

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE TELECOMUNICACIONES

Tema:

IDENTIFICACIÓN TEMPRANA DE PRESENCIA DE PLAGAS EN CULTIVOS DE AMBIENTE CONTROLADO EMPLEANDO VISIÓN ARTIFICIAL Y DEEP LEARNING

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Telecomunicaciones

ÁREA: Programación y redes

LÍNEA DE INVESTIGACIÓN: Tecnologías de la información y Sistemas de control.

AUTOR: Erick Fernando Lascano Villafuerte

TUTOR: Ing. Edgar Patricio Córdova Córdova, Mg.

Ambato - Ecuador

febrero – 2025

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: IDENTIFICACIÓN TEMPRANA DE PRESENCIA DE PLAGAS EN CULTIVOS DE AMBIENTE CONTROLADO EMPLEANDO VISIÓN ARTIFICIAL Y DEEP LEARNING, desarrollado bajo la modalidad Proyecto de Investigación por el señor Erick Fernando Lascano Villafuerte, estudiante de la Carrera de Telecomunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, febrero 2025.

Firmado electrónicamente por:
EDGAR PATRICIO
CORDOVA CORDOVA

Ing. Edgar Patricio Córdova Córdova, Mg.

TUTOR

AUTORÍA

El presente trabajo de titulación con el tema: IDENTIFICACIÓN TEMPRANA DE PRESENCIA DE PLAGAS EN CULTIVOS DE AMBIENTE CONTROLADO EMPLEANDO VISIÓN ARTIFICIAL Y DEEP LEARNING es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, febrero 2025.



Erick Fernando Lascano Villafuerte

C.C. 1805770599

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, febrero 2025.



Erick Fernando Lascano Villafuerte

C.C. 1805770599

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor Erick Fernando Lascano Villafuerte, estudiante de la Carrera de Telecomunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado IDENTIFICACIÓN TEMPRANA DE PRESENCIA DE PLAGAS EN CULTIVOS DE AMBIENTE CONTROLADO EMPLEANDO VISIÓN ARTIFICIAL Y DEEP LEARNING, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con el señor Presidente del Tribunal.

Ambato, febrero 2025.

Firmado electrónicamente por:
FRANKLIN OSWALDO
MAYORGA MAYORGA

Ing. Franklin Oswaldo Mayorga Mayorga, Mg.
PRESIDENTE DEL TRIBUNAL

Firmado electrónicamente por:
GUSTAVO DANILO
MACHADO MERINO

Ing. Gustavo Danilo Machado Merino MSc.
PROFESOR CALIFICADOR

Firmado electrónicamente por:
MARTHA ESPERANZA
SEVILLA ABARCA

Ing. Martha Esperanza Sevilla Abarca PhD.
PROFESOR CALIFICADOR

DEDICATORIA

El presente proyecto de investigación dedico a mis padres, por ser los pilares fundamentales al inculcarme los valores y guiarme siempre por el camino de esfuerzo y la dedicación.

A mis hermanas quienes, con su apoyo, y cariño y palabras de aliento me ayudaron para poder cumplir con la etapa del proyecto.

AGRADECIMIENTO

Agradezco a Dios por darme la vida, la fortaleza y la sabiduría necesarias para superar cada desafío en este camino. Su presencia ha sido mi guía en cada paso en la vida y en este proyecto.

A mis padres, quienes con su amor, esfuerzo y sacrificio han sido el pilar fundamental en mi vida. Gracias por ser mi inspiración y por enseñarme que con trabajo y dedicación todo es posible.

A mis hermanas, por su apoyo incondicional, sus palabras de aliento y su confianza en mí. Gracias por estar siempre presentes y por recordarme que nunca estoy solo en este camino.

A mis docentes, especialmente a mi tutor de tesis, por su guía, paciencia y valiosas enseñanzas a lo largo de este proyecto. Sus aportes han sido esenciales para la realización de este trabajo.

ÍNDICE GENERAL DE CONTENIDOS

PORTADA ..... **i**

APROBACIÓN DEL TUTOR ..... **ii**

AUTORÍA ..... **iii**

DERECHOS DE AUTOR ..... **iv**

APROBACIÓN DEL TRIBUNAL DE GRADO ..... **v**

DEDICATORIA ..... **vi**

AGRADECIMIENTO ..... **vii**

ÍNDICE GENERAL DE CONTENIDOS ..... **viii**

ÍNDICE DE TABLAS ..... **xii**

ÍNDICE DE FIGURAS ..... **xiv**

ÍNDICE DE ANEXOS ..... **xviii**

RESUMEN EJECUTIVO ..... **xix**

ABSTRACT ..... **xx**

CAPÍTULO I. MARCO TEÓRICO **1**

1.1 Tema de investigación 1

1.1.1 Planteamiento del problema 1

1.2 Antecedentes investigativos 2

1.3 Fundamentación teórica 4

1.3.1 Agricultura 5.0 4

1.3.2 Cultivos en Invernaderos 6

1.3.3 Tipos de plagas en invernaderos 6

1.3.4 Identificación de plagas 9

1.3.5 Métodos de monitoreo de plagas 10

1.3.6 Inteligencia Artificial 11

1.3.7 Deep Learning 14

1.3.8 Visión Artificial 16

1.3.9 Visión Artificial aplicado en la agricultura 19

1.3.10 Métricas de evaluación y rendimiento del modelo 19

1.3.11 Hardware y Software 21

1.4 Objetivos 26

1.4.1 Objetivo general 26

1.4.2 Objetivos específicos 26

CAPÍTULO II. METODOLOGÍA 27

2.1 Materiales 27

2.2 Métodos 28

2.2.1 Modalidad de la investigación 28

2.2.2 Recolección de información 28

2.2.3 Procesamiento y análisis de datos 29

CAPÍTULO III. RESULTADOS Y DISCUSIÓN 30

3.1 Desarrollo de la propuesta 30

3.2 Cultivos de plantas medicinales en ambiente controlado Susudel. 30

3.2.1 Plagas de las plantas medicinales 33

3.2.2 Características visuales de las plagas a detectar 34

3.3 Arquitectura del Sistema 34

3.4 Tecnologías de Hardware y Software Utilizados 36

3.4.1 Selección de Cámaras 36

3.4.2 Selección de microcomputador 37

3.4.3 Selección de la arquitectura de Deep Learning. 37

3.4.4 Selección de la base de datos 39

3.4.5 Selección de la plataforma para el desarrollo de la interfaz. 39

3.5 Descripción del sistema. 40

3.6 Entrenamiento de la Red Neuronal para la Detección de plagas 41

3.6.1 Dataset para el Entrenamiento. 41

3.6.2 Proceso de entrenamiento 45

3.7 Implementación del sistema 49

3.7.1 Librerías instaladas 49

3.7.2 Almacenamiento de los datos del sistema 50

3.7.3 Integración del sistema 52

3.7.4 Interfaz de Usuario 57

3.7.5 Instalación del sistema dentro del invernadero 60

3.8 Resultados del entrenamiento del modelo. 64

3.8.1 Predicción por lote de validación 66

3.8.2 Precisión del entrenamiento 67

3.8.3 Costo computacional 68

3.8.4 Pruebas de funcionamiento del modelo entrenado 68

3.9 Resultados del funcionamiento del sistema de detección de plagas 70

3.9.1 Resultados de las pruebas de Funcionamiento. 73

3.9.2 Comparación con sistemas similares 76

3.10 Presupuesto 77

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES 79

4.1 Conclusiones 79

4.2 Recomendaciones 81

REFERENCIAS BIBLIOGRÁFICAS 82

ANEXOS 90

ÍNDICE DE TABLAS

Tabla 1. Estructura de la matriz de confusión. 20

Tabla 2. Comparación de cámaras. 21

Tabla 3. Comparación de microcomputadoras. 22

Tabla 4. Comparación de modelos de Deep Learning. 23

Tabla 5. Comparación de técnicas para la identificación de plagas en cultivos con redes neuronales. 23

Tabla 6. Comparación de bases de datos. 25

Tabla 7. Comparación de software para la interfaz 25

Tabla 8. Materiales Utilizados. 27

Tabla 9. Criterios de recolección de información. 29

Tabla 10. Principales plagas de las plantas medicinales. 33

Tabla 11. Características visuales de las plagas 34

Tabla 12. Comparación de la v5 y v8 de YOLO. 38

Tabla 13. Conjunto de datos utilizados 42

Tabla 14. Características del dataset para la detección. 43

Tabla 15. Resultados de entrenamiento con distintas épocas 48

Tabla 16. Instalación de librerías para la implementación del sistema. 50

Tabla 17. Costo computacional 68

Tabla 18. Pruebas del funcionamiento del sistema 73

Tabla 19. Comparación con sistemas similares. 77

Tabla 20. Presupuesto. 78

ÍNDICE DE FIGURAS

Figura 1. Teledetección en la Agricultura 5.0 5

Figura 2. Cultivo en Invernadero 6

Figura 3. Estados de la mosca blanca 7

Figura 4. Plaga Botrytis 8

Figura 5. Etapas de vida de la araña roja 8

Figura 6. Estados de la plaga minador de hoja 9

Figura 7. Pulgones o también conocidos como Áfidos 9

Figura 8. Identificación de plagas 10

Figura 9. Relación de las distintas subáreas de la IA 12

Figura 10. Proceso de Machine learning (ML) 13

Figura 11. Número de soluciones en Big data agrícola y ML 14

Figura 12. Arquitectura CNN 15

Figura 13. Visión por computadora y sus subconceptos 17

Figura 14. Etapas de un sistema de visión artificial 18

Figura 15. Diagrama de bloques de un sistema de Visión artificial 19

Figura 16. Sistema de control de calidad, guiado por visión 19

Figura 17. Clavel (Dianthus caryophyllus) 31

Figura 18. Hierba Luisa (Cymbopogon citratus) 31

Figura 19. Manzanilla (Matricaria chamomilla) 32

Figura 20. Menta (Mentha spicata) 32

Figura 21. Cedrón (Aloysia citrodora) 33

Figura 22. Arquitectura del sistema. 35

Figura 23. Cámara IMX219. 36

Figura 24. Microcomputador NVIDIA Jetson Nano..... 37

Figura 25. Arquitectura del modelo YOLOv8. 39

Figura 26. Diagrama Funcional del sistema. 40

Figura 27. Estructura del dataset para Yolo. 41

Figura 28. Etiquetado de las imágenes. 44

Figura 29.Aumento de datos y Conversión de los datos a formato de Yolo. 44

Figura 30. Archivo data.yaml generado por roboflow. 45

Figura 31. Esquema del Entrenamiento del Modelo. 45

Figura 32. Comandos para montar el drive en Colab y descomprimir el dataset. 46

Figura 33. Instalación de la librería ultralytics. 46

Figura 34. Configuración de rutas del archivo data.yaml. 46

Figura 35. Comandos para el entrenamiento del modelo YOLO. 47

Figura 36. Ejecución del entrenamiento. 47

Figura 37. Carpera runs resultante del entrenamiento. 48

Figura 38. Capas del modelo para el sistema. 49

Figura 39. Tabla para las detecciones en tiempo real. 51

Figura 40.Tabla para las detecciones de las capturas programadas. 51

Figura 41. Diagrama entidad-relación. 52

Figura 42. Variables de Control. 52

Figura 43. Conexión a la base de datos. 53

Figura 44. Captura de los frames y gestión de las cámaras. 54

Figura 45. Detección de los objetos con YOLO. 54

Figura 46. Procesamiento de las detecciones. 54

Figura 47. Programar capturas automáticas. 55

Figura 48. Captura de las imágenes programadas. 56

Figura 49. Procesamiento de las capturas programadas. 56

Figura 50. Configuración de telegram y envió de notificaciones. 57

Figura 51. Interfaz de usuario para el sistema de detección de plagas. 58

Figura 52. Bloque 1, detección en tiempo real. 58

Figura 53. Bloque 2, Control de cámara IP y programación de horarios para las capturas. 59

Figura 54. Bloque 3, Visualización de las detecciones. 60

Figura 55. Representación 3D del sistema instalado. 60

Figura 56. Soportes para las cámaras. 61

Figura 57. Espacio designado para el servidor. 62

Figura 58. Instalación de las cámaras. 62

Figura 59. Soporte para el celular. 63

Figura 60. Estación de monitoreo instalada. 63

Figura 61. Instalación completa del sistema. 64

Figura 62. Métricas de rendimiento del modelo entrenado..... 65

Figura 63. Matriz de confusión. 65

Figura 64. Predicción por lotes para la clase araña roja. 66

Figura 65. Predicción por lotes para la clase mosca blanca. 66

Figura 66. Predicción por lotes para la clase Pulgón. 67

Figura 67. Precisión del modelo entrenado. 67

Figura 68. Prueba de detección, mosca blanca. 69

Figura 69. Prueba de detección, araña roja. 69

Figura 70. Prueba de detección, Pulgón. 70

Figura 71. Detección en Tiempo real. 70

Figura 72. Programar captura de imagen de la cámara. 71

Figura 73. Sistema de alerta en Telegram. 72

Figura 74. Funcionamiento de la estación de detección. 72

Figura 75. Análisis estadístico del sistema. 75

ÍNDICE DE ANEXOS

Anexo A. Especificaciones técnicas, NVIDIA Jetson Nano..... 90

Anexo B. Especificaciones técnicas, cámara IMX219 91

Anexo C. Instalación del Sistema 92

Anexo D. Estación de detección. 93

Anexo E. Estación de detección. 94

RESUMEN EJECUTIVO

La producción de cultivos en ambientes controlados proporciona condiciones ideales para su desarrollo, existen factores como las plagas, que afectan la producción y calidad de dicho cultivo. Lo que genera un impacto negativo en la economía de los agricultores. Con el fin de mejorar la producción de los agricultores se han integrado herramientas tecnológicas como Agricultura 5.0.

Basado en este contexto, se realizó un estudio con el objetivo de implementar un sistema para la detección de plagas, empleando tecnologías de Visión Artificial y Deep Learning. Es muy importante realizar detecciones tempranas de plagas en cultivos para no tener perdidas de producción, por ello, el sistema se basa en el uso de una red neuronal capaz de realizar detecciones precisas de las distintas plagas.

El sistema consta de cuatro etapas: adquisición, entrenamiento, procesamiento y visualización. En la etapa de adquisición, se utilizó cuatro cámaras que realizaran las capturas de las imágenes y video. El entrenamiento se realizó con los datos recolectados y un modelo que sea capaz de trabajar en equipos de recursos limitados y que tenga una buena precisión para la detección. Para el procesamiento de las imágenes se ocupa un microcomputador optimizado para trabajar con inteligencia artificial, para la visualización y gestión de información se realizó una interfaz gráfica que sea capaz de mostrar los datos obtenidos. El modelo entrenado demostró una precisión de 95.7% en la detección de plagas y después de realizar varias pruebas del sistema demostró una confiabilidad del 93.7%, siendo un sistema confiable.

Palabras clave: Visión Artificial, Deep Learning, detección de plagas, ambientes controlados.

ABSTRACT

The cultivation of crops in controlled environments offers optimal conditions for their growth. However, the presence of pests can adversely impact both the quantity and the quality of the produce. This phenomenon exerts a deleterious effect on the economic viability of farmers. To address these challenges, farmers have adopted technological solutions, such as Agriculture 5.0, to enhance their productivity and quality of produce.

A study was conducted with the objective of implementing a system for pest detection, utilizing Computer Vision and Deep Learning technologies. It is imperative to detect pests in crops at an early stage to avert production losses. Consequently, the system is predicated on a neural network capable of accurately detecting various pests.

The system is comprised of four distinct stages: acquisition, training, processing, and visualization. In the initial acquisition stage, four cameras were utilized to capture images and video. The training stage entailed the utilization of collected data in conjunction with a model adept at functioning with constrained resources while maintaining optimal detection accuracy. The image processing stage entailed the utilization of a microcomputer that had been optimized to operate in conjunction with artificial intelligence. The visualization and information management stage involved the development of a graphic interface capable of displaying the data obtained. The trained model demonstrated an accuracy of 95.7% in the detection of pests, and subsequent system tests yielded a reliability of 93.7%, thus confirming the system's reliability.

Keywords: Computer Vision, Deep Learning, pest detection, controlled environments.

CAPÍTULO I. MARCO TEÓRICO

1.1 Tema de investigación

IDENTIFICACIÓN TEMPRANA DE PRESENCIA DE PLAGAS EN CULTIVOS DE AMBIENTE CONTROLADO EMPLEANDO VISIÓN ARTIFICIAL Y DEEP LEARNING

1.1.1 Planteamiento del problema

La demanda de los alimentos crece con el aumento de la población mundial y gracias a este aumento se estima que se necesitara un crecimiento del 50% en la producción de alimentos para el año 2050, pero la producción de alimentos se ve afectada por plagas y enfermedades que amenazan gravemente a la seguridad alimentaria perdiendo hasta un 40% de los cultivos cada año, según datos de la Organización de las Naciones Unidas para la alimentación y la Agricultura (FAO). Ante dicha situación la FAO toma medidas como la contratación de seguros agrícolas y la implementación de tecnología sostenible [1].

En Ecuador, agricultores de los cantones Palenque y Vinces experimentaron fuertes pérdidas en los cultivos, por causa de distintos factores como plagas, enfermedades e inundaciones. Dichos eventos alertaron al Ministerio de Agricultura, Ganadería, Acuacultura y Pesca (MAGAP), lo que llevó a la activación del Seguro Agrícola del proyecto AgroSeguro. La evaluación de los distintos cultivos afectados reveló que el 78% de las pérdidas fueron por plagas, el 11% fueron por enfermedades, el 9% se atribuyó a inundaciones y el 2% a otras causas [2].

En la Sierra del Ecuador, especialmente en las provincias de Bolívar, Cotopaxi, Chimborazo, Azuay e Imbabura, es esencial para la seguridad alimentaria regional. A pesar de su importancia socioeconómica, el rendimiento de este cultivo en la región es de los más bajos, con alrededor de 3,68 toneladas por hectárea de choclo y 1,63 toneladas por hectárea de grano seco. Este bajo rendimiento se agrava debido a la escasa aplicación de tecnologías y recomendaciones de manejo. La necesidad de aumentar el rendimiento del maíz sube al menos un 18% para el año 2030, destaca la

urgencia de abordar los desafíos específicos que enfrentan este cultivo en la Sierra ecuatoriana [3].

La proliferación de plagas en cultivos de ambiente controlado cuenta con una serie de factores que contribuyen a este problema y sus consecuencias. Las condiciones ambientales, como temperaturas cálidas y humedad alta, que se presentan dentro de los invernaderos, son un habitad favorable para el aumento de plagas. Todos estos factores, descritos anteriormente, conducen a un aumento masivo de plagas en los cultivos, lo cual resulta desastroso para la producción agrícola.

La falta de un monitoreo continuo en los cultivos permite que las plagas crezcan sin control, esto da como resultado una deficiencia en la calidad y una reducción en la cantidad de los cultivos. Tanto la reducción de la calidad como la reducción de la cantidad de cultivos es un efecto negativo directo de la falta de monitoreo, generando pérdidas económicas para el agricultor y una disminución en alimentos para la población.

El uso excesivo de pesticidas como medida para el control de plagas presenta varios efectos que causan daño a los cultivos. Aunque el uso de pesticidas es una solución para el control de las plagas, su uso indiscriminado puede llevar a que las plagas desarrollen resistencia a dichos pesticidas. Por lo tanto, la eficacia en el control de plagas disminuye y se necesitara de pesticidas más potentes o el empleo de métodos agresivos para el control de las plagas, lo que causa un impacto negativo en la salud humana y el medio ambiente.

1.2 Antecedentes investigativos

A continuación, se referencian tesis o artículos científicos que se han realizado en la misma línea de investigación las cuales está relacionado con algoritmos de clasificación de imágenes para el sector agrícola, en especial para la detección de insectos y plagas.

En el año 2022, en la Universidad Nacional de Chimborazo, Jácome Jaime, en su investigación realiza un prototipo para la detección de plagas y enfermedades en el tomate riñón. Este prototipo está conformado por 3 módulos, el módulo principal en

el cual se encuentra la tarjeta de desarrollo Google Coral y los módulos complementarios los cuales son la cámara coral de 5 megapíxeles y una pantalla LCD. El entrenamiento se la red neuronal se realizó con 16 épocas las cuales fueron suficientes para alcanzar su exactitud. Durante las pruebas del prototipo se proyecta en la pantalla el video donde se visualiza el resultado de la predicción de enfermedad, plaga o de ser el caso una planta sana [4].

En el año 2022, en la Universidad Nacional de La Plata, Sergio Hernán Valenzuela Cámara, en su investigación, desarrolló un sistema para la detección y clasificación de enfermedades en el tomate utilizando Deep Learning y visión por computadora. El sistema utiliza dos redes neuronales pre-entrenadas mediante transfer learning: la primera red, Faster Mask R-CNN, se emplea para la detección de las hojas de tomate, y la segunda red, una red neuronal convolucional, se utiliza para la clasificación de la enfermedad a partir de la hoja detectada. El proceso de clasificación incluye la identificación de síntomas asociados y recomendaciones de prevención y tratamiento en tiempo real. Este enfoque permitió mejorar significativamente la precisión en la detección y tratamiento oportuno de las enfermedades en las plantas de tomate, contribuyendo a reducir las pérdidas económicas en la agricultura [5].

En el año 2021, en la Universidad Nacional Abierta y a Distancia, Viviana Yineth Sierra Guzmán, en su investigación, desarrolla un algoritmo para la detección de la plaga de mosca blanca en las hojas de plátano. Este algoritmo utiliza técnicas avanzadas de procesamiento de imágenes y un clasificador de máquina de soporte vectorial. Para la implementación, se emplea el software Matlab, en el cual se realizan procesos como cambios de modelo de color, umbralización, binarización y la aplicación de filtros tipo Sobel, entre otros. El clasificador supervisado permite distinguir entre hojas sanas y hojas afectadas por la plaga. Durante las pruebas del algoritmo, se logran identificar de manera precisa las hojas afectadas, permitiendo así un monitoreo eficiente y la toma de acciones correctivas para mejorar la producción del plátano [6].

En el año 2020, en la Universidad EIA, Natalia López Grisales desarrolló un sistema para la identificación y conteo de insectos (Mosca Blanca, Trips y Minador de Hoja) capturados en trampas adhesivas mediante visión artificial e inteligencia artificial en

los cultivos de flores de la empresa Flores El Trigal. El sistema incluye una aplicación móvil desarrollada en Android Studio que utiliza el modelo de detección de objetos YOLO (You Only Look Once) para clasificar y cuantificar los insectos en tiempo real. Utilizando la metodología Design Thinking, se evaluaron varios métodos computacionales, seleccionando YOLO por su mejor desempeño. La implementación del modelo en una aplicación móvil reduce significativamente el tiempo de monitoreo manual y facilita el almacenamiento digital de la información, apoyando la transformación digital de la empresa. La investigación también analiza el retorno de la inversión basado en las horas ahorradas por la automatización del proceso [7].

En el año 2019, en la universidad del Valle de Puebla, E. Berra Villaseñor, M. Pérez Castañeda y J.R. Cuautle Parodi, en su estudio implicó la selección de tres plagas específicas del cultivo de jitomate, sobre las cuales se entrenó una red neuronal artificial. Esta red fue capacitada para detectar y clasificar las plagas mediante el algoritmo de detección en imágenes de hojas del cultivo, utilizando la red neuronal artificial de convolución basada en Keras del framework TensorFlow. Para evaluar el rendimiento de la red neuronal, se realizaron pruebas de entrenamiento utilizando tanto CPU como el acelerador Intel Neuronal Stick 2. Durante estas pruebas, se midió el tiempo requerido para obtener pesos sinápticos óptimos que permitieran una detección mínima del ochenta y nueve por ciento de identificación de la plaga. Todos los algoritmos fueron implementados en lenguaje Python bajo el framework de Intel [8].

1.3 Fundamentación teórica

1.3.1 Agricultura 5.0

La Agricultura 5.0 es la evolución de la Agricultura 4.0, en donde se integra varias tecnologías como aprendizaje profundo, inteligencia artificial sistemas autónomos y robótica con el objetivo de aumentar la eficiencia de producción agrícola y reducir el impacto ambiental [9].

La agricultura digital, también conocida como Agricultura 5.0, se distingue de los métodos agrícolas tradicionales por varios aspectos clave. En primer lugar, mejora la eficiencia en la recopilación de datos, permitiendo obtener grandes volúmenes de

información en períodos y espacios determinados, lo que facilita un análisis detallado de los cultivos. Además, optimiza la precisión de las mediciones, ya que el uso de tecnologías avanzadas como drones, sensores e inteligencia artificial garantiza datos exactos con un margen de error mínimo. Finalmente, la rapidez en el procesamiento de información es un factor fundamental, pues una gestión ágil de los datos permite tomar decisiones en tiempo real, beneficiando directamente a los agricultores [10].

La Figura 1 muestra un modelo de teledetección (RS) aplicado a la agricultura de la quinta revolución industrial. Se integran varias tecnologías, como drones, inteligencia artificial, aprendizaje automático, robots autónomos, entre otros, para la recopilación de datos. Los datos son procesados generando información útil para una toma de decisión anticipada y eficiente [11].

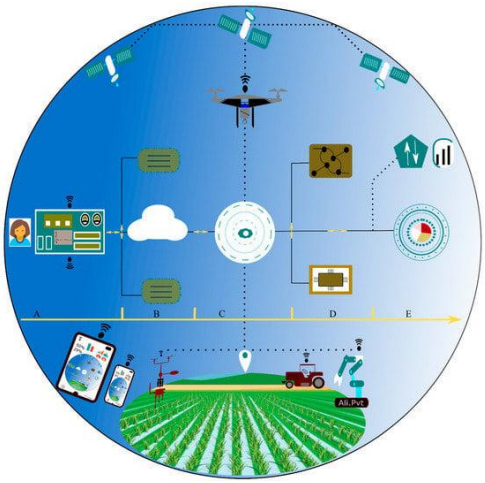


Figura 1. Teledetección en la Agricultura 5.0 [11]

a. Beneficios de la Agricultura 5.0

La Agricultura 5.0 contribuye a la producción agrícola con una amplia variedad de beneficios claves, entre los cuales destacan [12]:

- Aumento de la productividad.
- Sostenibilidad Ambiental.
- Reducción de Costos.
- Resiliencia ante el cambio Climático.

1.3.2 Cultivos en Invernaderos

La cosecha en invernaderos, se refieren la producción de cultivos en estructuras protegidas, estos invernaderos permiten controlar las condiciones ambientales como humedad, radiación solar, temperatura, concentración de CO2 entre otros. Uno de los beneficios que ofrece cultivar en invernaderos es que aumenta la precocidad de la fructificación, el cual permite cosechar de 50 – 60 días antes que la cosecha al aire libre [13]. En la Figura 2 se muestra en ejemplo de cultivos en invernaderos.



Figura 2. Cultivo en Invernadero

Si bien los invernaderos crean condiciones adecuadas para la producción de cultivos, a veces estas mismas condiciones llegan hacer propicias para la aparición de nuevas plagas en las plantas, esto sucede porque algunas plagas pueden adaptarse al cambio del ambiente o simplemente es su habitad natural.

1.3.3 Tipos de plagas en invernaderos

Si bien los invernaderos proporcionan ambientes propicios para la producción de cultivos, es necesario mantener un control estricto de plagas, debido a que las condiciones creadas dentro de estos invernaderos como humedad y temperaturas favorables, dan paso a la aparición de nuevas plagas y su rápida proliferación [14].

Las plagas que se encuentran habitualmente dentro de invernaderos son los siguientes:

a. Mosca blanca

Esta plaga llamada Mosca blanca o también conocida como (Bemisia tabaci) es una plaga de la familia Aleyrodidae, y se puede hospedar en una gran variedad de plantas, por lo que causa daños en una gran variedad de cultivos [14].

Tiene un alto nivel de resistencia, y generalmente se las visualiza en la parte inferior de las hojas de la planta, esta plaga se va acumulando alrededor de las venas y son de color blanco. Si bien para las personas esta plaga no causa ningún daño, para las plantas causan daños importantes reduciendo la producción y generando perdidas [14]. En la Figura 3 se puede ver los distintos estados que pasa la mosca blanca.

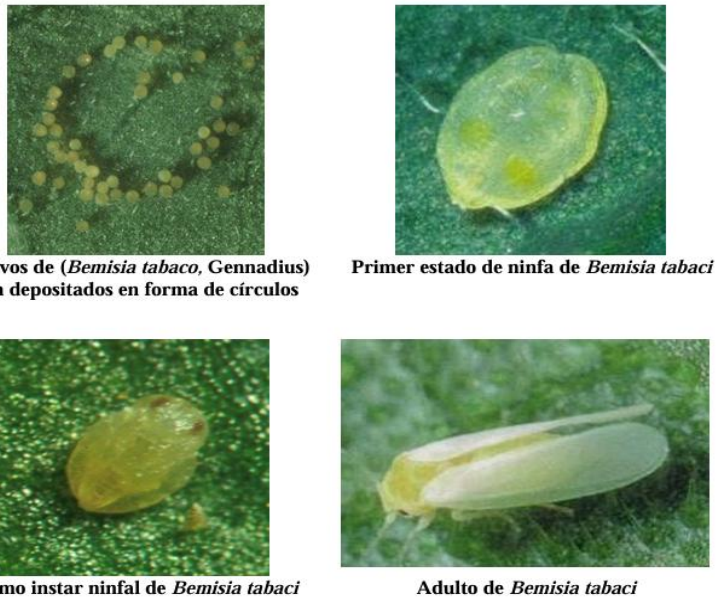


Figura 3. Estados de la mosca blanca [15]

b. Botrytis

También conocida como moho gris, es un tipo de plaga altamente destructiva que se puede encontrar en los cultivos que están en invernaderos. Son conocidos por atacar directamente a los tallos, las hojas y las flores de las plantas [14]. En la Figura 4 se muestra una representación de los efectos característicos de esta plaga que incluyen manchas grises, moho polvoriento, evidenciando el impacto negativo en la producción agrícola.



Figura 4. Plaga Botrytis [16]

c. Araña Roja

También conocida como tetanychus urticae, es otra de las plagas que se encuentran en los invernaderos, generalmente aparecen en invernaderos secos ya que no les gusta la humedad. Recibe el nombre de araña roja porque tiene la capacidad de tejer telarañas en las plantas reduciendo la capacidad fotosintética de la planta, los primeros síntomas de esta plaga aparecen unos puntos blancos o como cabeza de alfileres y se generan en el nervio principal de la hoja, después de que la plaga ya está avanzada es de un color rojizo [14]. En la Figura 5 se muestran las etapas de vida de la araña roja.

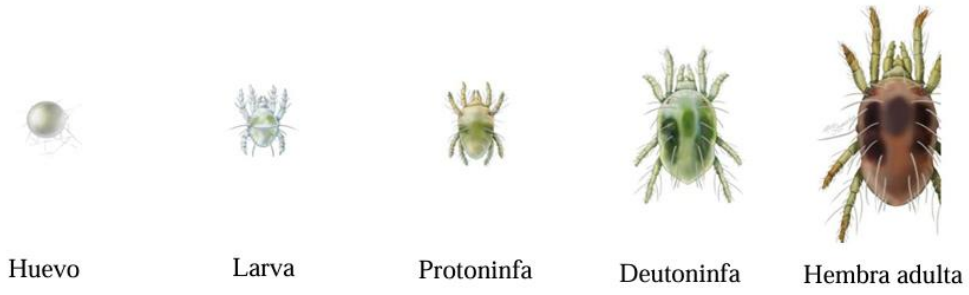


Figura 5. Etapas de vida de la araña roja [15]

d. Minador de hoja

Esta es una de las plagas más comunes en los invernaderos, si bien por sí sola no causa mucho daño a la planta, se puede dar una sobrepoblación y si no se detecta a tiempo es perjudicial para el cultivo, dificultando su control y afectando su desarrollo [14]. En la Figura 6 se presenta los estados de la plaga minador de hoja.



Figura 6. Estados de la plaga minador de hoja [15]

e. Áfidos

Son pulgones que su color vario entre blanco, amarillo, verde e incluso de color rosado. Se alimentan de las partes más ricas de las plantas que son las flores, raíces y hojas, cuando la planta se infecta de esta plaga se identifican por las hojas deformes o amarillentas. Esta plaga si es perjudicial para las personas ya que pueden crear reacciones alérgicas [14]. En la Figura 7 se muestra dos tipos de Áfidos.



Figura 7. Pulgones o también conocidos como Áfidos [15]

1.3.4 Identificación de plagas

Generalmente para la identificación de plagas en los cultivos, una persona debe monitorear las plantas para ver si contiene o no plagas y así planificar una estrategia

para el control de dicha plaga, en la mayoría de los casos los pasos utilizados para identificar plagas son los siguientes [17]:

- Tener un listado de plagas comunes que pueden afectar al cultivo.
- Aprender el ciclo de vida y el comportamiento que tiene la plaga.
- Conocer como se ve el daño que realiza la plaga en la planta.
- Saber reconocer las excreciones y secreciones de la plaga.
- Conocer muy bien los detalles de cada una de las plagas.

En la Figura 8 se presenta la identificación de plagas tradicional.



Figura 8. Identificación de plagas [17]

1.3.5 Métodos de monitoreo de plagas

a. Monitoreo en estaciones o grupos de plantas

Este tipo de monitoreo consiste en dividir en grupos de plantas uniformemente, para así observar la fluctuación de plagas o enfermedades que se encuentran en el cultivo. Este método permite dar respuesta a diversos factores como el manejo de las plagas, las condiciones climáticas o etapas fenológicas de las plantas [18].

b. Monitoreo al azar

Este método es utilizado para la detección temprana de plagas, ya que, si existe un foco de alguna plaga o enfermedad, este se debe monitorear por separado para evitar que se mezcle con el resto del cultivo, lo que afectaría a la toma de decisiones para el control [18].

c. Trampas amarillas y azules

Este método utiliza un plástico de color amarillo o azul cubierto de una sustancia adherente, el cual se coloca en lugares estratégicos para obtener muestras de los insectos, estos insectos al ser atraídos por el color de los plásticos se quedan pegados en las trapas, y posteriormente se realiza un análisis de la plaga y se toma acciones para el control de dicha plaga [18].

d. Trampas con feromonas

Este método se utiliza para el control y monitoreo de poblaciones de plagas específicas. Estas contienen feromonas sintéticas que atraen a los insectos, haciéndolos que se queden pegados en una superficie adherente, su uso no es para todas las especies solo a especies particulares, pero ofrecen un control de la población de plagas mientras se realiza el monitoreo, reduciendo las dispersión y reproducción de dichas plagas [18].

1.3.6 Inteligencia Artificial

La inteligencia artificial es un campo de estudio el cual busca el desarrollo de sistemas y máquinas que son capaces de presentar un comportamiento inteligente, simulado al de los seres humanos. En 1961, fue en donde se desarrolló el primer programa computacional de inteligencia artificial, llamado “Logic Theory Machine”, el cual se realizó con la finalidad de descubrir demostraciones de teoremas en lógica simbólica [19].

En la Figura 9 se presenta un diagrama de Venn, en el cual se muestra la relación jerárquica entre la Inteligencia Artificial, el aprendizaje automático (Machine Learning) y el aprendizaje profundo (Deep Learning), la IA es el concepto más amplio, el cual incluyen programas que pueden razonar, actuar y adaptarse. Dentro de ella, el aprendizaje automático son algoritomos que mejora suu rendimiento a medida que están expuestos a grandes cantidades de datos y por último el aprendizaje profundo, utiliza redes neuronales que aprenden a partir de grandes cantidades de datos con la finalidad de resolver problemas complejos [19].

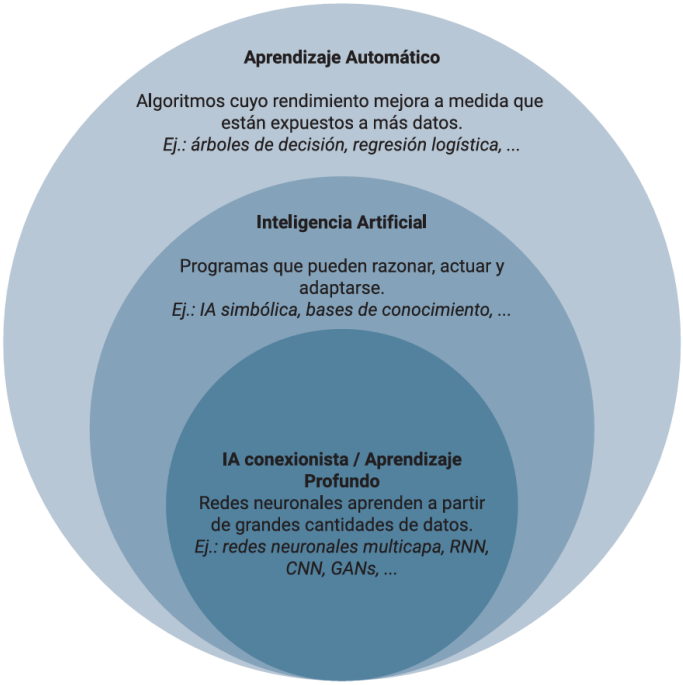


Figura 9. Relación de las distintas subáreas de la IA [19]

b. Limitaciones de la IA

Si bien la inteligencia artificial es una tecnología innovadora y con varios beneficios, existen algunas limitaciones que impiden que sea accesible para todas las personas, estos son [19]:

- Grandes necesidades de recursos computacionales y enormes cantidades de datos para el entrenamiento.
- Complejidad de los algoritmos de aprendizaje profundo, pues su interpretación puede resultar difícil para los seres humanos.

c. Machine Learning

El aprendizaje automático (ML), es un subcampo de la informática en el cual las computadoras son capaces de aprender sin ser programadas explícitamente para alguna tarea. Hoy en día los algoritmos de ML se utilizan para resolver problemas que los humanos les falta experiencia, como la predicción del clima, identificación de enfermedades, reconocimiento de patrones entre otros [20].

El proceso de ML se divide en tres aspectos claves: entrada de datos, construcción del modelo y generalización [20]. Como se muestra en la Figura 10.

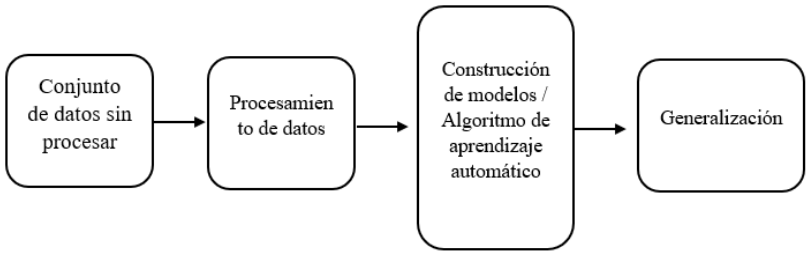


Figura 10. Proceso de Machine learning (ML) [20]

Los algoritmos de aprendizaje automático (ML) se categorizan en tres, aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [20].

- ***Aprendizaje supervisado.*** Este algoritmo realiza su aprendizaje con un supervisor o maestro. Trabaja con un conjunto de datos etiquetados, lo que significa que para cada entrada hay una salida [20].

- ***Aprendizaje no supervisado.*** Trabajan con datos no etiquetados, su objetivo es extraer conocimiento oculto del conjunto de datos con el que se entrenó [20].

- ***Aprendizaje por refuerzo.*** Este algoritmo tiene otro enfoque, el cual es aprender de su entorno a través de recompensas y castigos [20].

d. Machine learning aplicado a la Agricultura

El aprendizaje automático (ML) ha entrado ampliamente al campo de la agricultura, dando solución a distintos problemas que afectan la producción agrícola. Se utiliza para la predicción de rendimiento del cultivo, detección de enfermedades, detección de malezas que afectan el cultivo, reconocimiento de especies, manejo de suelo, entre otros. Junto con la combinación de sensores y ML permite el aumento de la calidad en la producción agrícola generando buenos resultados económicos para los agricultores [21].

En la Figura 11 se muestra la cantidad de soluciones que utilizan Big data y Machine learning en diferentes áreas de la agricultura.

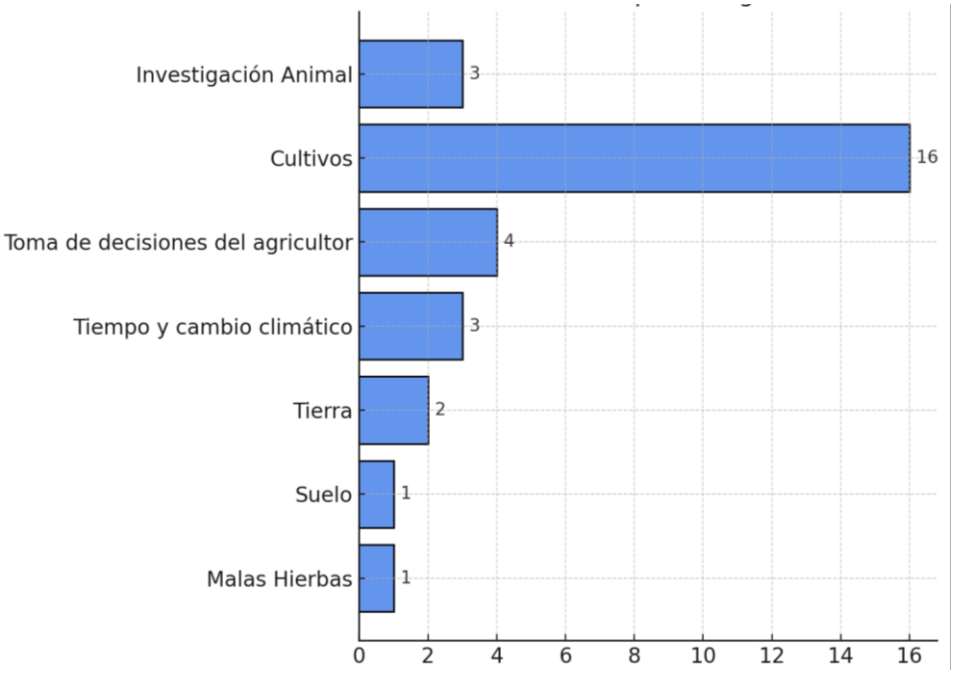


Figura 11. Número de soluciones en Big data agrícola y ML [21]

1.3.7 Deep Learning

El aprendizaje profundo con sus siglas en inglés (DL) se basa en Redes Neuronales Artificiales (RNA). Una RNA de aprendizaje profundo se diferencia de una RNA tradicional por su cantidad de capas ocultas, dado que puede contar con cientos de ellas, y cada una contiene múltiples unidades de procesamiento. La RNA de aprendizaje profundo más utilizada son el Perceptrón Multicapa [22].

a. Convolutional Neural Network (CNN)

Las redes neuronales convolucionales con sus siglas en inglés (CNN) es un algoritmo de aprendizaje profundo que se compone de múltiples capas convolucionales, capas de pooling y capas completamente conectadas. El diseño de La CNN se inspiró en la corteza visual de los animales, lo cual permite procesar imágenes de forma eficiente. La red neuronal convolucional es óptima para el procesamiento de imágenes, como detección de objetos, clasificación de imágenes, segmentación de imágenes entre otros [23]. En la Figura 12 se muestra la arquitectura de la red neuronal CNN.

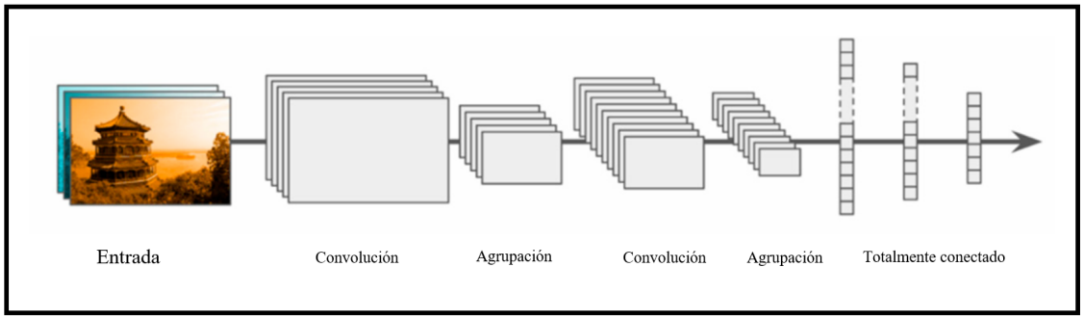


Figura 12. Arquitectura CNN [23]

- ***Capa Convolucional.*** Esta capa es la responsable de extraer las características visuales, es la capa más básica e importante de la red neuronal convolucional. Con la aplicación de filtros en la imagen obtenemos un mapa de activación, el cual tiene la ventaja de almacenar las características distintivas de una imagen determinada y también limita la cantidad de datos que deben procesarse simultáneamente [23].

- ***Capa de agrupamiento.*** Permite que una CNN agrupe todas distintas dimensiones de una imagen para reconocer el objeto suministrado, incluso si la forma del objeto esta distorsionada o tiene un ángulo diferente [23].

- ***Capa totalmente conectad.*** Esta es la capa final donde se alimenta la red neuronal, aquí ocurre todo el razonamiento y el cálculo de datos [23].

b. Deep Learning aplicado a la Agricultura

Con la incorporación del Deep Learning en el campo de la agricultura, se ha experimentado una transformación notable, especialmente en la estimación de rendimiento de los cultivos. Al integrar la teledetección con algoritmos de aprendizaje profundo, como las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN), los agricultores y científicos obtienen predicciones precisas y tempranas acerca de los cultivos y así tener una gestión eficaz de la producción agrícola [24].

- ***Identificación y clasificación de enfermedades en plantas.*** Las redes neuronales profundas como las redes neuronales convolucionales (CNN) se utilizan mucho para la detección temprana de enfermedades causadas por hongos, bacterias, plagas

en la planta, lo cual mejora significativamente en las pérdidas económicas para el agricultor [23].

- ***Identificación y clasificación de cultivos.*** El Deep Learning también se ocupa para la clasificación de especies de cultivos como malezas, plantas frutales, medicinales entre otras, toda esta clasificación se realiza mediante imágenes satelitales, drones, robots [23].

- ***Monitoreo de estrés hídrico.*** Se han propuesto modelos de Deep Learning para identificar si existe un estrés hídrico en ciertas áreas del terreno en donde se cultiva, con el fin de tener una buena gestión del agua en todo el cultivo [23].

- ***Predicción Meteorológica.*** Redes neuronales recurrentes como (RNN) se utilizan para predecir fenómenos meteorológicos como heladas, esto con el fin de planificar un plan para el manejo de sus cultivos y no tener pérdidas económicas [23].

1.3.8 Visión Artificial

En sus inicios, la visión por computadora se enfocaba en simular el sistema de visión humana. Sin embargo, con el pasar del tiempo ha evolucionado significativamente, generando una variedad de aplicaciones como la realidad virtual, reconocimiento de patrones, aprendizaje automático y la robótica. Gracias a su capacidad procesar y extraer características de un conjunto de datos de imágenes y videos, la visión por computadora es fundamental para el desarrollo de sistemas de aprendizaje profundo y aprendizaje automático [25].

En la Figura 13 se muestran los subconceptos que abarca la visión por computadora, cada uno de estos subconceptos ayuda a que se pueda procesar, interpretar y utilizar datos visuales de manera efectiva en varias aplicaciones tecnológicas como la detección de objetos, que identifica y localiza elementos dentro de una escena, en el apartado de la seguridad, aplicando el reconocimiento facial, la clasificación de imágenes, el cual organiza los datos en categorías predefinidas. Con el uso de todas estas tecnologías permite facilitar las tareas complejas que requieren de un análisis visual, ahorrando tiempo y aumentando la precisión [25].

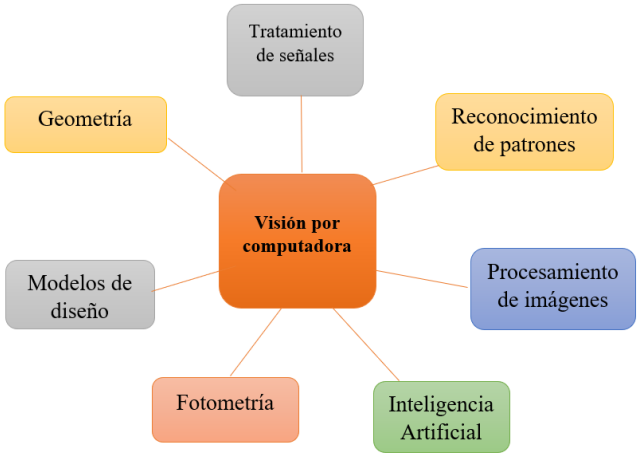


Figura 13. Visión por computadora y sus subconceptos [25]

a. Etapas en un proceso de Visión Artificial

Para llevar a cabo una tarea de visión artificial se deben cumplir varias etapas claves como se muestra en la Figura 14, cada una de estas etapas contribuyen al procesamiento y análisis efectivo de las imágenes, estas etapas son [26]:

- ***Adquisición de la imagen digital.*** En este paso se necesita de sensores y la capacidad para digitalizar la señal producida por el sensor [26].

- ***Etapa de procesado.*** El objetivo de esta etapa es realizar transformaciones y filtrados, con el fin de mejorar la imagen y tener mayores posibilidades de éxito [26].

- ***Etapa de segmentación.*** Esta etapa es muy importante, pues en ella se divide la imagen en regiones de interés, pero si se realiza una segmentación incorrecta existirá fallas en el sistema [26].

- ***Etapa de parametrización.*** Esta etapa es la encargada de extraer rasgos que producen alguna información cuantitativa de interés o rasgos que son básicos, con el fin de diferenciar diferentes clases de objetos [26].

- ***Etapa de reconocimiento y la interpretación.*** El reconocimiento o también llamado clasificación es el proceso en el cual se etiqueta a un objeto basado en la

información proporcionada por los descriptores. Con los rasgos extraídos se llega a una interpretación del objeto [26].

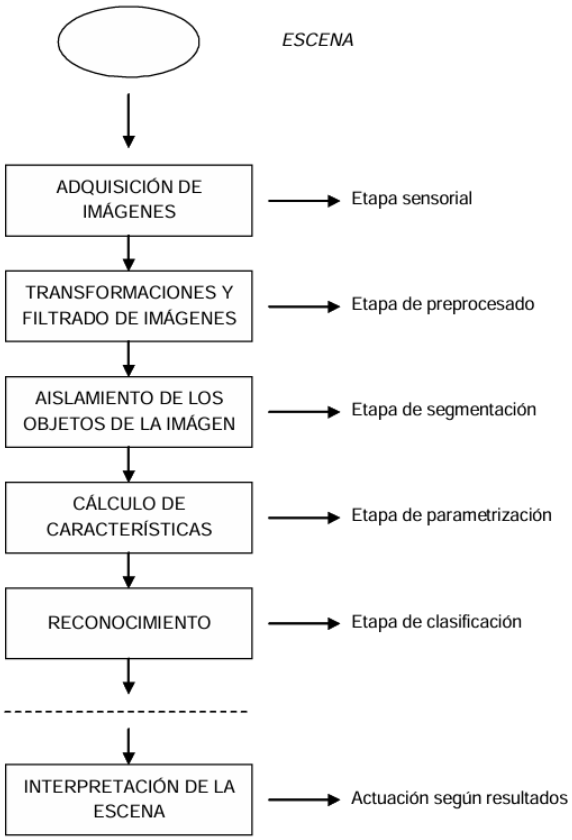


Figura 14. Etapas de un sistema de visión artificial [26]

b. Componentes de un sistema de visión

Para que un sistema de visión artificial funcione correctamente, requiere elementos de hardware mínimos necesarios, estos elementos son [26]:

- Sensor óptico
- Tarjeta de adquisición de imagen
- Computador
- Monitor de video

La Figura 15 muestra un diagrama de bloques, con los componentes principales de un sistema de visión artificial.

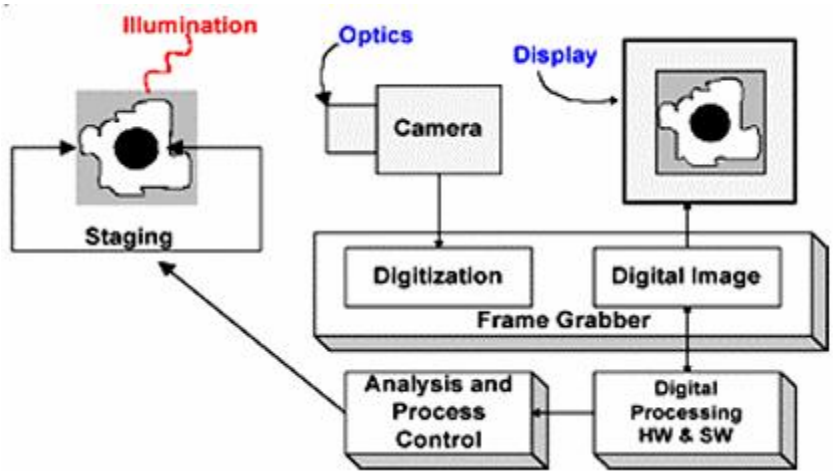


Figura 15. Diagrama de bloques de un sistema de Visión artificial [26]

1.3.9 Visión Artificial aplicado en la agricultura

En el sector agrícola, sistemas como visón artificial y robótica han dado lugar a la creación de máquinas inteligentes que son capaces de interactuar con el entorno en el que se encuentran, permitiendo una mejor gestión de recursos, cuidado de los cultivos hasta la planificación de la cosecha [27].

En la Figura 16 podemos observar un diagrama de un sistema de control de calidad el cual es guiado por visión artificial, el sistema cuenta con tres bloques claves, los cuales son, bloque de análisis, bloque de aprendizaje y bloque de decisión [27].

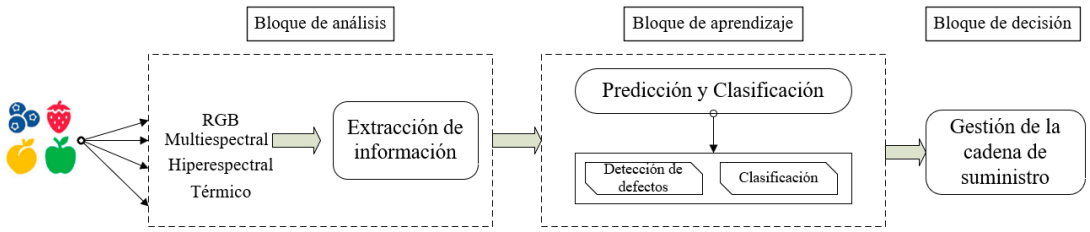


Figura 16. Sistema de control de calidad, guiado por visión [27]

1.3.10 Métricas de evaluación y rendimiento del modelo

Cuando se trabaja en detección de objetos con visión artificial, para evaluar el modelo es fundamental las métricas IoU y mAP. IoU (Intersection over Union) es la precisión para verificar la ubicación de los objetos que se detectaron dentro de la imagen y mAP

(mean Average Precision) es la precisión que se utiliza para la clasificación de los objetos [28].

- ***IoU (Intersection over Union).*** Esta métrica sirve para evaluar que tan bien predice la ubicación de un objeto detectado en la imagen, mide la superposición entre dos áreas, la caja delimitadora que es la predicción que genero el modelo entrenado y el Ground Truth que es la caja delimitadora verdadera, que fue marcada manualmente en el dataset y indica en donde está realmente el objeto [28].

- ***mAP (mean Average Precision).*** Esta métrica sirve para evaluar que tan buena es la clasificación y localización de los objetos detectados, en un conjunto de datos de prueba [28].

- ***Precision.*** Este mide que tan confiable son las detecciones positivas (TP) del modelo entrenado. Esto se mide con la siguiente formula [28]:

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)} \quad (1)$$

- ***Recall.*** Mide que tan capaz es el modelo de identificar correctamente todos los casos positivos que están presentes en un conjunto de datos. Un valor alto significa que el modelo detecta la mayoría de los casos positivos [28].

- ***Matriz de confusión.*** Es una tabla que se utiliza para evaluar el rendimiento de un modelo de clasificación que fue entrenado con un conjunto de datos. En la Tabla 1 se muestra la estructura de la matriz de confusión [28].

Tabla 1. Estructura de la matriz de confusión.

	Clase Positiva	Clase negativa
Detección	True Positive (TP)	False Positive (FP)
No Detección	False Negative (FN)	True Negative (TN)

A continuación, se describe los cuatro términos de la matriz de confusión [28]:

- **True Positives (TP). -** este representa casos en el que el modelo predice correctamente la clase positiva.

- **False Positive (FP). -** este representa los casos en el que el modelo predice la clase positiva, pero en verdad era la clase negativa.

- **True Negative (TN). –** este representa los casos en el que el modelo predice correctamente la clase negativa.

- **False Negative (FN). –** este representa los casos en el que el modelo predice la clase negativa, pero en realidad es la clase positiva.

1.3.11 Hardware y Software

La selección de tecnologías de hardware y software es clave para que el funcionamiento de sistema de detección de plagas sea optimo. A continuación, se elabora un análisis técnico y económico en el que se comparan varias opciones de hardware y software, con el fin de determinar la mejor alternativa que se adapte a los requerimientos del sistema.

a. Cámaras

En la Tabla 2 se muestra la comparación de tres cámaras que se podrían utilizar en la detección de plagas, se detalla las características más importantes de cada cámara.

Tabla 2. Comparación de cámaras [29], [30], [31].

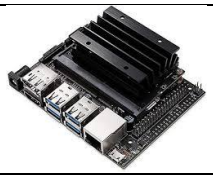
Características	Cámara web Logitech C270	Cámara IMX219	REOLINK RLC-811A
Imagen			
Resolución	3 MP	8MP	8MP
Sensor de la imagen	No	Sony IMX219	1/2.8" CMOS Sensor
Angulo de vision	180 grados	160 grados	123 Grados
Zoom	No	No	5x Optical Zoom
Conexión	USB 2.0	CSI (Camera Serial Interface) , USB2.0	Cableado/PoE

b. Microcomputador

La selección de un buen microcomputador es esencial para que el sistema de visión artificial funcione correctamente, ya que debe brindar un buen procesamiento de video

en una velocidad moderada. En la Tabla 3 se realiza la comparación de tres microcomputadoras, detallando las características más importantes para el análisis.

Tabla 3. Comparación de microcomputadoras.

Características	NVIDIA Jetson Nano	Coral Dev Board	Raspberry Pi 4
Imagen			
Fabricante	NVIDIA	Google LLC	Raspberry
Procesador	Quad-core ARM Cortex-A57 (4nucleos)	Quad-core ARM Cortex-A53 (4 núcleos)	Broadcom BCM2711 Cortex-A72 (4 nucleos)
Frecuencia del proocesador	Hasta 1.43 Ghz	Hasta 1.5 GHz	Hata 1.5 GHz
GPU	NVIDIA Maxwell de 128 núcleos	PowerVR GE8300 GPU	VideoCore VI 500 MHz
Memoria Ram	4 GB LPDDR4	2 GB LPDDR3	4GB LPDDR4
Aplicaciones	Inteligencia Artificial, IoT, robotica.	Desarrollo de software, Inteligencia Artificial.	Educación, Inteligencia artificial, Desarrollo de software.
Conectividad	Gigabit Ethernet, Bluetooth, 4 puertos USB 3.0, 1 puerto USB 2.0.	Gigabit Ethernet Bluetooth, 1 x USB 3.0 | 1 x USB 2.0.	Gigabit Ethernet, 2 puertos USB 3.0, 2 puertos USB 2.0, Bluetooth 5.0.
Salida de video	HDMI	HDMI	2 x micro-HDMI
Sistema Operativo	Linux Os	Linux	GNU / Linux
Almacenamiento	Slot para tarjeta microSD	Micro SD slot 8GB eMMC	Slot para tarjeta microSD

c. Modelo de Deep Learning en Visión Artificial.

Para la detección de plagas, es necesario escoger un modelo de Deep Learning que se adapte a todos los requerimientos del sistema. Las características específicas que se necesitan son, buena precisión, velocidad para trabajar en tiempo real y también que sea un modelo liviano para trabajar en recursos de hardware limitados. En la Tabla 4 se realiza la comparación de tres modelos de Deep Learning que pueden adaptarse al sistema.

Tabla 4. Comparación de modelos de Deep Learning [32], [33], [34].

Criterio /Arquitectura	YOLO (You Only Look Once)	SSD (Single Shot Multibox Detector)	Faster R-CNN
Precisión	Tiene una precisión Alta, más en las versiones actuales.	Alta, pero depende del tamaño del modelo.	Muy alta.
Velocidad	Su velocidad es muy rápida.	Su velocidad es rápida, tiene un balance entre precisión y velocidad.	Mas lenta, ya que necesita realizar un paso de propuestas de región.
Uso de recursos	Moderado.	Moderado.	Alto.
Compatibilidad con la placa NVIDIA Jetson Nano	Si es compatible	Si es Compatible	Si es compatible
Aplicación en tiempo real.	Buena velocidad en detección a tiempo real.	Adecuado para trabajar en tiempo real.	No es bueno en aplicaciones de tiempo real por su alta demanda de procesamiento.

Las tecnologías para el control de plagas en la agricultura evolucionan rápidamente, ofreciendo mejores resultados en la producción agrícola, estas innovaciones tecnológicas no solo están optimizadas para la detección y el control de plagas, sino que también contribuyen mucho en la reducción de usos de pesticidas, generando un cambio significativo en el impacto ambiental y aumentando la productividad de los cultivos. Por lo cual en la Tabla 5, realiza la comparación de técnicas avanzadas en la identificación de plagas en cultivos, haciendo énfasis en el uso de redes neuronales, como clave para la identificación de dichas amenazas.

Tabla 5. Comparación de técnicas para la identificación de plagas en cultivos con redes neuronales [35] [36], [37], [38], [39], [40].

Autores	Titulo	Algoritmo	Exactitud	Datos
Thanh-Nghi Doan, Thien-Hue Phan.	A Novel Smart System with Jetson Nano for Remote Insect Monitoring	Utiliza YOLOv7	74.7%	1633 imágenes
Yanlei Xu, Qingyan Chen, Shuolin Kong, Lu Xing, Qi Wang, Xue Cong, Yang Zhou.	Real-time object detection method of melon leaf diseases under complex background in greenhouse	Utiliza un método de detección de objetos en tiempo real Pruned – YOLO v5s + Shuffle (PYSS)	96.7%	1913 imágenes

Autores	Titulo	Algoritmo	Exactitud	Datos
Rong Ye, QuanGao, YeQian, Jihong Sun, Tong Li	Improved YOLOv8andSAHIModelfortheCollaborative Detection of Small Targets at the Micro Scale: A Case Study of Pest Detection in Tea	YOLOv8 and SAHI.	96.32%	2864 imágenes
Yang Zhang, Chenglong Song, Dongwen Zhang.	Deep Learning-Based Objet Detection Improvement for Tomato Disease	Utiliza una arquitectura de red neuronal profunda Faster RCNN, específicamente una red residual profunda ResNet. También aplica el algoritmo de agrupamiento k-means.	97.01%	4178 imágenes
Lin Jiao, Shifeng Dong, Shengyu Zhang, Chengjun Xie, Hongqiang Wang.	AF-RCNN: An anchor-free convolutional neural network for multi-categories agricultural pest detection	Utiliza una red neuronal convolucional sin anclas AF-RCNN	56.4%	20000 imágenes
Linhui Wang, Wangpeng Shi, Yonghong Tang, Zhizhuang Liu, Xiongkui He, Hongyan Xiao, YuYang.	Transfer Learning-Based Lightweight SSD Model for Detection of Pests in Citrus	Propone un modelo modificado de MobileNetV3 +RPBM, que se basa en el detector de objetos de un solo disparo (SSD)	86.10%	10000 imágenes

d. Base de datos

En la Tabla 6 se muestra una comparación de 4 sistemas de gestión de datos, el cual se enfoca en varios parámetros, como uso de recursos, lenguaje base, flexibilidad, escalabilidad entre otros, a través de la comparación de los distintos parámetros, se pueden tomar decisiones sobre la base de datos a utilizar, basándose en las necesidades específicas que el sistema necesita.

Tabla 6. Comparación de bases de datos [41], [42], [43], [44].

Parámetro	MySQL	MongoDB	SQLite	PostgreSQL
Tipo de Sistema	Relacional	NoSQL	Embebido	Relacional (AQL/ORDBMS)
Lenguaje Base	C, C++	C++	C	C
Consulta de datos	SQL	MQL (Mongo Query Lenguage)	SQL	SQL/PLpgSQL
Flexibilidad	Media require configuración	Esquema dinámico y flexible	Alta, no require de servidor.	Alta, admite datos estructurados y semi-estructurados.
Escalabilidad	Soporta grandes cantidades de datos	Exelente para Big Data y NoSQL.	Bueno para proyectos pequeños.	Buena escalabilidad y eficiente.
Uso de recursos	Moderado	Alto consumo	Bajo consumo	Balanceado.
Casos de Uso	Aplicaciones web y transaccionales	Big Data, IoT, Aplicaciones NoSQL.	Aplicaciones móviles.	Analitica empresarial y OLTP/OLAP.

e. Software para la interfaz

El software utilizado para el desarrollo de la interfaz es muy importante, ya que representa el contacto directo con el usuario para la visualización y gestión de toda la información del sistema. Por tal motivo en la Tabla 7 se realiza la comparación de diferentes softwares utilizados para el desarrollo de interfaces, evaluando características como compatibilidad, escalabilidad entre otros.

Tabla 7. Comparación de software para la interfaz [45], [46], [47]

Características	Flet	Flask	Qt (PyQt)
Facilidad de Uso	Muy fácil, sin necesidad de tener muchos conocimientos de HTML/CSS/JS	Requiere conocimientos de HTML/CSS/JS	Requiere concimiento de diseño de UI.
Interactividad en tiempo real	Usa WebSockets para actualizaciones en tiempo real	Requiere soluciones adicionales para tiempo real	Excelente para aplicaciones interactivas.
Escalabilidad	Ideal para prototipos y proyectos pequeños	Escalable para proyectos grandes con mayor control	Alta, ideal para aplicaciones robustas.
Configuración	Todo se realiza en Python.	Requiere la configuración de servidor y de plantilla.	Se realiza en Python.
Almacenamiento de datos	Compatible con SQlite y PostgresSQL.	Compatible con SQlite, MySQL y PostgresSQL.	Compatible con Mysql, MariaDB, PostgreSQL, SQlite.

1.4 Objetivos

1.4.1 Objetivo general

Implementar un sistema electrónico para la identificación temprana de presencia de plagas en cultivos de ambiente controlado empleando visión artificial y Deep Learning.

1.4.2 Objetivos específicos

- Determinar características que permitan la identificación de plagas en cultivos controlados en Susudel.
- Establecer la arquitectura del sistema de identificación de plagas en cultivos controlados empleando visión artificial y Deep Learning.
- Seleccionar herramientas de hardware y software y modelos de Deep Learning para el sistema de identificación de plagas en cultivos controlados.
- Diseñar el prototipo y entrenar el modelo de Deep Learning para la identificación de plagas en cultivos controlados.
- Desarrollar un sistema de gestión de información para la visualización de los resultados del proceso de identificación de plagas.
- Aportar con los resultados obtenidos al proyecto de investigación “Enhancing the Agrifood Supply chain towards Industry 5.0 (AGRO5) in the Ecuadorian Andes”.

CAPÍTULO II. METODOLOGÍA

2.1 Materiales

En el proyecto de investigación se ocuparon varios dispositivos, que se adapten a los requerimientos del sistema. Se desarrollo un sistema para la identificación temprana de plagas en cultivos de ambiente controlado con visión Artificial y Deep Learning. La arquitectura que se utilizó fue YOLO (You Only Look Once) en específico la versión 8, para la detección de imágenes, de igual manera se utilizó labelimg para el etiquetado de cada imagen del dataset.

En el entrenamiento del modelo se utilizó un dataset de 4803 imágenes, que se dividen en tres tipos de plagas. Para el proceso de entrenamiento se utilizó la plataforma de Google Colab, que ofrece buenos recursos de hardware y el lenguaje de programación Python que son útiles para agilizar el entrenamiento del modelo. Para el procesamiento de los datos del sistema se utilizó el microcomputador NVIDIA Jetson Nano, tres cámaras IMX219 y la cámara de un celular Redmi note 8.

Para la visualización de los resultados se ocupó el framework de desarrollo web Flask, y para el almacenamiento de los datos obtenidos se ocupó la base de datos MySQL. En la Tabla 8 se muestra los materiales utilizados para el desarrollo y funcionamiento del sistema.

Tabla 8. Materiales Utilizados.

Descripción de necesidades	Materiales
Modelo de Deep Learning.	YOLOv8n.
Adquisicion de datos.	Cámara IMX219.
Etiquetado de imagenes.	Labelimg.
Microcooomputador para el procesamiento.	NVIDIA Jetson Nano.
Platafoorma para el entrenamiento.	Google Colab.
Plataforma para gestión de la información.	Framework Flask para desarrollo web.
Base de datos para el almacenamiento de los resultados.	MySQL
Lenguaje de programación.	Python.

2.2 Métodos

2.2.1 Modalidad de la investigación

A continuación de describe la modalidad de investigación aplicada para el desarrollo del proyecto:

El proyecto de investigación se contempla como una investigación aplicada, debido a que se desarrolló un sistema para el monitoreo de plagas en cultivos de ambiente controlado, aplicando los conocimientos existentes.

Se desarrolló una investigación bibliográfica, mediante la revisión de artículos, tesis, libros, bases de datos y revistas indexadas de distintas universidades, con la finalidad de sustentar y dirigir adecuadamente el desarrollo del proyecto de investigación.

A parte de que el proyecto de investigación sea de campo, también se realizó una investigación de tipo experimental, en el que se enfocó en la recolección de datos para verificar la precisión y validación del sistema de detección de plagas, contribuyendo en una mejora continua para los cultivos y agricultores mediante las tecnologías aplicadas.

2.2.2 Recolección de información

Para el proyecto de investigación, la obtención de información, se realizó a través de repositorios de universidades, libros, revistas, bases de datos entre otros, con el fin de que aporten en la ejecución y desarrollo del proyecto. Po ende, se llevó a cabo una serie de preguntas para el plan de recolección de información el cual se detalla a continuación en la Tabla 9.

Tabla 9. Criterios de recolección de información.

Criterio	Descripción
¿Para qué?	Para identificar las plagas presentes en las plantas.
¿De qué personas u objetos?	De un invernadero de la parroquia de Susudel en la ciudad de Cuenca.
¿Cómo?	Mediante un sistema de monitoreo con visión artificial y Deep Learning.
¿Con qué?	Con cámaras y el entrenamiento de redes neuronales para identificar las plagas.
¿Cuándo?	Durante el periodo octubre 2024 a enero 2025
¿Dónde?	En un invernadero ubicado en la parroquia de Susudel en la ciudad de Cuenca.

2.2.3 Procesamiento y análisis de datos

Para el procesamiento y análisis de datos de la investigación se tomaron en cuenta varios aspectos importantes, estos aspectos son:

- Recolección detallada de los datos necesarios.
- Definición de un modelo específico para su entrenamiento.
- Preprocesamiento de los datos recolectados para garantizar su calidad y relevancia.
- Entrenamiento y pruebas del sistema utilizando los datos preprocesados.
- Análisis de los datos recopilados en las pruebas de funcionamiento.

CAPÍTULO III. RESULTADOS Y DISCUSIÓN

3.1 Desarrollo de la propuesta

La ejecución del proyecto sigue una modalidad investigativa, bibliográfica y de campo, enfocada en la detección temprana de plagas en cultivos de ambiente controlado en Susudel. Esta parroquia rural, ubicada en la región Sierra del Ecuador, se encuentra en el cantón Oña, provincia de Azuay.

El sistema se implementó en Python, gracias a que este lenguaje de programación es muy aplicado en proyectos de Inteligencia Artificial. El modelo seleccionado para el sistema esYOLOv8. Para el entrenamiento del modelo se utilizó la plataforma de Google Colab, el cual ofrece buenos recursos para el entrenamiento del modelo. El etiquetado de las imágenes se realizó con Labelimg, y después esas imágenes etiquetadas fueron subidas a la plataforma Roboflow para corregir errores del etiquetado, y generar el Dataset con los datos de entrenamiento, validación y test.

3.2 Cultivos de plantas medicinales en ambiente controlado Susudel.

En el invernadero de Susudel, se cultivan diversas plantas medicinales, que se destacan por ser buenas para el consumo humano por sus propiedades curativas. A continuación, se describe cada una de las plantas.

a. Clavel (Dianthus caryophyllus)

Es una planta herbácea y perenne, que puede alcanzar un tamaño de 80 cm, sus flores en cultivos al aire libre llegan a medir hasta 5cm, pero si se cultivan en invernaderos llegan a medir hasta 10cm. Las flores suelen ocuparse para la medicina tradicional, ya que tienen propiedades antialérgicas, analgésicas, antiespasmódicas, calmante del sistema nervioso entre otros beneficios[48]. En la Figura 17 se muestra la planta del clavel.



Figura 17. Clavel (Dianthus caryophyllus)

b. Hierba Luisa (Cymbopogon citratus)

Esta planta es muy utilizada en la industria alimentaria como en la industria no alimentaria. En la industria alimentaria, se utiliza para bebidas, energizantes, hidratantes y aditivos alimentarios, como aromatizantes y conservantes. Por otro lado, en la industria no alimentaria por todos sus beneficios se utiliza en la cosmética, perfumería, desodorantes, shampoo, y en la industria agrícola para realizar Insecticidas, Fungicidas y Herbicidas [49]. En la Figura 18 se muestra una planta de Hierba Luisa.



Figura 18. Hierba Luisa (Cymbopogon citratus)

c. Manzanilla

Es una de las plantas medicinales más antiguas, la manzanilla tienes dos variedades, la manzanilla alemana (Chamomilla recutita) y la manzanilla romana (Chamaemelum nobile). Esta planta tiene propiedades medicinales que son beneficiosas para la salud humana, ya que ayuda a tratar inflamaciones, fiebre, insomnio, ulceras, desordenes gastrointestinales entre otros [50]. La Figura 19 muestra la planta de Manzanilla.



Figura 19. Manzanilla (Matricaria chamomilla)

d. Menta (Mentha spicata)

Planta de color verde brillante, sus hojas son de forma puntiaguda y creen hasta 10 cm, pero si se cultivan en condiciones ideales alcanzaran una altura y anchura de 31 – 61 cm. Se utiliza mucho en la medicina, en la cosmética y en la cocina, todo esto gracias a sus propiedades [51]. En Figura 20 se muestra una planta de Menta.



Figura 20. Menta (Mentha spicata)

e. Cedrón (Aloysia citrodora)

Es un arbusto leñoso con hojas alargadas de color verde brillante, su característica importante es que tiene un aroma intenso con notas cítricas parecidas a la del limón. Tiene propiedades antiinflamatorias, antisépticas y sedantes, por lo cual es muy utilizada para remedios caseros por las personas. También se utiliza en la industria de la cosmética como en la perfumería, por su aroma intenso [52]. En la Figura 21 se presenta una planta de Cedrón.



Figura 21. Cedrón (Aloysia citrodora)

3.2.1 Plagas de las plantas medicinales

Identificar que tipos de plagas afectan a cada planta es crucial para la identificación de plagas. En Tabla 10 se muestra una lista de las principales plagas que afectan a cada planta, lo cual ayuda a tener una visión general de cuáles son las plagas que más se repiten y realizar un buen sistema de detección de plagas.

Tabla 10. Principales plagas de las plantas medicinales [53], [54], [55], [56].

Planta	Plagas
Clavel	▪ Minador ▪ Pulgón ▪ Araña Roja ▪ Mosca blanca ▪ Trips
Hierba Luisa	▪ Plugón.
Manzanilla	▪ Pulgón. ▪ Trips. ▪ Ácaros. ▪ Babosas. ▪ Mosca blanca.
Menta	▪ Mosca blanca. ▪ Pulgón. ▪ Orugas.
Cedron	▪ Pulgón. ▪ Gusano. ▪ Queresa Marron. ▪ Cigarrita,

3.2.2 Características visuales de las plagas a detectar

Las plagas presentan características visuales distintivas lo cual permite su identificación y posteriormente su control. Entre las características tenemos, colores específicos, comportamientos, patrones y formas que hacen que se puedan identificar. Gracias a estas características la identificación de las plagas no es muy difícil y se podrá controlar a tiempo.

En la Tabla 11 se presentan distintas características de las plagas lo cual facilitara a la identificación de plagas por medio de visión artificial, cada una de las plagas cuentan con características que las diferencian y también cada plaga causa distintos daños a las hojas de las plantas, todas estas características nos ayudaran a poder identificar qué tipo de plaga es.

Tabla 11. Características visuales de las plagas [14]

Plaga	Imagen	Color	Tamaño	Forma
Mosca Blanca		Blanco	Pequeños (1 – 2 mm)	Ovalados
Araña Roja		Rojo, Amarillo, Blanco y hojas amarillentas	Pequeños	Ovalados
Áfidos		Verde, amarillo, negro, rosado.	Pequeños (1-3mm)	Ovalados, periformes

3.3 Arquitectura del Sistema

En la Figura 22 se muestra la arquitectura del sistema de detección de plagas, este cuenta con 4 etapas las cuales se dividen en Adquisición de datos, Entrenamiento, Procesamiento y Visualización.

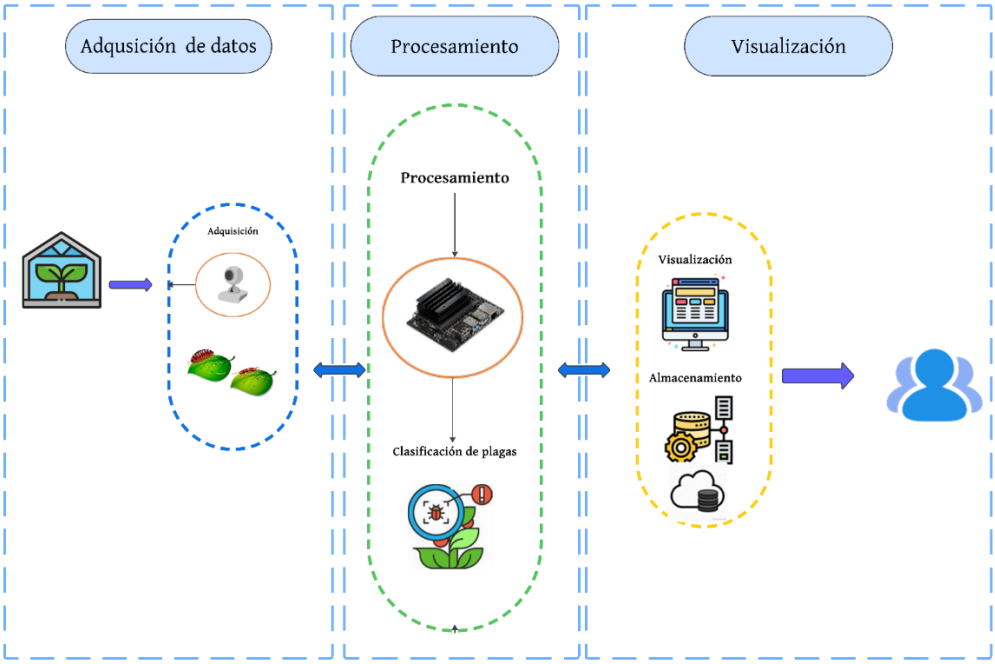


Figura 22. Arquitectura del sistema.

a. Etapa 1: Adquisición de datos.

Esta etapa es la parte inicial del sistema, en donde se realiza la adquisición de imágenes o video en tiempo real de las plantas, para la adquisición de datos, se realiza con tres cámaras IMX219 de alta resolución, las cuales se colocarán estratégicamente en puntos fijos para tener una buena visión del cultivo, en donde se detectará si la planta cuenta con alguna de las distintas clases de plagas. También se conecta una cámara de celular, el celular no va a estar fijo todo el tiempo, esta cámara está pensada solo para cuando se quiera analizar de manera más detallada las hojas de las plantas, por ende, hay una estación fija en donde colocará el celular y se conecta al servidor por medio de wifi y podrá realizar la detección de las plagas.

b. Etapa 2: Procesamiento de los datos.

Una vez finalizado el entrenamiento como se muestra en el esquema de la Figura 31, este se despliega en la Jetson Nano, donde se encarga de procesar en tiempo real las imágenes capturadas por las cámaras. Gracias a su capacidad de cómputo optimizada, la Jetson Nano ejecuta el modelo entrenado para detectar plagas, aplicando preprocesamiento a las imágenes, realizando la clasificación y almacenando los resultados para su visualización y análisis.

c. Etapa 3: Visualización y almacenamiento de los datos.

Como última etapa esta la visualización en donde se creará una interfaz gráfica en donde se mostrará la gestión de la información, detección de plagas en tiempo real y se podrá programar horarios para capturar imágenes y procesarlas para detectar plagas. Además, dicha información obtenida se podrá almacenar en una base de datos, el cual servida para un análisis posterior de los datos para los agricultores.

3.4 Tecnologías de Hardware y Software Utilizados

En el apartado 1.3.11 se realiza un análisis comparativo de las características de diversas opciones de hardware y software que se van a utilizar en el sistema. Después del análisis, se detalla a continuación el hardware y software seleccionado para la implementación del sistema.

3.4.1 Selección de Cámaras

Una vez analizado todas las características de cada cámara en la Tabla 2, se decidió utilizar la cámara IMX219. Se eligió esta cámara debido a su buena resolución que es de 8MP, lo que es muy importante para la detección precisa de plagas. Además, cuenta con un buen ángulo de visón, lo que ayudara mucho para cubrir un área amplia y realizar la detección, en la Figura 23, se muestra la cámara descrita.



Figura 23. Cámara IMX219.

3.4.2 Selección de microcomputador

Una vez analizado los tres microcomputadores en la Tabla 3, se optó por la placa NVIDIA Jetson Nano debido a su eficiencia en proyectos de Visión artificial e Inteligencia Artificial en el Anexo A se muestra las especificaciones técnicas. Las características que más destacadas que se tomó en consideración es su procesador Quad-core ARM Cortex-A57 (4nucleos) y su GPU NVIDIA MaxwelL de 128 núcleos, lo que proporciona un rendimiento óptimo para el sistema de detección de plagas. Además de la versatilidad de puertos con la que cuenta la hace muy factible para proyectos de visión artificial, como se muestra en la Figura 24



Figura 24. Microcomputador NVIDIA Jetson Nano.

3.4.3 Selección de la arquitectura de Deep Learning.

Después de realizar la comparación de los diferentes modelos en la Tabla 4 y ver los resultados obtenidos de los diferentes modelos en la identificación de plagas en cultivos de la Tabla 5, Se llega a la conclusión de que el modelo de YOLO, es el más adecuado para el sistema, gracias a su buena presión, su buena adaptación para trabajar con hardware de recursos limitados, para detecciones en tiempo real.

Las versiones de YOLO que más se adaptan a los requerimientos del sistema son YOLOv5 y YOLOv8. A continuación en la Tabla 12 se realiza una comparación entre las versiones de YOLO v5 y v8.

Tabla 12. Comparación de la v5 y v8 de YOLO [57], [58].

Características	YOLOv5	YOLOv8
Velocidad CPU ONNX (ms)	73.6	80.4
mAPval 50-95	34.3	37.3
Frameworks	PyTorch	PyTorch
Arquitectura	Es basada en CSPDarknet	Es basada en CSPDarknet53
Compatibilidad	Con CPU y GPU.	Con GPUs modernas y Edge devices.
Facilidad de uso	Es fácil de entrenar e implementar.	Mas fácil y con su sintaxis más simplificada.
Aplicaciones	Detecciones en tiempo real.	Detecciones en tiempo real con mayor precisión.

Después de analizar en la Tabla 12 las características de las versiones v5 y v8 de YOLO, se escogió la versión YOLOv8 debido a su buena precisión, su adaptabilidad para trabajar en tiempo real con dispositivos como la Jeton Nano y también por su facilidad de uso.

- ***Arquitectura de YOLOv8***

Dado que se usó el modelo de red neuronal YOLOv8, en la Figura 25 se expone una imagen de la arquitectura del modelo con sus tres secciones importantes, Backbone, Neck y Head. El backbone es donde se realiza la extracción de características de la imagen de entrada, esas imágenes son procesadas por varias capas convolucionales y capas de reducción de tamaño, lo que genera varios mapas de características en distintas resoluciones. El neck es el bloque en donde se combina todas las características extraídas a distintas escalas del backbone para obtener una mejor detección en objetos de distintos tamaños. Por ultimo tenemos el bloque Head en donde se realiza la clasificación y detección de los objetos utilizando los mapas de características generados por el bloque Neck, cada uno de los bloques que se encuentran dentro de Head funcionan para la detención de distintos tamaños de objetos, el Detect que proviene de C2f del bloque 15 es para la detección de objetos pequeños, el Detect que proviene de C2f del bloque 18 es para la detección de objetos medianos y el Detect que proviene de C2f del bloque 21 es para la detección de objetos grandes [59].

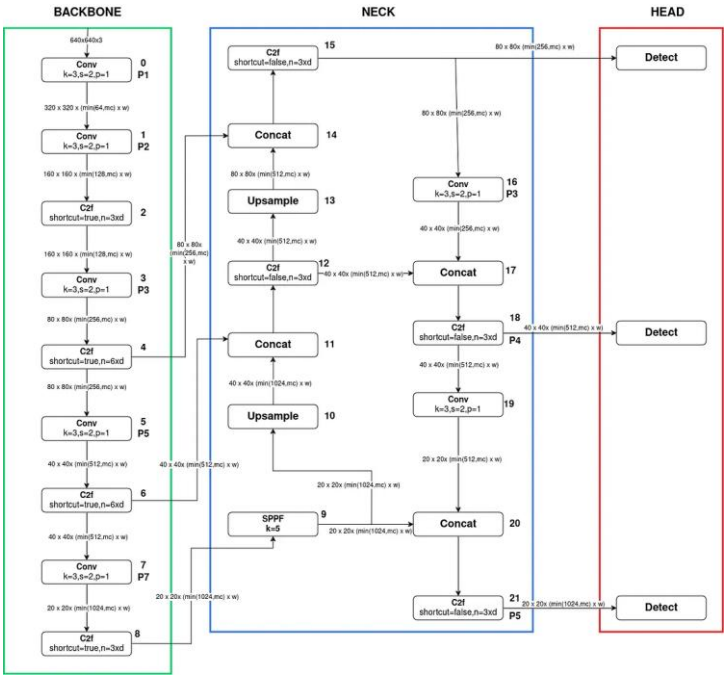


Figura 25. Arquitectura del modelo YOLOv8.

3.4.4 Selección de la base de datos

Una vez realizado el análisis de las características de las diferentes bases de datos en la Tabla 6, se escogió la base de datos MySQL debido a que soporta grandes cantidades de datos, es ampliamente utilizado para aplicaciones web, su consumo de recursos es adecuado para aplicarlo en hardware limitado y gracias a que utiliza el lenguaje SQL permite realizar consultas eficientes e integración con distintas aplicaciones.

3.4.5 Selección de la plataforma para el desarrollo de la interfaz.

Después de analizar las plataformas para realizar la interfaz gráfica en la Tabla 7, se escogió el framework Flask basado en Python, gracias a su escalabilidad para realizar proyectos con requerimientos más grades, compatibilidad con distintas bases de datos como, MySQL, SQLite y PostgreSQL, facilitando la integración de proyectos con cantidades de datos pequeños, moderados y grandes. También gracias a su flexibilidad para desarrollar interfaces que sean funcionales y personalizadas para el usuario.

3.5 Descripción del sistema

El sistema se compone de un algoritmo entrenado para la detección temprana de plagas en ambiente controlado, en el cual con la adquisición de video e imagen con las tres cámaras IMX219 y la cámara de un celular, se propone una detección de plagas en los cultivos. A partir de aquí procedemos a describir el proceso de implementación del sistema.

En la Figura 26, se plantea el diagrama funcional para la detección de plagas en cultivos de ambiente controlado, el sistema consta del equipo de procesamiento NVIDIA Jetson Nano que es la parte principal y más importante del sistema, ya que se encarga de procesar las imágenes capturadas y visualizar las detecciones. La adquisición de las imágenes se realiza con tres cámaras IMX219 que se conectan por medio de USB al equipo y la cámara del celular se conecta por medio de wifi, se descarga la aplicación IP Webcam para que el celular funcione como una cámara IP, el router domestico me asigna una IP al celular, la cual debemos especificar en el servidor y así poder obtener el video. Hay que tener en cuenta que tanto el celular como el servidor deben estar conectados a la misma red.

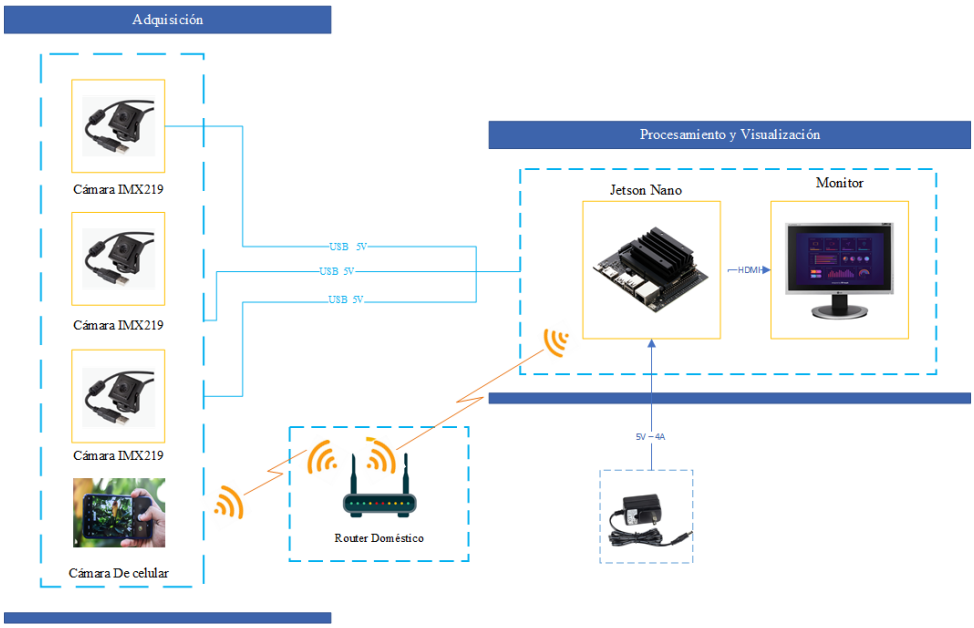


Figura 26. Diagrama Funcional del sistema.

3.6 Entrenamiento de la Red Neuronal para la Detección de plagas

Después de comparar varios modelos de Deep Learning en la Tabla 4 y verificar que el modelo YOLO (You Only Look Once) es muy bueno para trabajar en tiempo real y tienen buena precisión en la detección, se entrenó el modelo con el dataset generado. A continuación, se muestra el procedimiento de entrenamiento.

3.6.1 Dataset para el Entrenamiento.

Para construir un buen dataset, este debe tener una estructura específica para que el entrenamiento de la red neuronal sea capaz de detectar las plagas en los cultivos. Este dataset debe tener una gran variedad de imágenes de las plagas. En Figura 27, se muestra la estructura que debe tener el dataset para el entrenamiento del modelo Yolov8.

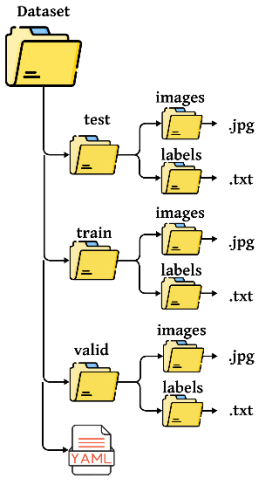


Figura 27. Estructura del dataset para Yolo.

Para que el entrenamiento de la red neuronal sea óptimo para el sistema, el dataset está compuesto de la siguiente manera. Como carpeta general tenemos a Dataset, dentro de esa carpeta contiene tres sub carpetas, test, train y valid y un archivo data.yaml. Cada carpeta tiene dos subcarpetas llamadas images y labels. En la carpeta images se almacena todas las imágenes que contienen los parámetros de detección, la carpeta labels almacena las coordenadas de las cajas delimitadoras en archivos .txt de cada imagen que fueron etiquetadas anteriormente con la herramienta labelimg. La carpeta train es una de las más importantes, ya que esta contiene las imágenes y labels para

realizar el entrenamiento, por eso esta carpeta debe contar con una variedad de imágenes de las distintas clases de plagas para obtener un buen resultado en la presión de detección.

En la carpeta valid se almacenan imágenes que no forman parte del conjunto de datos de train, esta carpeta se usa para evaluar el desempeño del modelo mientras se realiza el entrenamiento y ajustar hiperparámetros.

Por último, la carpeta test sirve para evaluar el modelo una vez que se haya finalizado su entrenamiento, de igual manera esta carpeta contiene imágenes que no fueron vistas durante el entrenamiento.

a. Características del dataset

Para la detección temprana de plagas en cultivos de ambiente controlado, se utilizó varios dataset que cumplen con los requerimientos para el sistema. En la Tabla 13 se muestra los datasets que fueron seleccionados.

Tabla 13. Conjunto de datos utilizados [60], [61], [62], [63]

Tema	Cantidad de imágenes	Clases	Institución
Tetranychus urticae, Aphididae and Healty leaf Dataset for Crop Pest Recognition (4185 images)	4185	▪ Tetranychus urticae ▪ Aphididae ▪ Healty leaf	Tecnológico Nacional de Mexico
LeLePhid: An Images Dataset for Aphids Detection and Infestation Severity on Lemons Leaf	665	▪ Aphids ▪ Healthy	Universidad Técnica de Manabi
A database of eight common tomato pest images	4263	▪ Tetranychus urticae ▪ Bemisia argentifolii ▪ Zeugodacus cucurbitae ▪ Thrips palmi ▪ Myzus persicae ▪ Spodoptera litura ▪ Spodoptera exigua	National Bureau of Agricultural Insect
Cassava Whitefly Dataset	3000	▪ Whitefly	Makerere University College of Computing and Information Sciences

Para la elaboración del dataset se utilizaron varios datasets que contienen las tres plagas que vamos a detectar, que son aphid, en español llamado pulgón, spider mite, en español llamado araña roja y whitefly, en español llamado mosca blanca. De cada dataset se realizó una selección de las mejores imágenes, obteniendo un dataset de 2795 imágenes para el entrenamiento de nuestro modelo. En Tabla 14 se muestra las características del dataset final para la detección de plagas.

Tabla 14. Características del dataset para la detección.

Clase	**Cantidad de imágenes**	**Categoría de la clase**	**Cantidad de imagenes por categoría**
Aphid	892	Baja abundancia	320
		Moderado	290
		Superabundancia	282
Spider mite (Tetranychus urticae)	907	Baja abundancia	320
		Moderado	302
		Superabundancia	285
Whitfly (Bemisia tabaco)	996	Baja abundancia	332
		Moderado	332
		Superabundancia	332

b. Etiquetado de las imágenes

Para el etiquetado de las imágenes se utilizó la herramienta Labelimg, cada imagen fue etiquetada con la región de interés correspondiente a cada clase de plaga, lo cual es muy útil para el entrenamiento de la red neuronal. Se bebe tener en cuenta de que el etiquetado de cada imagen debe ser precisa a la plaga que se quiere detectar, para que al momento de entrenar la red neuronal no tenga confusiones con etiquetas erróneas, con eso el modelo entrenado tendrá buenos resultados y al momento de utilizarlo en la detección de plagas no tendremos predicciones incorrectas de plagas o confusión entre las distintas plagas.

El etiquetado de cada imagen se la realizo en un formato .xml, en el cual contiene las coordenadas de las cajas delimitadoras o también llamados bounding box como se muestra en la Figura 28.

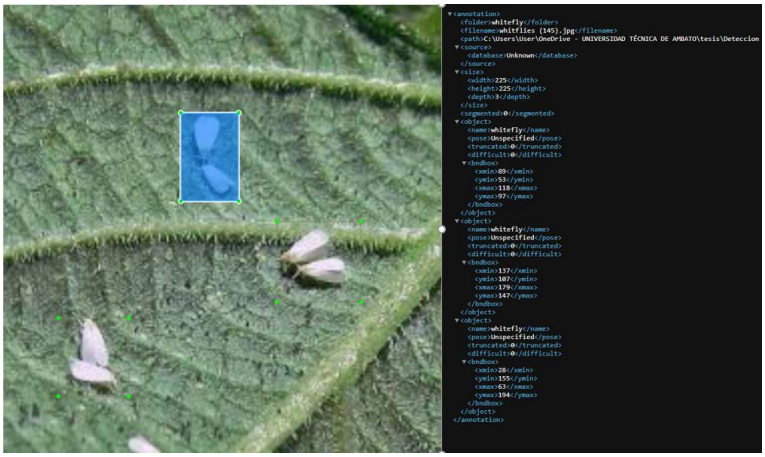


Figura 28. Etiquetado de las imágenes.

Ya que para el entrenamiento del modelo Yolo se necesita de la estructura de la Figura 27 con las imágenes en una carpeta y las anotaciones en formato .txt en otra carpeta, se procedió a subir todas las imágenes y etiquetas en formato .xml a la plataforma de roboflow, esta plataforma facilita el manejo de datasets para visión por computadora, puede convertir los datos subidos en formatos compatibles para distintos frameworks, como Yolo, tensorflow, PyTorch entre otros, y lo más importante es que se puede realizar un aumento de datos, por lo que se aplicó varios filtros y se obtuvo un dataset total de 4803 imágenes, en la Figura 29 se muestra los filtros aplicados y la conversión a formato YOLO.

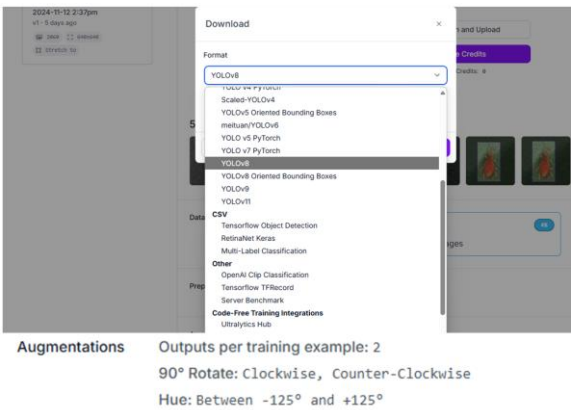


Figura 29.Aumento de datos y Conversión de los datos a formato de Yolo.

Una vez descargado en el formato de Yolo nos proporciona una carpeta con la estructura de la Figura 27, con todas sus imágenes, labels y el archivo data.yaml como se muestra en la Figura 30.

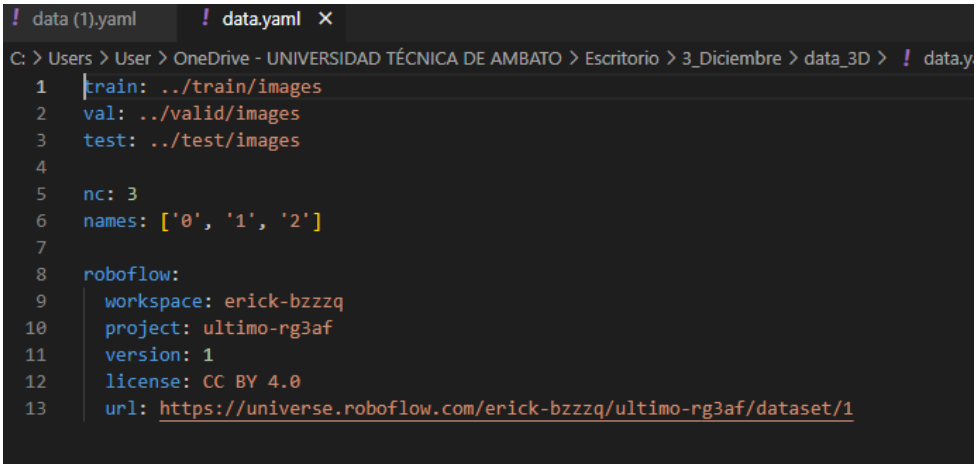


Figura 30. Archivo data.yaml generado por roboflow.

3.6.2 Proceso de entrenamiento

Se realizó el entrenamiento y validación del modelo de detección de plagas en cultivos, aplicando múltiples configuraciones de parámetros como “batch size” y “epochs”. El modelo fue evaluado, utilizando métricas de validación. En la Figura 31 se muestra el esquema general del entrenamiento del modelo.

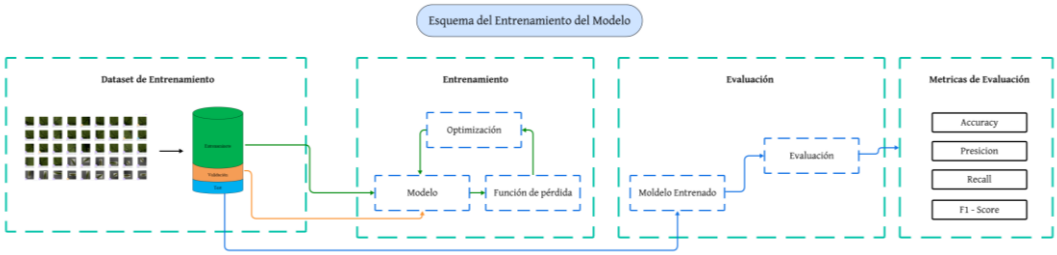


Figura 31. Esquema del Entrenamiento del Modelo.

Para el entrenamiento se utilizó la plataforma de desarrollo de Google colab el cual proporciona un ambiente de ejecución de un notebook de Jupyter en la nube. Este ambiente proporciona una CPU de 12.7 GB, una GPU de 15 GB específicamente NVIDIA T4 Tensor Core y un disco de 112.6 GB, en su forma gratuita.

Para empezar con el entrenamiento de la red neuronal primero debemos subir el dataset en formato .zip a un drive, para posteriormente descomprimirlo en el entorno de ejecución de Google colab, para montar el drive y descomprimir el dataset en el entorno de Google colab, ocupamos lo siguientes comandos que se muestran en la Figura 32.

```
from google.colab import drive
drive.mount('/content/gdrive')
!unzip -q /content/gdrive/MyDrive/Yolo/insect_detection.v2i.yolov8.zip -d /content/gdrive/MyDrive/Yolo/dataTraing
```

Figura 32. Comandos para montar el drive en Colab y descomprimir el dataset.

Después de montar el drive y descomprimir la carpeta del dataset en el entorno de Google colab, procedemos a instalar la librería de ultralytics con el comando **!pip install ultralytics,** como se muestra en la Figura 33**.**

```
Downloading ultralytics-8.3.31-py3-none-any.whl (886 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 886.3/886.3 kB 26.8 MB/s eta 0:00:00
Downloading ultralytics_thop-2.0.11-py3-none-any.whl (26 kB)
Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.31 ultralytics-thop-2.0.11
```

Figura 33. Instalación de la librería ultralytics.

Procedemos a configurar las rutas del dataset en el archivo data.yaml para posteriormente realizar el entrenamiento de la red neuronal. En la Figura 34 se muetra la configuración de las rutas del archivo data.yaml

```
: > Users > User > Downloads > ! data (2).yaml
1   train: /content/gdrive/MyDrive/tesis/train/images
2   val: /content/gdrive/MyDrive/tesis/valid/images
3   test: /content/gdrive/MyDrive/tesis/test/images
4
5   nc: 3
6   names: ['pulgon', 'arana_roja', 'mosca_blanca']
7
```

Figura 34. Configuración de rutas del archivo data.yaml.

A continuación, descargamos y colocamos en el entorno de Google colab los pesos preentrenados del modelo YOLOv8 desde el repositorio de ultralytics con el siguiente comando:

!wget https://github com/ultralytics/assets/releases/download/v8.1.0/yolov8n pt

Por último, importamos la librería Yolo, cargamos el modelo preentrenado, y procedemos al entrenamiento del modelo como se muestra en la Figura 35.



Figura 35. Comandos para el entrenamiento del modelo YOLO.

Los hiperparámetros establecidos para el entrenamiento fueron de 80 épocas, un tamaño del lote o también llamado batch de 64, un tamaño de pixeles de entrada de 640 y por último se especifica en donde ejecutar el entrenamiento, en la GPU por lo que se pone device=cuda.

En el proceso de entrenamiento se puede observar el consumo de la GPU en cada época, que varía un poco entre 8.3G y 8.7eG como se observa en la Figura 36.

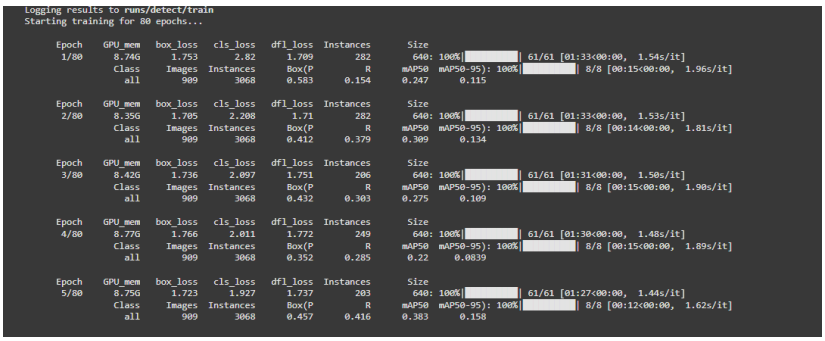


Figura 36. Ejecución del entrenamiento.

Una vez que se completen todas las épocas y el proceso de entrenamiento finalice, los resultados de entrenamiento como graficas de perdidas, de precisión y los pesos del modelo recién entrenado best.pt y last.pt se guardan en el entorno de Google colab en una carpeta llamada runs, en la Figura 37 se muestra la carpeta runs con todos sus archivos generados después del entrenamiento.

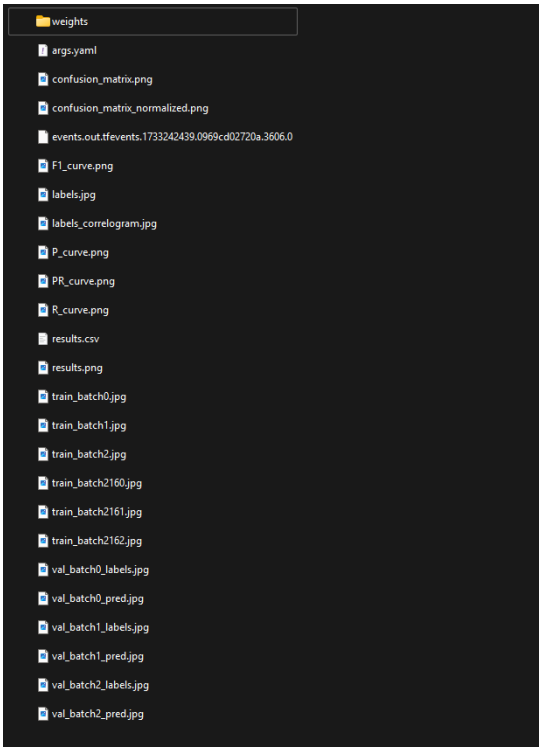


Figura 37. Carpera runs resultante del entrenamiento.

En la Tabla 15 se presenta los resultados de entrenamiento con distintas épocas para el modelo YOLOv8n.

Tabla 15. Resultados de entrenamiento con distintas épocas

Modelo	**Épocas**	**Métricas de validación**				**Tiempo (horas)**
		Presición	**Recall**	**mAP50**	**mAP50-95**	
YOLOv8n	30	0.90	0.86	0.66	0.33	1.30
	60	0.92	0.88	0.70	0.40	1.55
	80	**0.95**	**0.9**	**0.77**	**0.46**	**2.30**

Después de revisar los resultados de entrenamiento con distintas épocas para el modelo YOLOv8n se eligió el entrenamiento con 80 épocas, siendo el mejor entrenamiento con mejores resultados respecto a los otros entrenamientos.

- ***Capas de la arquitectura YOLOv8***

Las redes neuronales están estructuradas con una variedad de capas con las que optimizan de manera muy eficiente la detección de los objetos. En la Figura 38, se muestra las entradas, capas del modelo y las salidas que fueron utilizadas para el sistema de detección de plagas.

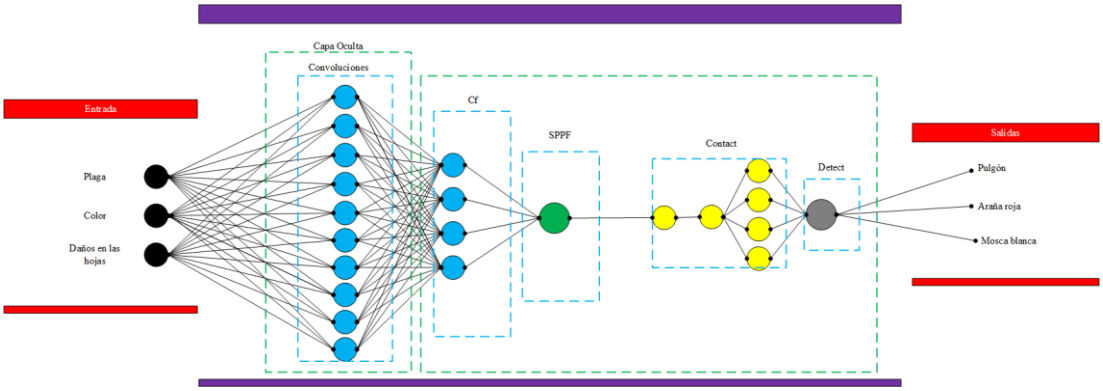


Figura 38. Capas del modelo para el sistema.

- ***Convolución (Conv).*** Son las capas que se encargan de extraer patrones básicos.
- ***C2f (Cross Stage Partial Networks).*** Realiza una mejora en la extracción de las características gracias a sus conexiones residuales.
- ***SPPF (Spatial Pyramid Pooling Fast).*** Realiza la extracción de las características de globales mediante el submuestreo, mejorando en la precisión para la detección de objetos en diferentes tamaños.
- ***Contact.*** realiza una combinación de varias características de distintos niveles de profundidad, logrando enriquecer las representaciones multiescala.
- ***Detect.*** aquí se realiza la clasificación y detección de los objetos, para eso utiliza los mapas de características que fueron procesados por el Neck.

3.7 Implementación del sistema

Para la implementación del sistema de detección de plagas, se realizó una serie de pasos estructurados, que permitieron el desarrollo y funcionalidad del sistema. A continuación, se describen los pasos realizados.

3.7.1 Librerías instaladas

En la Tabla 16 podemos observar las librerías instaladas y utilizadas en la implementación del sistema. Cada una de las librerías cumplen una función importante ya que ayudaron a que el sistema funcione de forma adecuada.

Tabla 16. Instalación de librerías para la implementación del sistema.

Librería	Función
Flask	Framework web en Python para desarrollo de interfaces web.
render_template	Función de flask que permite renderizar archivos HTML.
Response y jsonify	Son funciones que ayudan a manejar respuestas HTTP y para la conversión de datos en formato JSON.
request	Permite acceder a datos enviados a través de las solicitudes HTTP. Permite también realizar solicitudes HTTP para enviar mensajes a la API de Telegram.
Cv2 (OpenCV)	Procesamiento de captura de imágenes y videos.
mysql.connector	Ayuda a realizar consultas y manipular datos de la base de datos mysql.
threading	Permite ejecutar varias tareas en hilos separados.
ultralytics (YOLO)	Biblioteca para la implementación del modelo de Yolov8.
schedule	Biblioteca para realizar tareas en tiempos determinados, por ejemplo, la captura de imágenes en varios horarios.
time	Maneja las funciones que están relacionadas con el tiempo.

3.7.2 Almacenamiento de los datos del sistema

El sistema para la detección de plagas incorpora una interfaz en la web para la gestión de información desarrollada con el framework Flask, mientras que los datos obtenidos se almacenan en la base de datos de MySQL, permitiendo un manejo adecuado y eficiente de la información generada.

El almacenamiento de datos se realiza con MySQL, en el cual se creó una base de datos llamada detecciones, dentro de esa base de datos contiene 2 tablas para guardar las detecciones, la tabla “detecciones”, guarda todas las detecciones que se realizan en tiempo real, en donde se registra el tipo de plaga, el nivel de confianza de la detección y la fecha en que se realizó la detección, como se muestra en la Figura 39.

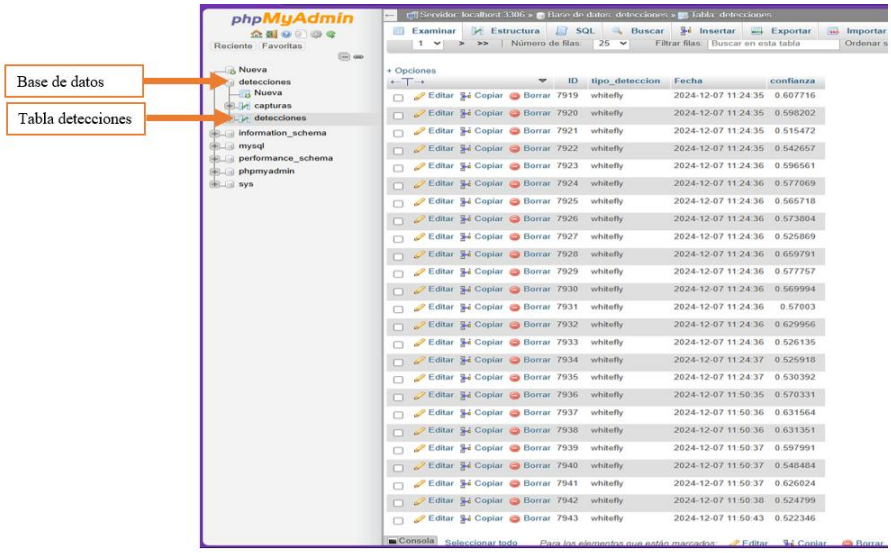


Figura 39. Tabla para las detecciones en tiempo real.

La tabla “capturas”, se creó para almacenar las detecciones que provienen de las imágenes capturadas que fueron programadas para que el sistema funcione de manera autónoma, en esa tabla se registra el tipo de plaga detectada, la confianza, ID de la cámara, la ruta de la imagen y la fecha en que se realizó la captura. Cabe recalcar que los datos solo se guarnan en la base de datos si se detectó alguna plaga, pero si no se detectaron plagas en la captura no se guarda nada en la base de datos, en la Figura 40 se muestra la tabla “capturas” de la base de datos “detecciones”.

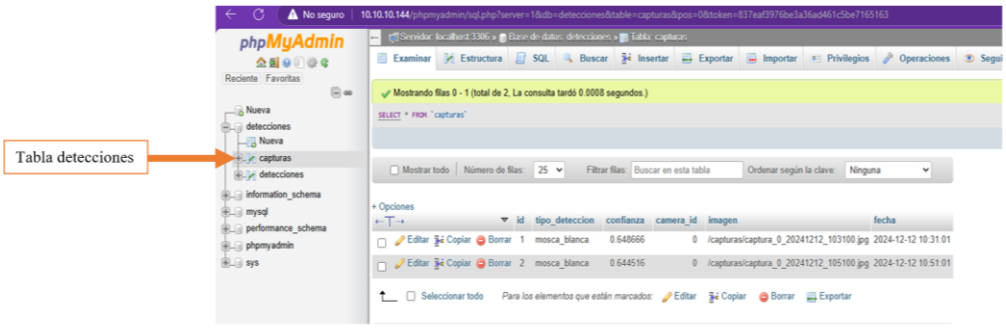


Figura 40.Tabla para las detecciones de las capturas programadas.

En la Figura 41 se muestra la relación que tiene la tabla capturas con la tabla detecciones, que es una relación de uno a muchos. Eso significa que una captura puede tener muchas detecciones asociadas, por ejemplo, una imagen puede contener múltiples detecciones de diferentes objetos, pero una detección solo puede estar asociada a una captura.

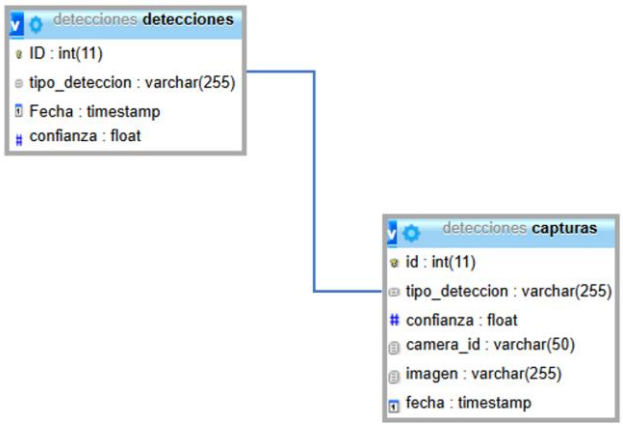


Figura 41. Diagrama entidad-relación.

3.7.3 Integración del sistema

Una vez que tenemos listo el entorno con todas las librerías instaladas, procedemos a declarar las variables de control las cuales ayudan a organizar y gestionar el funcionamiento del sistema como de muestra en la Figura 42, **selected_camera**, esta variable define a la cámara 1 para que este activa, **IP_CAMARA_RL**, en esta variable se almacena la dirección IP de la cámara de celular que proporciona la aplicación IP Webcam, **model**, esta variable contiene el modelo anteriormente entrenado para la detección de plagas, **real_time_detection** y **real_time_thread**, son variables que trabajan en conjunto para la detección en tiempo real, **activar_camara** y **camera_lock**, estas variables funcionan para manejar eficientemente las cámaras y no saturar los recursos de hardware del sistemas, tenemos las variables **camara_horarios** y **almacenar_tareas**, que funcionan para la gestión de los horarios en la mañana y tarde y por último la variable **dir_almacenamiento_img,** aquí es donde se guardan todas las imágenes capturadas.

```
# Variables de control
seleccionar_camara = 0  # Cámara por defecto (Cámara 1)
IP_CAMARA_URL = "http://192.168.1.2:8080/video"  # URL para la camara de celular
model = YOLO('best.pt')  # Ruta del modelo YOLOv8
real_time_detection = False
real_time_thread = None
activar_camara = None  # Variable para manejar la cámara activa
camara_lock = threading.Lock()  # Control de acceso a la cámara

camara_horarios = {
    "morning": {},  # Horarios para la mañana
    "afternoon": {}  # Horarios para la tarde
}
almacenar_tareas = {}  # Diccionario para almacenar las tareas programadas
# Ruta donde se guardarán las imágenes
dir_almacenamiento_img = "/var/www/html/capturas/"
```

Figura 42. Variables de Control.

Una vez creada la base de datos y las tablas como se muestra en la Figura 39 y Figura 40, procedemos a conectarnos con el script que se muestra en la Figura 43. El script consta de una función llamada **connect_db**, el cual utiliza la biblioteca **mysql.connector** para poder realizar la conexión a la base de datos que está localizada en el **host = "localhost",** con el usuario "**root"**, su clave "**erick"** y accede a la base de datos "**detecciones".** Si la conexión falla manda un mensaje de "**Error al conectar la base de datos"** y los detalles del error. Gracias a esta función podemos realizar las funciones de guardar o recuperar datos.

```python
# Conectar a la base de datos MySQL
def connect_db():
    try:
        return mysql.connector.connect(
            host="localhost",
            user="root",
            password="erick",
            database="detecciones"
        )
    except mysql.connector.Error as err:
        print(f"Error al conectar a la base de datos: {err}")
        return None
```

Figura 43. Conexión a la base de datos.

Para la captura de los frames de video tenemos una función llamada "**gen_frames"**, en donde se realiza la gestión de las cámaras para la captura de video y prepara dichos frames para su procesamiento como se muestra en la Figura 44. Primero verificamos si existe una cámara activa y si está abierta la liberamos con **"activar_camara.release()"**, con eso evitamos conflictos al abrir una nueva cámara y nos ayuda a poder elegir entre las distintas cámaras que están conectadas al sistema. Después verificamos si la variable "**camera_source"** es un índice o una URL ya que tenemos cámaras USB y una cámara que se conecta por medio de una IP, todo eso con la finalidad de realizar la detección en tiempo real. Con el bucle **"while real_time_detection:"**, realiza la captura de los frames en la detección en tiempo real, este bucle solo funciona si la variable global de control **"real_time_detection"** es True.

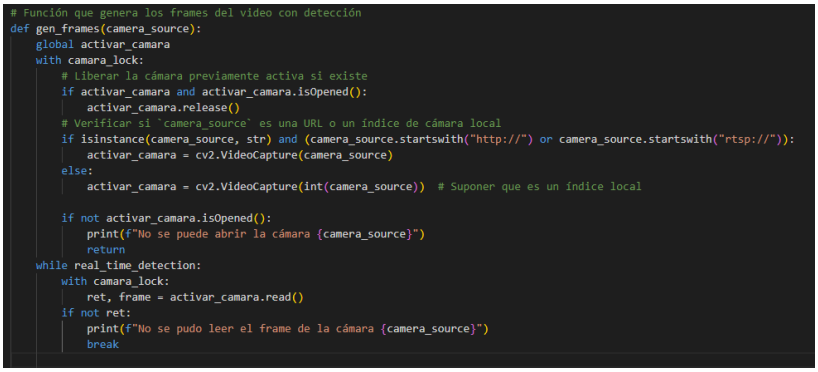


Figura 44. Captura de los frames y gestión de las cámaras.

Una vez que se obtiene la captura de los frames, porocedemos a realizar la detección en cada frame con **"results = model(frame)"**, con **boxes = results[0].boxes**, **class_ids = boxes.cls.numpy().astype(int)** y **confidences = boxes.conf.numpy()**, obtenemos las cajas de detección, los IDs de las clases y la confianza de las detecciones respectivamente como se muestra en la Figura 45.

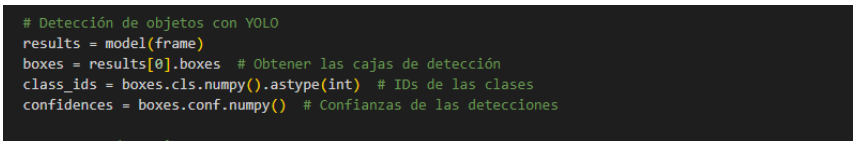


Figura 45. Detección de los objetos con YOLO.

Después de obtener las detecciones, los IDs de las clases y los niveles de confianza, para cada detección se extrae las coordenadas de la caja delimitadora y el nivel de confianza con **x1, y1, x2, y2 = map(int, box)** y **confidence = confidences[i]**. Después, se filtra las detecciones que tengan un nivel de confianza menor a 0.5 y gurda en la base de datos con **save_detection(tipo_deteccion, confidence)**. Por último, dibuja los rectángulos alrededor del objeto detectado, se añade la etiqueta y el nivel de confianza. En la Figura 46 se muestra el script anteriormente descrito.

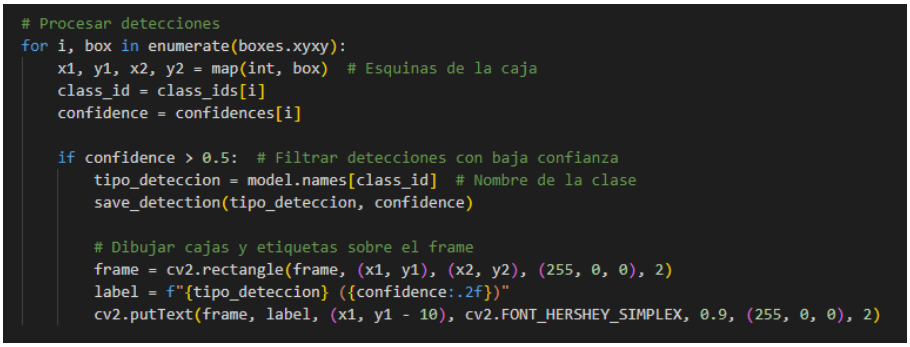


Figura 46. Procesamiento de las detecciones.

En la Figura 47 podemos observar los pasos para que el sistema sea autónomo, vamos a realizar la programación para que el sistema tome capturas de las cámaras, procese las capturas y detecte si tiene plagas para enviar alertas. Todo eso lo realizamos con la biblioteca schedule, primero obtenemos los datos que son enviados en una solicitud HTTP en formato JSON desde la interfaz, en donde se extrae time, camera_id y period. A continuación, realiza una validación de los datos para saber si están bien o si no están vacíos. Si los datos no son válidos devuelve un error, pero si está bien procede a verificar si la cámara seleccionada tiene horarios asignados en el periodo que se asigna y si no tiene horarios inicializa una lista vacía. Una vez que se verificó se agregan los nuevos horarios y se programa la captura con **job = schedule.every().day.at(time).do(capture_image, camera_id=int(camera_id))**, para que la función **capture_image** que se muestra en la Figura 49, se ejecute todos los días a la hora especifica en las distintas cámaras.

```python
def add_schedule():
    global camara_horarios, almacenar_tareas
    data = request.json
    time = data.get('time')
    camera_id = data.get('camera_id')  # Este debe ser un string
    period = data.get('period')  # "morning" o "afternoon"

    if time and camera_id is not None and period in ["morning", "afternoon"]:
        camera_id = str(camera_id)
        if camera_id not in camara_horarios[period]:
            camara_horarios[period][camera_id] = []

        if time not in camara_horarios[period][camera_id]:
            camara_horarios[period][camera_id].append(time)
            camara_horarios[period][camera_id] = sorted(camara_horarios[period][camera_id])

            # Programar la captura
            job = schedule.every().day.at(time).do(capture_image, camera_id=int(camera_id))
            almacenar_tareas[f"{period}_{camera_id}_{time}"] = job  # Guardar la referencia a la tarea

            return jsonify({"message": f"Hora añadida para cámara {camera_id} en la {period}"}), 200

    return jsonify({"message": "Hora ya existe o inválida"}), 400
```

Figura 47. Programar capturas automáticas.

Después de programar los horarios para las capturas, en la Figura 48 se procede a realizar la captura de la imagen de cada cámara en el horario programado. Primero, captura la imagen de la cámara seleccionada a la hora programada, genera un nombre a la imagen y lo guarda en la ruta especificada que se declara en la variable **“dir_almacenamiento_img”**, después analiza la imagen capturada con **process_captured_image**, función que se muestra en la Figura 49, esta función procesa la imagen para detectar si tiene o no plagas. Si la captura contiene plagas envía una alerta a telegram detallando el tipo de detección y la confianza de la detección y si no tiene plagas no envía nada.

```python
def capture_image(camera_id):
    cap = cv2.VideoCapture(camera_id)
    try:
        if cap.isOpened():
            ret, frame = cap.read()
            if ret:
                # Nombre del archivo y ruta completa
                filename = f"captura_{camera_id}_{time.strftime('%Y%m%d_%H%M%S')}.jpg"
                full_path = f"{dir_almacenamiento_img}{filename}"

                # Guardar la imagen
                cv2.imwrite(full_path, frame)
                print(f"Imagen capturada y guardada: {full_path}")

                # Procesar la imagen capturada
                detecciones = process_captured_image(filename, camera_id)
                if detecciones:
                    print(f"Se detectaron plagas en la captura: {detecciones}")

                    #Enviar alerta de detección a telegram
                    message = f"⚠️ Alerta de detección de plagas ⚠️\n"
                    for det in detecciones:
                        message += f"Tipo: {det['tipo_deteccion']}, Confianza: {det['confianza']:.2f}\n"

                    #enviar mensaje y la imagen a telegram
                    send_telegram_alert(message,full_path)

                else:
                    print("No se detectaron plagas en la captura.")
    finally:
        cap.release()
```

Figura 48. Captura de las imágenes programadas.

En la Figura 49 se muestra el script para realizar el procesamiento de las capturas programadas. Consta de una función llamada **process_captured_image**, primero carga la imagen desde el directorio en el que se guardó, esa dirección está almacenada en la variable **"dir_almacenamiento_img"** de la Figura 42, luego realiza la detección de plagas en la imagen con el modelo YOLO, extrayendo las coordenadas de la caja delimitadora, los IDs de las clases y el nivel de confianza. Después solo se procesan las detecciones con una confianza mayor a 0.5, estas detecciones se añaden a una lista de detecciones para posteriormente guardarlas en la base de datos con **save_capture_detection**.

```python
# Función para procesar imágenes capturadas y detectar plagas
def process_captured_image(filename, camera_id):
    frame = cv2.imread(f"{dir_almacenamiento_img}{filename}")
    results = model(frame)
    boxes = results[0].boxes
    class_ids = boxes.cls.numpy().astype(int)
    confidences = boxes.conf.numpy()

    detecciones = []
    for i, box in enumerate(boxes.xyxy):
        x1, y1, x2, y2 = map(int, box)
        class_id = class_ids[i]
        confidence = confidences[i]

        if confidence > 0.5:
            tipo_deteccion = model.names[class_id]
            detecciones.append({
                "tipo_deteccion": tipo_deteccion,
                "confianza": confidence,
                "camera_id": camera_id,
                "imagen": filename
            })
    print(f"Detecciones procesadas: {detecciones}")
    for det in detecciones:
        save_capture_detection(det["tipo_deteccion"], det["confianza"], camera_id, filename)

    return detecciones
```

Figura 49. Procesamiento de las capturas programadas.

En la Figura 50 se muestra la configuración y envío de notificaciones para telegram. Primero se debe configurar un bot en la aplicación de telegram para poder enviar las notificaciones, de ese bot obtenemos el Token y el ID, los cuales colocaremos en las variables **TELEGRAM_BOT_TOKEN** y **TELEGRAM_CHAT_ID**. Para el envío de mensajes primero se debe generar una alerta en la función **capture_image**, que se muestra en la Figura 48. Una vez generado el mensaje de alerta, envía el mensaje mediante una solicitud HTTP POST a la API de telegram, con todos los datos que fueron generados en el mensaje y se envía conjuntamente la imagen capturada que contiene las plagas.

```python
# Configuración de Telegram
TELEGRAM_BOT_TOKEN = '8143234351:AAH70A9iKGNBGRkUO2GV3uV0qZesFbZ-WC8'
TELEGRAM_CHAT_ID = 5495137846  # Reemplaza con tu ID de usuario de Telegram

# Enviar notificaciones a Telegram
def send_telegram_alert(message, image_path=None):
    url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
    data = {'chat_id': TELEGRAM_CHAT_ID, 'text': message}

    # Enviar mensaje de texto
    try:
        response = requests.post(url, data=data)
        print(f"Notificación enviada: {response.json()}")
    except Exception as e:
        print(f"Error al enviar notificación: {e}")

    # Enviar imagen, si está disponible
    if image_path:
        url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendPhoto"
        try:
            with open(image_path, 'rb') as image_file:
                files = {'photo': image_file}
                data = {'chat_id': TELEGRAM_CHAT_ID}
                response = requests.post(url, data=data, files=files)
                print(f"Imagen enviada: {response.json()}")
        except Exception as e:
            print(f"Error al enviar la imagen: {e}")
```

Figura 50. Configuración de telegram y envió de notificaciones.

3.7.4 Interfaz de Usuario

Para el sistema de diseño una interfaz gráfica que sea amigable con el usuario como se muestra en la Figura 51, el cual consta de tres bloques importantes, bloque 1 para la detección en tiempo real, bloque 2 para control de la cámara y programar los horarios en los que se va a realizar capturas de los cultivos y el bloque 3 en donde se visualiza las detecciones en tablas. La interfaz fue diseñada para que sea muy intuitiva, garantizando que un usuario que no conozca nada del sistema pueda manejarlo sin ningún problema.

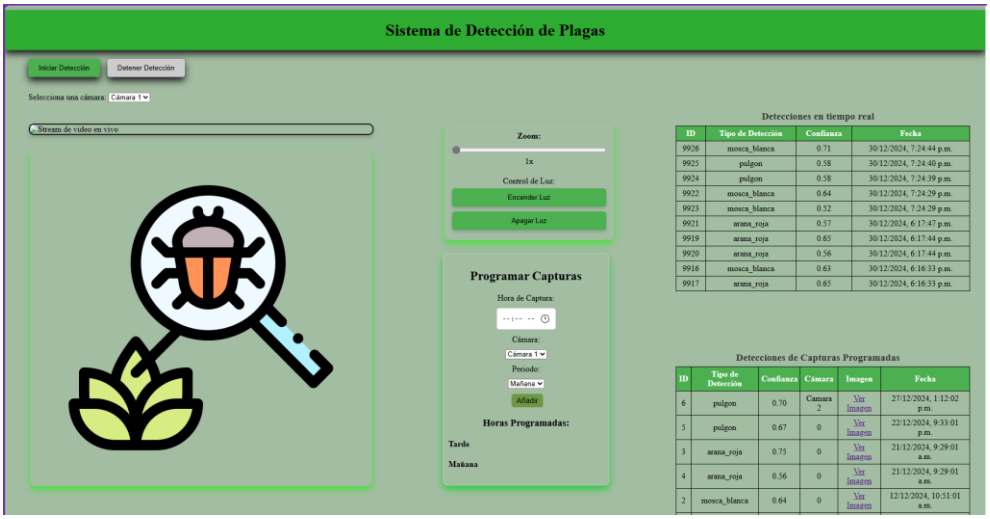


Figura 51. Interfaz de usuario para el sistema de detección de plagas.

- ***Bloque 1 (Detección en tiempo real).*** En este bloque se realiza la detección de plagas en tiempo real, como se puede observar en la Figura 52, tenemos dos botones que tienen la función de iniciar y detener la detección en tiempo real, esto con el fin de no saturar los recursos de hardware, también cuenta con un sistema de selección de cámara, el cual permite intercambiar entre las distintas cámaras que están conectadas al sistema. Este bloque ayuda mucho al usuario si desea realizar la detección en cualquier hora del día.

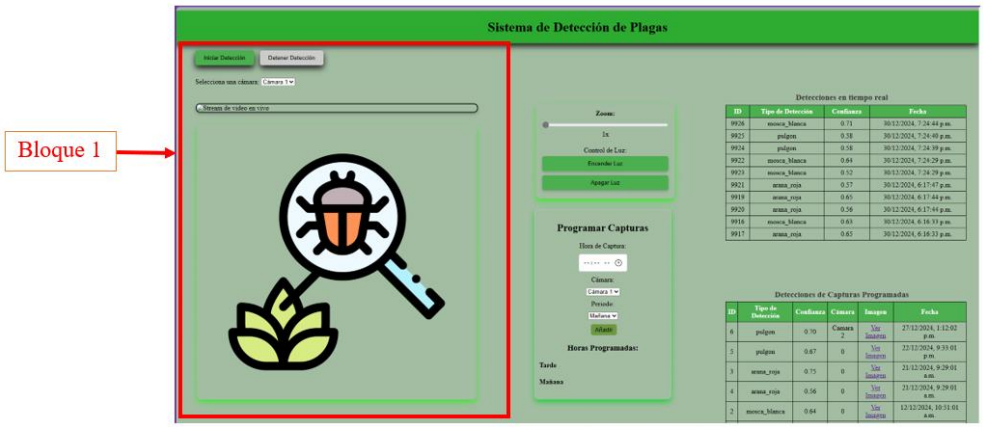


Figura 52. Bloque 1, detección en tiempo real.

- ***Bloque 2 (Control de cámara IP y programar horarios para capturas).*** En este bloque existen dos secciones una para controlar el zoom y la luz de la cámara IP y la sección 2 es para programar los horarios para la captura de las imágenes de cada cámara USB como se muestra en la Figura 53. La sección 2 de este bloque contiene un campo para seleccionar la hora en la que el usuario desee programar la captura

de la imagen, un campo para elegir qué cámara quiere que capture en dicha hora y un campo para seleccionar si la captura se realiza en la mañana o en la tarde. También cuenta con un botón de añadir el cual guardara la hora y la cámara para realizar la captura y un botón de eliminar si desea cambiar de hora. El sistema esta pesado para que funcione de manera autónoma, entonces si el usuario programo las capturas en la mañana y en la tarde el sistema realizara la toma de las capturas todos los días y procesara las imágenes como se mostró en la Figura 49, todo esto mientras el sistema este ejecutándose.

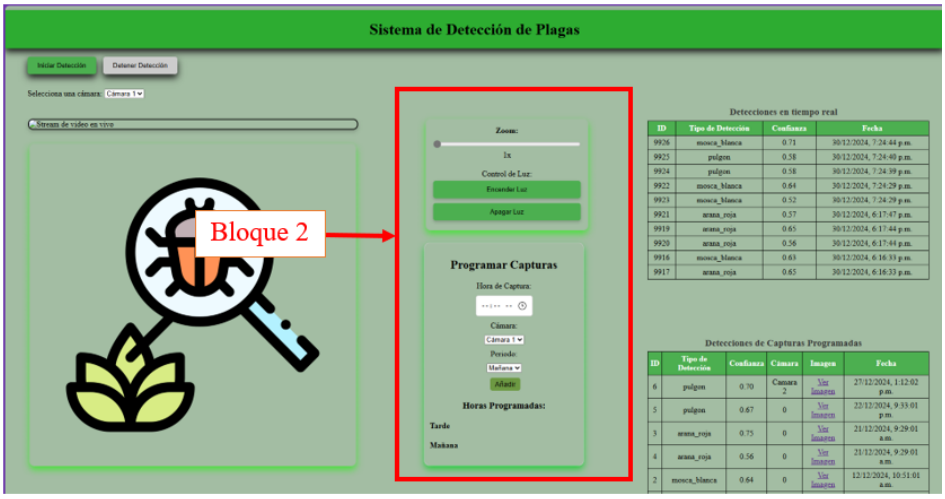


Figura 53. Bloque 2, Control de cámara IP y programación de horarios para las capturas.

- ***Bloque 3 (Visualización de detecciones).*** Por último, en el bloque 3 se realiza la visualización de las detecciones en tablas como se muestra en la Figura 54. En este bloque tenemos 2 tablas, tabla para las detecciones en tiempo real, aquí es donde se mostrarán las ultimas detecciones adquiridas por el sistema mostrando el tipo de detección, confianza y la fecha en la que se hizo la detección. Tabla para detecciones de capturas programadas, aquí se muestra las ultimas detecciones que se hicieron en las capturas que fueron programadas en diferentes horarios por el usuario, mostrando el tipo de detección, confianza, imagen y la fecha en la que se realizó la detección.

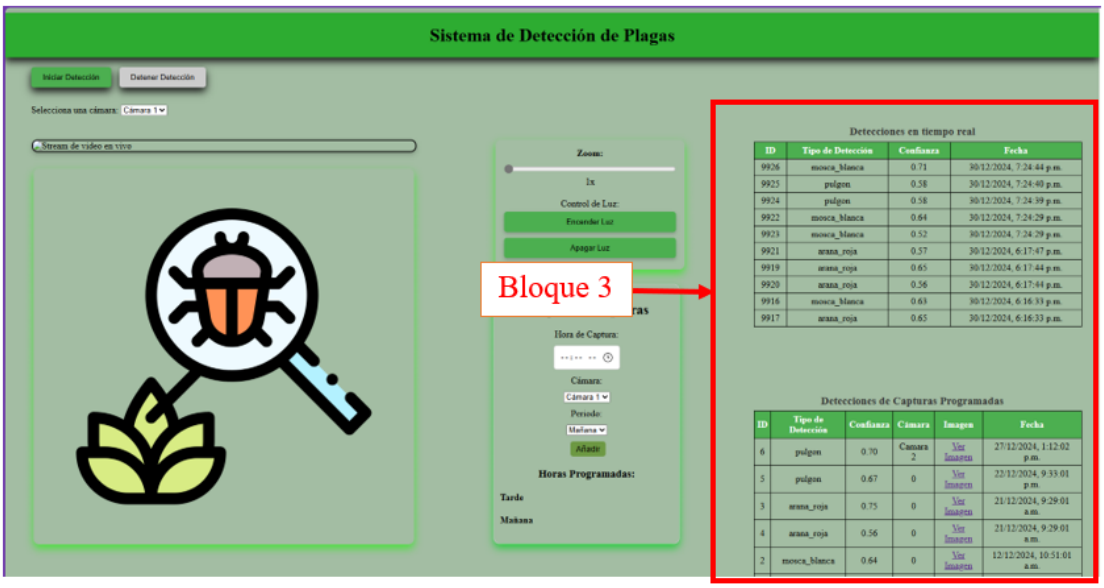


Figura 54. Bloque 3, Visualización de las detecciones.

3.7.5 Instalación del sistema dentro del invernadero

Para la implementación del sistema dentro del invernadero, se realizó un bosquejo de cómo queremos que se vea una vez que se instale el sistema completo. En la Figura 55 tenemos una referencia de cómo se quiere que quede instalado el sistema.



Figura 55. Representación 3D del sistema instalado.

a. Preparación del invernadero

Primero se debe verificar que haya conexiones a internet y conexiones de energía, después se debe inspeccionar la estructura del invernadero para identificar los puntos exactos en donde se colocará cada parte del sistema, para la instalación de las cámaras se debe encontrar puntos estratégicos en donde tenga un soporte y buen visón para colocar los soportes de las cámaras.

En la Figura 56 podemos observar los puntos estratégicos para colocar los soportes de las cámaras, es recomendable emplear soportes resistentes para que aseguren la estabilidad de la cámara y también es muy importante que los soportes se coloquen en puntos donde la cámara tenga una buena visión hacia las plantas a monitorear.



Figura 56. Soportes para las cámaras.

Se ubica el punto exacto en donde se colocará el servidor, el cableado de las cámaras que irán conectadas a la computadora y la estación fija para la detección detallada con la cámara del celular. En la Figura 57 se muestra la ubicación seleccionada dentro del invernadero.



Figura 57. Espacio designado para el servidor.

b. Instalación de las cámaras

Después de ubicar los puntos exactos para los soportes de las cámaras IMX219, procedemos a ajustar los soportes, configuramos la altura y el ángulo de las cámaras para que tengan una buena visión de las plantas y conectamos las cámaras mediante cables USB, que irán enrutados por canaletas para que no estén expuestos. En la Figura 58 se muestra la instalación de las cámaras.



Figura 58. Instalación de las cámaras.

Para la cámara de celular cuenta con una caja de madera adaptada para que el celular encaje correctamente y se pueda utilizar la cámara. La caja es de un tamaño de 20 x

15 cm y tiene un agujero en la parte superior en donde se coloca el celular y realiza la detección de las plagas, esta cámara se controla desde la interfaz gráfica y se podrá controlar el zoom y el encendido de luz. En la Figura 59 se muestra el soporte del celular.



Figura 59. Soporte para el celular.

c. Instalación de la estación de monitoreo

Una vez ubicado en donde va a ir la estación de monitoreo, procedemos a instalarla en una mesa, debemos tener en cuenta que debe haber un tomacorriente para energizar el computador y también tener conexión a internet, para que el sistema funcione correctamente. La estación fija para detección con el celular lo colocamos a un lado del computador, con el fin de que sea una estación en donde se realicen detecciones más detalladas. En la Figura 60 se muestra la estación de monitoreo instalada.



Figura 60. Estación de monitoreo instalada.

Finalmente, en Figura 61, podemos observar el sistema completamente instalado en el invernadero.



Figura 61. Instalación completa del sistema.

3.8 Resultados del entrenamiento del modelo.

En la Figura 62 podemos observar los resultados de las métricas de rendimiento del modelo entrenado en 80 épocas para la detección de plagas. Como se observa las pérdidas de train/box_loss empieza en 1.7 y desciende a 1.2, train/cls_loss empieza mayor a 2.5 y desciende a 0.6, mientras que train/dfl_loss empieza en 1.4 y baja a 1.1. Todas estas pérdidas indica una mejora progresiva en la localización y clasificación de los objetos a detectar. Las métricas de precisión (B) y recall (B), muestran incrementos significativos, pasando de valores de 0.3 y 0.4 a valores aproximadamente de 0.85 y 0.8, respectivamente. De igual manera en la parte de validación las pérdidas de val/box_loss, val/cls_loss y val/dfl_loss, también disminuyen de 2.4, 2.5 y 2.6 a 1.6, 1.4 y 1.8 respectivamente. Las métricas de validación mAP50(B) aumentan de 0.2 a 0.75, mientras que mAP50-95(B) tiene una mejora de 0.1 a 0.5, lo que confirma un rendimiento robusto y estable del modelo en diferentes niveles de precisión. En general todos estos resultados reflejas un modelo bien entrenado, que tiene un aprendizaje constante y un buen desempeño.

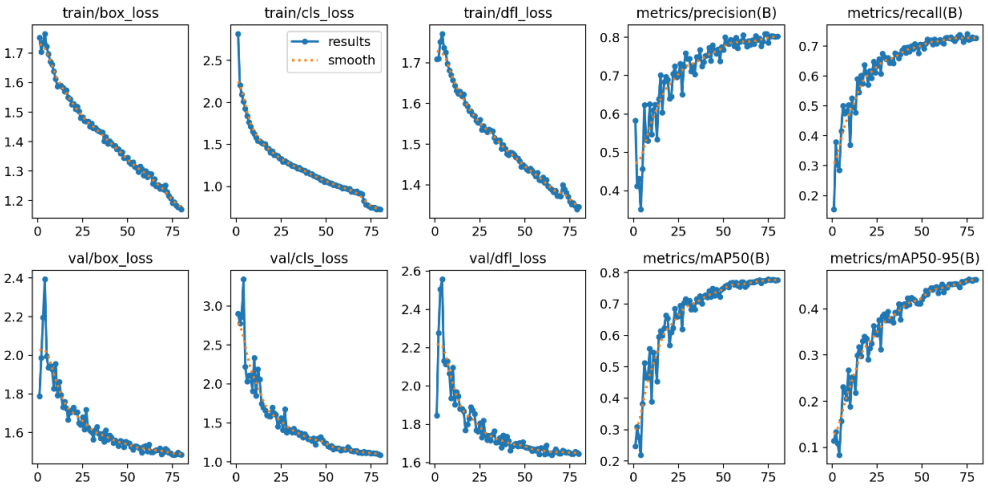


Figura 62. Métricas de rendimiento del modelo entrenado

En la Figura 63 podemos observar la matriz de confusión, nos muestra que el modelo tiene un buen desempeño en la clase araña_roja con un valor de 86% de precisión y en mosca_blanca con un 84% de precisión. Por otro lado, la clase pulgón con un valor de 66% de precisión, presenta confusiones significativas con el background (52%), esto sugiere que el modelo puede estar clasificando incorrectamente algunas instancias de pulgón como background. Para mejorar este problema, podemos ajustar el umbral de confianza, elevándole a un valor más alto entre 50% y 70%. Con eso el modelo aceptara solo las predicciones seguras de todas las clases.

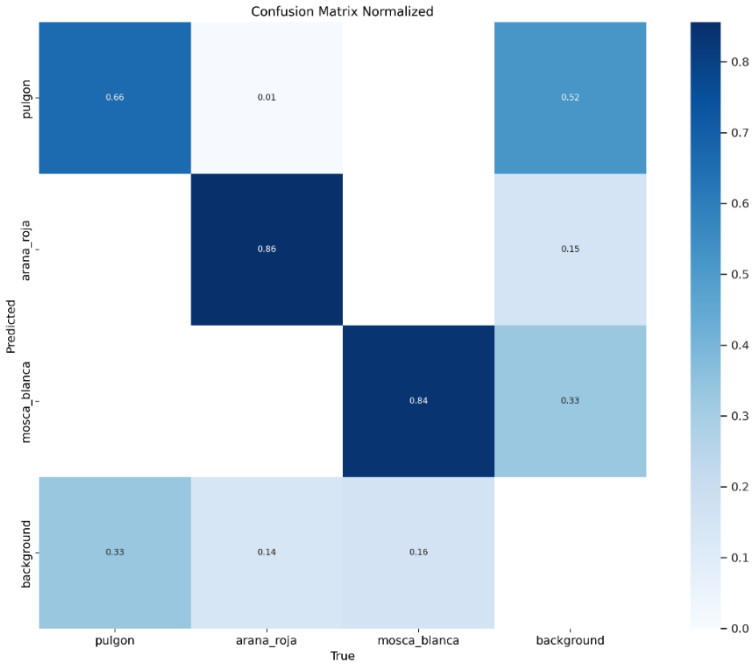


Figura 63. Matriz de confusión.

3.8.1 Predicción por lote de validación

La predicción por lote del conjunto de validación, se refiere a que el modelo realiza predicciones sobre datos que no ha visto durante el entrenamiento, lo que funciona para evaluar el rendimiento del modelo. Para esta predicción se toma un lote de imágenes de 8, 16 o 32 imágenes del conjunto de datos de validación y realiza las predicciones de cada imagen del lote tomado, identificando los objetos y sus clases. En las Figura 64, Figura 65, y Figura 66 se muestra las predicciones por lotes del conjunto de validación para cada una de las clases a detectar.



Figura 64. Predicción por lotes para la clase araña roja.



Figura 65. Predicción por lotes para la clase mosca blanca.

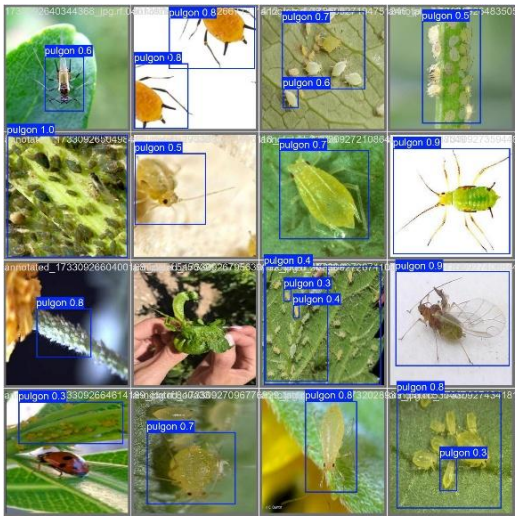


Figura 66. Predicción por lotes para la clase Pulgón.

3.8.2 Precisión del entrenamiento

La Precision-Confidence Curve muestra cómo varía la precisión en función del nivel de confianza de las predicciones del modelo. Cada línea representa una clase: azul claro para pulgón, naranja para araña roja, verde para mosca blanca y una línea azul gruesa para el desempeño general. En el eje X se indica el nivel de confianza, mientras que en el eje Y se representa la precisión, es decir, la proporción de predicciones correctas. La Figura 67 ilustra esta relación, evidenciando que el rendimiento global del modelo alcanza un 95.7% de precisión, lo que refleja un desempeño sólido en la detección de plagas.

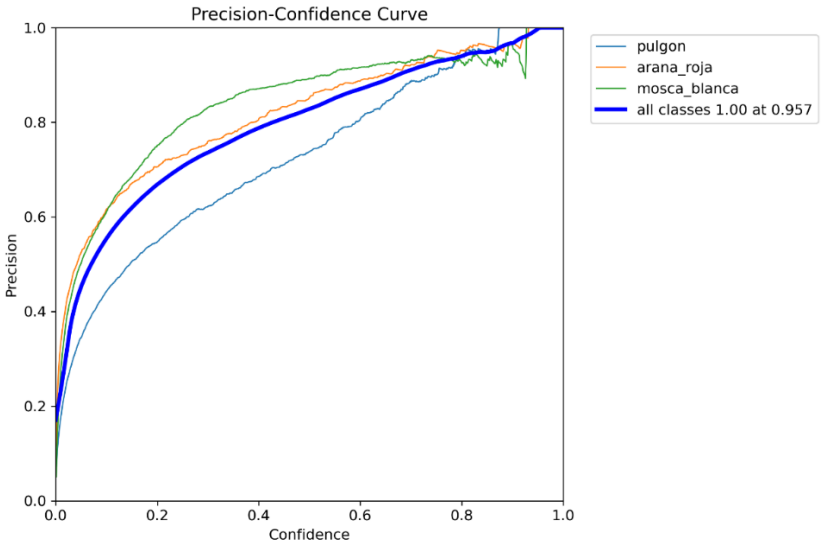


Figura 67. Precisión del modelo entrenado.

3.8.3 Costo computacional

Para el entrenamiento del modelo de detección de plagas se utilizó la plataforma de Google Colab ya que cuenta con recursos computacionales que aceleran el procesamiento del entrenamiento. En la Tabla 17 se muestra el costo computacional al entrenar una red neuronal en distintas épocas.

Tabla 17. Costo computacional

Parámetro del notebook				Descripción
CPU				CPU Intel Xeon a 2.20 GHz
GPU				NVIDIA Tesla T4 Tensor Core
RAM				13 GB
Memoria de la GPU				15 GB
Sistema Operativo				Linux
Almacenamiento				112.6 GB
Modelo entrenado				Yolov8n
Número de épocas				**Total**
Épocas	30	60	80	170
Tiempo (horas)	1:30	1:55	2:30	5:15 (horas)
Uso de la GPU		8.60 GB / 15 GB		57.33% de 100%

3.8.4 Pruebas de funcionamiento del modelo entrenado

Después de finalizar con el proceso de entrenamiento del modelo YOLOv8 para la detección de plagas, sometemos al modelo a varias pruebas para verificar su funcionamiento, para las pruebas se realizó con un script básico con Python, las respectivas librerías de Ultralytics y la librería de Cv2, las pruebas fueron realizadas con imágenes o videos de las plagas con el fin de que el modelo detecte correctamente.

Como podemos observar en la Figura 68, Figura 69 y Figura 70, estamos realizando pruebas de detección de las diferentes plagas, se puede observar que al realizar las detecciones la mayor parte tiene una confiabilidad entre 0.7 y 0.9 lo que significa que el modelo está identificando correctamente las plagas, por otro lado, las detecciones menores a 0.7 pueden ser filtradas para no obtener falsos positivos, siendo una buena estrategia para mejorar la fiabilidad del sistema.

Hay que tomar en cuenta que la confiabilidad puede ser superior si la detección se realiza en un ambiente controlado, como un laboratorio, con una mejor iluminación y mejor estabilidad.

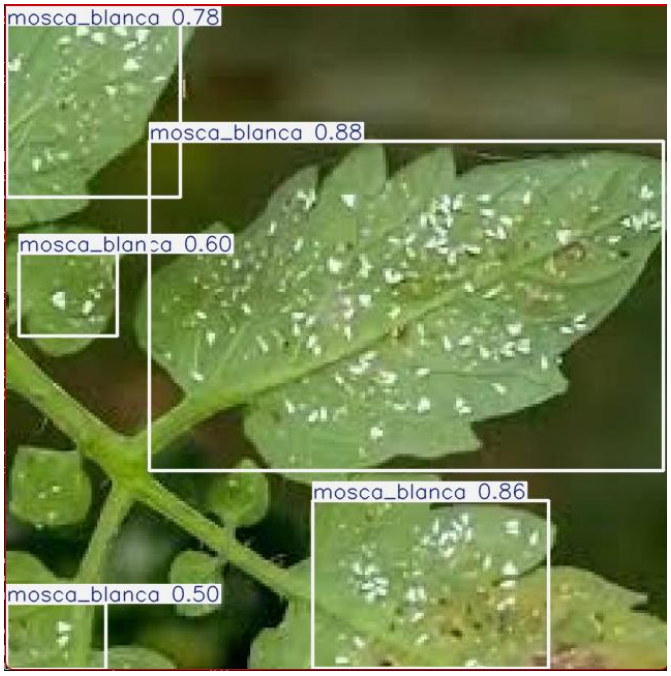


Figura 68. Prueba de detección, mosca blanca.

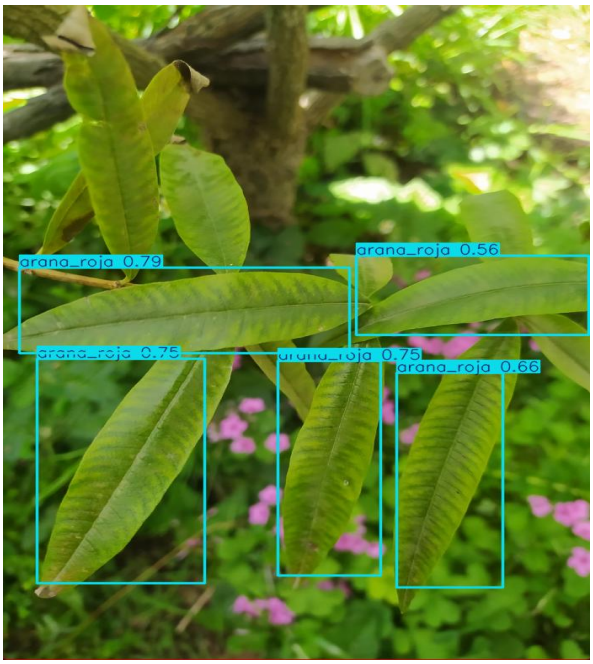


Figura 69. Prueba de detección, araña roja.

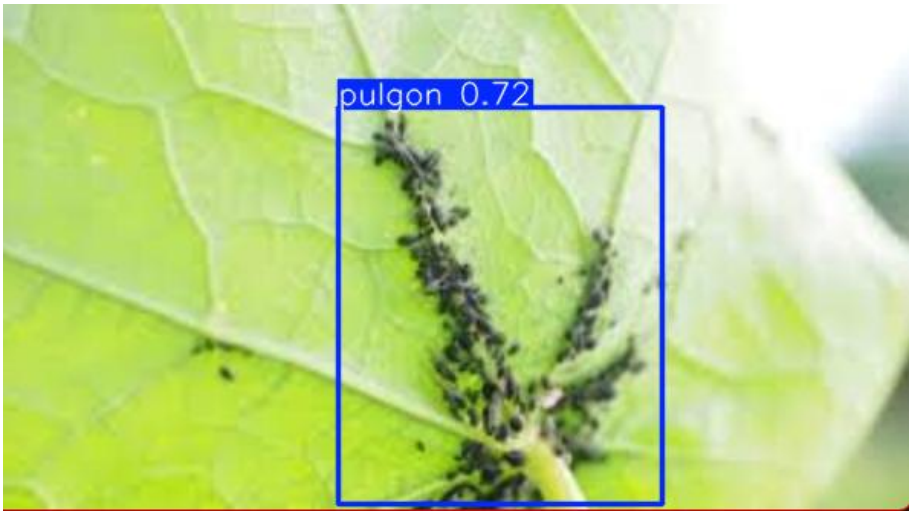


Figura 70. Prueba de detección, Pulgón.

3.9 Resultados del funcionamiento del sistema de detección de plagas

Como se puede observar en la Figura 71, se pone a prueba el funcionamiento de la detección de plagas en tiempo real, en la interfaz web primero se inicia la detección en tiempo real dando clic en el botón, comienza a grabar video y con el modelo entrenado procesa cada frame del video para verificar si tiene o no plagas, en la figura podemos ver que detecta un daño que causa la plaga araña roja, de igual manera esa detección es guardada en la base de datos y se muestra en la tabla “Detección en tiempo real” con la fecha , hora , tipo de plaga y la confianza de la detección.

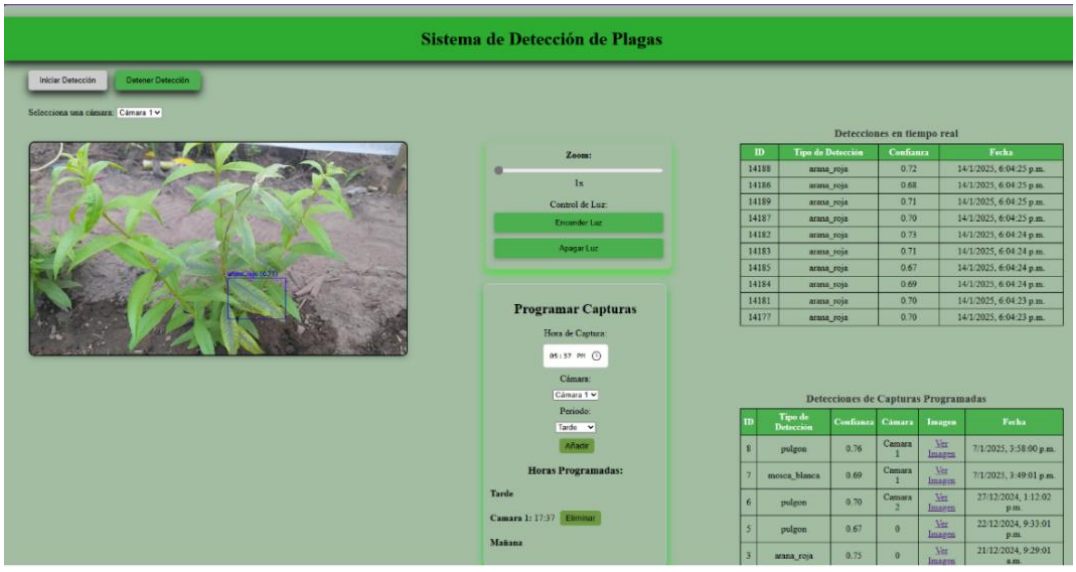


Figura 71. Detección en Tiempo real.

Como se puede observar en la Figura 72, realizamos una captura programada en una hora especifica, para eso debemos dirigirnos al apartado de “Programar Capturas” y elegimos la hora, la cámara y si es en la mañana o en la tarde, le damos a añadir y con eso ya tenemos listo para que la cámara elegida tome la captura, esa captura será procesada por la red neuronal y verificará si es que la imagen contiene plagas. Si es que después de ser procesada la red neuronal detectó alguna plaga en la imagen, los datos como el tipo de detección, confianza, cámara, imagen y fecha son enviados a la base de datos, para después ser mostrado en la tabla “Detecciones de captura programadas” en la interfaz web.

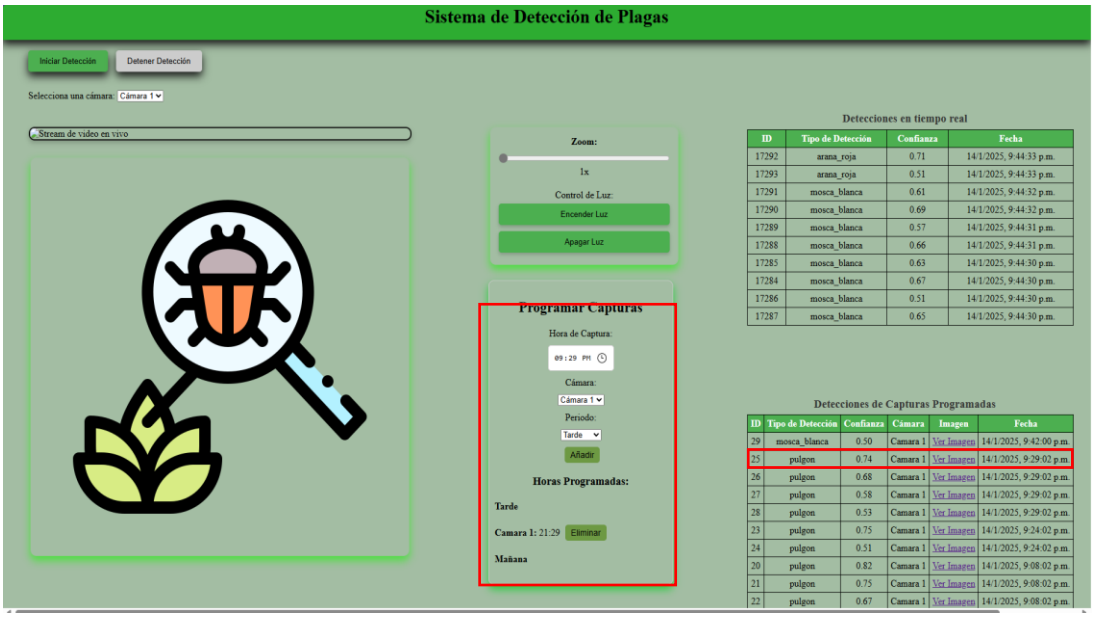


Figura 72. Programar captura de imagen de la cámara.

Al mismo tiempo que se guarda en la base de datos y se muestra en la interfaz web las detecciones que se hicieron en la imagen capturada, también envía un mensaje de alerta a Telegram, en ese mensaje se muestra un mensaje de “Alerta de detección de plagas”, con el tipo de plaga y la confianza de la detección. Después del mensaje también se adjunta la imagen con la detección realizada. En la Figura 73, podemos observar el funcionamiento del sistema de alerta con lo mencionado anteriormente.



Figura 73. Sistema de alerta en Telegram.

En la Figura 74 se puede observar el funcionamiento de la detección de plagas en la estación, gracias a la utilización del zoom de la cámara podemos detectar la clase mosca blanca, en el Anexo D se muestra como está compuesto la estación de detección y en la Figura 59 se muestra instalado en el invernadero.

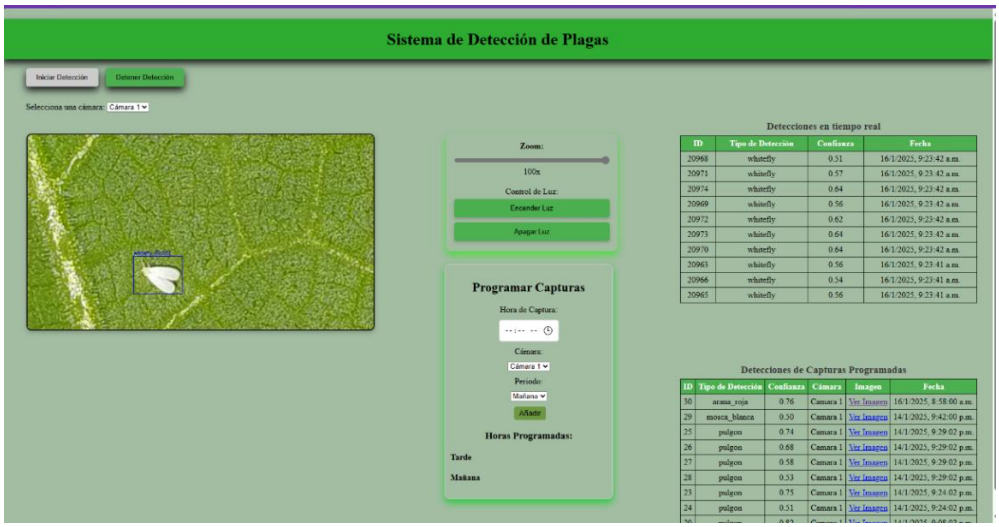


Figura 74. Funcionamiento de la estación de detección.

3.9.1 Resultados de las pruebas de Funcionamiento.

Para la interpretación de los resultados del sistema de detección de plagas, se realizó dos comparaciones entre los resultados obtenidos con el sistema y los resultados obtenidos con una inspección visual por el humano. En la Tabla 18 se muestra las pruebas realizadas del funcionamiento del sistema.

Tabla 18. Pruebas del funcionamiento del sistema

e	Hora	# Plantas Analizadas	Tipo de Plaga Esperada	Cantidad Real (Zonas Infestadas)	Inspección Humana Visual	Inspección Artificial
1	8:00 AM	1	Araña roja	2	2	2
2	8:15 AM	1	Pulgón	3	3	3
3	8:30 AM	1	Araña roja	2	2	2
4	8:45 AM	1	Mosca blanca	6	6	6
5	9:00 AM	1	Mosca blanca	4	2	3
6	9:15 AM	1	Pulgón	2	2	2
7	9:30 AM	2	Araña roja	6	6	5
8	9:45 AM	3	Pulgón	2	2	2
9	10:00 AM	1	Araña roja	2	2	2
10	10:15 AM	2	Mosca blanca	4	4	3
11	10:30 AM	3	Pulgón	2	2	1
12	10:45 AM	1	Mosca blanca	3	2	3
13	11:00 AM	2	Araña roja	6	4	6
14	11:15 AM	3	Pulgón	2	2	2
15	11:30 AM	1	Araña roja	2	2	3
16	11:45 AM	2	Mosca blanca	6	6	6
17	12:00 PM	3	Pulgón	3	3	3
18	12:15 PM	1	Pulgón	2	2	2
19	12:30 PM	2	Araña roja	6	6	6
20	12:45 PM	3	Mosca blanca	1	1	1
21	13:00 PM	1	Mosca blanca	2	2	1
22	13:15 PM	2	Araña roja	3	3	3
23	13:30 PM	3	Araña roja	2	1	2
24	13:45 PM	1	Araña roja	5	5	4
25	14:00 PM	2	Araña roja	3	3	3
26	14:15 PM	3	Pulgón	2	2	2
27	14:30 PM	1	Pulgón	1	1	1
28	14:45 PM	2	Pulgón	6	6	5
29	15:00 PM	3	Pulgón	2	2	1

e	Hora	# Plantas Analizadas	Tipo de Plaga Esperada	Cantidad Real (Zonas Infestadas)	Inspección Humana Visual	Inspección Artificial
30	15:15 PM	1	Pulgón	2	2	3
31	15:30 PM	2	Mosca blanca	6	6	6
32	15:45 PM	3	Mosca blanca	2	2	2
33	16:00 PM	1	Mosca blanca	2	2	2
34	16:15 PM	2	Mosca blanca	6	6	6
35	16:30 PM	3	Mosca blanca	3	2	3
36	16:45 PM	1	Mosca blanca	2	2	2
37	17:00 PM	2	Araña roja	6	6	6
38	9:00 AM	3	Araña roja	5	5	4
39	9:10 AM	1	Araña roja	2	2	2
40	9:20 AM	2	Araña roja	6	6	5
41	9:30 AM	3	Araña roja	2	2	2
42	9:40 AM	1	Araña roja	2	2	1
43	9:50 AM	2	Araña roja	6	6	6
44	10:00 AM	3	Pulgón	1	1	1
45	10:10 AM	1	Pulgón	3	2	3
46	10:20 AM	2	Pulgón	2	1	2
47	10:30 AM	3	Pulgón	3	3	3
48	10:40 AM	1	Pulgón	2	2	2
49	10:50 AM	2	Pulgón	2	2	2
50	11:00 AM	3	Plugón	4	4	3

En la Tabla 18, se muestra una comparación de detecciones realizadas con el sistema y detecciones realizadas por el humano con el número real de las zonas infestadas con plagas. En la Figura 75, se muestra un gráfico de barras titulado "Detección de plagas", donde se comparan tres métodos de identificación de infestaciones: Cantidad Real (Zonas Infestadas) (en azul), Inspección Humana Visual (en naranja) e Inspección Artificial (en gris). Se analizan tres tipos de plagas: Pulgón, Araña Roja y Mosca Blanca, además del total de detecciones. Los datos reflejan que tanto la inspección humana como la artificial tienen valores cercanos a la cantidad real, con pequeñas diferencias en la detección.

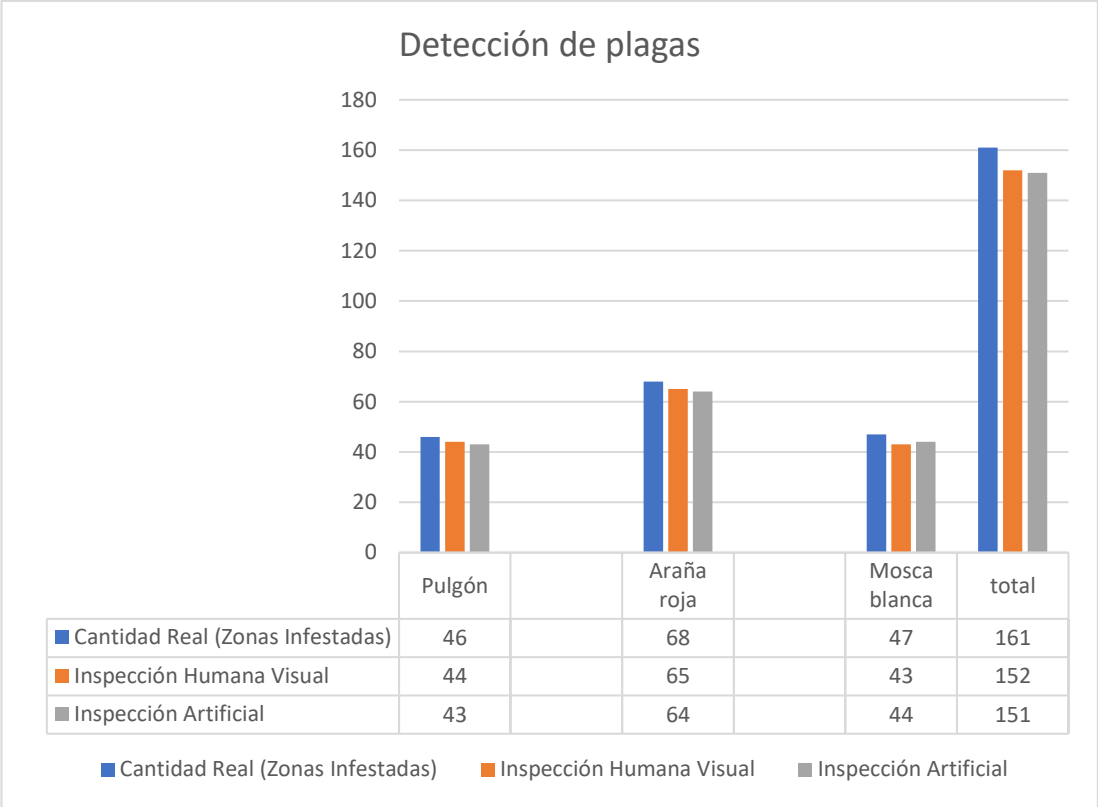


Figura 75. Análisis estadístico del sistema.

Según los datos de la Figura 75, para el Pulgón, la cantidad real de zonas infestadas fue de 46, mientras que la inspección humana detectó 44 y la inspección artificial 43. En el caso de la Araña Roja, la cantidad real fue de 68, con detecciones de 65 por inspección humana y 64 por inspección artificial. Para la Mosca Blanca, se registraron 47 zonas infestadas reales, mientras que la inspección humana identificó 43 y la artificial 44. Finalmente, en el total de detecciones, la cantidad real de infestaciones fue de 161, con 152 identificadas por inspección humana y 151 por inspección artificial. Con estos resultados se calcula la confiabilidad del sistema, como se muestra a continuación.

a. Cálculo del error absoluto

$$\boldsymbol{Error\ absoluto = |Valor\ real - valor\ obtenido|} \quad (2)$$

$$Error\ absoluto = |161 - 151|$$

$$Error\ absoluto = 10$$

b. Cálculo del error porcentual

$$\boldsymbol{Error\ porcentual\ =\ \frac{Error\ absoluto}{valor\ real} * 100} \quad (3)$$

$$Error\ porcentual\ =\ \frac{10}{161} * 100$$

$$Error\ porcentual\ =\ 6.21\ \%$$

c. Confiabilidad del sistema

$$\boldsymbol{Confiabilidad_{sistema}\ =\ 100\% - error\ porcentual} \quad (4)$$

$$\boldsymbol{Confiabilidad_{sistema}\ =\ 100\% - 6.21\%}$$

$$\boldsymbol{Confiabilidad_{sistema}\ =\ 93.79\ \%}$$

El sistema tiene un valor de confiabilidad del 93.79% en detecciones de plagas, garantizando un sistema fiable para el control de plagas.

3.9.2 Comparación con sistemas similares

En la tabla Tabla 19, se muestra la comparación del sistema desarrollado con otros sistemas similares. La comparación de sistemas indica que el sistema desarrollado, basado en YOLOv8n, alcanza una confiabilidad del 93.79%, un valor similar al de otros enfoques que utilizan YOLOv7 y YOLOv8. Aunque su precisión es ligeramente inferior a la del modelo con mejor desempeño (93.83%), sigue siendo competitiva. Además, mientras que otros sistemas se centran en una variedad más amplia de plagas, el sistema desarrollado está especializado en la detección de pulgón, araña roja y mosca blanca, plagas relevantes en cultivos de ambiente controlado. Esto demuestra que el modelo es adecuado para su aplicación específica y que su rendimiento es comparable al de estudios previos, validando su efectividad en el contexto en el que se emplea.

Tabla 19. Comparación con sistemas similares [35], [37].

Sistema desarrollado			
Tema	**Confiabilidad del Sistema**	**Algoritmo Utilizado**	**Plagas**
Identificación temprana de presencia de plagas en cultivos de ambiente controlado empleando visión artificial y Deep Learning	93.79%	YOLOv8n	Pulgón Araña roja Mosca blanca
Otros sistemas			
Tema	**Confiabilidad del Sistema**	**Algoritmo Utilizado**	**Plagas**
A Novel Smart System with Jetson Nano for Remote Insect Monitoring	93%	Yolov7	Escarabajo rayado Caracol gigante Saltamontes Mantis Chinche de calabaza Abeja de espárrago
Improved YOLOv8 and SAHI Model for the Collaborative Detection of Small Targets at the Micro Scale: A Case Study of Pest Detection in Tea	93.83%	YOLOv8	Pulgón negro de los cítricos Escarabajo del té Chicharrita del peral Chicharrita amarilla

3.10 Presupuesto

El sistema de identificación de plagas tiene un presupuesto el cual se calcula considerando los costos de los equipos, diseño y fabricación del sistema. Primero calculamos la remuneración, para eso ocupamos la ecuación (5), considerando que un mes cuenta con 21 días laborables y se trabajan 8 horas diarias. El sueldo de un ingeniero recién egresado es de $858 según el ministerio del trabajo, con todos estos datos realizamos los cálculos.

$$\boldsymbol{Remuneración\ por\ día} = \frac{\boldsymbol{remuneracíon\ por\ mes}}{\boldsymbol{dias\ laborables}} \tag{5}$$

$$Remuneración\ por\ hora = \frac{858}{21} = \$40.86$$

$$Remuneración\ por\ hora = \frac{40.86}{8} = \$5.11$$

Se considero que las horas empleadas para el desarrollo del proyecto fue de 120 horas, por lo tanto, el presupuesto total del diseño es de $613.2.

En la Tabla 20 se muestra el presupuesto con todos los precios bien detallados de cada equipo y de los distintos materiales adquiridos.

Tabla 20. Presupuesto.

Dispositivos Utilizados					
Ítem	Detalle	Unidad	Cantidad	Valor Unitario	Total
1	NVIDIA Jetson Nano	c/u	1	$ 263,00	$263.00
2	Cámara IMX219	c/u	3	$40	$120
3	Monitor	c/u	1	$ 85,99	$ 85,99
4	Mouse y teclado	c/u	1	$ 21,99	$ 21,99
5	Cable HDMI	c/u	1	$ 6,65	$ 6,65
6	Tarjeta de memoria	c/u	1	$15,94	$15,94
7	Extensor USB	c/u	3	$6.70	$20.10
Otros					
8	Tubo PCB	c/u	2	$3	$6
9	Codos de tubo PBC	c/u	4	$0.60	$2.40
10	MDF	c/u	3	$1	$3
				Total	$545.07

El presupuesto total para la implementación del sistema es la suma del presupuesto del diseño y el presupuesto de implementación por lo que calculamos el presupuesto total con la ecuación (6).

$$\boldsymbol{Presupuesto_{total}} = \boldsymbol{Presupuesto\ dise\tilde{n}o} + \boldsymbol{Presupuesto\ implementaci\acute{o}n} \quad (6)$$

$$Presupuesto_{total} = 613.2\ d\acute{o}lares + 545.07\ d\acute{o}lares$$

$$Presupuesto_{total} = 1158.27\ \ d\acute{o}lares$$

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Mediante el análisis de las plagas presentes en los cultivos de ambiente controlado en Susudel, se identificaron características visuales distintivas que facilitan su detección, como las tonalidades verdes y marrones del pulgón, el color rojizo de la araña roja junto al amarillamiento foliar, y el blanco traslúcido de la mosca blanca. A pesar de que métodos tradicionales como trampas de feromonas, monitoreo por grupo de plantas, monitoreo al azar y trampas cromáticas (amarillas o azules) son utilizados para el monitoreo, su implementación resulta compleja debido a la necesidad de personal capacitado, lo que resalta la importancia de identificar adecuadamente estas características para mejorar el control de plagas en ambientes controlados.

- Se logró el diseño de una arquitectura sólida y eficiente para el sistema de identificación de plagas en cultivos de ambiente controlado, incorporando técnicas avanzadas de visión artificial y Deep Learning. El sistema alcanzó una confiabilidad del 93.7% en la identificación de plagas, lo que garantiza detecciones precisas.

- Se incorporaron módulos de captura de imágenes mediante cámaras IMX219 y procesamiento con una Jetson Nano, lo que permitió la detección de plagas de manera eficiente. Gracias a la implementación de redes neuronales optimizadas, como YOLOv8n, el sistema puede operar en hardware de recursos limitados sin comprometer la precisión de las detecciones. Además, la incorporación de herramientas como Flask y MySQL facilita la visualización y gestión de la información, permitiendo a los usuarios interpretar los resultados de manera intuitiva.

- Se realizó una selección meticulosa de las herramientas de hardware, software y modelos de Deep Learning optimizados para el sistema de identificación de plagas. Para el hardware, se eligieron cámaras IMX219 de 8MP, con una resolución máxima de 3264 × 2448 píxeles, enfoque automático y capacidad

de captura de vídeo a 15 FPS en 3264 × 2448 píxeles o 30 FPS en 1080P/720P, lo que permite registrar detalles específicos de las plagas con alta precisión. Como unidad de procesamiento, se utilizó el microcomputador NVIDIA Jetson Nano, optimizado para trabajar con inteligencia artificial. En cuanto al modelo de Deep Learning, se seleccionó YOLOv8n, debido a su buena precisión, velocidad y bajo peso, ideal para tareas de detección en hardware de recursos limitados. Este modelo fue entrenado con 80 épocas, alcanzando una precisión del 95.7%, un recall de 0.9, un mAP50 de 0.776 y un mAP50-95 de 0.46, garantizando un rendimiento óptimo para el sistema.

- Se logró diseñar un prototipo funcional en el que se integran componentes claves, como las cámaras para la captura de imágenes de alta resolución, un microcomputador que sea compatible para trabajar con redes neuronales profundas y una interfaz para la gestión y visualización de los resultados obtenidos. El modelo de Deep Learning que se entreno fue YOLOv8n, para el cual se utilizó un conjunto de datos específico con imágenes de las diferentes plagas.

- Se desarrolló un sistema de gestión de información que facilita la visualización de los resultados obtenidos, integrando herramientas visuales e interactivas que permiten al usuario interpretar y manejar los datos de manera eficiente. La interfaz gráfica se organiza en tres bloques clave: el primero para la detección en tiempo real, permitiendo al usuario elegir entre las cámaras conectadas y realizar detecciones en directo; el segundo para controlar el zoom, la luz de la cámara del celular y programar horarios de captura a lo largo del día, ofreciendo autonomía al sistema al procesar las imágenes capturadas sin intervención constante del usuario, notificando cualquier detección de plagas a través de Telegram; y el tercero, que muestra las detecciones registradas tanto en tiempo real como en las capturas programadas, las cuales se almacenan en una base de datos para su posterior análisis.

4.2 Recomendaciones

- Para el entrenamiento del modelo de Deep Learning, se recomienda crear un dataset propio con imágenes del entorno en el que se va a realizar la detección de las plagas y que el número de datos del dataset sea mayor a 2,000 imágenes para realizar un buen entrenamiento.

- Para las detecciones, se recomienda utilizar cámaras de alta resolución, puesto que para la detección de plagas se necesita visualizar con claridad las características de cada plaga.

- Se recomienda explorar la integración de sistemas de detección automatizados con cámaras móviles que se desplacen automáticamente por rieles, junto con robots automáticos o drones, para permitir un monitoreo continuo sin la necesidad de utilizar muchos equipos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Organización de las Naciones Unidas para la Alimentación y la Agricultura, “Producción y protección vegetal”. Consultado: el 15 de septiembre de 2024. [En línea]. Disponible en: https://www.fao.org/plant-production-protection/about/es

[2] Ministerio de Agricultura y Ganadería, “Agricultores de Los Ríos afectados por invierno recibieron indemnizaciones de Seguro Agrícola”. Consultado: el 15 de septiembre de 2024. [En línea]. Disponible en: https://www.agricultura.gob.ec/agricultores-de-los-rios-afectados-por-invierno-recibieron-indemnizaciones-de-seguro-agricola/

[3] J. L. Zambrano Mendoza *et al.*, “Guía para la producción sustentable de maíz en la Sierra ecuatoriana”, 2021, Consultado: el 20 de septiembre de 2024. [En línea]. Disponible en: http://repositorio.iniap.gob.ec/handle/41000/5796

[4] J. L. Jinez Tapia y J. F. Jácome Garcia, “Detección temprana de minador, mosca blanca y fusarium en el tomate riñón, aplicando técnicas de visión artificial y machine learning”, 2022. Consultado: el 23 de septiembre de 2024. [En línea]. Disponible en: http://dspace.unach.edu.ec/handle/51000/9102

[5] S. H. Valenzuela Cámara, “Detección y clasificación de enfermedades en el tomate mediante Deep Learning y Computer Vision”, Universidad Nacional de La Plata, 2022. Consultado: el 23 de septiembre de 2024. [En línea]. Disponible en: http://sedici.unlp.edu.ar/handle/10915/139770

[6] V. Y. Sierra, “Algoritmo de detección de mosca blanca por medio de inteligencia artificial en las hojas de plátano.”, 2021. Consultado: el 23 de septiembre de 2024. [En línea]. Disponible en: https://repository.unad.edu.co/handle/10596/42374

[7] N. López Grisales, “Identificación y conteo de insectos (mosca blanca, trips y minador de hoja) capturados en trampa adhesiva mediante visión e

inteligencia artificial. Caso Flores El Trigal", Universidad EIA, 2020. Consultado: el 23 de septiembre de 2024. [En línea]. Disponible en: https://repository.eia.edu.co/handle/11190/2654

[8] E. Berra Villaseñor, M. Pérez Castañeda, y J. R. C. Parodi, "Redes neuronales artificiales para la detección de plagas en cultivo de jitomate bajo invernadero", *Coloquio de Investigación Multidisciplinaria 2019 Journal CIM*, vol. 7, núm. 1, 2019.

[9] D. Haloui, K. Oufaska, M. Oudani, y K. El Yassini, "Bridging Industry 5.0 and Agriculture 5.0: Historical Perspectives, Opportunities, and Future Perspectives", *Sustainability 2024, Vol. 16, Page 3507*, vol. 16, núm. 9, p. 3507, abr. 2024, doi: 10.3390/SU16093507.

[10] Sergieieva Kateryna, "Tecnología Agrícola". Consultado: el 18 de septiembre de 2024. [En línea]. Disponible en: https://eos.com/es/blog/tecnologias-en-la-agricultura/

[11] V. Martos, A. Ahmad, P. Cartujo, y J. Ordoñez, "Ensuring Agricultural Sustainability through Remote Sensing in the Era of Agriculture 5.0", *Applied Sciences 2021, Vol. 11, Page 5911*, vol. 11, núm. 13, p. 5911, jun. 2021, doi: 10.3390/APP11135911.

[12] AGROZAPIENS, "La Revolución de la Agricultura 5.0: Innovación y Sostenibilidad para el Futuro". Consultado: el 18 de septiembre de 2024. [En línea]. Disponible en: https://agrozapiens.com/la-revolucion-de-la-agricultura-5-0-innovacion-y-sostenibilidad-para-el-futuro/

[13] P. Martínez, S. Rahimi, J. Salazar, J. López, M. Rubio, y P. Martínez, "Principles and prospects of prunus cultivation in greenhouse", el 4 de marzo de 2021, *MDPI AG*. doi: 10.3390/agronomy11030474.

[14] FERTRI invernaderos, "Tipos de plagas en invernaderos ¿Cómo controlar las plagas en invernaderos?" Consultado: el 18 de septiembre de 2024. [En línea]. Disponible en: https://fertri.com/2024/04/tipos-plagas-invernaderos-como-controlar-plagas/

[15] E. S. Jiménez Martínez, *Plagas de Cultivos*, vol. 2. Managua: Universidad Nacional Agraria., 2021.

[16] D. Montero, "Botrytis - Consejos para mi huerto". Consultado: el 26 de septiembre de 2024. [En línea]. Disponible en: https://www.consejosparamihuerto.com/vegetales/botrytis/

[17] Sela Guy, "Identificación de plagas | Cropaia". Consultado: el 19 de septiembre de 2024. [En línea]. Disponible en: https://cropaia.com/es/blog/identificacion-de-plagas/

[18] Intagri S.C., "El Monitoreo Herramienta Basica en Los Programas de MIP y MIE en Hortalizas". Consultado: el 19 de septiembre de 2024. [En línea]. Disponible en: https://www.intagri.com/articulos/fitosanidad/el-monitoreo-herramienta-basica-en-los-programas-mip-mie

[19] A. Abeliuk y C. Gutiérrez, "Historia y evolución de la inteligencia artificial", 2021.

[20] A. Sharma, A. Jain, P. Gupta, y V. Chowdary, "Machine Learning Applications for Precision Agriculture: A Comprehensive Review", *IEEE Access*, vol. 9, pp. 4843–4873, 2021, doi: 10.1109/ACCESS.2020.3048415.

[21] A. Cravero, S. Pardo, S. Sepúlveda, y L. Muñoz, "Challenges to Use Machine Learning in Agricultural Big Data: A Systematic Literature Review", el 1 de marzo de 2022, *MDPI*. doi: 10.3390/agronomy12030748.

[22] J. Díaz-Ramírez, "Aprendizaje Automático y Aprendizaje Profundo", *Ingeniare. Revista chilena de ingeniería*, vol. 29, núm. 2, pp. 180–181, jun. 2021, doi: 10.4067/S0718-33052021000200180.

[23] M. Altalak, M. A. Uddin, A. Alajmi, y A. Rizg, "Smart Agriculture Applications Using Deep Learning Technologies: A Survey", *Applied Sciences (Switzerland)*, vol. 12, núm. 12, jun. 2022, doi: 10.3390/app12125919.

[24] K. Steven y G. Abril, "Deep Learning en agricultura: conceptos y aplicaciones en la identificación de cultivos sobre imágenes satelitales", 2023.

[25] V. Kakani, V. H. Nguyen, B. P. Kumar, H. Kim, y V. R. Pasupuleti, “A critical review on computer vision and artificial intelligence in food industry”, el 1 de diciembre de 2020, *Elsevier B.V.* doi: 10.1016/j.jafr.2020.100033.

[26] A. Gonzales *et al.*, *Técnicas y algoritmos básicos de visión artificial*. 2006.

[27] S. Ghazal, A. Munir, y W. S. Qureshi, “Computer vision in smart agriculture and precision farming: Techniques and applications”, *Artificial Intelligence in Agriculture*, vol. 13, pp. 64–83, jun. 2024, doi: 10.1016/J.AIIA.2024.06.004.

[28] J. I. Rivera Chinchuña, “Diseño e implementación de un sistema de detección de tomates mediante un robot agrícola autónomo”, Universidad Loyola, 2024. Consultado: el 8 de enero de 2025. [En línea]. Disponible en: https://repositorio.uloyola.es/handle/20.500.12412/6136

[29] logitech, “Cámara web HD C270, video 720p con micrófono con reducción de ruido”. Consultado: el 18 de octubre de 2024. [En línea]. Disponible en: https://www.logitech.com/es-roam/products/webcams/c270-hd-webcam.960-000947.html?srsltid=AfmBOop1TaOmZhzvU8O-mbVDO9pm7DxIkAoH8EfR39u5W4ad64q_1FSR

[30] Waveshare, “IMX219-77 Camera”. Consultado: el 4 de enero de 2025. [En línea]. Disponible en: https://www.waveshare.com/wiki/IMX219-77_Camera

[31] Reolink, “RLC-811A | Cámara de seguridad PoE UHD 4K con alarma y focos”. Consultado: el 18 de octubre de 2024. [En línea]. Disponible en: https://reolink.com/es/product/rlc-811a/?srsltid=AfmBOortQlOTD6SiLAgN7cdKRXzk_SwdhW0iCM3n9_BWz74faIIAVK98

[32] Ultralytics, “YOLOv8 - Ultralytics YOLO Documentos”. Consultado: el 19 de octubre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/models/yolov8/

[33] E. Forson, “Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning”. Consultado: el 9 de diciembre de 2024. [En línea]. Disponible en: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

[34] V. Kumar Thakur, “Faster R-CNN: Object Detection”. Consultado: el 19 de octubre de 2024. [En línea]. Disponible en: https://medium.com/thedeephub/faster-r-cnn-object-detection-5dfe77104e31

[35] T.-N. Doan y T.-H. Phan, “A Novel Smart System with Jetson Nano for Remote Insect Monitoring”, *International Journal of Advanced Computer Science and Applications*, vol. 15, núm. 7, p. 2024, jun. 2024, doi: 10.14569/IJACSA.2024.0150798.

[36] Y. Xu *et al.*, “Real-time object detection method of melon leaf diseases under complex background in greenhouse”, *J Real Time Image Process*, vol. 19, núm. 5, pp. 985–995, oct. 2022, doi: 10.1007/s11554-022-01239-7.

[37] R. Ye, Q. Gao, Y. Qian, J. Sun, y T. Li, “Improved YOLOv8 and SAHI Model for the Collaborative Detection of Small Targets at the Micro Scale: A Case Study of Pest Detection in Tea”, *Agronomy 2024, Vol. 14, Page 1034*, vol. 14, núm. 5, p. 1034, may 2024, doi: 10.3390/AGRONOMY14051034.

[38] Y. Zhang, C. Song, y D. Zhang, “Deep Learning-Based Object Detection Improvement for Tomato Disease”, *IEEE Access*, vol. 8, pp. 56607–56614, 2020, doi: 10.1109/ACCESS.2020.2982456.

[39] L. Jiao, S. Dong, S. Zhang, C. Xie, y H. Wang, “AF-RCNN: An anchor-free convolutional neural network for multi-categories agricultural pest detection”, *Comput Electron Agric*, vol. 174, jul. 2020, doi: 10.1016/j.compag.2020.105522.

[40] L. Wang *et al.*, “Transfer Learning-Based Lightweight SSD Model for Detection of Pests in Citrus”, *Agronomy 2023, Vol. 13, Page 1710*, vol. 13, núm. 7, p. 1710, jun. 2023, doi: 10.3390/AGRONOMY13071710.

[41] MySQL, “MySQL 8.4 Reference Manual”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/

[42] MongoDB, “MongoDB Documentation”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://www.mongodb.com/docs/

[43] SQLite, “SQLite Documentation”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://sqlite.org/docs.html

[44] PostgreSQL, “PostgreSQL: Documentation”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://www.postgresql.org/docs/

[45] Flet, “Build multi-platform apps in Python powered by Flutter”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://flet.dev/

[46] Flask, “Welcome to Flask — Flask Documentation (3.1.x)”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://flask.palletsprojects.com/en/stable/

[47] Qt, “Qt for Python”. Consultado: el 14 de diciembre de 2024. [En línea]. Disponible en: https://doc.qt.io/qtforpython-6/

[48] J. Moreno, “Clavel, Dianthus caryophyllus, características, cuidados y cultivo”. Consultado: el 1 de octubre de 2024. [En línea]. Disponible en: https://naturaleza.animalesbiologia.com/plantas/clavel-dianthus-caryophyllus#:~:text=Clavel,%20Dianthus%20caryophyllus,%20conoce%20sus%20caracter%C3%ADsticas,%20c%C3%B3mo%20es%20su%20cultivo,

[49] M. Á. Enríquez, S. E. Poveda, y G. I. Alvarado, “Bioactivos de la hierba luisa utilizados en la industria”, *Rev Mex De Cienc Agric*, vol. 14, núm. 1, pp. 1–11, ene. 2023, doi: 10.29312/REMEXCA.V14I1.3249.

[50] A. y P. Instituto Nacional de Investigaciones Forestales, “Plantas medicinales, tradición ancestral”. Consultado: el 3 de octubre de 2024. [En

línea]. Disponible en: https://www.gob.mx/inifap/articulos/plantas-medicinales-tradicion-ancestral

[51] S. Patterson, “Spearmint Care: Learn How To Grow Spearmint Herbs”. Consultado: el 3 de octubre de 2024. [En línea]. Disponible en: https://www.gardeningknowhow.com/edible/herbs/mint/growing-spearmint-plants.htm

[52] A. Ochoa, “Cedrón: Todo lo que debes saber sobre esta planta medicinal”, Admagazine. Consultado: el 3 de octubre de 2024. [En línea]. Disponible en: https://www.admagazine.com/articulos/cedron-todo-lo-que-debes-saber-sobre-esta-planta-medicinal

[53] D. Sordo, “¿Cómo combatir las plagas y enfermedades del clavel?”, Leroy Merlin. Consultado: el 4 de octubre de 2024. [En línea]. Disponible en: https://www.leroymerlin.es/ideas-y-consejos/consejos/como-combatir-las-plagas-y-enfermedades-del-clavel.html

[54] J. García Cochagne y E. V. Oré Eusebio, “Guía Ilustrada de Plagas en plantas Medicinales”, Lima, dic. 2017.

[55] D. Estrada *et al.*, “Manzanilla, Anthemis nobilis”, Guatemala.

[56] P. Otero, “HIERBABUENA: Plagas y Enfermedades más Importantes”, AgroHuerto. Consultado: el 4 de octubre de 2024. [En línea]. Disponible en: https://www.agrohuerto.com/hierbabuena-plagas-enfermedades-comunes/

[57] Ultralytics, “YOLOv8 - Ultralytics YOLO Documentos”. Consultado: el 17 de diciembre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/models/yolov8/

[58] Ultralytics, “YOLOv5 - Ultralytics YOLO Docs”. Consultado: el 17 de diciembre de 2024. [En línea]. Disponible en: https://docs.ultralytics.com/es/models/yolov5/#overview

[59] A. Timilsina, “YOLOv8 Architecture Explained!” Consultado: el 18 de diciembre de 2024. [En línea]. Disponible en: https://abintimilsina.medium.com/yolov8-architecture-explained-a5e90a560ce5

[60] H. Cobián-López, “Tetranychus urticae, Aphididae and Healty leaf Dataset for Crop Pest Recognition (4185 images)”, *Mendeley Data*, vol. 2, jun. 2023, doi: 10.17632/RBGX99YM6J.2.

[61] J. Parraga-Alava, “LeLePhid: An Images Dataset for Aphids Detection and Infestation Severity on Lemons Leaf”, *Mendeley Data*, vol. 1, jun. 2021, doi: 10.17632/TNDHS2ZNG4.1.

[62] M.-L. Huang y T. C. Chuang, “A database of eight common tomato pest images”, *Mendeley Data*, vol. 1, may 2020, doi: 10.17632/S62ZM6DJD2.1.

[63] J. Tusubira, “Cassava Whitefly Dataset”, *Mendeley Data*, vol. 2, jun. 2022, doi: 10.17632/5G38399Z9P.2.

ANEXOS

Anexo A. Especificaciones técnicas, NVIDIA Jetson Nano

En la Figura A1 se muestra las especificaciones técnicas del microcomputador NVIDIA Jetson Nano que se ocupó para el sistema de detección.



NVIDIA JETSON NANO DEVELOPER KIT
TECHNICAL SPECIFICATIONS

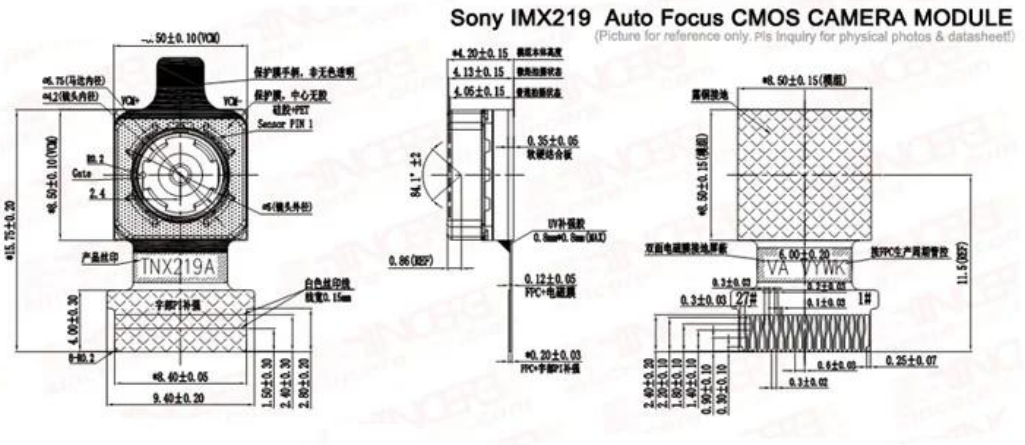
DEVELOPER KIT	
GPU	128-Core Maxwell
CPU	Quad-Core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (Not Included)
Video Encoder	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decoder	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30| (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI 2.0 and eDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I^2C, I^2S, SPI, UART
Mechanical	100 mm x 80 mm x 29 mm

*Please refer to NVIDIA documentation for what is currently supported.

Figura A1. Especificaciones técnicas de la NVIDIA Jetson Nano.

Anexo B. Especificaciones técnicas, cámara IMX219

En la Figura A2 se muestra las especificaciones técnicas de la cámara IMX219



Nombre de marca	Arducam
Dimensiones del producto	1,73 x 0,98 x 1,73 pulgadas
País de origen	China
Número de modelo del producto	B029201
Nombre del color	8MP IMX219, enfoque automático con mini caja de metal
Características especiales	Luz baja

Figura A2. Especificaciones técnicas, cámara IMX219.

Anexo C. Instalación del Sistema

En la Figura A3 se muestra el sistema de detección de plagas instalado en el invernadero.



Figura A3. Instalación del sistema en el invernadero.

Anexo D. Estación de detección.

En la Figura A4 se muestra la estación de detección para elaborar una detección más certera.

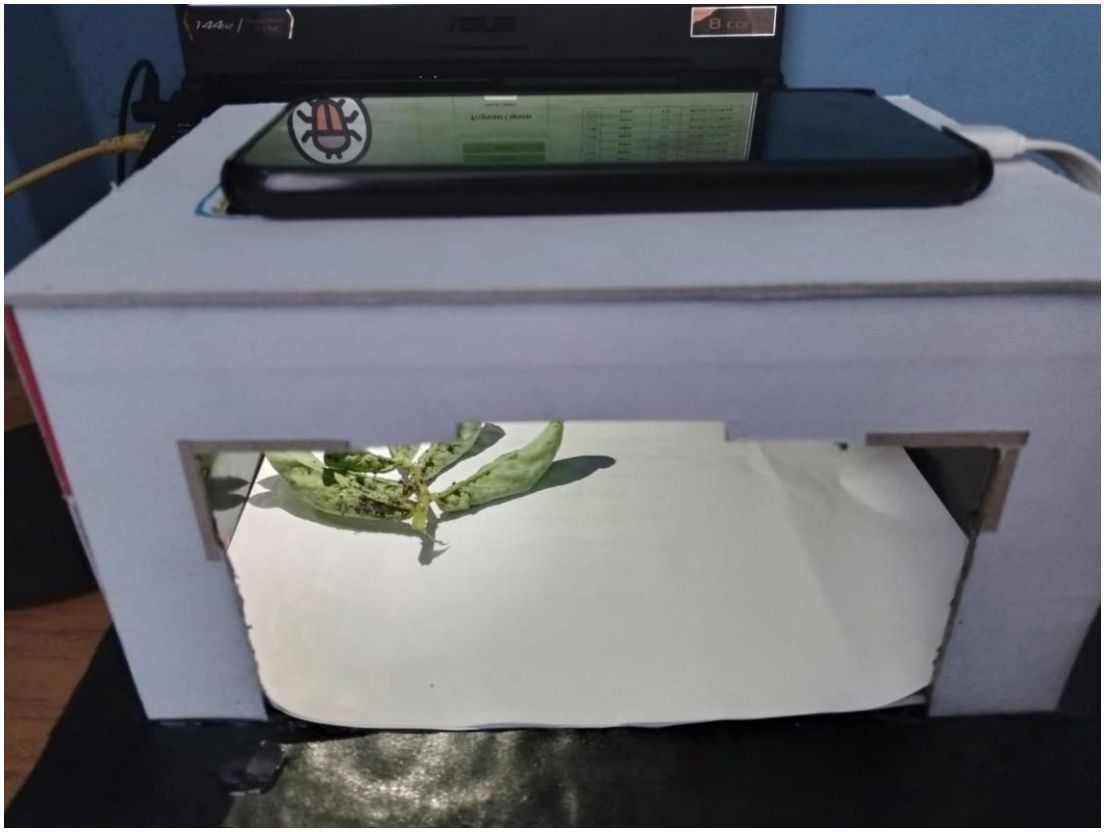


Figura A4. Estación de detección.

```
from flask import Flask, render_template, Response, jsonify, request
import cv2
import mysql.connector
import threading
from ultralytics import YOLO
import schedule
import time
import requests
from pytz import timezone

# Inicializamos Flask
app = Flask(__name__)

# Variables de control
seleccionar_camara = 0  # Cámara por defecto (Cámara 1)
IP_CAMARA_URL = "http://192.168.1.2:8080/video"  # URL para la camara de celular
model = YOLO('best.pt')  # Ruta del modelo YOLOv8
real_time_detection = False
real_time_thread = None
activar_camara = None  # Variable para manejar la cámara activa
camara_lock = threading.Lock()  # Control de acceso a la cámara
camara_horarios = {
    "morning": {},  # Horarios para la mañana
    "afternoon": {}  # Horarios para la tarde
}
almacenar_tareas = {}  # Diccionario para almacenar las tareas programadas
# Ruta donde se guardarán las imágenes
dir_almacenamiento_img = "/var/www/html/capturas/"

# Configuración de Telegram
TELEGRAM_BOT_TOKEN = '8143234351:AAH70A9iKGNBGRkUO2GV3uV0qZesFbZ-WC8'  # Reemplaza con el token de tu bot
TELEGRAM_CHAT_ID = 5495137846  # Reemplaza con tu ID de usuario de Telegram
# Enviar notificaciones a Telegram
def send_telegram_alert(message, image_path=None):
    url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
    data = {'chat_id': TELEGRAM_CHAT_ID, 'text': message}
    # Enviar mensaje de texto
    try:
        response = requests.post(url, data=data)
        print(f"Notificación enviada: {response.json()}")
    except Exception as e:
        print(f"Error al enviar notificación: {e}")
```

```
    # Enviar imagen, si está disponible
    if image_path:
        url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendPhoto"
        try:
            with open(image_path, 'rb') as image_file:
                files = {'photo': image_file}
                data = {'chat_id': TELEGRAM_CHAT_ID}
                response = requests.post(url, data=data, files=files)
                print(f"Imagen enviada: {response.json()}")
        except Exception as e:
            print(f"Error al enviar la imagen: {e}")

# Conectar a la base de datos MySQL
def connect_db():
    try:
        return mysql.connector.connect(
            host="localhost",
            user="root",
            password="erick",
            database="detecciones"
        )
    except mysql.connector.Error as err:
        print(f"Error al conectar a la base de datos: {err}")
        return None

# Función que genera los frames del video con detección
def gen_frames(camera_source):
    global activar_camara
    with camara_lock:
        # Liberar la cámara previamente activa si existe
        if activar_camara and activar_camara.isOpened():
            activar_camara.release()
        # Verificar si `camera_source` es una URL o un índice de cámara local
        if isinstance(camera_source, str) and (camera_source.startswith("http://") or
camera_source.startswith("rtsp://")):
            activar_camara = cv2.VideoCapture(camera_source)
        else:
            activar_camara = cv2.VideoCapture(int(camera_source))  # Suponer que es un
índice local

        if not activar_camara.isOpened():
            print(f"No se puede abrir la cámara {camera_source}")
            return
    while real_time_detection:
        with camara_lock:
            ret, frame = activar_camara.read()
        if not ret:
            print(f"No se pudo leer el frame de la cámara {camera_source}")
```

```
            break

        # Detección de objetos con YOLO
        results = model(frame)
        boxes = results[0].boxes  # Obtener las cajas de detección
        class_ids = boxes.cls.numpy().astype(int)  # IDs de las clases
        confidences = boxes.conf.numpy()  # Confianzas de las detecciones

        # Procesar detecciones
        for i, box in enumerate(boxes.xyxy):
            x1, y1, x2, y2 = map(int, box)  # Esquinas de la caja
            class_id = class_ids[i]
            confidence = confidences[i]

            if confidence > 0.5:  # Filtrar detecciones con baja confianza
                tipo_deteccion = model.names[class_id]  # Nombre de la clase
                save_detection(tipo_deteccion, confidence)

                # Dibujar cajas y etiquetas sobre el frame
                frame = cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
                label = f"{tipo_deteccion} ({confidence:.2f})"
                cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.9, (255, 0, 0), 2)


        ret, jpeg = cv2.imencode('.jpg', frame)
        if ret:
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')

    with camara_lock:
        try:
            if activar_camara:
                activar_camara.release()
        finally:
            activar_camara = None

# Guardar la detección en la base de datos
def save_detection(tipo_deteccion, confianza):
    conn = connect_db()
    if conn:
        try:
            cursor = conn.cursor()
             query = "INSERT INTO detecciones (tipo_deteccion, confianza) VALUES
(%s, %s)"
            cursor.execute(query, (tipo_deteccion, float(confianza)))
            conn.commit()
        except mysql.connector.Error as err:
            print(f"Error al guardar la detección: {err}")
```

```
        finally:
            cursor.close()
            conn.close()

# Recuperar datos de detecciones desde la base de datos
def get_detections():
    conn = connect_db()
    if conn:
        try:
            cursor = conn.cursor()
              query = "SELECT id, tipo_deteccion, confianza, fecha FROM detecciones
ORDER BY fecha DESC LIMIT 10"
            cursor.execute(query)
            detecciones = cursor.fetchall()

            local_tz = timezone('America/Guayaquil')
            results = []

            for d in detecciones:

                local_date = d[3].astimezone(local_tz)  # Convierte la fecha
                results.append({
                    "id": d[0],
                    "tipo_deteccion": d[1],
                    "confianza": d[2],
                    "fecha": local_date.strftime('%Y-%m-%d %H:%M:%S')
                })
            return results
        except mysql.connector.Error as err:
            print(f"Error al recuperar detecciones: {err}")
            return []
        finally:
            cursor.close()
            conn.close()
    return []

# Función para iniciar la detección en tiempo real
def start_real_time_detection(camera_id):
    global real_time_detection
    real_time_detection = True
    return gen_frames(camera_id)  # Retorna el generador de frames

# Ruta para iniciar la detección
@app.route('/start_detection', methods=['POST'])
def start_detection():
    global real_time_thread
    camera_source = request.json.get("camera_source", 0)  # Aceptar índices o URLs
    if real_time_thread is None or not real_time_thread.is_alive():
```

```
            real_time_thread = threading.Thread(target=start_real_time_detection, args=(camera_source,))
        real_time_thread.daemon = True
        real_time_thread.start()
    return jsonify({"status": "detection_started"})

# Ruta para detener la detección
@app.route('/stop_detection', methods=['POST'])
def stop_detection():
    global real_time_detection
    real_time_detection = False
    return jsonify({"status": "detection_stopped"})

# Ruta para transmitir video
@app.route('/video')
def video():
    camera_source = request.args.get('camera', default=0, type=str)
        return Response(gen_frames(camera_source), mimetype='multipart/x-mixed-replace; boundary=frame')

# API para obtener detecciones en formato JSON
@app.route('/detecciones', methods=['GET'])
def detecciones():
    detecciones = get_detections()
    return jsonify(detecciones)

# Función para procesar imágenes capturadas y detectar plagas
def process_captured_image(filename, camera_id):
    frame = cv2.imread(f"{dir_almacenamiento_img}{filename}")
    results = model(frame)
    boxes = results[0].boxes
    class_ids = boxes.cls.numpy().astype(int)
    confidences = boxes.conf.numpy()

    detecciones = []
    for i, box in enumerate(boxes.xyxy):
        x1, y1, x2, y2 = map(int, box)
        class_id = class_ids[i]
        confidence = confidences[i]

        if confidence > 0.5:
            tipo_deteccion = model.names[class_id]
            detecciones.append({
                "tipo_deteccion": tipo_deteccion,
                "confianza": confidence,
                "camera_id": camera_id,
                "imagen": filename
            })
    print(f"Detecciones procesadas: {detecciones}")
```

```
    for det in detecciones:
        save_capture_detection(det["tipo_deteccion"], det["confianza"], camera_id, filename)

    return detecciones

# Guardar detecciones específicas de capturas programadas
def save_capture_detection(tipo_deteccion, confianza, camera_id, filename):
    relative_path = f"/capturas/{filename}"
    print(f"Intentando guardar en la base de datos: {tipo_deteccion}, {confianza}, {camera_id}, {relative_path}")
    conn = connect_db()
    if conn:
        try:
            cursor = conn.cursor()
            query = """
                INSERT INTO capturas (tipo_deteccion, confianza, camera_id, imagen, fecha)
                VALUES (%s, %s, %s, %s, NOW())
            """
            print(f"Consulta SQL: {query}")
            print(f"Valores: {tipo_deteccion}, {confianza}, {camera_id}, {relative_path}")
            cursor.execute(query, (tipo_deteccion, float(confianza), camera_id, relative_path))
            conn.commit()
            print("Detección guardada correctamente en la base de datos.")
        except mysql.connector.Error as err:
            print(f"Error al guardar en la base de datos: {err}")
        finally:
            cursor.close()
            conn.close()
    else:
        print("No se pudo conectar a la base de datos.")

#Funcion para realizar la captura de imagenes de la camara
def capture_image(camera_id):
    cap = cv2.VideoCapture(camera_id)
    try:
        if cap.isOpened():
            ret, frame = cap.read()
            if ret:
                # Nombre del archivo y ruta completa
                filename = f"captura_{camera_id}_{time.strftime('%Y%m%d_%H%M%S')}.jpg"
                full_path = f"{dir_almacenamiento_img}{filename}"

                # Guardar la imagen
                cv2.imwrite(full_path, frame)
                print(f"Imagen capturada y guardada: {full_path}")
```

```
            # Procesar la imagen capturada
            detecciones = process_captured_image(filename, camera_id)
            if detecciones:
                print(f"Se detectaron plagas en la captura: {detecciones}")

                #Enviar alerta de detección a telegram
                message = f"⚠ Alerta de detección de plagas ⚠\n"
                for det in detecciones:
                                message += f"Tipo: {det['tipo_deteccion']}, Confianza: {det['confianza']:.2f}\n"

                #enviar mensaje y la imagen a telegram
                send_telegram_alert(message,full_path)

            else:
                print("No se detectaron plagas en la captura.")
    finally:
        cap.release()

#Obtener las ultimas detecciones de la base de datos para mostrar en la tabla de la interfaz
@app.route('/capturas', methods=['GET'])
def capturas():
    conn = connect_db()
    if conn:
        try:
            cursor = conn.cursor()
            query = """
                SELECT id, tipo_deteccion, confianza, camera_id, imagen, fecha
                FROM capturas
                ORDER BY fecha DESC
                LIMIT 10
            """
            cursor.execute(query)
            capturas = cursor.fetchall()

            local_tz = timezone('America/Guayaquil')
            results = []

            for c in capturas:
                # Convierte la fecha al huso horario local
                local_date = c[5].astimezone(local_tz) if c[5] else None
                results.append({
                    "id": c[0],
                    "tipo_deteccion": c[1],
                    "confianza": c[2],
                    "camera_id": c[3],
                    "imagen": c[4],
```

```
                "fecha": local_date.strftime('%Y-%m-%d %H:%M:%S') if local_date else
None
            })

        return jsonify(results)
    except mysql.connector.Error as err:
        print(f"Error al recuperar capturas: {err}")
        return jsonify([])
    except Exception as ex:
        print(f"Error general: {ex}")
        return jsonify([])
    finally:
        cursor.close()
        conn.close()
  else:
    print("No se pudo conectar a la base de datos.")
    return jsonify([])

#agregar horarios para realizar las capturas de las camras
@app.route('/add_schedule', methods=['POST'])
def add_schedule():
  global camara_horarios, almacenar_tareas
  data = request.json
  time = data.get('time')
  camera_id = data.get('camera_id')  # Este debe ser un string
  period = data.get('period')  # "morning" o "afternoon"

  if time and camera_id is not None and period in ["morning", "afternoon"]:
    camera_id = str(camera_id)
    if camera_id not in camara_horarios[period]:
      camara_horarios[period][camera_id] = []

    if time not in camara_horarios[period][camera_id]:
      camara_horarios[period][camera_id].append(time)
                                        camara_horarios[period][camera_id]      =
sorted(camara_horarios[period][camera_id])

      # Programar la captura
                          job    =    schedule.every().day.at(time).do(capture_image,
camera_id=int(camera_id))
          almacenar_tareas[f"{period}_{camera_id}_{time}"] = job  # Guardar la
referencia a la tarea

        return jsonify({"message": f"Hora añadida para cámara {camera_id} en la
{period}"}), 200

  return jsonify({"message": "Hora ya existe o inválida"}), 400

#Eliminar horarios que ya fueron programados
```

```
@app.route('/delete_schedule', methods=['POST'])
def delete_schedule():
    global camara_horarios, almacenar_tareas
    data = request.json
    time = data.get('time')
    camera_id = data.get('camera_id')  # Este debe ser un string
    period = data.get('period')  # "morning" o "afternoon"

        if camera_id and period in camara_horarios and camera_id in
camara_horarios[period]:
        if time in camara_horarios[period][camera_id]:
            # Eliminar el horario de camera_schedules
            camara_horarios[period][camera_id].remove(time)
            if not camara_horarios[period][camera_id]:
                del camara_horarios[period][camera_id]

            # Cancelar la tarea programada
            job_key = f"{period}_{camera_id}_{time}"
            if job_key in almacenar_tareas:
                schedule.cancel_job(almacenar_tareas[job_key])  # Cancela la tarea
                del almacenar_tareas[job_key]  # Elimina la referencia de scheduled_jobs

            return jsonify({"message": f"Hora {time} eliminada para cámara {camera_id}
en la {period}"}), 200

    return jsonify({"message": "Hora no encontrada o inválida"}), 400

#consultar los horarios que fueron programados para las camaras en la interfaz.
@app.route('/get_schedules', methods=['GET'])
def get_schedules():
    global camara_horarios
    return jsonify(camara_horarios), 200

#Controlar el zoom de la camara
@app.route('/control_zoom', methods=['POST'])
def control_zoom():
    data = request.json
    zoom_level = data.get('zoom_level')  # Nivel de zoom enviado desde el frontend

    if not zoom_level:
        return jsonify({"error": "Falta el parámetro 'zoom_level'"}), 400

    try:
        # Enviar comando HTTP a la cámara para ajustar el zoom
        url = f"{IP_CAMARA_URL.replace('/video', '')}/ptz?zoom={zoom_level}"
        response = requests.get(url)
        if response.status_code == 200:
            return jsonify({"message": f"Zoom ajustado a {zoom_level}"}), 200
        else:
```

```
            return jsonify({"error": f"Error al ajustar el zoom: {response.status_code}"}),
500
    except Exception as e:
        return jsonify({"error": str(e)}), 500

#Controlar el encendido de la linterna
@app.route('/control_light', methods=['POST'])
def control_light():
    data = request.json
    light_state = data.get('light_state')  # Puede ser 'on' o 'off'

    # Si no se recibe la URL de la cámara desde el frontend, usar IP_CAMERA_URL
    camera_url = data.get('camera_url', IP_CAMARA_URL)

    if not camera_url:
        return jsonify({"error": "URL de la cámara no proporcionada"}), 400

    # Define las rutas para encender y apagar la luz
    if light_state == 'on':
        url = f"{camera_url.replace('/video', '')}/enabletorch"
    elif light_state == 'off':
        url = f"{camera_url.replace('/video', '')}/disabletorch"
    else:
        return jsonify({"error": "Estado de luz inválido"}), 400

    try:
        print(f"Enviando solicitud a {url}")  # Depuración
        response = requests.get(url)
        if response.status_code == 200:
            return jsonify({"message": f"Luz {'encendida' if light_state == 'on' else
'apagada'} correctamente"})
        else:
            return jsonify({"error": f"Error al controlar la luz: {response.status_code}"}),
500
    except Exception as e:
        print(f"Error al enviar solicitud: {str(e)}")  # Depuración
        return jsonify({"error": str(e)}), 500

# Ejecutar la programación en un hilo
def run_schedule():
    while True:
        schedule.run_pending()
        time.sleep(1)

# Ejecutar la programación en segundo plano
threading.Thread(target=run_schedule, daemon=True).start()

# Página principal con las detecciones
@app.route('/')
```

```
def index():
    return render_template('final.html')

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=5000)
```