral de NVIDIA que proporciona un conjunto de herramientas y bibliotecas para crear aplicaciones de inteligencia artificial.

En cuanto a las dependencias necesarias para el despliegue, debido a que la imagen que se usó ya contenía python 3.6.9, OpenCV y tensorflow, únicamente se instaló PyQt5, necesario para realizar la implementación del modelo en la interfaz de usuario. Además, la Jetson Nano fue configurada con una memoria *Swap* de 5 GB, la memoria *Swap* se utiliza como una extensión de la memoria RAM física para aumentar la capacidad de almacenamiento temporal de datos en el sistema y sirve para optimizar el rendimiento del sistema, permitiendo que se ejecuten más aplicaciones o procesos simultáneamente sin causar una disminución significativa del rendimiento.

Para usar el modelo, inicialmente se creó un *script* de prueba que permitiera comprobar el correcto funcionamiento del modelo para realizar los ajustes y correcciones necesarias, dicho *script* fue construido para cargar una imagen desde el sistema y reajustar su tamaño a 256 px x 256 px, para posteriormente ingresar al modelo y obtener los resultados de la inferencia. Finalmente, se construyó una interfaz usando PyQt5 que permitiera ingresar a la cámara del sistema para capturar la imagen, posteriormente pasarla al modelo y mostrar en pantalla el resultado de la inferencia. Dicha interfaz será mostrada detalladamente en en la sección 5.4.

5.2.2. Despliegue en RASPBERRY PI 4. En esta tarjeta se instaló la imágen disponible para este sistema mediante el *software* Raspberry PI imager desde un computador con Ubuntu 20.04. Este sistema operativo de Raspberry ya contiene una versión de Python 3 y algunas de sus librerías.

Luego de realizar la configuración necesaria en el entorno de la Raspberry, se actualizó Python a la versión 3.9.2 y se instaló *pip*, que es un sistema de gestión de paquetes que se utiliza para instalar y administrar paquetes de software escritos en Python. Usando *pip* se instalaron las librerías necesarias tales como Tensorflow, OpenCV y PyQt5.

Para la ejecución del modelo al igual que en la Jetson Nano se hizo mediante el *script* de prueba y posteriormente en la interfaz programada anteriormente mencionada.

5.3. MODELO EN 3D.

Entre los diversos programas de software libre para el diseño y modelado 3D se escogió FreeCAD, ya que este dispone de una mayor comunidad, documentación y soporte.

Para el diseño de la carcasa se realizaron distintos modelos, modificando varios aspectos y dimensiones, obteniendo un diseño compacto y óptimo que permite la integración de diversos elementos y equipos electrónicos. El diseño final se encuentra en el repositorio de GitHub, ver en el Anexo 5,o podrá observar los planos a detalle en el Anexo 1.

Como primera versión se diseñó una carcasa que se compone de una caja rectangular y una tapa. Por medio de la herramienta de extrusión se dio profundidad y espesor a las paredes del cuerpo de la caja, de esta forma se permite ubicar la Jetson nano junto a la “power bank”. Con la ayuda de la herramienta extrusión se dio profundidad a la tapa tomando como base las medidas del rectángulo de la carcasa, esta se soporta con torres que permiten el ajuste mediante tornillos.

Esta carcasa se descartó ya que en la parte inferior del soporte, la pantalla estaba expuesta, lo que causaba que no estuviera ajustada, de esta misma manera los cables de conexión se veían desordenados, y se hacía necesario adicionar más

agujeros a la carcasa.

Posteriormente, se creó un segundo encapsulado teniendo en cuenta los problemas en el primer diseño. Sin embargo, este modelo fue descartado, debido a que el tamaño no era adecuado para la manipulación, porque al estar la Jetson Nano sobre la power Bank sobraba espacio.

Finalmente, se diseñó un tercer encapsulado resolviendo todos los inconvenientes presentados, además, se pulieron y optimizaron los espacios para su correcto funcionamiento y manipulación.

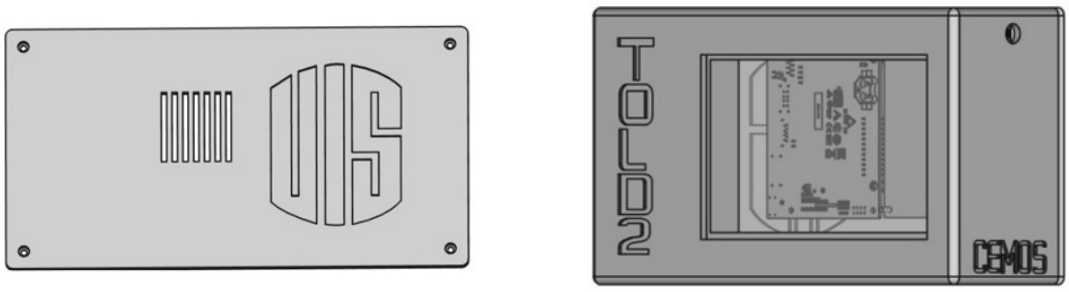
El diseño tiene como partes importantes una carcasa que contiene una tarjeta de desarrollo Jetson Nano, una “Power Bank”, una pantalla táctil de 7 pulgadas y los cables de conexión, adicional a esto, cuenta con un soporte giratorio para la cámara, sosteniendo un porta muestra con forma de pirámide cuadrada, facilitando la toma de imágenes en un ambiente controlado.

A este diseño se le agrego en forma de bajo relieve el logo institucional de la UIS, los nombres de los grupos de investigación interesados DICBOT y CEMOS, y el nombre del encapsulado TOLD2 que son las siglas en ingles de “Tomato Leaf Desease Detector”.

5.3.1. Encapsulado. En este primer módulo se inició con el diseño de un rectángulo de 250 mm x 130 mm mediante la extrusión del croquis de forma rectangular de 70 mm y así dando la forma de cubo rectangular, se realizó un vaciado en una de las caras para darle grosor a la pared dejando 4 mm de espesor. Luego se implementó un gabinete donde se incorpora la pantalla de manera ajustada, evitando cualquier fractura o deterioro a largo plazo, para permitir la visualización de la pantalla se realizó un corte en la cara principal.

Se elaboro la base donde se ajusta y soporta la tapa en las esquinas superiores, realizando un cuarto de circunferencia entre las paredes de la caja, para fijar un orificio centrado de 1.5 mm de radio el cual permite atornillar la tapa.

Figura 14. Encapsulado: (a) Tapa del encapsulado, (b) Carcasa del encapsulado



(a) (b)

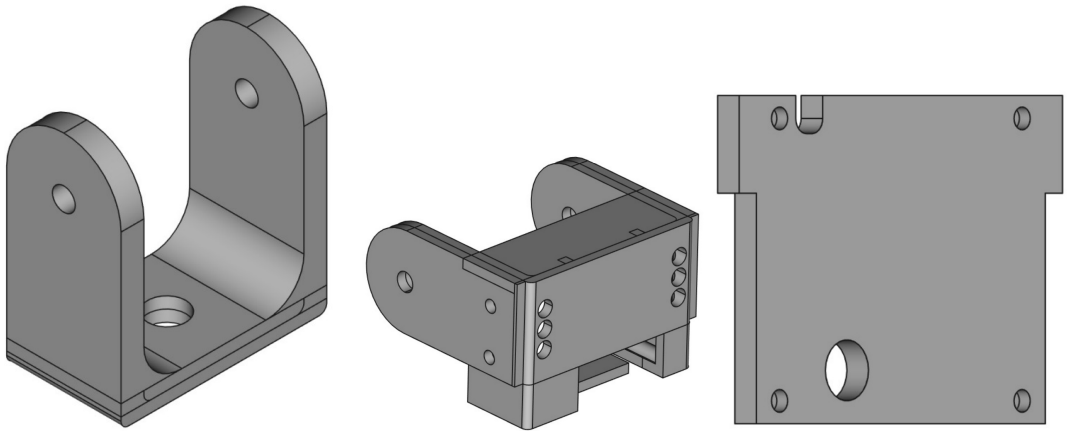
Se acomodó los distintos dispositivos Jetson Nano, Ventilador, Power Bank y cables de conexión de forma ordenada, optimizando espacios.

Posteriormente en un cuerpo adicional se hizo otro rectángulo con las mismas dimensiones de la carcasa del encapsulado, esta vez para ser utilizado como tapa, en este cuerpo se hizo la extrusión de 4 mm, luego se realizaron los respectivos orificios con la herramienta de taladro abocado en las 4 esquinas de la tapa, para que de esta manera se pudiera atornillar.

Finalmente, se elaboró las rejillas de ventilación de 3 mm x 30 mm, donde van separadas cada 3 mm permitiendo que toda la electrónica cuente con un flujo de aire que facilite la refrigeración de este, de esta misma manera se hicieron los orificios para el conector de alimentación y los interruptores. Ver Figura 14.

5.3.2. Soporte Cámara. El soporte de la cámara se compone de 3 partes principalmente; la primera es la base del soporte de la cámara, la cual es una pieza en forma de “U” , tiene un espesor de 6 mm, un alto de 60 mm, un ancho de 54 mm y una profundidad de 30 mm, la pieza se construyó partiendo de un croquis rectangular de 54 mm x 30 mm, se realizó una extrusión de las dos aletas laterales partiendo de una de las caras de la extrusión anterior, luego se hizo un corte lateral

Figura 15. Soporte cámara: (a) Base del soporte de la cámara, (b) Cabeza del soporte de la cámara, (c) Tapa del soporte de la cámara



(a) (b) (c)

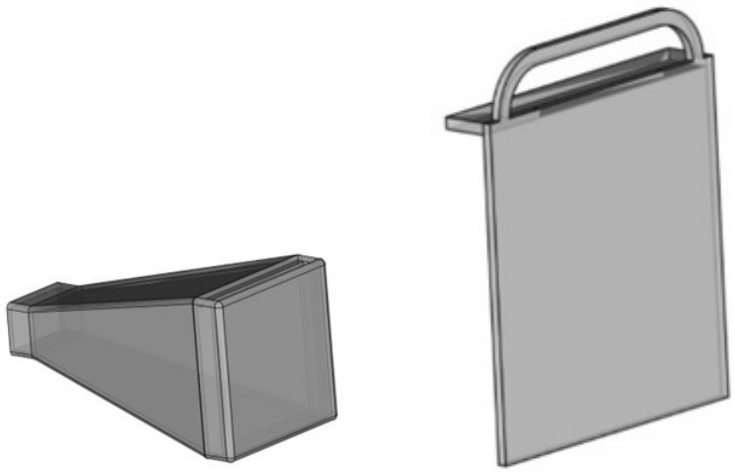
para redondear los extremos de las aletas, finalmente se realizaron las perforaciones para fijar la base a la cabeza del soporte y para atornillar la base del soporte al encapsulado para permitir un giro rotatorio de 360°.

La segunda pieza es la cabeza del soporte de la cámara, esta parte se realizó de forma similar a la pieza descrita anteriormente, se inició realizando un croquis con forma rectangular de 66 mm x 30 mm y se extruyó 6 mm, a continuación se realizaron las aletas en los extremos de la extrusión anterior con un espesor de 6 mm, luego se fueron agregando los soportes para atornillar la tapa que protege la electrónica interior para la iluminación usando extrusión con croquis circular, finalmente se hicieron las perforaciones para la salida de los LEDs, para atornillar la cabeza a la base y atornillar al porta muestras.

La última pieza del soporte para la cámara es la tapa del soporte de la cámara la cual cubre y protege la electrónica, fue construida partiendo de un croquis en forma de “T” que en su parte superior mide 60 mm y en su parte inferior mide 54 mm, la pieza tiene un espesor de 4 mm y tiene las perforaciones necesarias para atornillar, ver Figura 15.

5.3.3. Porta muestras Las piezas que componen el porta muestras son una carcasa, que aísla la luz exterior, y una bandeja donde se coloca la muestra. La carcasa tiene forma de tronco de pirámide cuadrada y fue construida a partir de la cara rectangular más grande. Se inició con un croquis de 122 mm x 122 mm y se realizó una extrusión de 220 mm. Posteriormente, se realizaron cortes transversales que permitieran darle la forma de tronco de pirámide cuadrada. Luego, se realizó un vaciado para darle un grosor de 6 mm a la pieza. A continuación, se realizó un corte rectangular en uno de los costados de la parte inferior de la pieza para que pudiera entrar la bandeja. Finalmente, se realizaron las perforaciones para sujetar la pieza al soporte de la cámara y se construyeron guías para poder desplazar la bandeja.
La bandeja fue fabricada en acrílico de 3 mm de espesor. Su forma parte de un rectángulo de 114 mm x 109 mm y en su parte superior tiene un arco que permite extraer e introducir la bandeja. También, en la parte superior del rectángulo, se añade otra lámina de acrílico de 16 mm x 109 mm en forma perpendicular, de tal forma que evite totalmente la entrada de luz al interior de la carcasa, ver Figura 16 .

Figura 16. Porta muestras: (a) Porta muestras, (b) Bandeja del porta muestras



(a) (b)

Cada una de estas partes se fabricaron por medio de impresiones 3D en donde mediante la superposición de capas sucesivas de material permiten la creación de los diversos objetos, para este caso se utilizó un filamento PLA de 1.75 mm ya que este no es toxico y ofrece gran resistencia a la humedad.

En el diseño final se verificó que este no tuviera ángulos rectos, los sólidos internos no contuvieran huecos, la orientación de la pieza de forma que pueda soportar cargas y la verificación del tamaño del archivo STL, debido a que este formato construye los cuerpos mediante mallas de triángulos, de forma que entre más triángulos mejor resolución será el objeto que se imprima, pero a su vez entre mayor tamaño de archivo, mayor procesamiento de datos.

5.4. ENTORNO DE USUARIO.

Con la finalidad de hacer la experiencia de usuario mas agradable, cómoda e intuitiva, se configuró el sistema del dispositivo de detección con tres botones en su inicio, los primeros dos ejecutan la interfaz de toma de datos y la interfaz para realizar inferencia respectivamente y el tercer botón tiene la funcionalidad de apagar el sistema, ver Figura 17. Los códigos correspondientes a cada interfaz de usuario se encuentran disponibles en el Anexo 3.

Figura 17. Pantalla de inicio

5.4.1. Interfaz de toma de datos El código told2ds_v0.2 contiene la interfaz gráfica de usuario (GUI) por sus siglas en inglés, que fue programada en Python usando la biblioteca PyQt5. La interfaz muestra un *stream* de vídeo en vivo de la cámara del sistema y proporciona tres botones: “Tomar Foto”, “Guardar Foto” y “Volver a tomar Foto”. Cuando se hace clic en el botón "Tomar Foto", se detiene la transmisión de vídeo y se muestra la imagen capturada en la interfaz. Los botones “Guardar Foto” y “Volver a tomar Foto” se muestran para permitir al usuario guardar la foto o volver a tomar una nueva foto, respectivamente. Una vez que se guarda la foto, la transmisión de vídeo se reinicia. Además, se muestra un contador de imágenes guardadas y se organizan las imágenes en carpetas según la opción seleccionada por el usuario en un menú desplegable, la pantalla inicial de la interfaz para toma de datos se puede observar en la Figura 18.

Figura 18. Interfaz de usuario para toma de datos

5.4.2. Interfaz de detección La interfaz told2_v0.1 utiliza la red neuronal para clasificar los diferentes tipos de enfermedades de las plantas a partir de las imágenes capturadas. La interfaz permite al usuario tomar una foto de una hoja de la planta y enviarla para su clasificación. Esta interfaz gráfica está creada con el “framework” PyQt5. El modelo de red neuronal utilizado está previamente entrenado y

almacenado con extensión h5 llamado "modelo_ResNet50".

Cuando el usuario ejecuta la interfaz, se inicia la captura de vídeo de la cámara web del sistema. Al hacer clic en el botón “Tomar Foto”, se toma una instantánea del vídeo actual y se redimensiona a 256x256 píxeles. Al hacer *click* en el botón “Hacer Detección” La imagen redimensionada se envía a la red neuronal para su inferencia. El resultado de la inferencia se muestra en la interfaz gráfica de usuario junto con un comentario sobre el tipo de enfermedad detectada. También se guarda una copia de la imagen clasificada en la carpeta "detections", junto con un archivo de texto que contiene el resultado de la detección, la pantalla inicial de la interfaz de usuario para realizar inferencia se puede observar en la Figura 19.

Figura 19. Interfaz de usuario para realizar inferencia

6. RESULTADOS.

En este capítulo se presentan los resultados obtenidos durante el desarrollo del proyecto. Se analizan y discuten los datos recopilados durante la implementación y se comparan con los objetivos planteados inicialmente. Se describen las principales tendencias y patrones observados en las diferentes gráficas, matrices de confusión y medidas de rendimiento. Además se muestra los resultados obtenidos en campo usando las tarjetas de desarrollo. Finalmente se realiza una comparativa del consumo de potencia eléctrica durante la inferencia entre las tarjetas de desarrollo.

Al sistema del sistema de diagnóstico de enfermedades en la hoja del tomate se le asignó el nombre de TOLD2 que significa *Tomato Leaf Desease Detector*, se puede observar en la Figura 20.

Figura 20. TOLD2

Además en la Figura 21, se encuentra el sistema con su accesorio de porta muestras incorporado.

Figura 21. TOLD2 con porta muestras.

6.1. DESEMPEÑO DE LAS REDES NEURONALES.

En la evaluación de un modelo de *deep learning*, es fundamental conocer las métricas que permiten medir su desempeño y determinar su capacidad de generalizar el error. Entre las principales métricas se encuentran la exactitud de entrenamiento y validación, el número de épocas y la evaluación final obtenida con el método *compile* de Keras.

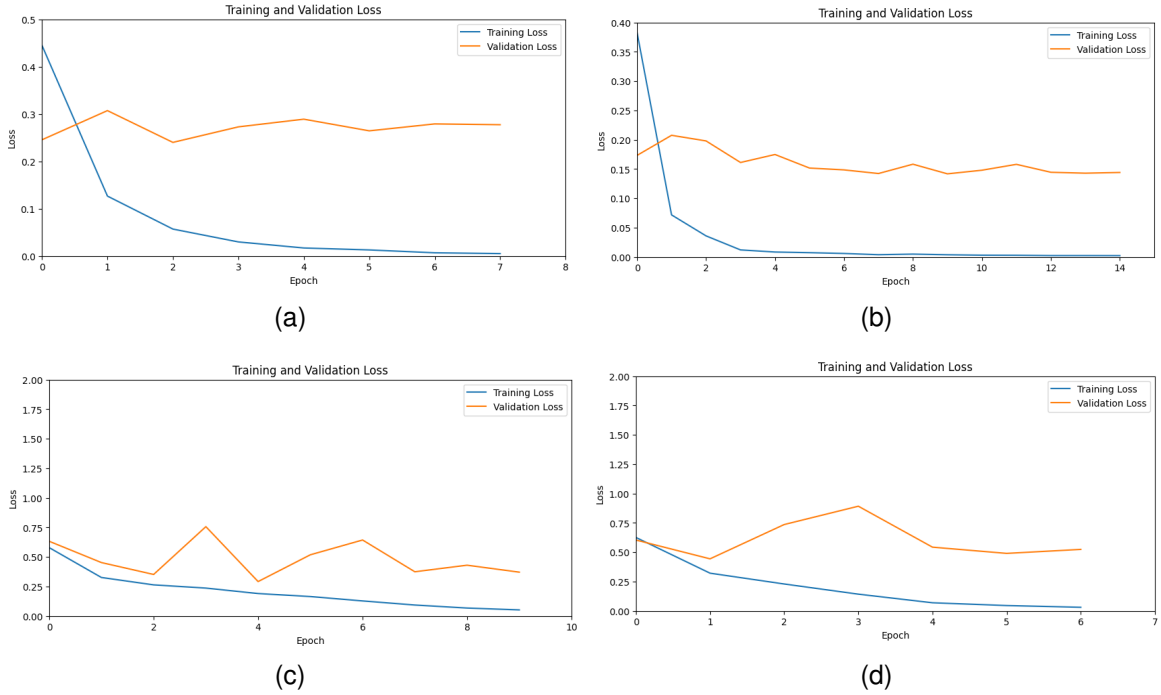
Tabla 14. Medidas de rendimiento de cada una de las redes.

Red neuronal	Épocas	Exactitud	Exactitud en validación	Evaluación del modelo
Red propuesta	9	0.9866	0.8783	0.8952
Red MobileNetV2	6	0.9953	0.8366	0.8336
Red ResNet50	14	1.00	0.9583	0.9569
Red VGG19	7	0.9997	0.9289	0.9261

Como se puede apreciar en la Tabla 14, se observa que las redes presentan un bajo número de épocas, esto se debe a que en el conjunto de datos de entrenamiento generaliza el error y en el conjunto de datos de validación este proceso es complejo. Esto genera una convergencia pronta en los modelos y una actualización de los mejores pesos con ayuda del *callback EarlyStopping*.

Una de las herramientas de evaluación de un modelo de aprendizaje profundo, son las gráficas de la función de perdida de entrenamiento y validación con respecto a las épocas entrenadas, las cuales fueron definidas usando el *callback EarlyStopping*.

Figura 22. Gráficas de la función de perdida para cada modelo: (a) Modelo VGG19, (b) Modelo ResNet50, (c) Red propuesta, (d) Modelo MobileNetV2



En la Figura 22, se evidencia la falta de épocas de entrenamiento, generando en la gráfica de la función de perdida de validación, una tendencia a disminuir con una pendiente pequeña. Esto debido a la rápida convergencia de los modelos.

Otra herramienta que permite la visualización del desempeño de un algoritmo de *deep learning* es la matriz de confusión, a continuación se muestran las matrices de confusión correspondientes a cada uno de los modelos entrenados.

En la Figura 23 se puede observar que todos los modelos presentan una precisión superior al 95 % en la predicción de la clase correspondiente a la hoja sana. Sin

embargo, se puede notar que el desempeño de los modelos en la clasificación de la enfermedad del tizón tardío es mejorable, ya que se confunde en la mayoría de los casos con la enfermedad del tizón temprano. Esto se debe a que ambas enfermedades tienen síntomas similares. Por otro lado, en la clasificación de la enfermedad del virus del mosaico, los modelos presentan un desempeño adecuado, ya que esta enfermedad no tiene similitudes con las otras clases.

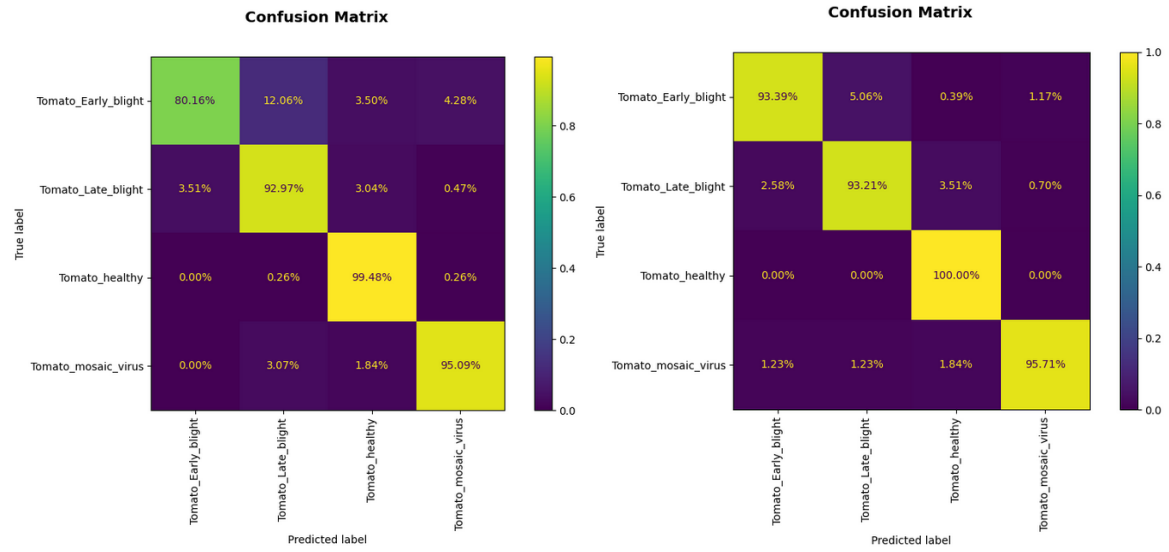
El cálculo de las medidas de rendimiento se hizo teniendo en cuenta el *weighted average*, que se refiere a un promedio ponderado. En este tipo de promedio, los valores con mayor peso tienen un impacto mayor en el resultado final, mientras que los valores con menor peso tienen un impacto menor.

En la tabla 15 las medidas de rendimiento *Precision*, *Recall* y *F1 score*, permiten observar que las redes con mejor desempeño son la red VGG19 y la red ResNet50, debido a presentan un *F1 score* alto, lo cual representa un equilibrio entre *Precision* y *Recall*. Lo anterior indica que estas redes tienen poca tendencia a presentar falsos positivos y falsos negativos. Por otra parte, las redes que presentaron un menor desempeño fueron la red propuesta y la MobileNetv2, debido a que presentan un desbalance entre las medidas *Precision* y *Recall*, lo cual se traduce en un bajo *F1 score*.

Tabla 15. Medidas de rendimiento de cada una de los modelos.

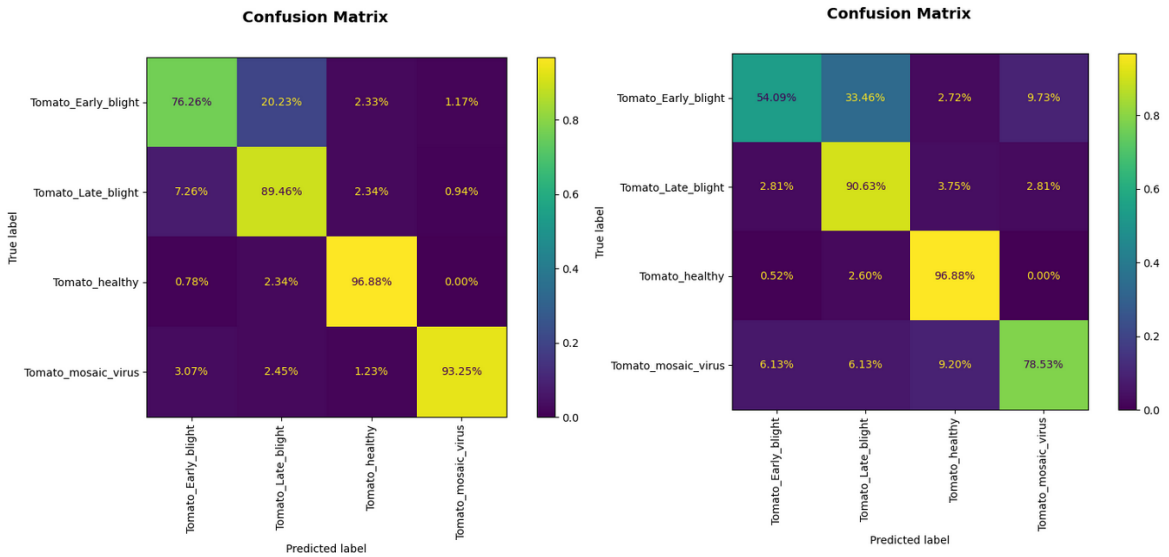
Red neuronal	**Precision**	**Recall**	**F1 score**	**Total de imágenes de test**
Red propuesta	0.89	0.90	0.89	1232
Red MobileNetV2	0.84	0.83	0.83	1232
Red ResNet50	0.96	0.96	0.96	1232
Red VGG19	0.93	0.93	0.92	1232

Figura 23. Matriz de confusión de cada uno de los modelos.: (a) Modelo VGG19, (b) Modelo ResNet50, (c) Modelo propuesto, (d) Modelo MobileNetV2



(a)

(b)



(c)

(d)

6.2. DESEMPEÑO DEL MODELO EN DIFERENTES CONDICIONES DE ILUMINACIÓN.

Con base en los análisis realizados en la sección anterior se definió el modelo ResNet50 como el modelo para implementar en cada una de las tarjetas de desarrollo. Para evaluar el desempeño del modelo en campo se tuvieron en cuenta dos tipos diferentes de iluminación, con luz controlada usando el accesorio porta muestras incorporado y con luz natural.

La evaluación del modelo se realizó de la siguiente forma: Inicialmente se hizo inferencia con 20 muestras diferentes para cada clase distribuidas de la siguiente manera, 10 muestras con iluminación controlada y 10 muestras en condiciones de iluminación natural. Luego se determinó la cantidad de diagnósticos correctos del modelo para realizar el cálculo del porcentaje de aciertos obtenidos.

Tabla 16. Resultados de desempeño en diferentes condiciones de iluminación.

Clase	**Aciertos con iluminación controlada**	**Aciertos con iluminación natural**	**Total de aciertos**
Tizón tardío	100	50	15
Tizón temprano	50	90	14
Virus del mosaico	100	90	19
Hoja sana	100	90	19
Total			67

En la Tabla 16 se muestra que la cantidad de aciertos obtenidos es de 67, en total se realizaron pruebas con 80 muestras lo cual implica que el modelo tiene un porcentaje de aciertos del 83.75 %, cabe aclarar que este resultado se debe a que la evaluación no se hizo con un conjunto de datos significativo, debido a que fue difícil encontrar un cultivo que tuviera las cuatro clases el día de la visita técnica destinada para realizar pruebas en campo, también debido a que dicha visita técnica fue destinada para comprobar de que forma se afectaba el modelo en diferentes condiciones de iluminación, sin embargo, lo ideal es realizar estas pruebas con una cantidad de

datos significativa.
También se puede observar que el porcentaje de aciertos es mayor en la en tres de las cuatro clases cuando se usa el accesorio de porta muestras, lo anterior permite comprobar que el modelo se desempeña de una mejor manera cuando se controla la iluminación del sistema de detección TOLD2.

6.3. COMPARATIVA ENTRE TARJETAS.

El consumo de energía es un factor importante a considerar en dispositivos de cómputo, especialmente en aplicaciones que requieren un alto rendimiento. En este proyecto, se comparó el consumo de corriente eléctrica entre la Jetson Nano de 4GB y la Raspberry PI 4B mientras se realiza la inferencia, ya que dicho proceso es el que mas requiere de potencia eléctrica.
El proceso para la medición del consumo de corriente en cada dispositivo fue el siguiente: primero se realizó el montaje en laboratorio del sistema TOLD2 fuera de la carcasa para facilitar la manipulación del cableado, lo cual es necesario para la medición de la corriente, además, se verificó que todos los periféricos del dispositivo estuvieran conectados. Luego de realizar el montaje, se procedió a realizar inferencias con imágenes capturadas en el momento usando la interfaz de usuario dispuesta para este fin. Finalmente haciendo uso de una pinza amperimétrica se midió la corriente a través del cable de alimentación que va desde el banco de beaterías hasta la tarjeta de desarrollo. Cabe aclarar que dicho procedimiento se realizó de la misma manera para cada tarjeta de desarrollo. Los resultados de las mediciones de consumo se muestran en la Tabla 17
Aunque el modelo de TensorFlow utilizado fue el mismo en ambas tarjetas de desarrollo, se observó una diferencia en el consumo de corriente eléctrica durante la inferencia, demostrando que la Jetson Nano consume 0.2 A más que la Raspberry PI 4B. Esto puede deberse a que la Jetson Nano dispone de GPU, tambien puede

Tabla 17. Comparativa entre tarjetas.

	Jetson Nano	**Raspberry PI 4B**
Precio	151 USD	144 USD
Procesador	ARM Cortex-A57 de 4 núcleos a 1.43 GHz	Broadcom BCM2711, Quad core Cortex-A72 a 1.5 GHz
RAM	4 GB	4 GB
GPU	NVIDIA Maxwell de 128 núcleos CUDA	No tiene
Consumo de corriente durante la inferencia	1.8 A	1.6 A

ser debido a las diferencias en la arquitectura de los procesadores, la configuración del sistema y la eficiencia energética de los componentes. Es importante tener en cuenta que este estudio se realizó con un modelo ResNet50 y que los resultados pueden variar con otros modelos y aplicaciones.

7. RECOMENDACIONES

Durante la experiencia adquirida en el desarrollo de este trabajo, se proponen diversas indicaciones con el fin de mejorar la calidad del trabajo y su aplicación en el futuro.

- En relación con el diseño, es importante examinar la optimización de los espacios para la adaptación de todos los dispositivos electrónicos, la longitud de los cables existentes, el material empleado y la calidad de la impresión 3D. También se sugiere manipular el encapsulado con cuidado para evitar fracturas y protegerlo de la exposición al agua y la humedad.

- Se sugiere aumentar la cantidad de imágenes de la base de datos de cada enfermedad y las visitas de campo. Además, de incluir más enfermedades que se manifiestan en la hoja, tallo y fruto para considerar una mayor cantidad de variables que permitan realizar un mejor diagnostico.

- Con respecto a los sistemas embebidos, se recomienda realizar pruebas en otras tarjetas de desarrollo diferentes a las del proyecto, con el fin de comparar aspectos como el consumo de energía, la eficiencia, el rendimiento, entre otros.

- En cuanto a la parte de interfaz y despliegue, se propone integrar la interfaz de inferencia y detección con la de toma de base de datos para formar una sola aplicación, lo que permitirá gestionar ambos aspectos en un único entorno, brindar la opción de crear nuevas carpetas en caso de presentarse nuevas enfermedades.

8. CONCLUSIONES

La presente tesis de investigación se enfocó en la detección temprana de enfermedades en las hojas del tomate, con el objetivo de aplicar los resultados obtenidos a la agroindustria. A partir del análisis de los resultados obtenidos y la revisión de los objetivos planteados, se pueden destacar las siguientes conclusiones:

- Gran parte de los agricultores que fueron visitados en los municipios presentan una actitud escéptica hacia la adopción de nuevas tecnologías. En su experiencia, no ven la necesidad de incorporar dispositivos que les proporcionen predicciones o indicaciones para llevar a cabo sus actividades agrícolas. No obstante, un porcentaje de estos agricultores perciben la tecnología como una herramienta innovadora que les ayuda a llevar a cabo sus labores de campo de forma más práctica, rápida y fiable. La transferencia tecnológica entre la agroindustria ha permitido el desarrollo de sistemas de inteligencia artificial que pueden resultar beneficiosos para los agricultores. En este proyecto, se ha creado un sistema de detección temprana de enfermedades en las hojas del tomate, el cual es una herramienta valiosa para agricultores y expertos, ya que les permite realizar un diagnostico temprano y prevenir daños importantes en lo cultivos. Este sistema también es una excelente herramienta didáctica para agrónomos jóvenes y estudiantes, facilitando el aprendizaje de manera más accesible y dinámica.

- Los invernaderos son lugares diseñados para controlar diversos factores que influyen en el desarrollo de las plantas como la humedad, además, permiten prevenir enfermedades causadas por los cambios climáticos bruscos y plagas.

- La obtención de hojas del tomate con la enfermedad deseada es un proceso complejo, debido a que se pueden presentar hasta 3 enfermedades en una misma hoja, dificultando la selección de la enfermedad requerida.

- La tarjeta de desarrollo Jetson Nano cuenta con una GPU de 4 GB que le permite realizar una amplia variedad de tareas, mientras que la Raspberry Pi 4B no cuenta con esta capacidad, lo que la limita en ciertos aspectos. Debido a esto, es razonable pensar que la Jetson nano consumirá más potencia eléctrica durante el procesamiento. Aunque es cierto que existe una pequeña diferencia en el consumo de potencia eléctrica entre la Jetson Nano y la Raspberry PI 4B, esta diferencia no es significativa para tareas de procesamiento de corta duración. Sin embargo, para tareas de larga duración la Raspberry PI 4B tendrá menor consumo energético que la Jetson Nano.

- A partir de los resultados obtenidos en este estudio, se puede afirmar que los modelos de red neuronal creados con técnicas de *transfer learning*, específicamente ResNet50 y VGG19, mostraron un mejor desempeño en la resolución de problemas relacionados con la clasificación de enfermedades en la hoja de la planta del tomate, en comparación con el modelo MobileNetV2 y el modelo de jerarquía reducida. Esto se debe a que su arquitectura fue diseñada para abordar este tipo de problemas y, por lo tanto, se adaptó mejor a la tarea en cuestión.

- A pesar de que el modelo de jerarquía reducida no obtuvo los resultados deseados, su desempeño no es el peor, ya que, sus medidas de rendimiento, gráficas de función de perdida y matriz de confusión fueron mejores que las del modelo MobileNetV2. Lo anterior permite seguir con la investigación sobre modelos de redes de jerarquía reducida, que permitan solucionar este tipo de problemas con la menor cantidad de parámetros posibles.

- En cada uno de los cuatro modelos de inteligencia artificial entrenados se pudo observar que la clase con mejores medidas de rendimiento es la hoja sana, por lo anterior es poco probable que la inteligencia artificial se equivoque al predecir si una hoja presenta alguna enfermedad o no, ya que normalmente antes de realizar el diagnostico de una enfermedad, lo primero es determinar si la hoja se encuentra enferma.

- En base a los resultados obtenidos en las pruebas de campo se demostró que el modelo de red neuronal implementado en las tarjetas de desarrollo, muestra un mejor rendimiento cuando se controlan las condiciones de iluminación, debido a que cuando se hace uso del porta muestras se logra dar una iluminación más uniforme sobre la hoja, evitando rayos de luz externos que puedan generar brillos que afecten la calidad de la imagen capturada.

BIBLIOGRAFÍA

Aggarwal, Charu C. *Neural Networks and Deep Learning*. 1.ª ed. NY: Springer International Publishing AG, 2018. 512 págs. (vid. págs. 27, 30, 32).

Aguado Martinez, Ana et al. *EL VIRUS DEL MOSAICO DEL TOMATE*. Ene. de 2014 (vid. pág. 24).

Alexis, Martinez Sarmiento Edgar. "DESARROLLO DE UN SOFTWARE PARA EL DISEÑO ASISTIDO DE EJES SOMETIDOS A CARGAS ESTÁTICAS Y DINÁMICAS". Tesis doct. Quito: Escuela Politécnica Nacional, 2016. 179 págs. (vid. pág. 39).

Bernal, Roberto. *ENFERMEDADES DE TOMATE (Lycopersicum esculentum Mill.) EN INVERNADERO EN LAS ZONAS DE SALTO Y BELLA UNION*. 2010 (vid. pág. 22).

Bharate, Anil A. y M. S. Shirdhonkar. "A review on plant disease detection using image processing". En: *2017 Internationa Conference on Intelligent Sustainable Systems (ICISS)*. Dic. de 2017, págs. 103-109. DOI: 10.1109/ISS1.2017.8389326 (vid. pág. 21).

Blancard, Dominique. *Enfermedades del tomate*. Mundi-Prensa Libros, 1 de ene. de 2011. 682 págs. (vid. pág. 22).

blender.org - Home of the Blender project - Free and Open 3D Creation Software. 2023. URL: https://www.blender.org/ (visitado 20-03-2023) (vid. pág. 40).

C, Lucás. *El filamento de ABS en la impresión 3D*. 3Dnatives. 6 de jun. de 2019. URL: https://www.3dnatives.com/es/filamento-de-abs-impresion-3d-06062019/ (visitado 23-01-2023) (vid. pág. 42).

— *Guía completa: el filamento PLA en la impresión 3D*. 3Dnatives. 18 de ago. de 2019. URL: https://www.3dnatives.com/es/guia-filamento-pla-en-la-impresion-3d-190820192/ (visitado 23-01-2023) (vid. pág. 42).

Chollet, François. *Deep Learning with Python*. 2.ª ed. NY: Manning Publications Co, 2021. 504 págs. (vid. págs. 26, 30-32, 47).

Flores, Ceferino, Sebastian Buono y Sergio Giorgini. *Guía de consulta Enfermedades de Tomate*. Argentina: Instituto Nacional de Tecnologia Agropecoria Yuto, 10 de nov. de 2012. 138 págs. (vid. pág. 24).

Frayre, González et al. "*Estudio y comparativa de algoritmos de detección de objetos con redes neuronales artificiales convolucionales para la detección de enfermedades en hojas*". En: (2019). Publisher: Asociación Mexicana de la Industria Automotriz, pág. 1. DOI: https://doi.org/10.48779/p41k-fx69 (vid. pág. 15).

Fuentes, Alvaro et al. "A Robust Deep Learning Based Detector for Real Time Tomato Plant Diseases and Pests Recognition". en. En: *Sensors* 17.9 (sep. de 2017). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, pág. 2022. DOI: 10.3390/s17092022 (vid. pág. 20).

G., Geetharamani y Arun Pandian J. "Identification of plant leaf diseases using a nine-layer deep convolutional neural network". En: *Computers & Electrical Engineering* 76 (1 de jun. de 2019), págs. 323-338. DOI: 10.1016/j.compeleceng.2019.04.011 (vid. págs. 18, 33, 48).

Gajjar, Ruchi et al. “Real-time detection and identification of plant leaf diseases using convolutional neural networks on an embedded platform”. En: *The Visual Computer* 38.8 (ago. de 2022), págs. 2923-2938. DOI: 10.1007/s00371-021-02164-9 (vid. pág. 20).

Gonzalez, Sol. *¿Qué es un sistema embebido?* Medium. 10 de mayo de 2021. URL: https://ardebytes.medium.com/que-es-un-sistema-embebido-9fd6d3ac94d9 (visitado 11-04-2023) (vid. pág. 35).

Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3.ª ed. O’Reilly Media, Inc, 2022. 1196 págs. (vid. págs. 26, 29-32, 45).

Hughes, David, Marcel Salathé et al. “An open access repository of images on plant health to enable the development of mobile disease diagnostics”. En: *arXiv preprint arXiv:1511.08060* (2015) (vid. pág. 43).

Maryum, Alina, Muhammad Usman Akram y Anum Abdul Salam. “Cassava Leaf Disease Classification using Deep Neural Networks”. En: *2021 IEEE 18th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*. 2021, págs. 32-37. DOI: 10.1109/HONET53078.2021.9615488 (vid. pág. 18).

Murphy, Kevin P. *Probabilistic Machine Learning An Introduction*. 1.ª ed. London: The MIT Press Cambridge, Massachusetts, 2022. 856 págs. (vid. pág. 29).

Paz, Nercy Sita Ricardo, Angel Gustavo Polanco Aballe y Sonia Reyes Gómez. “Comportamiento del tizón temprano del tomate (Alternaria solani) en las condiciones del municipio de Holguín, Cuba”. En: (2013-8) (vid. pág. 23).

Phillips, Gavin. *The Best Single-Board Computers for Chrome OS and Android*. 22 de ene. de 2020. URL: https://www.makeuseof.com/tag/best-single-board-computers/ (visitado 11-04-2023) (vid. pág. 37).

Pérez-Aguilar, Daniel Alexis, Redy Henry Risco-Ramos y Luis Casaverde-Pacherrez. "Transfer learning en la clasificación binaria de imágenes térmicas". En: *Ingenius* 26 (29 de jun. de 2021), págs. 71-86. DOI: 10.17163/ings.n26.2021.07 (vid. pág. 15).

Rosenblatt, F. "The perceptron: A probabilistic model for information storage and organization in the brain." En: *Psychological Review* 65 (1958). Place: US Publisher: American Psychological Association, págs. 386-408. DOI: 10.1037/h0042519 (vid. pág. 27).

Ruiz, Carlos. *Guía completa: el filamento PLA en la impresión 3D*. AUROS Colombia. 1 de ene. de 2020. URL: https://www.auros.com.co/guia-completa-filamento-pla-la-impresion-3d/ (visitado 20-03-2023) (vid. pág. 42).

Shaha, Manali y Meenakshi Pawar. "Transfer Learning for Image Classification". En: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA). Mar. de 2018, págs. 656-660. DOI: 10.1109/ICECA.2018.8474802 (vid. págs. 20, 34).

Sistemas embebidos: qué son y para qué se utilizan. InnovaciónDigital360. 16 de dic. de 2022. URL: https://www.innovaciondigital360.com/iot/sistemas-embebidos-que-son-y-para-que-se-utilizan/ (visitado 11-04-2023) (vid. pág. 35).

Soberania Alimentaria, Centro de Estudios para el Desarrollo Rural Sustentable y la. *Análisis de la producción y consumo de hortalizas*. 2020 (vid. pág. 22).

Tian, Yunong et al. “Apple detection during different growth stages in orchards using the improved YOLO-V3 model”. en. En: *Computers and Electronics in Agriculture* 157 (feb. de 2019), págs. 417-426. DOI: 10.1016/j.compag.2019.01.012 (vid. pág. 19).

Uso básico del microcontrolador para medición y control. es. Oct. de 2022 (vid. pág. 36).

Valdes, Fernando y Ramón Pallás Areny. *Microcontroladores Fundamentos y Aplicaciones con PIC*. España: Marcombo, 28 de feb. de 2007. 346 págs. (vid. pág. 36).

Vallejo Cabrera, Franco Alirio. *Mejoramiento genético y producción de tomate en Colombia*. Colombia: Universidad Nacional de Colombia, nov. de 1999 (vid. pág. 15).

Vesga, Juan Carlos. *Microcontroladores motorolafreescale*. Mexico: Alpha Editorial, jun. de 2008 (vid. pág. 36).

Zhang, Yanchao et al. “Real-time strawberry detection using deep neural networks on embedded system (rtsd-net): An edge AI application”. En: *Computers and Electronics in Agriculture* 192 (ene. de 2022), pág. 106586. DOI: 10.1016/j.compag.2021.106586 (vid. pág. 21).

Zhang, Yu-Dong et al. “Image based fruit category classification by 13-layer deep convolutional neural network and data augmentation”. en. En: *Multimedia Tools and Applications* 78.3 (feb. de 2019), págs. 3613-3632. DOI: 10.1007/s11042-017-5243-3 (vid. pág. 19).

ANEXOS

Anexo A. Planos del encapsulado.

El presente anexo describe los planos correspondientes al encapsulado utilizado en el desarrollo de esta tesis, proporcionando información detallada acerca de sus características y especificaciones técnicas. Los planos presentan las dimensiones, formas y detalles constructivos del encapsulado, lo que permite una mejor compresión de su diseño y funcionamiento.

La inclusión de los planos en este anexo es fundamental para facilitar la compresión de los detalles técnicos del encapsulado, lo que resulta relevante para futuras investigaciones y desarrollos en el campo. De esta forma, se garantiza que la información contenida en los planos sea accesible y comprensible para cualquier interesado en el tema.

Anexo B. Diseño PCB

En la presente tesis de grado se desarrolló el diseño y construcción de una PCB de iluminación con LEDs utilizando el software KiCAD. Como parte del proceso de documentación, se incluye un anexo que contiene los planos y archivos necesarios para replicar el diseño y construcción de la PCB. Esto incluye el archivo de perforación de la PCB, el archivo de enmascaramiento de soldadura, el archivo de serigrafía y el archivo de corte. Todos estos archivos están en formato PDF y están listos para ser utilizados por un fabricante de PCB.

Además, se incluyen las especificaciones de los componentes electrónicos utilizados en la PCB, así como una lista de materiales completa que indica la cantidad y el número de pieza de cada componente.

Este anexo es una herramienta para aquellos que deseen replicar el diseño y construcción de la PCB de iluminación con LEDs. Los archivos de KiCAD se encuentran en el Anexo 5, dispuestos de forma que permitan una fácil visualización y edición del diseño.

Anexo C. Códigos interfaz

Para poder desplegar la red neuronal en cada uno de los sistemas embebidos es necesario hacer uso de la interfaz de usuario que permita darle uso, de forma sencilla, práctica y cómoda. Además para realizar la toma de datos que permitan aumentar la base de datos también se requiere de una interfaz que permita hacer esta tarea de forma eficiente.

El presente Anexo contiene los códigos a cada una de las interfaces desplegadas en las tarjetas de desarrollo, dichos códigos funcionan tanto para Jetson Nano como para Raspberry PI 4B. También se encuentra los archivos de recursos usados por cada interfaz, tales como iconos e imágenes que se muestran. También se incluye todo archivo necesario para el funcionamiento de cada interfaz.

Anexo D. Códigos para la construcción de las redes neuronales

Para construir una red neuronal es importante seguir una serie de pasos previos para asegurar una buena metodología en el *machine learning*. Uno de los pasos más importantes es la limpieza de la base de datos, la cual consiste en eliminar aquellos datos que no representen la característica deseada o que puedan generar ruido en el modelo. Una vez que se cuenta con una base de datos limpia, es necesario dividirla en tres grupos: entrenamiento, validación y prueba, utilizando el código Division_datos. Este código permite realizar la división de los datos según un porcentaje específico.

Posteriormente, se realizan transformaciones a las imágenes con el objetivo de crear nuevos datos que ayuden al modelo a generalizar el error. Esto se logra con el código Aumento_datos, el cual aplica diferentes técnicas de aumento de datos, como rotación, desplazamiento, cambio de brillo, entre otros.

Finalmente, se cuenta con los códigos para construir la red neuronal, incluyendo la red neuronal de jerarquía reducida y las tres arquitecturas de *transfer learning*.

Anexo E. Repositorio en GitHub

Se ha creado un repositorio en GitHub con fines académicos para fomentar la contribución al proyecto. También se ha incluido un manual de usuario que describe cómo utilizar el dispositivo, así como todos los códigos y diseños necesarios para replicar el proyecto fácilmente.