



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLÁHUAC

“LA ESENCIA DE LA GRANDEZA RADICA EN LAS RAICES”

TITULACIÓN INTEGRAL POR:

INFORME TÉCNICO POR RESIDENCIA PROFESIONAL

NOMBRE DE LA EMPRESA:

INFOTEC Centro de Investigación e Innovación en Tecnologías de la
Información y Comunicación

TÍTULO DEL PROYECTO:

Diseño e implementación de un contenedor de basura inteligente basado en
Inteligencia Artificial para INFOTEC Centro de Investigación e Innovación en
Tecnologías de la Información y Comunicación.

PARA OBTENER EL TÍTULO DE:

INGENIERO EN SISTEMAS COMPUTACIONALES

P R E S E N T A: (N)

Alvarado Márquez Francisco Javier
Mejía Gaucín Audrey Melissa
Pérez Juárez Carlos Iván
Soto Chávez José Luis

ASESORES:

Ing. Fabiola Rueda Segunda



Tláhuac, Cd. Mx. Noviembre - 2025

ANEXO XXII

ANEXO III

Agradecimientos

Queremos dedicar un agradecimiento muy especial a nuestros padres, quienes han sido pilares fundamentales a lo largo de todo este proceso académico y personal. Este trabajo no habría sido posible sin su constante apoyo, paciencia y compromiso incondicional con nuestro crecimiento.

Desde nuestros primeros años de formación, ellos han estado presentes guiándonos con firmeza, brindándonos no solo los recursos materiales necesarios, sino también el aliento emocional y la motivación para seguir adelante incluso en los momentos más exigentes. Gracias a su ejemplo de perseverancia, responsabilidad y trabajo honesto, hemos aprendido a enfrentar los desafíos con determinación y a valorar el esfuerzo como camino hacia el logro.

Durante los años que hemos transitado en nuestra formación profesional, nuestros padres nos ofrecieron un acompañamiento silencioso, pero siempre presente, sosteniéndonos en cada etapa, celebrando nuestros logros y dándonos fuerza ante las dificultades. En muchas ocasiones fueron ellos quienes creyeron en nosotros incluso cuando nosotros mismos dudábamos, y esa fe es, sin duda, una de las razones por las que hoy podemos presentar con orgullo este proyecto.

Esta tesis representa el cierre de una etapa que no hemos recorrido solos. Es también el resultado del amor, la entrega y el sacrificio de quienes han estado detrás de nosotros cada día, impulsándonos a ser mejores personas y profesionistas. Por todo esto, y por tanto más, les dedicamos con todo nuestro cariño y respeto este logro académico, que es tan suyo como nuestro.

Resumen

En México, la adecuada gestión y clasificación de desechos supone un reto importante, el cual tiene un impacto directo en la eficacia del reciclaje y en el medio ambiente. Hoy en día, la mayoría de los residuos sólidos urbanos no son separados correctamente, lo que reduce su uso y hace que haya más contaminación en los rellenos sanitarios. En respuesta a esta problemática, se hace necesario desarrollar soluciones tecnológicas que ayuden a gestionar los desechos de una manera más eficaz y sostenible.

El propósito de este proyecto es crear un sistema automatizado que, a través de la inteligencia artificial, logre distinguir y clasificar desechos orgánicos e inorgánicos, aumentando así la rapidez y exactitud en esta labor diaria. Con este fin, se creó y puso en marcha un prototipo que emplea microcontroladores, al cual se le añadió un sensor ultrasónico para identificar objetos y un servomotor para el aislamiento físico de los residuos. Además, se incorporó una pantalla que muestra el historial de clasificaciones en tiempo real, lo que proporciona una interfaz informativa y útil.

El proyecto fue llevado a cabo en INFOTEC, que es el Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación. Esta es una entidad pública del Gobierno Federal Mexicano, cuyo propósito es promover la transformación digital de la nación a través de la investigación, la innovación, el desarrollo de soluciones tecnológicas y la formación académica. En este marco, el trabajo actual se sitúa en las esferas de aplicación de las TIC con miras a la sustentabilidad urbana y al bienestar medioambiental.

SCRUM, un marco ágil que posibilitó la organización del trabajo en sprints (fases iterativas y controladas de desarrollo), fue el método utilizado para desarrollar el sistema. Esto promovió la colaboración, la adaptación a las modificaciones y la entrega constante de resultados útiles. Esta metodología posibilitó que la creación del software de clasificación y el ensamblaje del hardware se unieran eficazmente, asegurando un proceso flexible y estructurado.

Los resultados iniciales indican que el sistema obtiene una clasificación exacta de los desechos comunes, conservando un adecuado ritmo de comunicación y procesamiento entre sus elementos. Asimismo, se posibilita una supervisión nítida y sencilla de las operaciones que se llevan a cabo gracias a la interfaz visual implementada. En general, el proyecto ayuda a promover prácticas tecnológicas sostenibles y a desarrollar soluciones novedosas en el área de gestión inteligente de desechos en México.

Palabras clave: Gestión de residuos, clasificación automatizada, inteligencia artificial, aprendizaje profundo, microcontroladores, ESP32, sensor ultrasónico, servomotor, pantalla TFT, SCRUM, TIC

Abstract

The proper management and classification of waste represent a significant challenge in Mexico, directly affecting recycling efficiency and generating considerable environmental impact. Currently, most urban solid waste is not properly separated, which limits its reuse and increases pollution in landfills. In response to this issue, there is a need to develop technological solutions that contribute to a more efficient and sustainable waste management process.

This project aims to develop an automated system capable of identifying and classifying organic and inorganic waste through artificial intelligence, optimizing accuracy and speed in this daily task. To achieve this, a prototype based on microcontrollers was designed and implemented, complemented by an ultrasonic sensor for object detection and a servomotor responsible for the physical separation of waste. Likewise, a display was integrated to show the classification history in real time, providing an informative and functional interface.

The project was developed at INFOTEC, the Research and Innovation Center in Information and Communication Technologies, a public institution of the Federal Government of Mexico that promotes the country's digital transformation through research, innovation, academic training, and the development of technological solutions. In this context, the present work is framed within the lines of application of ICT for the benefit of the environment and urban sustainability.

The methodology applied for the system development was SCRUM, an agile framework that allowed organizing the work into sprints (iterative and controlled development stages), promoting collaboration, adaptation to changes, and continuous delivery of functional results. This methodology made it possible to effectively integrate the development of the classification software with the assembly of the hardware, ensuring a structured and flexible process.

Preliminary results show that the system achieves accurate classification of common waste (paper, plastic, metal, and organic), maintaining good processing speed and communication between components. In addition, the implemented visual interface allows clear and simple monitoring of the operations performed. Overall, the project contributes to the promotion of sustainable technological practices and the creation of innovative solutions in the field of smart waste management in Mexico.

Keywords: Waste management, automated classification, artificial intelligence, deep learning, microcontrollers, ESP32, ultrasonic sensor, servomotor, TFT display, SCRUM, ICT, sustainability, environment, INFOTEC, smart recycling, computer vision, prototype, embedded system.

Contenido

Agradecimientos	I
Resumen	II
Abstract	III
Contenido	IV
Índice de Figuras	VI
Índice de Tablas	VII
Índice de acrónimos	VIII
Introducción	1
Generalidades	2
2.1 Planteamiento del problema.....	2
2.2 Objetivos	3
2.2.1 Objetivo general	3
2.2.2 Objetivos específicos	3
2.3 Justificación.....	4
2.4 Caracterización de la empresa en la que participo.....	5
2.4.1 Datos generales de la empresa	5
2.4.2 Descripción del departamento o área de trabajo.....	7
2.5 Problemas a resolver	7
2.6 Alcances y limitaciones.....	8
2.6.1 Alcances.....	8
2.6.2 Limitaciones.....	9
Marco teórico.....	10
3.1 Investigaciones previas.....	10
3.2 Bases teóricas	11
3.2.1 Sistemas de Clasificación de Residuos Basados en Inteligencia Artificial.....	11
3.2.2 Redes Neuronales Convolucionales (CNN)	12
3.2.3 Transfer Learning	12
3.2.4 Microcontroladores ESP32 y ESP32-CAM	12
3.2.5 Aplicaciones en la Gestión de Residuos.....	13
3.4 Antecedentes	15
3.4.1 Contexto General del Problema.....	15
3.4.2 Evolución de la Clasificación de Residuos.....	16
3.4.3 Problemática de Clasificación de Residuos	17
3.4.4 Avances Tecnológicos en Clasificación y Automatización de Residuos	18
3.4.5 Proyectos y Aplicaciones Específicas	19
3.5 Relevancia del Proyecto.....	19
3.5.1 Importancia del proyecto en el contexto actual	20
3.5.2 Por qué se debería invertir en este proyecto.....	20
3.5.3 Metodología SCRUM aplicada al desarrollo	21

3.5.5 Valor social, tecnológico y ambiental.....	22
Desarrollo	23
4.1 Entrevistas y Análisis de Requerimientos.....	24
4.1.1 Aspectos generales del código.....	26
4.1.2 Hardware a utilizar.....	27
4.2 Planificación y Ejecución de la Metodología Scrum.....	30
4.2.1 Construcción del servidor Flask.	31
4.2.2 Implementación del sistema mecánico.	32
4.2.3 Integración del ESP32 con el backend mediante un endpoint.	34
4.2.4 Elección del dataset y entrenamiento del modelo de IA	36
4.3 Diseño de la Arquitectura del Sistema.....	38
4.3.1 Módulo C++: Esp32ToPythonOledHttpCAM.ino.....	38
4.3.2 Módulo Python: appHTTP.py.....	54
4.3.3 Módulo Python: pruebaModelosTF.py.....	61
4.3.4 Módulo Python: ManejadorHTTP.py.....	65
4.4 Diseño del Modelo de Inteligencia Artificial	72
4.4.1 Elección de arquitectura de inteligencia artificial.....	72
4.4.2 Selección y Evolución del Dataset.....	73
4.4.3 Estrategia y Procesamiento del Conjunto de Datos.....	74
4.4.4 Scripts de procesamiento.....	77
4.4.5 Scripts de entrenamiento.....	80
4.5 Ensamblaje e integración del sistema embebido.....	91
4.5.1 Conexión eléctrica y lógica del sistema	91
4.5.2 Distribución física de los componentes	93
4.5.3 Render en 3D del producto final para producción	97
Pruebas y resultados	102
5.1 Pruebas Realizadas.....	103
5.1.1 Pruebas Unitarias por Módulo: Hardware	103
5.1.2 Pruebas Unitarias por Módulo: Software.....	104
5.1.3 Pruebas de Integración	¡Error! Marcador no definido.
5.2 Evaluación de resultados.....	105
5.2.1 Prueba Hardware	105
5.2.2 Pruebas del Servidor y Arquitectura Backend.....	109
5.2.3 Pruebas de Rutas (API REST).....	110
5.2.4 Prueba Modelo de Inteligencia Artificial	113
5.3 Resultados finales.....	122
Glosario	123
Conclusiones	126
Referencias	127

Índice de Figuras

Ilustración 1 Croquis de INFOTEC.....	6
Ilustración 2 Objetivos de desarrollo sostenible	16
Ilustración 3 Servomotor mg90s	27
Ilustración 4 Sensor ultrasónico HC-SR04	28
Ilustración 5 ESP32-S3 N16R8	28
Ilustración 6 Pantalla oled 128x64 i2c	29
Ilustración 7 Cámara esp32 OV2640	30
Ilustración 8 Diagrama de clases esp32	42
Ilustración 9 Diagrama secuencia de esp32 y servidor	50
Ilustración 10 Diagrama de flujo del loop esp32.....	51
Ilustración 11 Diagrama secuencia Long Polling.....	58
Ilustración 12 Diagrama de clases python.....	60
Ilustración 13 Diagrama de flujo AcomodoDataset.....	78
Ilustración 14 Diagrama de flujo AcomodoDatasetTestTrain	79
Ilustración 15 Código entrenamiento RestNet-50	85
Ilustración 16 Código entrenamiento MobileNetV2.....	87
Ilustración 17 Consola de entrenamiento FT	90
Ilustración 18 Diagrama de electrónico.....	92
Ilustración 19 ESP32 S3 Cam ubicación	93
Ilustración 20 Iluminación prototipo.....	94
Ilustración 21 Cableado y protoboard prototipo.....	95
Ilustración 22 Sensor ultrasónico frente al prototipo.....	96
Ilustración 23 Pantalla Oled en el prototipo	96
Ilustración 24 Servomotor en el prototipo y conexión a la tapa	97
Ilustración 25 Modelo 3D construcción inicial.....	98
Ilustración 26 Modelado en tres posiciones	99
Ilustración 27 Modelo 3D finalizado versión manufacturable.....	100
Ilustración 28 Visión interna del modelo 3D.....	101
Ilustración 29 Prueba servomotor.....	105
Ilustración 30 Prueba sensor	106
Ilustración 31 Prueba pantalla oled	107
Ilustración 32 Prueba cámara	108
Ilustración 33 Matriz de confusión de ResNet50.....	114
Ilustración 34 Prueba real ResNet50.....	114
Ilustración 35 Curva ROC de ResNet50.....	115
Ilustración 36 Matriz de confusión de MobileNetV2 primera iteración.	116
Ilustración 37 Prueba real MobileNetV2 primera iteración.	116
Ilustración 38 Curva Roc MobileNetV2 primera iteración.	117
Ilustración 39 Matriz de confusión de MobileNetV2 segunda iteración.	118
Ilustración 40 Prueba real MobileNetV2 segunda iteración.	118
Ilustración 41 Curva Roc MobileNetV2 segunda iteración.	119
Ilustración 42 Matriz de confusión de MobileNetV2 tercera iteración.	120
Ilustración 43 Prueba real MobileNetV2 tercera iteración.	120
Ilustración 44 Curva Roc MobileNetV2 tercera iteración.....	121

Índice de Tablas

Tabla 1 Nombre de instancias	26
Tabla 2 Construcción del Servidor Flask	32
Tabla 3 Implementación del sistema mecánico	34
Tabla 4 Integración del ESP32 con el backend mediante un endpoint.	35
Tabla 5 Elección del dataset y entrenamiento del modelo de IA.	38
Tabla 6 Definición de variables iniciales ESP32	40
Tabla 7 Atributos del setup del microcontrolador	41
Tabla 8 Funcionalidad de Esp32ToPythonOledHttpCAM.....	54
Tabla 9 Definición de variables iniciales appHTTP.....	55
Tabla 10 Resumen funcionalidad main	60
Tabla 11 Atributos del constructo de “pruebaModelosIATF”	62
Tabla 12 Funcionalidad de pruebaModelosTF	64
Tabla 13 Atributos del constructo de “manejadorHTTP”	66
Tabla 14 Significado códigos http.....	67
Tabla 15 Funcionalidad: ManejadorHTTP.....	72
Tabla 16 Tabla de Conexiones	92
Tabla 17 Resultados servomotor	105
Tabla 18 Resultados sensor	106
Tabla 19 Resultados pantalla oled	107
Tabla 20 Resultados cámara	108
Tabla 21 Métricas de los modelos de IA.....	113

Índice de acrónimos

Acrónimo	Significado
3D	Tres Dimensiones. Representación de objetos o modelos con ancho, alto y profundidad.
IA	Inteligencia Artificial Rama de la informática que desarrolla sistemas capaces de realizar tareas que normalmente requieren de la inteligencia humana, como el reconocimiento de patrones, la toma de decisiones o el aprendizaje a partir de datos.
CNN	Red Neuronal Convolutacional. Tipo de red neuronal profunda especializada en procesar datos en forma de imágenes, extrayendo características visuales como bordes, texturas y formas para tareas de clasificación o detección.
CONACYT	Consejo Nacional de Ciencia y Tecnología Organismo público de México encargado de promover y financiar actividades relacionadas con la investigación científica, el desarrollo tecnológico y la innovación.
IoT	Internet de las Cosas. Paradigma tecnológico que conecta objetos físicos cotidianos a internet, permitiendo la recolección y el intercambio de datos en tiempo real para mejorar procesos y servicios.
OpenCV	Open Source Computer Vision Library Librería de visión por computadora de código abierto ampliamente utilizada para el procesamiento de imágenes y videos.
SEDEMA	Secretaría del Medio Ambiente de la Ciudad de México Institución local encargada de diseñar e implementar políticas ambientales en la Ciudad de México, incluyendo la gestión de residuos sólidos y la mitigación de la contaminación.
SEMARNAT	Secretaría de Medio Ambiente y Recursos Naturales Institución local encargada de diseñar e implementar políticas ambientales en la Ciudad de México, incluyendo la gestión de residuos sólidos y la mitigación de la contaminación.
TFT	Thin Film Transistor Tecnología de pantalla basada en transistores de película delgada, utilizada en monitores y dispositivos portátiles como la pantalla ILI9341, que ofrece buena resolución y bajo consumo.
TIC	Tecnologías de la Información y la Comunicación Conjunto de recursos, herramientas y sistemas tecnológicos que permiten gestionar, procesar, almacenar y transmitir información, como computadoras, redes e internet.

Capítulo 1

Introducción

La creciente generación de residuos sólidos urbanos representa uno de los retos ambientales más urgentes a nivel mundial. El manejo inadecuado de la basura no solo contribuye a la contaminación del suelo, agua y aire, sino que también dificulta los procesos de reciclaje y aprovechamiento de materiales. En la actualidad, gran parte de la separación de residuos depende del conocimiento y voluntad de las personas, lo que provoca errores frecuentes en la clasificación y reduce la eficiencia del tratamiento posterior.

Ante esta problemática, la tecnología y, en particular, la IA, se ha convertido en una herramienta clave para optimizar y automatizar procesos que antes dependían exclusivamente de la intervención humana. El presente proyecto propone el desarrollo de un sistema capaz de identificar y clasificar distintos tipos de basura mediante el uso de visión por computadora e IA, integrando un entorno de hardware y software que permita un funcionamiento rápido, eficiente y confiable.

Este trabajo se enfoca en el desarrollo de un prototipo que une un modelo clasificador de imágenes ya entrenado, un microcontrolador para la recolección y procesamiento de datos, así como una interfaz visual que exhiba los resultados al instante. La meta es establecer las bases para soluciones escalables que sean aplicables en ambientes urbanos, industriales y domésticos; de este modo, se podrá avanzar en la gestión de residuos y promover prácticas sustentables.

Capítulo 2

Generalidades

2.1 Planteamiento del problema

La producción de residuos sólidos es un problema fundamental tanto ambiental como social a escala global. La ausencia de una cultura de separación, el aumento demográfico y la alta demanda de productos han llevado a un aumento considerable en el número de desechos que terminan en basurales a cielo abierto y rellenos sanitarios. De acuerdo con entidades internacionales, si estos desechos se clasificaran adecuadamente desde su origen, una gran parte de ellos podrían reciclarse o reutilizarse.

La separación de basura es una tarea que muchos usuarios o trabajadores del sector todavía realizan a mano en varias comunidades, lo que trae consigo múltiples problemas: clasificación errónea por parte de humanos, exposición a materiales peligrosos, lentitud en los procesos y escasa motivación para ejecutar esta práctica correctamente. Esto tiene un efecto inmediato en la efectividad de los sistemas de reciclaje y recolección, así como en la sostenibilidad del medio ambiente.

En este contexto, se hace necesario idear e implementar soluciones tecnológicas novedosas que posibiliten la optimización y automatización del proceso de clasificación de residuos. Utilizar dispositivos de bajo costo y la inteligencia artificial es una opción factible para crear sistemas que puedan clasificar y detectar basura en tiempo real.

2.2 Objetivos

2.2.1 Objetivo general

Desarrollar y diseñar un software y hardware que permita identificar y clasificar distintos tipos de residuos mediante inteligencia artificial, para optimizar la gestión de basura, reducir errores de separación y contribuir a la sostenibilidad ambiental en INFOTEC Centro de investigación e innovación en tecnologías de la información y comunicación.

2.2.2 Objetivos específicos

Análisis

- Investigar las herramientas y tecnologías disponibles para desarrollar un sistema eficiente de clasificación de basura, asegurando una base técnica sólida que permita un diseño confiable.
- Evaluar diferentes opciones de materiales y métodos para seleccionar la alternativa más efectiva en términos de calidad, costo y funcionamiento, optimizando los recursos del proyecto.

Planificación

- Evaluar diferentes opciones de materiales y métodos para seleccionar la alternativa más efectiva en términos de calidad, costo y funcionamiento, optimizando los recursos del proyecto.
- Establecer prioridades en la construcción del prototipo, la implementación del software y la integración del hardware para asegurar una ejecución eficiente.
- Definir un plan de trabajo estructurado mediante la metodología ágil SCRUM, estableciendo sprints de desarrollo que permitan avanzar de forma iterativa y controlada en las distintas etapas del proyecto.

Diseño

- Diseñar el modelo 3D del sistema para definir su estructura y funcionamiento, optimizando la disposición del hardware y la lógica de operación para un rendimiento eficiente.

- Definir la arquitectura del software y su integración con el hardware, asegurando la compatibilidad entre componentes para un funcionamiento estable y escalable.

Desarrollo

- Ensamblar los componentes electrónicos necesarios para el funcionamiento del sistema de clasificación de residuos, garantizando robustez y operatividad.
- Implementar el software de reconocimiento de imágenes con inteligencia artificial y programar el microcontrolador con el objetivo de lograr una clasificación automática y precisa.

Pruebas

- Realizar pruebas de funcionamiento del sistema para verificar la precisión del modelo de IA y la respuesta del hardware ante los tipos de residuos, validando la efectividad de la solución.
- Identificar los desafíos y barreras en la implementación del sistema para proponer soluciones que optimicen su desempeño y aseguren su viabilidad en escenarios reales.

2.3 Justificación

Durante nuestra formación en la carrera de Ingeniería en Sistemas Computacionales, adquirimos las bases teóricas y prácticas necesarias. Estas capacidades nos han permitido identificar y abordar una problemática crítica: la ineficiente gestión de residuos, causada por la dificultad de clasificar la basura de manera correcta. Los métodos de separación actuales son manuales, lo que los hace imprecisos y lentos, limitando la recuperación de materiales valiosos. Esta situación es alarmante, ya que solo el 5% de los residuos se separa adecuadamente en la fuente. Por lo tanto, el desarrollo de un sistema automatizado para la clasificación de residuos se vuelve fundamental. Este proyecto se justifica por la necesidad de optimizar los procesos de reciclaje, fomentar prácticas más sostenibles y mejorar la gestión de desechos, aprovechando los conocimientos adquiridos y los recursos proporcionados por INFOTEC Centro de investigación e innovación en tecnologías de la información y comunicación, para ofrecer una solución tecnológica a este desafío ambiental y social.

2.4 Caracterización de la empresa en la que participo

La residencia profesional se llevará a cabo en las instalaciones de INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, específicamente en la planta baja dentro del área de la biblioteca. Este espacio cuenta con recursos bibliográficos especializados en inteligencia artificial y programación general, brindando un entorno adecuado para el desarrollo del proyecto.

2.4.1 Datos generales de la empresa

Nombre: INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación

Domicilio: Av. San Fernando 37, Toriello Guerra, Tlalpan, 14050 Ciudad de México.

Teléfono: 55 5624 2800.

Titular de la empresa: Dr. Juan Mario Beltrán Valle (Director Adjunto de Administración)

Actividades que realiza:

INFOTEC Centro de investigación e innovación en tecnologías de la información y comunicación combina investigación científica, desarrollo de soluciones TIC y formación profesional especializada y presta servicios de infraestructura, IoT, desarrollo de software y educación continua (maestrías y doctorados en TIC avalados por CONACYT)

Misión

Contribuir a la transformación digital de México, a través de la investigación, la innovación, la formación académica y el desarrollo de productos y servicios TIC.

Visión

Ser un Centro Público de Investigación que habilite los caminos que conduzcan hacia un México Moderno y de Inclusión Digital, a través de su presencia en los sectores público y privado.

Valores

- Colaboración: Es la capacidad que tenemos de sumar nuestro talento dentro y fuera de la organización, y una excelente forma de crear añadir valor.

- Innovación: Como valor de nuestra organización es la capacidad de idear, diseñar, experimentar y construir nuevas prácticas, procesos, productos y estructuras capaces de crear valor e implantar con éxito soluciones a retos.
- Transparencia: Cualidad de compartir y revelar con honestidad la información que poseemos mientras caminamos hacia la consecución de los objetivos y respetando las reservas legales.
- Apertura: Capacidad de escuchar y aprender de otras personas y organizaciones para interactuar con el entorno, adaptarnos, crecer y organizarnos.
- Compromiso: Es la capacidad de cumplir con los resultados planteados, tomando en cuenta las condiciones y obligaciones que implican para la institución y las personas. Es una parte integral de la forma en que las personas actúan con convicción en su relación con la Institución.
- Confiabilidad: Es la capacidad de cumplir con lo prometido en tiempo y forma, lo cual implica credibilidad, respeto, justicia e igualdad de oportunidades en todos los campos.

Croquis:

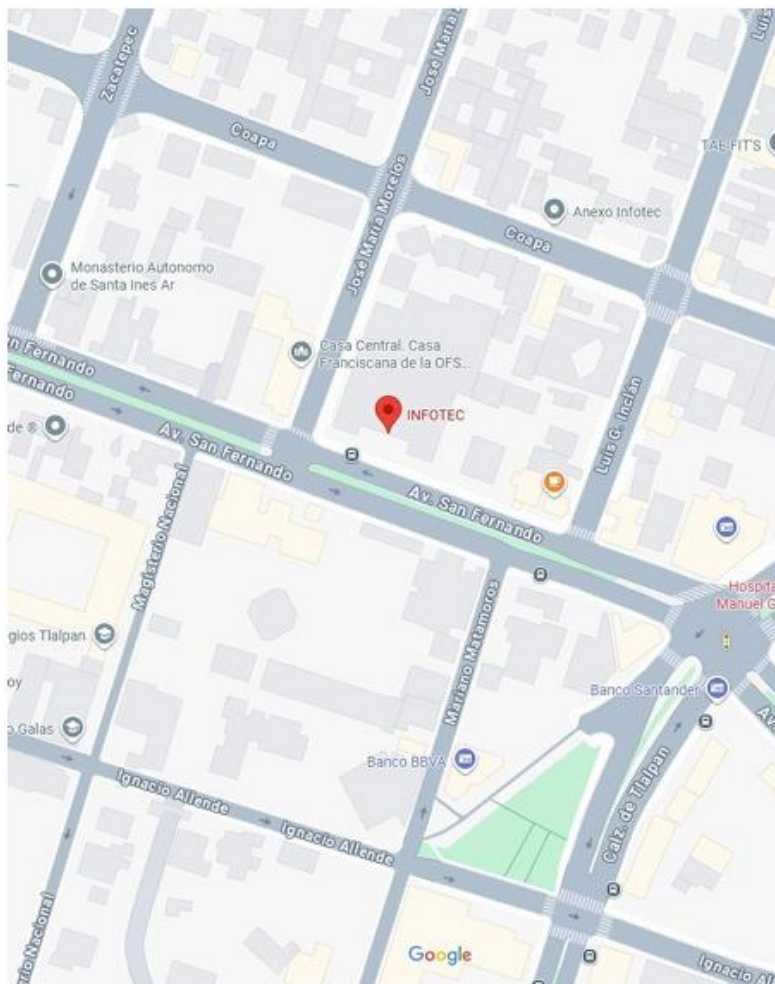


Ilustración 1 Croquis de INFOTEC.

2.4.2 Descripción del departamento o área de trabajo

- Nombre del departamento: Biblioteca.
- Descripción del área en que se participó: Este espacio cuenta con recursos bibliográficos especializados en inteligencia artificial y programación general, brindando un entorno adecuado para el desarrollo del proyecto.

2.5 Problemas a resolver

Previo al inicio del proyecto, se detectaron diversas problemáticas que obstaculizan la correcta clasificación de residuos, afectando tanto la eficiencia del proceso como la posibilidad de implementar soluciones sostenibles. Estas problemáticas se han priorizado de acuerdo con su impacto en el desarrollo del sistema y se vinculan directamente con los objetivos específicos planteados.

- Falta de herramientas tecnológicas adecuadas para la clasificación de residuos. Actualmente, los métodos manuales son poco eficientes, dependen del criterio del usuario y generan errores frecuentes en la separación de basura. Esta problemática se relaciona con el objetivo de investigar herramientas y tecnologías disponibles que permitan diseñar un sistema confiable y eficiente.
- Limitaciones en la selección de materiales y métodos de implementación. Existen múltiples alternativas para construir el prototipo, pero no todas ofrecen un balance entre calidad, costo y rendimiento. Resolver este problema es clave para optimizar recursos, asegurando que la solución sea funcional y económicamente viable.
- Ausencia de una estructura física y lógica definida del sistema. Sin un diseño previo, la disposición de los componentes electrónicos y el software podría generar incompatibilidades y bajo rendimiento. Este problema motiva el objetivo de diseñar el modelo en 3D y definir la arquitectura del software y hardware, priorizando la estabilidad y escalabilidad del sistema.
- Dificultades en la integración del hardware con el software. La falta de sincronización entre la cámara, sensores, actuadores y el modelo de IA puede limitar la operatividad del sistema. Por ello, se debe ensamblar e integrar los componentes electrónicos de manera robusta, además de programar el microcontrolador para garantizar una clasificación automática y precisa.
- La clasificación errónea de residuos comprometería la utilidad del sistema, generando incertidumbre en la precisión y confiabilidad del modelo de inteligencia artificial.

2.6 Alcances y limitaciones

2.6.1 Alcances

Funcionalidad Principal

Se desarrolló un prototipo funcional utilizando el microcontrolador ESP32, capaz de clasificar automáticamente residuos en las categorías de orgánicos e inorgánicos. El sistema incluye una pantalla como interfaz de retroalimentación para informar al usuario sobre el resultado de la clasificación.

Inteligencia Artificial

Se integró un modelo de inteligencia artificial (basado en MobileNetV2 preentrenado), ajustado y entrenado con un dataset que incluye imágenes de residuos comunes como PET, papel bond, papel higiénico, diversos tipos de cáscaras, huesos, latas de aluminio y cobre. El modelo es capaz de identificar y clasificar los residuos automáticamente, garantizando un nivel de precisión aceptable según las métricas definidas en la fase de pruebas.

Diseño y Estructura Física

Se generó un Diseño 3D completo del sistema. Este diseño permite una visualización detallada de la disposición del hardware (incluyendo la cámara, servomotor, sensor ultrasónico y pantalla), la estructura física del contenedor y la lógica de operación espacial del prototipo.

Integración de Hardware y Software

Se completó la integración total del hardware (cámara, servomotor, sensor ultrasónico y pantalla) con el software (código de control y modelo MobileNetV2). Se estableció y garantizó la comunicación eficiente entre todos los elementos, asegurando la coordinación básica de los componentes para ejecutar el ciclo de detección y clasificación.

Validación y Pruebas

Se llevaron a cabo pruebas de funcionamiento rigurosas en condiciones controladas, enfocadas en dos aspectos clave:

- Evaluación de la precisión del modelo de IA frente a una muestra variada de residuos.
- Verificación de la respuesta coordinada del hardware ante las instrucciones de clasificación emitidas por el microcontrolador. Los resultados obtenidos fueron analizados y validados, permitiendo identificar fortalezas, debilidades y áreas de mejora específicas para el desarrollo de futuras versiones del sistema.

2.6.2 Limitaciones

Delimitación de Materiales

El alcance de clasificación se restringió a las categorías generales definidas, excluyendo subcategorías específicas de plásticos (como HDPE, PVC, PS), metales complejos, vidrio, cartón y residuos menores (colillas, chicles), debido a las limitaciones en la extensión del dataset y la arquitectura del modelo.

Condiciones Ambientales

La precisión del sistema presenta una dependencia de condiciones controladas; factores externos como variaciones drásticas de iluminación o la presencia de residuos deformados, excesivamente sucios o mezclados pueden impactar la fiabilidad de la inferencia.

Restricciones de Hardware

El uso del ESP32 impuso límites técnicos en memoria y procesamiento, lo que restringió la complejidad del modelo de IA y la resolución de las imágenes procesadas, obligando a optimizar el sistema para un funcionamiento básico en lugar de alto rendimiento.

Escalabilidad y Enfoque

El diseño se concibió como un prototipo experimental y no como una solución industrial. Por tanto, la capacidad de procesamiento y la resistencia de los materiales están dimensionadas para pruebas de laboratorio, con una escalabilidad reducida en esta fase.

Dependencia de Datos

La efectividad del sistema está directamente vinculada a la calidad del entrenamiento, lo que implica que la precisión depende de la diversidad de las imágenes recopiladas en el dataset, pudiendo fallar ante objetos no representados previamente.

Limitaciones Presupuestales

Las pruebas y la construcción física se limitaron al entorno académico y al presupuesto disponible, lo que impidió el uso de componentes de grado industrial o la creación de una versión más robusta con mayores prestaciones mecánicas.

Capítulo 3

Marco teórico

3.1 Investigaciones previas

La gestión de residuos en México enfrenta retos importantes: se generan más de 100 mil toneladas diarias, pero sólo ~84% se recolecta y solo ~10% se recicla [13]. La inteligencia artificial (IA) surge como herramienta clave para mejorar esta situación. Por ejemplo, la IA permite sistemas automáticos de clasificación visual que aceleran el reciclaje [5]. En el contexto mexicano, las investigaciones destacan la potencialidad de redes neuronales y visión por computadora para reconocer y separar residuos sólidos.

Proyectos académicos e investigativos en México

Redes neuronales convolucionales (CNN) para residuos. En México se han entrenado modelos CNN sobre cámaras en tiempo real. Por ejemplo, investigadores del IPN desarrollaron un algoritmo CNN que clasifica cuatro tipos de residuos reciclables, logrando una precisión superior al 89% [11]. Este sistema incluso activa salidas de hardware (Arduino) para accionar máquinas de separación, demostrando su aplicabilidad en procesos reales. De manera similar, un grupo del TecNM empleó YOLOv8 (un detector de objetos avanzado) con una webcam para clasificar basura doméstica. El modelo entrenado alcanzó una precisión global del 86% con inferencia rápida (~50 ms por imagen) [12], lo cual es suficiente para implementaciones en sensores en contenedores inteligentes.

Prototipos y puntos inteligentes. Se han desarrollado prototipos de puntos de reciclaje inteligentes (smart bins) en universidades mexicanas. Por ejemplo, Sánchez Yáñez y Márquez (2024) mencionan el diseño de un “punto ecológico inteligente” basado en MobileNet que clasificó residuos con ~97.9% de acierto usando una Raspberry Pi [11]. Estos dispositivos combinan visión artificial con microcontroladores (IoT) para asistir al usuario en la separación correcta de plástico, papel, metal, etc. Además, alumnos del Tec Milenio (NL) crearon “RevoTrash”, un basurero inteligente con IA que identifica y clasifica automáticamente la basura y educa a los usuarios. En pruebas piloto en colegios de Monterrey, 9 de cada 10 personas usaron este dispositivo correctamente [13], evidenciando el impacto social de integrar IA en el reciclaje.

Análisis de datos y normativa. Las investigaciones académicas también resaltan la importancia de la infraestructura de datos para IA en residuos. Según Sánchez Yáñez y Márquez (2024), la falta de datos etiquetados y de infraestructura adecuada son desafíos clave en México. Por ello, se propone recopilar grandes bases de imágenes de residuos locales y aprovechar transfer learning para adaptar modelos globales (como YOLO o DenseNet) al contexto de la región.

Innovaciones tecnológicas e industriales

Equipos robóticos avanzados. Empresas líderes en clasificación de residuos llevan sus soluciones a México. Por ejemplo, la noruega TOMRA presentó su línea AUTOSORT™ en la Expo Residuos 2023 (Ciudad de México). Estos sistemas combinan sensores de alta resolución (hasta 320 000 puntos de escaneo por segundo) con cámaras de color y visión infrarroja cercana (NIR) [14]. Esta tecnología permite identificar plásticos, metales y papeles con altísima precisión y altos niveles de pureza al separarlos, optimizando la rentabilidad del reciclaje. TOMRA reporta que sus equipos ya están instalados en miles de plantas globales y ayudan a recuperar una gran cantidad de materiales valiosos en México y el mundo.

Aplicaciones prácticas locales. Además de la investigación, surgen iniciativas de emprendimiento. El basurero “RevoTrash” del Tec Milenio (NL) es un ejemplo local: utiliza IA para clasificar residuos y ganó un premio estatal en 2024 [13]. Este prototipo no solo separa basura por tipo, sino que también genera estadísticas de uso y conciencia ambiental. A nivel de industria nacional, otros startups mexicanos exploran IA para gestión de residuos, aunque muchas aún están en fase experimental. Sin embargo, la creciente demanda por procesos más limpios ha llevado al gobierno y empresas a invertir en demos tecnológicas (como la participación de TOMRA en expos locales) y en proyectos piloto a pequeña escala.

IoT y sensores inteligentes. Más allá de robots, la visión artificial se integra con el Internet de las cosas. Sensores en contenedores «smart» pueden monitorear niveles de llenado, y cámaras con IA pueden alertar cuando se arrojan residuos no reciclables. Según un estudio reciente, dispositivos inteligentes permiten clasificar materiales con alta precisión sin depender de servidores centrales [5]. Este enfoque edge computing es prometedor para zonas urbanas de México, donde la conectividad puede ser limitada.

3.2 Bases teóricas

3.2.1 Sistemas de Clasificación de Residuos Basados en Inteligencia Artificial

Un sistema de clasificación de residuos con inteligencia artificial es una solución tecnológica que integra modelos de aprendizaje profundo, visión por computadora y hardware especializado para identificar y separar distintos tipos de desechos. Su funcionamiento se basa en el uso de CNN, entrenadas con bases de datos como TrashNet o WasteNet, que permiten reconocer patrones visuales en imágenes de basura y clasificarlos en categorías definidas (papel, plástico, metal, orgánico, etc.).

Estos sistemas permiten automatizar una tarea que tradicionalmente depende de la intervención humana, mejorando la eficiencia, rapidez y precisión del proceso de separación. Además, pueden integrarse con dispositivos como cámaras, microcontroladores (ESP32, Arduino) y actuadores (servomotores o brazos robóticos) para realizar la separación física de los residuos.

3.2.2 Redes Neuronales Convolucionales (CNN)

Las CNN convencionales son un tipo de modelo de aprendizaje profundo ampliamente utilizado en tareas de visión por computadora. Su arquitectura está diseñada para procesar imágenes mediante capas convolucionales que extraen características como bordes, texturas y formas, permitiendo identificar patrones complejos.

En el contexto de clasificación de residuos, las CNN se entrenan con miles de imágenes de diferentes categorías de basura. Al final del entrenamiento, el modelo puede generalizar y reconocer residuos en tiempo real con alta precisión. Esto convierte a las CNN en la base fundamental de los sistemas modernos de clasificación inteligente.

3.2.3 Transfer Learning

El aprendizaje por transferencia (*Transfer Learning*) es una técnica que permite reutilizar un modelo previamente entrenado en una tarea general (por ejemplo, MobileNetV2 entrenado en ImageNet) y adaptarlo a un nuevo dominio con menos datos.

En este proyecto, se aplica para entrenar un modelo de clasificación de residuos sin necesidad de contar con millones de imágenes propias. De esta manera, se optimizan los recursos computacionales y se acelera el desarrollo del sistema, logrando resultados confiables con menos tiempo de entrenamiento.

3.2.4 Microcontroladores ESP32 y ESP32-CAM

El ESP32 es un microcontrolador de bajo costo con conectividad Wifi y Bluetooth, ampliamente usado en proyectos de IoT y automatización. Su versión ESP32-CAM incluye una cámara integrada, lo que permite capturar imágenes directamente desde el dispositivo y enviarlas a un servidor para su procesamiento con inteligencia artificial.

En un sistema de clasificación de basura, el ESP32-CAM cumple la función de capturar las imágenes de los residuos y comunicarse con un servidor (Flask + TensorFlow) encargado de procesarlas. Además, puede controlar periféricos como servomotores o pantallas TFT para mostrar los resultados de la clasificación, haciendo posible una integración completa de hardware y software.

3.2.5 Aplicaciones en la Gestión de Residuos

En los últimos años, la inteligencia artificial (IA) ha adquirido un papel cada vez más relevante en la gestión ambiental, particularmente en la clasificación y manejo de residuos sólidos urbanos. El uso de algoritmos de aprendizaje profundo (Deep Learning), redes neuronales convolucionales (CNN) y visión por computadora ha permitido el desarrollo de sistemas capaces de identificar materiales reciclables con alta precisión, reduciendo significativamente la intervención humana y los errores asociados al proceso manual de separación.

La implementación de estas tecnologías contribuye directamente a la eficiencia de los procesos de reciclaje, al permitir la automatización de tareas que anteriormente dependían de la inspección visual. Los modelos de IA pueden ser entrenados para reconocer diferentes tipos de residuos como plástico, metal, papel, vidrio u orgánicos mediante el análisis de imágenes capturadas en tiempo real, facilitando su clasificación en líneas de procesamiento o incluso en entornos domésticos mediante dispositivos embebidos.

En el ámbito de la sostenibilidad, la IA también ha impulsado la transición hacia una economía circular, optimizando la reutilización de materiales y disminuyendo la cantidad de desechos enviados a vertederos. Sistemas de visión por computadora integrados con microcontroladores, como el ESP32-CAM, y sensores periféricos permiten desarrollar prototipos accesibles y energéticamente eficientes, lo que abre la posibilidad de implementar soluciones de bajo costo en comunidades, centros educativos y pequeñas empresas.

Además, los avances en el Internet de las Cosas (IoT) han permitido conectar estos sistemas inteligentes a plataformas en la nube, generando redes de monitoreo ambiental capaces de recolectar datos sobre volúmenes de residuos, frecuencia de recolección y niveles de contaminación. Estos datos pueden emplearse para diseñar políticas públicas más efectivas y fomentar una gestión urbana sostenible.

En conjunto, la aplicación de la inteligencia artificial en la gestión de residuos no solo representa un avance tecnológico, sino también una herramienta estratégica para mitigar el impacto ambiental, optimizar recursos y fortalecer la conciencia ecológica. La combinación de IA, visión por computadora e IoT está sentando las bases para ciudades más limpias, sostenibles e inteligentes.

A pesar de los beneficios que la inteligencia artificial aporta a la gestión ambiental, su implementación también conlleva una serie de desafíos y consecuencias que deben ser considerados para garantizar su sostenibilidad a largo plazo. Entre los principales problemas asociados se encuentran el elevado consumo energético, la demanda de recursos naturales y la generación indirecta de emisiones de carbono.

El entrenamiento de modelos de aprendizaje profundo, como redes neuronales convolucionales, requiere una cantidad significativa de energía eléctrica debido al uso intensivo de unidades de procesamiento gráfico (GPU) o aceleradores especializados. Estudios recientes han estimado que el entrenamiento de un modelo de gran escala puede consumir tanta energía como la utilizada por varios hogares durante un año completo. Este consumo energético, cuando proviene de fuentes no renovables, incrementa la huella de carbono y contradice parcialmente los objetivos de sostenibilidad ambiental que estos sistemas buscan promover. [1]

Además, el proceso de enfriamiento de los centros de datos donde se entrenan y despliegan los modelos de IA requiere grandes volúmenes de agua para mantener las temperaturas operativas adecuadas. Se estima que algunos centros tecnológicos pueden consumir millones de litros de agua al año, lo que representa un impacto ambiental adicional en regiones con estrés hídrico. [1]

Otro aspecto relevante es el ciclo de vida del hardware utilizado. Los microcontroladores, sensores, cámaras y demás componentes electrónicos empleados en sistemas embebidos generan residuos tecnológicos que deben gestionarse adecuadamente para evitar la contaminación por metales pesados o compuestos químicos. Sin una estrategia de reciclaje electrónico, el impacto ambiental del desarrollo tecnológico podría superar sus beneficios. [1]

Por otro lado, el uso masivo de datos (datasets) para el entrenamiento de modelos plantea retos en materia de almacenamiento, transmisión y procesamiento de información. Esto incrementa la demanda de infraestructura digital y, con ella, el consumo energético global.

En consecuencia, aunque la IA representa una herramienta poderosa para mejorar la gestión de residuos y promover prácticas sostenibles, su aplicación debe ir acompañada de políticas de eficiencia energética, uso de energías renovables y estrategias de diseño responsable de hardware y software. Solo mediante un equilibrio entre innovación y sostenibilidad será posible aprovechar plenamente los beneficios de la inteligencia artificial sin comprometer el medio ambiente.

3.4 Antecedentes

3.4.1 Contexto General del Problema

La gestión de residuos sólidos en México representa uno de los principales retos ambientales y sociales de las últimas décadas. El acelerado crecimiento urbano, el consumo masivo y la limitada infraestructura para el manejo adecuado de los desechos han provocado un aumento sostenido en la generación de basura. De acuerdo con la Secretaría de Medio Ambiente y Recursos Naturales (SEMARNAT), en el país se producen diariamente más de 120 mil toneladas de residuos sólidos urbanos, de los cuales una parte considerable no se dispone adecuadamente, afectando la salud pública, los ecosistemas y la calidad de vida de la población. [1]

En este contexto, la gestión eficiente de residuos se ha convertido en una prioridad dentro de las políticas ambientales nacionales y locales, promoviendo estrategias basadas en los principios de la economía circular: reducir, reutilizar y reciclar. No obstante, la implementación de estas prácticas aún enfrenta desafíos significativos, como la falta de infraestructura tecnológica, la escasa participación ciudadana y la ineficiencia en los procesos de separación y clasificación manual.

Ante esta problemática, INFOTEC ha impulsado proyectos orientados a la innovación tecnológica y la sostenibilidad ambiental, fomentando el uso de herramientas digitales, inteligencia artificial y microcontroladores para desarrollar soluciones prácticas que apoyen la gestión de residuos.

En este marco, el presente proyecto se centra en el diseño e implementación de un sistema automatizado de clasificación de residuos, empleando visión por computadora e inteligencia artificial para identificar diferentes tipos de basura orgánica e inorgánica. Este desarrollo no solo busca aportar una alternativa eficiente y accesible para el tratamiento de residuos, sino también fortalecer las competencias tecnológicas y científicas de los estudiantes mediante la aplicación de metodologías ágiles como SCRUM y tecnologías emergentes como el ESP32, sensores ultrasónicos y pantallas.

De esta forma, el proyecto integra un enfoque interdisciplinario y formativo, combinando innovación, sostenibilidad y tecnología para contribuir al manejo responsable de los residuos y al cumplimiento de los Objetivos de Desarrollo Sostenible (ODS), particularmente el ODS 12: Producción y consumo responsables y el ODS 13: Acción por el clima.



Ilustración 2 Objetivos de desarrollo sostenible

3.4.2 Evolución de la Clasificación de Residuos

La Ciudad de México, a través de la SEDEMA, implementa programas específicos para la recolección, tratamiento y disposición final de residuos sólidos urbanos, incluyendo campañas de separación en origen y educación ambiental para los ciudadanos.

La clasificación de residuos ha experimentado una evolución significativa en México durante las últimas décadas, impulsada por la necesidad de reducir los impactos ambientales derivados del creciente volumen de desechos urbanos. Tradicionalmente, la gestión de residuos se basaba en la recolección conjunta de basura orgánica e inorgánica, lo que complicaba los procesos de reciclaje, aumentaba la contaminación y saturaba los sitios de disposición final.

A nivel nacional, diversas iniciativas han promovido la separación en origen como estrategia fundamental para mejorar el aprovechamiento de materiales reciclables y disminuir la carga ambiental de los rellenos sanitarios. Este enfoque ha sido reforzado por normativas federales y programas de educación ambiental que buscan modificar los hábitos de la población en torno al manejo responsable de los residuos.

En este contexto, la Ciudad de México ha desempeñado un papel destacado en la modernización de la clasificación de residuos. La Secretaría del Medio Ambiente (SEDEMA) ha implementado lineamientos y campañas orientadas a fortalecer la cultura de separación en los hogares, comercios y espacios públicos. Entre los avances más relevantes se encuentra la adopción del sistema de separación en cuatro categorías principales: residuos orgánicos, inorgánicos reciclables, inorgánicos no reciclables y residuos de manejo especial y voluminosos. Este sistema ha permitido una mayor eficiencia en los procesos de recolección y ha incrementado la cantidad de materiales aprovechados.

Además, SEDEMA ha impulsado programas de valorización de residuos, centros de acopio, jornadas de reciclaje comunitarias y campañas permanentes de sensibilización ambiental. Estas acciones forman parte de una estrategia integral que busca no solo optimizar la gestión de residuos, sino también promover la corresponsabilidad ciudadana y avanzar hacia modelos más sostenibles alineados con políticas ambientales globales.

En conjunto, la evolución de la clasificación de residuos en México y en la Ciudad de México refleja un proceso de transición hacia prácticas más eficientes, sostenibles y orientadas a la economía circular. Este marco contextual sienta las bases para el desarrollo de soluciones tecnológicas, como el sistema automatizado propuesto en este proyecto, que contribuyen a la mejora continua en la gestión de residuos urbanos.

Esta evolución se ha visto reforzada por los lineamientos federales establecidos por la SEMARNAT y por los programas implementados por la SEDEMA en la Ciudad de México, que buscan fortalecer la separación en origen, mejorar la infraestructura de recolección y promover prácticas sostenibles de consumo y disposición final de residuos [2][3].

Un ejemplo puede ser la reforma impulsada por la cámara de gobierno capitalina, la cual establece que, a partir del 1° de enero de 2026, la separación de residuos será obligatoria en la Ciudad de México. Bajo el programa denominado “Transforma tu ciudad, cada basura en su lugar”, el nuevo esquema exige que los hogares, comercios, escuelas y oficinas dividan sus desechos en tres categorías: residuos orgánicos, residuos inorgánicos reciclables y residuos inorgánicos no reciclables. Como parte de esta estrategia, se prevé que al menos el 50 % de las aproximadamente 8 600 toneladas de basura generadas a diario sean correctamente recicladas o aprovechadas.

Esta iniciativa también incluye un calendario de recolección diferenciada por días, campañas de educación ambiental y una mayor inversión en infraestructura de recolección. [4]

3.4.3 Problemática de Clasificación de Residuos

A nivel mundial, la separación inadecuada de residuos continúa siendo uno de los principales obstáculos para lograr sistemas de reciclaje eficientes y sostenibles. En la mayoría de los países, el proceso de clasificación depende en gran medida de la participación ciudadana, lo que introduce un alto margen de error debido a la falta de información, hábitos inadecuados o la ausencia de infraestructura adecuada. Esta situación provoca contaminación cruzada, es decir, la mezcla de residuos orgánicos e inorgánicos, o de materiales reciclables con desechos no reciclables, lo que disminuye considerablemente la calidad del material recuperado y aumenta los costos operativos de separación y tratamiento [2].

En México, este problema se refleja en la baja tasa de separación efectiva en los hogares y espacios públicos, así como en la saturación de los centros de transferencia y rellenos sanitarios. A pesar de los programas de educación ambiental y las normativas que promueven la separación en origen, gran parte de los residuos sigue llegando mezclada a los sitios de disposición final, dificultando su aprovechamiento y generando impactos ambientales como emisiones de gases de efecto invernadero, proliferación de fauna nociva y contaminación del suelo y del agua.

Ante este panorama, iniciativas recientes impulsadas por el Gobierno de la Ciudad de México buscan reforzar la cultura de separación mediante campañas informativas y nuevas estrategias de gestión de residuos, como la campaña “Transforma tu ciudad, cada basura en su lugar”, que pretende mejorar la correcta clasificación desde el hogar y los espacios públicos [4].

3.4.4 Avances Tecnológicos en Clasificación y Automatización de Residuos

En los últimos años, la incorporación de inteligencia artificial (IA), visión por computadora y sensores IoT ha transformado significativamente los procesos de gestión y clasificación de residuos a nivel mundial. Estas tecnologías permiten optimizar la separación de materiales, reducir costos operativos y aumentar la precisión en la identificación de desechos.

A nivel internacional, sistemas avanzados como AMCS Vision AI han demostrado su capacidad para detectar residuos contaminados, identificar incorrectamente clasificados y monitorear niveles de sobrellenado en contenedores, lo que contribuye a mejorar la logística de recolección y la calidad del material recuperado [6].

Del mismo modo, empresas como Recycleye han desarrollado soluciones basadas en IA capaces de clasificar residuos automáticamente en plantas industriales, utilizando modelos entrenados con millones de imágenes. Estas plataformas reducen significativamente la intervención manual, agilizan la operación de centros de reciclaje y permiten obtener tasas de eficiencia superiores a las del método tradicional [7].

A nivel nacional, México ha comenzado a integrar estas tecnologías dentro de proyectos piloto y estudios académicos orientados a modernizar la gestión de residuos urbanos. Investigaciones recientes demuestran que la combinación de microcontroladores, visión por computadora y redes neuronales permite automatizar la clasificación en tiempo real, facilitando la separación entre residuos orgánicos e inorgánicos directamente desde el punto de generación. Sin embargo, estos avances aún enfrentan retos significativos, como la falta de infraestructura tecnológica sólida, el costo de implementación en municipios con recursos limitados y la necesidad de capacitar a operadores y personal técnico. Aun así, el progreso científico y el creciente interés

gubernamental en estrategias de digitalización ambiental indican un escenario favorable para la adopción gradual de estas soluciones en los próximos años [8].

3.4.5 Proyectos y Aplicaciones Específicas

A nivel internacional, diversos países han desarrollado proyectos innovadores que integran inteligencia artificial, sensores avanzados y robótica para mejorar la eficiencia en la gestión de residuos.

En Finlandia, empresas como Remeo y Zen Robotics han liderado la automatización del reciclaje mediante sistemas robóticos equipados con visión por computadora, cámaras infrarrojas y algoritmos de clasificación en tiempo real. Estos robots son capaces de identificar y separar más de 2 000 objetos por hora, lo que demuestra la viabilidad de la robótica industrial en los centros de reciclaje modernos [8].

En Estados Unidos, el uso de contenedores inteligentes ha permitido optimizar tanto la recolección como el monitoreo de residuos urbanos. Estos dispositivos emplean sensores de nivel de llenado, comunicación inalámbrica y modelos de predicción para notificar a los servicios municipales cuando es necesario vaciarlos, reduciendo costos operativos y evitando la saturación de contenedores [9].

En América Latina, diversas iniciativas impulsadas por organismos como el Banco Interamericano de Desarrollo (BID) han aplicado técnicas de inteligencia artificial para predecir la generación de desechos, optimizar rutas de recolección y analizar patrones de comportamiento ciudadano. Estos proyectos buscan avanzar hacia modelos de economía circular, promoviendo la reutilización y el reciclaje de materiales, así como la mejora de la infraestructura de gestión de residuos en países de la región [10].

3.5 Relevancia del Proyecto

A pesar de los avances tecnológicos logrados a nivel internacional en materia de clasificación automatizada de residuos, la mayoría de estas soluciones continúan siendo costosas, de difícil acceso y diseñadas principalmente para entornos industriales o plantas especializadas de reciclaje. Equipos basados en robótica avanzada, sensores industriales y sistemas de visión artificial de alto rendimiento representan inversiones significativas que resultan inviables para la gran mayoría de municipios, pequeñas comunidades o incluso hogares. Esta brecha tecnológica genera un rezago considerable en la adopción de prácticas eficientes de separación de residuos, especialmente en

países como México, donde la infraestructura de gestión de desechos es limitada y persisten desafíos como la contaminación cruzada, la falta de separación en origen y la saturación de sitios de disposición final.

En este contexto, el presente proyecto adquiere una relevancia particular al proponer una alternativa accesible, escalable y de bajo costo, empleando tecnologías como ESP32, servidores ligeros en Flask e inteligencia artificial basada en MobileNetV2, un modelo optimizado para dispositivos de bajo consumo. A diferencia de sistemas industriales, este prototipo está pensado para operar en entornos domésticos, escolares o comunitarios, donde puede funcionar como herramienta de apoyo para la correcta clasificación de residuos, reducir la dependencia del trabajo manual y fomentar una cultura de reciclaje más efectiva entre la población.

3.5.1 Importancia del proyecto en el contexto actual

El manejo inadecuado de residuos representa uno de los principales retos ambientales en México. Según SEMARNAT, el país genera más de 120 mil toneladas de residuos sólidos urbanos al día, y una parte considerable termina sin separación adecuada. Al facilitar la correcta clasificación en el punto de origen, este proyecto contribuye de manera directa a:

- Reducir la contaminación cruzada entre residuos orgánicos e inorgánicos.
- Mejorar la calidad del material reciclado.
- Disminuir la cantidad de desechos que llegan a rellenos sanitarios.
- Apoyar políticas públicas de separación obligatoria que se implementarán en los próximos años.
- Incrementar la educación ambiental mediante un sistema interactivo y visual.

Además, al basarse en hardware económico, reutilizable y ampliamente documentado, este prototipo puede ser adoptado por instituciones educativas, comunidades rurales y centros urbanos con recursos limitados, generando un impacto social significativo.

3.5.2 Por qué se debería invertir en este proyecto

La inversión en soluciones accesibles como esta resulta estratégica por múltiples motivos:

1. Escalabilidad y bajo costo.

El ESP32, junto con sensores económicos y componentes fácilmente disponibles, permite crear sistemas replicables con un presupuesto reducido, lo que facilita su implementación en múltiples entornos.

2. Reducción de costos municipales.

Al mejorar la separación en origen, se disminuyen los costos de clasificación manual, transporte y tratamiento posterior.

3. Alineación con objetivos de sostenibilidad nacionales e internacionales.

Contribuye a los Objetivos de Desarrollo Sostenible (ODS), principalmente los relacionados con ciudades sostenibles, producción responsable y acción climática.

4. Fortalecimiento de la innovación nacional.

Desarrollar tecnología desde una institución como INFOTEC impulsa la autonomía tecnológica del país y reduce la dependencia de sistemas costosos importados.

5. Potencial educativo.

Promueve el aprendizaje en áreas como electrónica, IA, programación y sostenibilidad ambiental, pudiendo utilizarse en talleres, universidades y espacios comunitarios.

En conjunto, estos factores justifican ampliamente la inversión en este tipo de iniciativas, que no solo abordan un problema ambiental urgente, sino que también generan conocimiento e infraestructura tecnológica propia.

3.5.3 Metodología SCRUM aplicada al desarrollo

El desarrollo del proyecto se llevó a cabo mediante la metodología SCRUM, permitiendo una gestión ágil, iterativa y colaborativa del proceso. SCRUM facilitó la descomposición del proyecto en sprints definidos, cada uno enfocado en objetivos específicos, tales como:

- **Construcción del servidor Flask.**
- **Implementación del sistema mecánico.**
- **Integración del ESP32 con el backend mediante un endpoint.**
- **Elección del dataset y entrenamiento del modelo de IA**

Gracias a SCRUM, el equipo pudo adaptar el desarrollo conforme aparecían nuevos retos técnicos, priorizar funcionalidades críticas y mantener un flujo constante de entregables funcionales.

Esto permitió evolucionar desde un sistema básico de clasificación local hasta un prototipo completamente integrado capaz de operar de forma autónoma.

3.5.5 Valor social, tecnológico y ambiental

El valor del proyecto trasciende la simple automatización. Su relevancia se fundamenta en que:

- Democratiza el acceso a tecnologías de IA para la gestión de residuos.
- Fomenta la cultura de separación desde el hogar.
- Crea un camino para futuros sistemas inteligentes de reciclaje de bajo costo.
- Sirve como base para mejorar políticas públicas y estrategias de sostenibilidad.

En suma, este proyecto representa una contribución concreta a la transformación digital y ambiental del país, alineándose con el trabajo realizado en INFOTEC y respondiendo a una problemática real que afecta tanto a México como al resto del mundo.

Capítulo 4

Desarrollo

El desarrollo del sistema de clasificación automatizada de residuos se llevó a cabo mediante una serie de etapas secuenciales e iterativas que permitieron asegurar la correcta integración entre los componentes de software, hardware y el modelo de inteligencia artificial. Con el fin de garantizar un proceso organizado, eficiente y controlado, se empleó una metodología basada en análisis colaborativo, entrevistas internas y la adopción del marco ágil **SCRUM**, lo que permitió dividir la solución en incrementos funcionales que fueron evaluados de manera continua.

La fase inicial consistió en la identificación de las necesidades reales dentro del contexto de INFOTEC, por medio de reuniones y pláticas con nuestro asesor académico, personal de la institución y compañeros residentes que compartían problemáticas relacionadas con la gestión de residuos y el manejo responsable dentro de los espacios de trabajo. Estas interacciones permitieron reconocer puntos clave tales como:

- Falta de mecanismos de separación eficiente
- Escasa sensibilización sobre el reciclaje
- Necesidad de soluciones accesibles para instituciones educativas y centros de investigación

así como el interés de INFOTEC en promover tecnologías emergentes aplicadas a problemas ambientales.

A partir de este análisis colaborativo se consolidaron los requisitos funcionales y no funcionales del sistema, estableciendo un camino claro para el desarrollo tecnológico. Posteriormente, se desarrolló el modelo de inteligencia artificial, entrenado con un conjunto de datos compuesto por imágenes de residuos orgánicos e inorgánicos capturadas bajo diversas condiciones de luz y ángulos.

Se implementaron técnicas de *data augmentation* para mejorar la capacidad de generalización y se evaluaron varias arquitecturas ligeras basadas en MobileNetV2, buscando un equilibrio adecuado entre rendimiento y velocidad en dispositivos embebidos como el ESP32. Tras múltiples iteraciones de entrenamiento, validación y ajuste de hiperparámetros, se seleccionó el modelo final que ofreció la mayor precisión manteniendo tiempos de inferencia óptimos.

Una vez obtenido el modelo final, se procedió a la construcción del entorno de pruebas local, mediante un servidor Flask encargado de recibir las imágenes enviadas por el ESP32, procesarlas, ejecutar la inferencia y devolver la clasificación correspondiente. Este entorno permitió depurar

eficazmente el flujo de comunicación, medir los tiempos de respuesta HTTP, validar el funcionamiento del pipeline de inteligencia artificial y realizar pruebas controladas antes de la implementación en hardware.

Con el software base implementado, la siguiente etapa fue la integración del hardware, que incluyó el ESP32, el sensor ultrasónico, el servomotor para el mecanismo de separación física y la pantalla para mostrar el historial de clasificaciones y el estado del sistema. Durante esta fase se realizaron pruebas de conexión eléctrica, medición de consumo energético, calibración del sensor de distancia y verificación del movimiento del servomotor. También se desarrolló la interfaz visual que permite al usuario monitorear el funcionamiento del sistema en tiempo real.

Para organizar y controlar el proceso, se utilizó la metodología SCRUM, estructurando el trabajo en sprints semanales. Cada sprint incluyó sesiones de planificación, ejecución de tareas, reuniones diarias de seguimiento (*daily scrum*), revisiones al finalizar cada incremento y retrospectivas para identificar mejoras en el flujo de trabajo. Este enfoque permitió corregir problemas oportunamente, priorizar funcionalidades críticas y asegurar un avance continuo y ordenado del proyecto.

Finalmente, se realizó el ensamblaje físico del prototipo, integrando todos los componentes dentro de una estructura diseñada para optimizar el flujo de residuos, la captura de imágenes y la clasificación automática. Se llevaron a cabo pruebas de campo en condiciones reales, donde se midió la precisión del modelo, la estabilidad de la conexión WiFi, los tiempos de respuesta del servidor, la operación del servomotor y la fiabilidad general del sistema ante distintos tipos de residuos.

El resultado de este proceso de desarrollo fue un sistema funcional, accesible y escalable, capaz de realizar clasificación automatizada en tiempo real. Este prototipo demuestra la viabilidad de aplicar inteligencia artificial y microcontroladores de bajo costo dentro de instituciones como INFOTEC, fomentando prácticas sostenibles y apoyando la transición hacia una gestión de residuos más eficiente y responsable.

4.1 Entrevistas y Análisis de Requerimientos

Para el diseño del sistema de clasificación automatizada de residuos, fue indispensable realizar un proceso inicial de levantamiento de información mediante entrevistas y conversaciones con actores clave dentro de INFOTEC. Se llevaron a cabo reuniones informales y formales con el asesor académico, personal administrativo, personal de operación y compañeros residentes, quienes aportaron una visión directa sobre los retos reales relacionados con la separación y gestión de residuos dentro del centro.

Durante estas entrevistas se identificaron diversas problemáticas, entre ellas:

- Falta de uniformidad en la separación de residuos dentro de las áreas comunes.
- Ausencia de herramientas tecnológicas que apoyen la correcta clasificación.
- Escasa sensibilización del personal respecto a la importancia de la separación.
- Necesidad de soluciones accesibles que puedan escalarse a otras instalaciones.
- Interés institucional por integrar tecnologías emergentes en procesos ambientales.

Con base en estas observaciones se definieron los requerimientos funcionales, entre los que destacaron:

1. El sistema debe capturar imágenes de residuos automáticamente.
2. El modelo debe clasificar en tiempo real entre residuo orgánico e inorgánico.
3. El servidor debe procesar imágenes y devolver resultados inmediatos.
4. Debe integrarse un mecanismo físico para mover o separar los residuos.
5. Se requiere una interfaz visual para mostrar estado de la clasificación.

Asimismo, se establecieron requerimientos no funcionales:

- Bajo consumo energético.
- Tiempo de inferencia menor a 2 segundos.
- Capacidad de operar con WiFi local.
- Uso de hardware económico y fácilmente reemplazable.
- Tolerancia básica a condiciones ambientales variables.

Este análisis permitió sentar las bases del diseño del prototipo, alineando los objetivos técnicos con las necesidades reales de INFOTEC.

4.1.1 Aspectos generales del código.

En este proyecto se implementa la nomenclatura en su mayoría CamelCase para la creación de clases, con el objetivo de mejorar la legibilidad y mantener un estilo de código uniforme.

Además de CamelCase, se utiliza una nomenclatura de tipo militar (por letras como Alpha, Bravo, etc.) para nombrar las instancias principales del sistema. Esta estrategia facilita la identificación de módulos clave y permite distinguir rápidamente las responsabilidades de cada componente.

Hasta el momento, se han utilizado dos identificadores principales:

Letra utilizada	Clase encargada
Alpha	ManejadorHTTP Asignado al backend principal, encargado del manejo general del sistema.
Bravo	pruebaModelosTF Asignado al backend de identificación mediante inteligencia artificial.

Tabla 1 Nombre de instancias

La nomenclatura general adoptada combina claridad semántica, orden y una estructura que facilita la escalabilidad del proyecto. De esta manera, cada nueva clase o instancia podrá integrarse sin perder consistencia ni generar ambigüedades en el código.

4.1.2 Hardware a utilizar

El hardware que se llegó a utilizar durante el desarrollo del prototipo fue el siguiente:



Ilustración 3 Servomotor mg90s

El Servomotor MG90S es una elección común y efectiva debido a su excelente relación tamaño-par. Es un micro servo de engranajes metálicos que, a pesar de su tamaño reducido, puede proporcionar suficiente fuerza (torque) para tareas ligeras a medianas, como mover pequeños brazos robóticos, accionar interruptores o posicionar sensores y cámaras con precisión. Su capacidad para ser controlado mediante una simple señal PWM (Pulse Width Modulation) desde el ESP32-S3 permite un control angular preciso y lo convierte en una opción económica y fácil de integrar para funciones de movimiento en el proyecto.



Ilustración 4 Sensor ultrasonico HC-SR04

El Sensor Ultrasonico HC-SR04, destaca por su habilidad para medir distancias de manera no intrusiva, confiable, y además a un costo bajo. Emplea pulsos sónicos ultrasónicos, permitiendo la determinación de distancia a un objeto, exhibiendo una admirable resistencia ante variaciones luminosas, lo que le otorga una ventaja sobre otros sensores, por ejemplo, los infrarrojos. Su operación simple, dependiente del "Time of Flight", o el tiempo transcurrido, y su interfaz de dos pines Trigger y Echo simplifican la integración en tareas tales como detección de obstáculos, y la medición de nivel, encajando a la perfección con las capacidades de procesamiento del ESP32-S3.

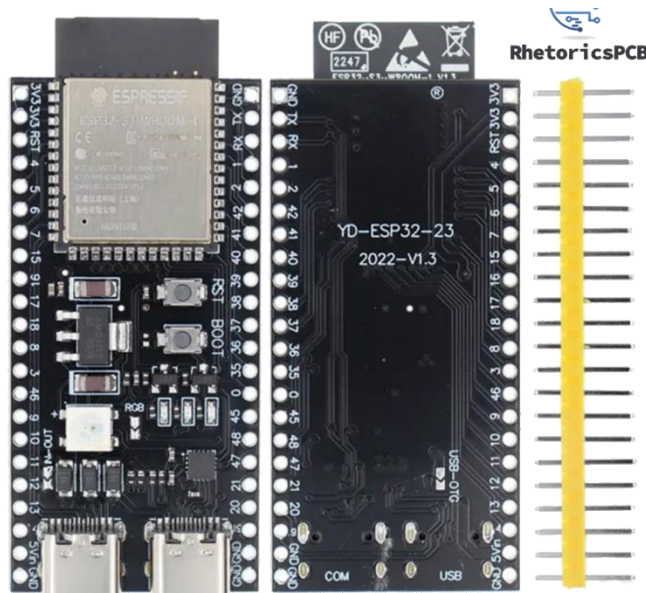


Ilustración 5 ESP32-S3 N16R8

El ESP32-S3 N16R8 fue seleccionado como el cerebro de esta iniciativa debido a su capacidad de conectividad avanzada y alto rendimiento. La selección concreta del N16R8 señala que cuenta con 8MB de PSRAM (RAM pseudoestática) y 16MB de memoria Flash, lo cual le otorga una enorme capacidad para almacenar y procesar datos que es crucial para gestionar la transmisión de video de la cámara OV2640 y llevar a cabo otras funciones al mismo tiempo, como el control del servo, la lectura del sensor ultrasónico y la administración de la pantalla. Asimismo, su soporte incorporado para Bluetooth 5 (LE) y Wi-Fi posibilita la comunicación sin cables y el control a distancia.



Ilustración 6 Pantalla oled 128x64 i2c

Se optó por la pantalla OLED de 128x64 con interfaz I2C debido a su bajo consumo energético y su elevada legibilidad. Su tecnología OLED (diodo emisor de luz orgánico) implica que cada píxel genera su propia luz, lo que da lugar a un contraste sobresaliente y negros impecables, lo cual es perfecto para presentar información nítida en distintas situaciones de iluminación. Asimismo, su tamaño pequeño y la sencillez de la interfaz I2C (que solo necesita dos pines de reloj/datos, además del suministro eléctrico) la hacen ideal para proyectos en los que el espacio y la reducción del cableado son significativos, liberando así pines valiosos del ESP32-S3.



Ilustración 7 Cámara esp32 OV2640

La Cámara OV2640 fue elegida para añadir la capacidad de visión al proyecto. Es una opción muy popular y bien soportada por la plataforma ESP32, ofreciendo una resolución máxima de 2 Megapíxeles (UXGA - 1600x1200), que es más que suficiente para la mayoría de las aplicaciones de visión integrada o streaming de baja a media resolución. Su interfaz SCCB/I2C para control de configuración y la capacidad de emitir datos de imagen en varios formatos la hacen compatible y configurable con el ESP32-S3, permitiendo que el microcontrolador no solo tome decisiones basadas en sensores, sino también en el análisis del entorno visual.

4.2 Planificación y Ejecución de la Metodología Scrum.

El desarrollo del proyecto se estructuró bajo la metodología Scrum, un marco ágil que facilita la gestión de proyectos complejos mediante la división del trabajo en iteraciones cortas y de duración fija llamadas Sprints. Cada Sprint se enfoca en entregar un incremento de producto funcional y potencialmente entregable.

La priorización de los elementos del Product Backlog (tales como la construcción del servidor, la implementación mecánica y el entrenamiento del modelo) se definió para asegurar que el Mínimo Producto Viable (MVP) — la clasificación funcional de residuos con respuesta del servidor — se completara en los primeros Sprints.

4.2.1 Construcción del servidor Flask.

El objetivo de este sprint fue establecer la base del sistema: el Servidor Flask que actuaría como el backend central del sistema de clasificación. Se definió y probó el endpoint de recepción de datos y se integró el core de la Inteligencia Artificial.

Historia de Usuario / Elemento	Tareas Clave	Criterios de Aceptación
Construcción del Servidor Flask y Core de Clasificación.	1.1 Crear la aplicación Flask e inicializar el manejador ManejadorHTTP cargando el modelo de IA (model_retrained_REALDATA_v2.h5).	El servidor se inicia correctamente en la IP local y el modelo de IA está cargado en memoria.
Implementación del Endpoint de Clasificación (POST).	1.2 Crear la ruta principal (/) que maneja solicitudes POST. 1.3 Implementar la lógica de recepción del mensaje JSON (<u>recepcionMensaje</u>). 1.4 Manejar el evento "imagen bytes": guardar la imagen y pasarla al modelo de IA.	El endpoint (/) recibe un JSON con imagen_bytes, guarda el archivo JPG (guardarImagen) y lo procesa.
Integración del Modelo de Clasificación (MobileNetV2).	1.5 Implementar la pre-carga (<u>loadModel</u>) y pre-procesamiento de la imagen (tamaño 224x224). 1.6 Ejecutar la predicción (<u>predictImage</u>) y obtener la clase y probabilidad. 1.7 Implementar la regla de confianza insuficiente (≤ 60) y devolver el estado "reintentar".	La predicción devuelve una clase y una probabilidad. El servidor responde con "ok" y clasificacion ('B' o 'N') o "reintentar".

Desarrollo del Servicio de Actualización (Long Polling).	1.8 Crear la ruta /verificar_actualizacion para la actualización del frontend.	El endpoint espera hasta que haya una nueva clasificación disponible (ultima_actualizacion > timestamp_cliente) o hasta un timeout de 30 segundos.
	1.9 Implementar la lógica de Long Polling con un while True para esperar nuevos datos.	
	1.10 Actualizar la variable global ultima_actualizacion al recibir una nueva clasificación.	

Tabla 2 Construcción del Servidor Flask

4.2.2 Implementación del sistema mecánico.

El objetivo de este Sprint fue consolidar la capa física del sistema de clasificación. Se integraron y probaron los componentes esenciales bajo el control del ESP32-S3 CAM, asegurando su funcionamiento coordinado. Asimismo, se validó la comunicación HTTP bidireccional entre el dispositivo embebido y el servidor Flask, estableciendo así una interacción confiable entre el hardware y el backend del sistema.

Historia de Usuario / Elemento	Tareas Clave	Criterios de Aceptación
Integración del ESP32 con el Backend Mediante un Endpoint	<p>2.1 Configurar la conectividad Wi-Fi del ESP32.</p> <p>2.2 Inicializar la cámara (init_camera) y ajustarla para la captura de imagen.</p> <p>2.3 Implementar el envío de una notificación de detección.</p> <p>2.4 Implementar la función <u>enviarImagenComoBytes</u>.</p>	El ESP32 se conecta al Wi-Fi, la cámara inicializa con ajustes optimizados y el microcontrolador toma un frame y procesa la imagen en bytes para su incorporación en un JSON.

	leyendo la imagen de la cámara (esp_camera_fb_get) y serializándola en un array de bytes JSON.	
Implementación del Sistema Mecánico (Servomotor y Sensor Ultrasónico)	<p>2.5 Inicializar el servomotor y el sensor ultrasónico (NewPing sonar).</p> <p>2.6 Implementar la lógica de detección de objeto: si la distancia es menor a 10 cm y ha pasado un delay de 3 segundos desde la última detección.</p> <p>2.7 Implementar el control suave del servomotor (<u>moverServoSuave</u>) hacia los ángulos definidos (0 y 180).</p>	<p>El sensor ultrasónico activa la función <u>enviarDeteccionYClasificacion</u>.</p> <p>El servomotor se mueve suavemente al ángulo de clasificación y regresa a la posición central (100)</p>
Procesamiento de Respuesta y Feedback Visual (OLED).	<p>2.8 Implementar la inicialización robusta de la pantalla OLED.</p> <p>2.9 Implementar la función <u>procesarComando</u> para mover el servo al recibir 'B' o 'N' y mostrar el resultado en la OLED.</p>	<p>El ESP32 parsea la respuesta del servidor. Si recibe 'B' o 'N', mueve el servo e imprime "Basura Biodegradable/No-Biodegradable" en la OLED. Si hay un error, se notifica en la OLED.</p>
Modelado 3D para Visualización de Escalabilidad y Diseño Final	<p>2.10 Diseñar la estructura completa (carcasa y soporte) del prototipo final con dimensiones comerciales en software CAD.</p> <p>2.11 Generar renders de alta calidad que muestren el</p>	<p>El modelo 3D debe integrar todos los componentes electrónicos —ESP32, cámara, servomotor y sensor— junto con el mecanismo de separación de residuos de forma viable, reflejando además la capacidad de mejora del</p>

	prototipo industrializado y apto para producción. 2.12 Documentar las vistas clave del modelo 3D para su inclusión en el informe y como material para futuras presentaciones de financiamiento.	proyecto en entornos reales mediante una apariencia robusta y una estética moderna.
--	--	---

Tabla 3 Implementación del sistema mecánico

4.2.3 Integración del ESP32 con el backend mediante un endpoint.

El enfoque de este Sprint fue establecer la comunicación bidireccional robusta entre el ESP32 y el servidor Flask, utilizando la biblioteca HTTPClient y el formato de datos JSON. La clave fue desarrollar el protocolo de envío de imágenes capturadas como un array de bytes y la lógica de respuesta para procesar la clasificación y ejecutar la acción física.

Historia de Usuario / Elemento	Tareas Clave	Criterios de Aceptación
Implementación del Protocolo de Envío de Notificación.	<p>3.1 Configurar el endpoint HTTP del ESP32 usando la URL del servidor Flask (ej. <code>http://<ip_servidor>:5000/</code>).</p> <p>3.2 Implementar <code>enviarNotificacionDeteccion()</code> : Iniciar la conexión HTTP y enviar un POST al servidor con el evento "objeto identificado".</p> <p>3.3 Incluir el identificador del dispositivo (<code>WiFi.macAddress()</code>) en la notificación JSON.</p>	<p>El ESP32 se conecta directamente al servidor mediante un endpoint y este como primer parte envía una solicitud JSON inicial al servidor para indicar que hay un objeto y está listo para enviar la imagen.</p>

Envío de la Imagen de la Cámara como Bytes.	<p>3.4 Implementar <u>enviarImagenComoBytes</u> para obtener el frame buffer JPEG (camera_fb_t).</p> <p>3.5 Construir el payload JSON con el evento "imagen bytes" y crear un JsonArray para almacenar cada byte de la imagen.</p> <p>3.6 Enviar el payload JSON mediante HTTP POST al servidor con un timeout amplio (15 segundos).</p>	El servidor Flask recibe el array de bytes de la imagen de forma exitosa y devuelve un código de respuesta HTTP válido.
Manejo de Respuestas del Servidor.	<p>3.7 Procesar la respuesta HTTP del servidor y deserializar el JSON de respuesta (responseDoc).</p> <p>3.8 Analizar el status de la respuesta: si es "reintentar", mostrar el mensaje de confianza baja.</p> <p>3.9 Si el status es "ok", extraer el campo clasificacion y llamar a <u>procesarComando</u> para ejecutar la acción física ('B' o 'N').</p>	El ESP32 es capaz de distinguir entre una clasificación exitosa ("ok") y una solicitud de repetición ("reintentar") basada en la respuesta JSON del servidor.
Manejo de Errores de Conexión/JSON.	3.10 Implementar manejo de errores para timeout de HTTP y fallos en el código de respuesta.	Los errores de conexión o de formato JSON son detectados, reportados a la consola Serial, y se muestra un mensaje temporal en la pantalla OLED.

Tabla 4 Integración del ESP32 con el backend mediante un endpoint.

4.2.4 Elección del dataset y entrenamiento del modelo de IA

Este Sprint abordó la fase crucial de investigación y desarrollo del Modelo de Clasificación de Residuos. Se priorizó el uso de modelos de Transfer Learning tanto como MobileNetV2 y ResNet50 para acelerar el entrenamiento y optimizar los recursos. El trabajo incluyó la gestión y preparación de diferentes datasets (públicos y propios) y la ejecución de cuatro ciclos de entrenamiento iterativos para afinar la arquitectura y el rendimiento del modelo.

Historia de Usuario / Elemento	Tareas Clave	Criterios de Aceptación
Elección y Preparación de los Datasets.	4.1 Selección de un Dataset Público inicial (e.g., Waste Classification Data V2 en el notebook). 4.2 Creación de un Dataset Propio (REALDATA) de residuos bajo condiciones controladas para Fine-tuning. 4.3 Definición y aplicación de la Aumentación de Datos (ImageDataGenerator) para ambas fuentes, incluyendo rotación y zoom.	Los datasets están estructurados con directorios de entrenamiento/validación. El generador de imágenes está configurado para la normalización.
Entrenamiento 1: ResNet50 con Dataset Público.	4.4 Cargar la arquitectura ResNet50 (pesos 'imagenet'). 4.5 Congelar las capas base y añadir capas densas (GlobalAveragePooling2D, Dense con 2 clases) para clasificación binaria.	Se ejecuta el entrenamiento con el dataset público, y los resultados iniciales se guardan para comparar con los demás modelos.

	4.6 Compilar el modelo con el optimizador Adam y callbacks (ModelCheckpoint, ReduceLROnPlateau).	
Entrenamiento 2: MobileNetV2 con Dataset Público.	<p>4.7 Cargar y adaptar la arquitectura MobileNetV2 para la clasificación binaria (similar a 4.5).</p> <p>4.8 Ejecutar el entrenamiento con el mismo dataset público para comparar la precisión y velocidad con ResNet50.</p>	El modelo MobileNetV2 entrenado existe y su precisión de validación es registrada, para su posterior uso en pruebas.
Entrenamiento 3: MobileNetV2 con Dataset Personalizado Fusionado.	<p>4.9 Fusionar el Dataset "RealWaste Image Classification" con el Dataset "TrashNet" para crear el Dataset Personalizado.</p> <p>4.9 Cargar nuevamente la arquitectura del MobileNetV2.</p> <p>4.10 Entrenar el modelo MobileNetV2 con el Dataset Personalizado Fusionado.</p>	El modelo MobileNetV2 ha sido expuesto a la diversidad de imágenes de dataset públicos de fotografías reales y presenta un aumento en la precisión de validación.
Entrenamiento 4: Fine-tuning Iterativo por Dominio (MobileNetV2).	<p>4.11 En este entrenamiento se buscó aplicar la técnica de Dominio Iterativa (Iterative Domain Technique).</p> <p>4.12 Cargar el modelo anterior para conseguir un entrenamiento utilizando fine-tuning.</p>	El modelo es ajustado finamente para las condiciones reales de iluminación y perspectiva del proyecto. El modelo final (model_retrained_REALDATA_v2.h5) está listo para ser integrado en el backend.

	4.13 Reentrenar el modelo MobileNetV2 exclusivamente con el Dataset Propio (Imágenes propias y el dataset personalizado) a una tasa de aprendizaje baja para fine tuning.	
--	---	--

Tabla 5 Elección del dataset y entrenamiento del modelo de IA.

4.3 Diseño de la Arquitectura del Sistema.

4.3.1 Módulo C++: Esp32ToPythonOledHttpCAM.ino.

Este módulo constituye el firmware principal del sistema embebido, diseñado para operar sobre el microcontrolador ESP32-S3 CAM. Su función es orquestar la interacción entre el entorno físico y el servidor de inteligencia artificial, integrando la captura de imágenes, la detección de proximidad y la actuación mecánica en un flujo automatizado.

El código gestiona el ciclo completo de clasificación de residuos: monitorea constantemente un sensor ultrasónico para detectar la presencia de objetos, captura y serializa imágenes en tiempo real para su transmisión vía HTTP, e interpreta las respuestas de clasificación recibidas para controlar un servomotor. Este actuador segrega físicamente el material en sus respectivas categorías (biodegradable o no biodegradable), mientras el sistema proporciona retroalimentación visual continua sobre el estado de la conexión y el proceso mediante una pantalla OLED.

Paqueterías

WiFi.h: Para gestionar la conexión a la red inalámbrica del dispositivo.

HTTPClient.h: Permite al ESP32 actuar como cliente para enviar peticiones HTTP POST (incluyendo los bytes de la imagen) al servidor Python.

ArduinoJson.h: Esencial para la serialización de la imagen capturada en un array de bytes JSON para su envío, y para la deserialización de la respuesta del servidor.

esp_camera.h: Librería de bajo nivel para la configuración y captura de frames del sensor de cámara (OV2640/OV5640) del ESP32-CAM.

Adafruit_SSD1306.h / Adafruit_GFX.h: Para el control gráfico y del hardware de la pantalla OLED I2C, mostrando el estado del sistema y los resultados.

NewPing.h: Utilizada para realizar una lectura precisa y estable del sensor ultrasónico.

ESP32Servo.h: Para el control del servomotor que ejecuta la segregación física del material clasificado.

Variables predefinidas y su significado

Variable / Constante	Tipo	Propósito
SCREEN_WIDTH / SCREEN_HEIGHT	#define	Definen las dimensiones físicas de la pantalla OLED (128x64 píxeles) para el manejo gráfico.
I2C_SDA_PIN / I2C_SCL_PIN	#define	Asignan los pines GPIO 42 y 41 del ESP32-S3 para la comunicación I2C con la pantalla OLED.
TRIG_PIN / ECHO_PIN	#define	Definen los pines GPIO (1 y 2) conectados al sensor ultrasónico HC-SR04 para medir la distancia.
SERVO_PIN	#define	Pin GPIO 3 utilizado para enviar la señal PWM que controla la posición angular del servomotor.
MAX_DISTANCE	#define	Establece el umbral de distancia máxima (10 cm) para que el sensor ultrasónico dispare el evento de "Objeto Detectado".
CAM_PIN_	#define	Conjunto extenso de definiciones de pines que mapean las líneas de datos y

		control del sensor de cámara (OV2640/OV5640) al chip ESP32-S3.
ssid / password	const char*	Almacenan las credenciales necesarias (nombre y contraseña) para conectarse a la red WiFi local.
serverUrl	const char*	Dirección IP y puerto estático del servidor Flask (ej. http://192.168.0.121:5000/) al que el ESP32 enviará las peticiones HTTP.
displayIniciada	bool	Variable bandera que indica si la pantalla OLED se inicializó correctamente; se usa para evitar fallos de escritura si la pantalla no está conectada.

Tabla 6 Definición de variables iniciales ESP32

Configuración setup()

Componente/Objeto	Tipo/Librería	Descripción
Serial	HardwareSerial	Inicializa la comunicación serie a 115200 baudios para depuración en consola.
display	Adafruit_SSD1306	Inicializa el bus I2C y configura la pantalla OLED, verificando las direcciones 0x3C o 0x3D.
servoMotor	Servo (ESP32Servo)	Asigna el servomotor al pin GPIO 3 y lo coloca en una posición neutral inicial de 100 grados.
esp_camera	esp_camera.h	Configura los pines, frecuencia del reloj (20MHz) y ajustes de

		imagen (brillo/contraste) del sensor.
WiFi	WiFiClass	Establece la conexión a la red inalámbrica utilizando las credenciales (SSID/Password) definidas.

Tabla 7 Atributos del setup del microcontrolador

Métodos Principales

init_camera(): Configura los pines, la frecuencia y los ajustes de calidad de imagen (brillo, contraste, saturación) necesarios para inicializar el sensor de la cámara .

inicializarOLED(): Establece la comunicación I2C con la pantalla, verificando las direcciones de memoria (0x3C o 0x3D) para asegurar su correcto encendido .

mostrarMensajeTemporal(mensaje, tamano, tiempo): Despliega texto informativo en la pantalla OLED durante un intervalo definido para dar retroalimentación visual al usuario .

moverServoSuave(anguloObjetivo): Controla el servomotor para que se desplace hacia la posición deseada de manera incremental y fluida, evitando movimientos bruscos.

moverYVolverSuave(anguloObjetivo): Ejecuta la secuencia mecánica completa, llevando el servomotor a la posición de clasificación y retornándolo automáticamente a la posición neutra (100°)

procesarComando(cmd): Interpreta el carácter de clasificación recibido ('B' o 'N') y activa la función mecánica correspondiente para segregar el residuo.

enviarNotificacionDeteccion(): Envía una solicitud HTTP inicial al servidor para notificar que se ha detectado un objeto y preparar el backend .

enviarImagenComoBytes(): Captura la imagen, serializa sus bytes en un JSON, la envía al servidor y procesa la respuesta para determinar si se debe clasificar o reintentar .

enviarDeteccionYClasificacion(): Coordina el flujo lógico de red, ejecutando secuencialmente el envío de la notificación y posteriormente el envío de la imagen .

setup(): Inicializa todos los componentes del sistema (Serial, OLED, Servo, Cámara, WiFi) al arrancar el dispositivo .

loop(): Monitorea cíclicamente el sensor ultrasónico para detectar la presencia de objetos y detonar el proceso de clasificación si se cumplen las condiciones de distancia y tiempo .

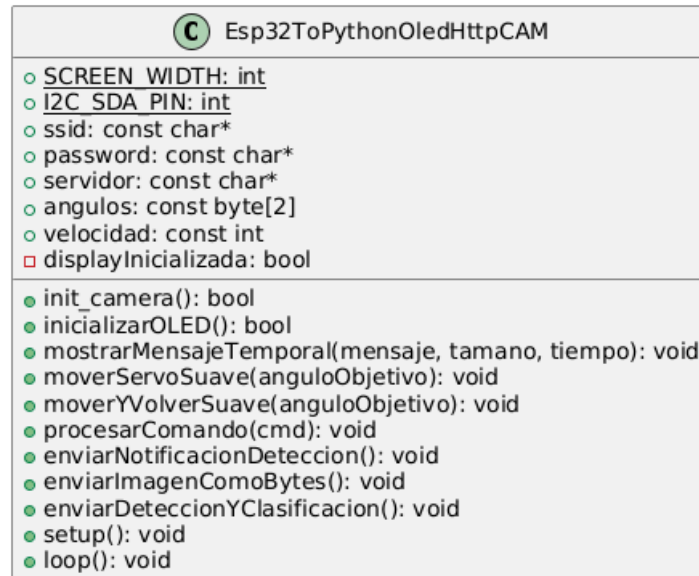


Ilustración 8 Diagrama de clases esp32

Lógica de init_camera()

Configuración de Estructura:

Crea un objeto de configuración (camera_config_t) y asigna los canales de temporizador LEDC.

Mapea todos los pines físicos de la cámara (D0-D7, XCLK, PCLK, VSYNC, HREF, SDA, SCL) a los pines correspondientes del ESP32-S3.

Parámetros de Captura:

Define la frecuencia de reloj XCLK en 20MHz y el formato de píxeles como JPEG.

Establece la resolución de imagen en 240x240 (FRAMESIZE_240X240) y la calidad de compresión en 10 (donde menor número es mayor calidad) .

Configura el almacenamiento del frame buffer en la memoria DRAM.

Inicialización del Driver:

Ejecuta esp_camera_init() con la configuración establecida.

Verifica si el resultado es distinto a ESP_OK; si hay error, imprime un mensaje y retorna false

Ajustes del Sensor:

Obtiene el puntero del sensor de la cámara para realizar ajustes finos.

Modifica los parámetros de imagen: establece la saturación en 0, reduce el brillo a -2 y aumenta el contraste a 1 para mejorar la definición de bordes .

Retorno:

Devuelve true indicando que la cámara está lista para operar.

Lógica de inicializarOLED()

Reinicio del Bus I2C:

Reinicia la comunicación Wire y la inicia explícitamente con los pines SDA (42) y SCL (41) definidos para este hardware.

Búsqueda de Dirección:

Intenta iniciar la pantalla en la dirección I2C 0x3C.

Si falla, realiza un segundo intento con la dirección 0x3D.

Confirmación Visual:

Si se logra la conexión, limpia el buffer de la pantalla, configura el texto y muestra el mensaje "OLED OK" por un segundo.

Manejo de Errores:

Si ambas direcciones fallan, imprime un error en serial y retorna false.

Lógica de mostrarMensajeTemporal(mensaje, tamano, tiempo)

Verificación de Pantalla:

Comprueba la variable displayInicializada para confirmar que la pantalla OLED se conectó correctamente durante la inicialización (setup).

Si la pantalla no fue inicializada, la función termina inmediatamente.

Preparación de Pantalla:

Limpia el buffer de la pantalla (`display.clearDisplay()`) para borrar cualquier mensaje previo.

Establece el color del texto en blanco (`display.setTextColor(SSD1306_WHITE)`) y la posición del cursor en el origen (0, 0).

Configuración de Texto:

Define el tamaño de la fuente (`display.setTextSize(tamano)`) basándose en el argumento de entrada.

Escritura:

Escribe el mensaje de entrada en el buffer de la pantalla mediante `display.print(mensaje)`.

Visualización:

Muestra el contenido del buffer en el hardware de la pantalla con `display.display()`.

Temporización:

Mantiene el mensaje visible durante el tiempo especificado por el parámetro tiempo (en milisegundos) usando la función `delay()`.

Limpieza Final:

Limpia de nuevo el buffer de la pantalla (`display.clearDisplay()`) después de la pausa temporal.

Lógica de moverServoSuave(anguloObjetivo)

Lectura de Posición Inicial:

Obtiene la posición actual del servomotor (en grados) usando `servoMotor.read()`.

Asigna esta posición inicial a la variable `anguloActual`.

Determinación de Dirección:

Compara el `anguloObjetivo` con el `anguloActual`.

Establece la dirección del movimiento (incremento) como +1 si el objetivo es mayor, o -1 si el objetivo es menor.

Bucle de Movimiento:

Inicia un bucle while que continúa mientras el `anguloActual` no haya alcanzado el `anguloObjetivo`.

Paso Incremental: Incrementa (o decrementa) el `anguloActual` en la cantidad definida por la velocidad (generalmente 5 grado por paso).

Control del Servo: Envía el nuevo valor del `anguloActual` al servomotor mediante `servoMotor.write(anguloActual)`.

Temporización:

Introduce un pequeño retardo (`delay(10)`) en cada paso para suavizar el movimiento y controlar la velocidad.

Finalización:

Una vez que el `anguloActual` alcanza o supera el `anguloObjetivo`, el bucle termina.

Asegura que el servo quede posicionado exactamente en el `anguloObjetivo` al salir del bucle.

Lógica de moverYVolverSuave(int anguloObjetivo)

Movimiento de Clasificación:

Invoca a la función `moverServoSuave(anguloObjetivo)`.

Esto causa que el servomotor se mueva desde su posición actual (generalmente 100°) hacia el ángulo de clasificación definido (0 para Biodegradable o 180 para No Biodegradable).

Pausa de Segregación:

Aplica una pausa de medio segundo (`delay(500)`).

Este retardo asegura que el servomotor permanezca en la posición de clasificación el tiempo suficiente para que el residuo caiga en el contenedor correspondiente.

Retorno a Posición Neutra:

Invoca nuevamente a la función `moverServoSuave(100°)`.

Esto hace que el servomotor regrese a la posición central (100°), dejando el sistema listo para la siguiente detección y evitando el bloqueo del mecanismo.

Lógica de procesarComando(cmd)

Validación de Entrada:

Recorre el arreglo de comandos válidos ({'B', 'N'}) para verificar si el carácter recibido coincide.

Ejecución Mecánica:

Si hay coincidencia, llama a moverYVolverSuave() pasando el ángulo correspondiente (0° o 180°) según el índice del comando.

Retroalimentación Visual:

Si el comando es 'B', muestra "Basura Biodegradable" en la OLED; si es 'N', muestra "Basura No-Biodegradable".

Manejo de Error:

Si el comando no es reconocido, muestra un mensaje de error en la pantalla indicando clasificación inválida.

Logica de enviarNotificacionDeteccion()

Inicialización HTTP:

Crea una instancia de HTTPClient y establece la conexión con la URL del servidor (serverUrl).

Configura la cabecera HTTP Content-Type como application/json para indicar el formato de los datos que se van a enviar.

Construcción JSON:

Crea un documento JSON estático (StaticJsonDocument) de 512 bytes, suficiente para la carga útil.

Define el par clave-valor {"evento": "objeto identificado"} en el documento JSON, que es la señal que espera el servidor para iniciar el proceso de clasificación.

Serializa el documento JSON a un String (jsonPayload) listo para la transmisión.

Envío y Validación:

Envía la petición HTTP POST con el jsonPayload serializado.

Almacena el código de respuesta HTTP (httpResponseCode).

Manejo de Errores:

Si el httpResponseCode es positivo (generalmente 200), la notificación se considera exitosa y se imprime el código en la consola serial.

Si el código es negativo (indicando un error de conexión o envío), imprime el error en la consola y muestra "Error Notificacion" en la pantalla OLED.

Limpieza:

Cierra la conexión HTTP con http.end() para liberar los recursos de red utilizados por la solicitud .

Logica de enviarImagenComoBytes()

Captura de Imagen:

Obtiene el frame buffer actual de la cámara mediante esp_camera_fb_get().

Introduce una pausa de estabilización de 500ms para asegurar la integridad de los datos.

Valida que la captura sea exitosa; si el puntero del buffer (fb) es nulo, aborta la función inmediatamente para evitar errores de memoria.

Serialización JSON:

Calcula el tamaño necesario para el documento JSON (tamaño de la imagen + 1024 bytes de cabecera) y crea un DynamicJsonDocument.

Define el tipo de evento como "imagen bytes" y agrega un campo "tamano_imagen" con la longitud total del buffer.

Crea un arreglo anidado (imagen_bytes) e itera byte por byte sobre el buffer de la imagen para llenarlo.

Transmisión HTTP:

Inicializa un cliente HTTP, configura la cabecera Content-Type: application/json y establece un tiempo de espera (timeout) de 15 segundos.

Serializa el documento JSON a un String y lo envía mediante una petición POST al servidor.

Procesamiento de Respuesta:

Si el código HTTP es positivo, descarga el cuerpo de la respuesta (http.getString()) e intenta deserializar el JSON recibido.

Manejo de Estados:

Caso "reintentar": Si el servidor devuelve confianza insuficiente, extrae el mensaje y el valor de confianza. Muestra "Confianza baja" en la pantalla OLED y aplica una espera de 3 segundos antes de continuar.

Caso "ok": Si la clasificación es exitosa, extrae el carácter de clasificación. Valida que sea exactamente un carácter ('B' o 'N') antes de llamar a procesarComando(). Si el carácter no es válido, muestra un error en pantalla.

Errores de Parseo: Si la respuesta no es un JSON válido o el estado es desconocido, alerta sobre "Error respuesta del servidor" en la OLED.

Error de Envío: Si la petición HTTP falla (código de error negativo), imprime la causa en el monitor serial y muestra "Error envio imagen" en la pantalla OLED.

Limpieza:

Libera obligatoriamente la memoria del frame buffer usando esp_camera_fb_return(fb) para prevenir fugas de memoria en el ESP32.

Cierra la conexión HTTP con http.end() para liberar recursos de red .

Lógica de enviarDeteccionYClasificacion()

Verificación de Conexión:

Comprueba el estado de la conexión WiFi (WiFi.status()).

Notificación al Servidor:

Invoca a la función `enviarNotificacionDeteccion()`.

Este paso es una señal rápida al servidor Python que establece el evento: "objeto identificado", indicando que la secuencia de clasificación va a comenzar.

Retraso:

Genera un retraso de 2 segundos para que haya un tiempo de espera para el vaciado del buffer del anterior frame.

Procesamiento de Imagen:

Invoca a la función `enviarImagenComoBytes()`.

Este paso es el flujo pesado que ejecuta la captura de imagen, la serialización en JSON, la transmisión HTTP y el manejo de la respuesta.

Actualización de Estado (Implícita):

Después de completar todo el proceso, el control regresa al `loop()`, y la variable `ultima_deteccion` se actualiza para evitar que el proceso se dispare inmediatamente de nuevo (debounce).

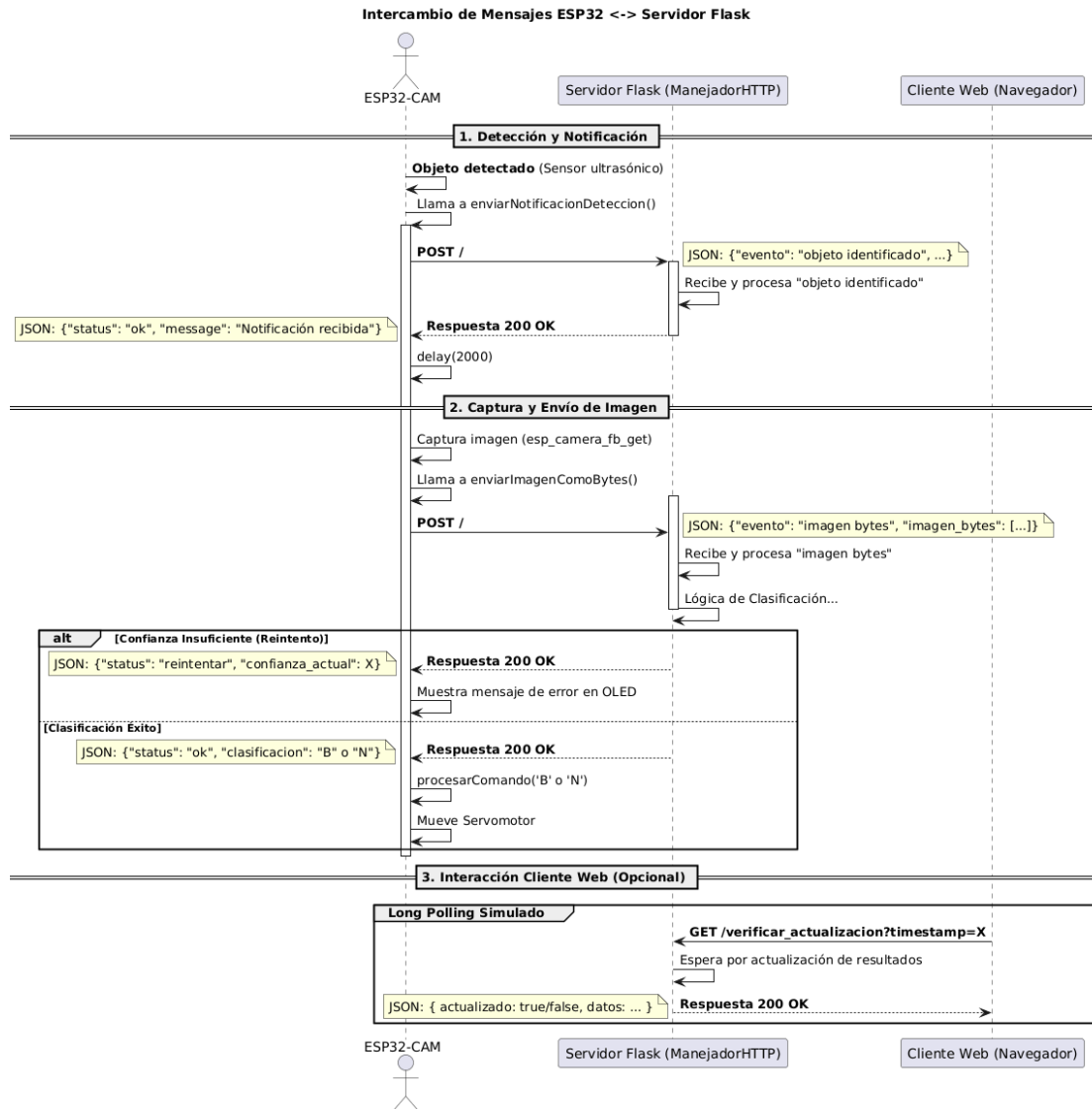


Ilustración 9 Diagrama secuencia de esp32 y servidor

Lógica de loop()

Monitoreo de Sensor:

Calcula el tiempo actual y realiza una lectura de distancia en centímetros usando el objeto sonar.

Evaluación de Condiciones:

Verifica tres condiciones simultáneas: que la distancia sea válida (>0), que esté dentro del rango máximo (10cm), y que hayan pasado más de 3 segundos desde la última detección.

Disparo de Acción:

Si se cumplen las condiciones, notifica en la pantalla "Objeto detectado".

Invoca la función `enviarDeteccionYClasificacion()` para iniciar el proceso de red .

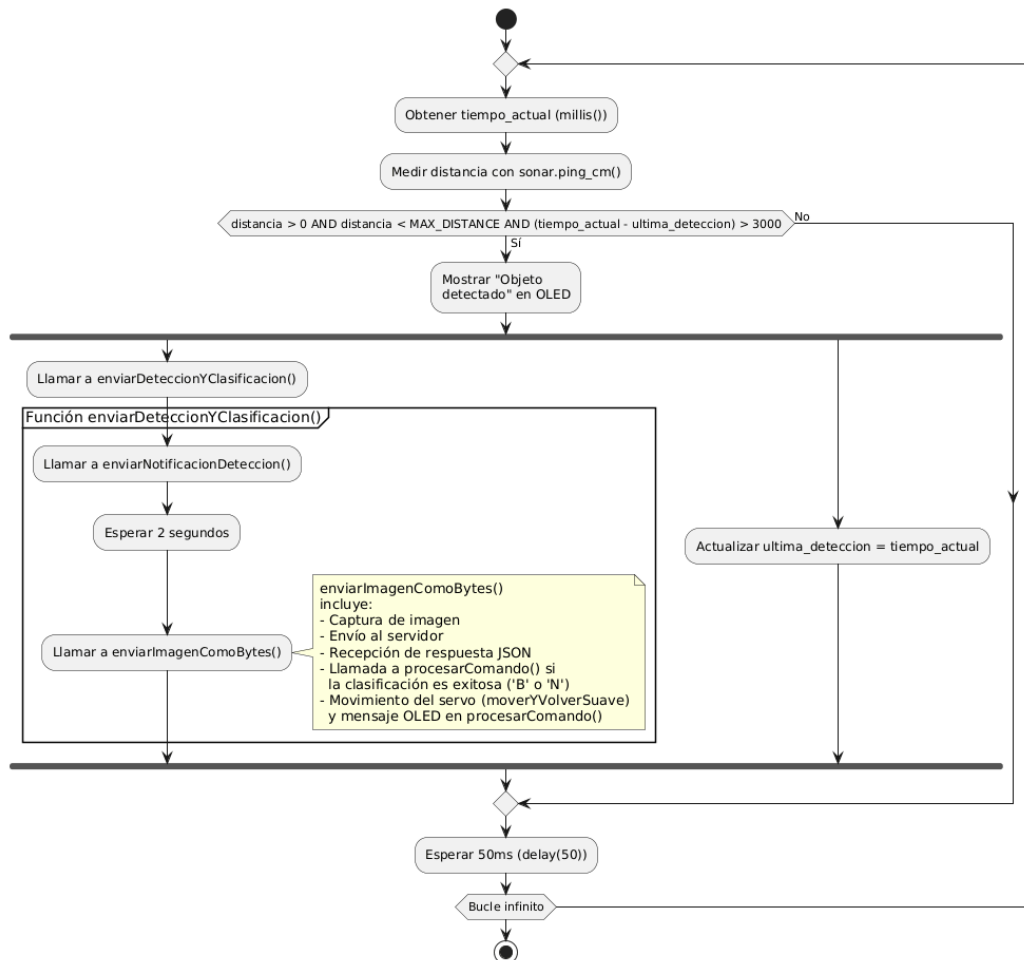


Ilustración 10 Diagrama de flujo del loop esp32

Lógica de setup()

Inicialización de Comunicación Serial:

Inicia la comunicación serial a 115200 baudios (`Serial.begin(115200)`) para permitir el debugging y la impresión de estados.

Inicialización de OLED:

Llama a inicializarOLED () para configurar el bus I2C y la pantalla.

Si la inicialización falla, imprime el mensaje de error "OLED Fallo" en la consola serial y aborta la ejecución para evitar errores posteriores.

Configuración del Servomotor:

Asigna el objeto servoMotor al pin GPIO 3 (SERVO_PIN) mediante servoMotor.attach().

Mueve el servo a la posición neutra inicial de 100 grados (servoMotor.write(100)) y añade un breve retardo de 500ms para estabilizar el componente.

Inicialización de la Cámara:

Muestra el mensaje "Iniciando camara" en la pantalla OLED.

Llama a la función init_camera() para configurar el sensor .

Validación de Cámara: Si init_camera() retorna false (fallo), imprime "Fallo inicializacion camara" y muestra "Error Camara" en la OLED. Si es exitosa, imprime y muestra "Camara OK".

Conexión WiFi:

Imprime "Conectando a WiFi..." en la consola y muestra "Conectando WiFi" en la OLED.

Inicia la conexión (WiFi.begin(ssid, password)) usando las credenciales definidas.

Bucle de Espera: Entra en un bucle que espera hasta que el estado sea WL_CONNECTED o hasta que se exceda el límite de 20 intentos (con una pausa de 500ms en cada intento).

Validación de WiFi:

Si la conexión es exitosa, imprime la dirección IP local en la consola y muestra "WiFi OK" junto con la IP en la pantalla OLED.

Si la conexión falla después de los 20 intentos, imprime "Error WiFi" en la consola y muestra un mensaje de error en la pantalla.

Finalización:

Finaliza el proceso de configuración con los mensajes "Sistema listo" en la consola y la pantalla OLED.

El programa procede a ejecutar la función loop().

Resumen de funcionalidad

Funcionalidad / Método	Descripción Técnica
Configuración del microcontrolador loadModel(ruta_modelo)	Inicialización de todo el hardware (WiFi, Cámara, OLED, Servomotor) al encender el dispositivo.
Ciclo de control loop()	Ciclo de control principal. Monitorea continuamente el sensor ultrasónico y, al detectar un objeto dentro del rango, dispara la secuencia de clasificación.
Iniciador de cámara init_camera()	Configura los pines, formato (JPEG 240x240), y ajusta los parámetros ópticos (brillo, contraste) del sensor de la cámara. Retorna true si es exitosa.
Iniciador de pantalla Okled inicializarOLED()	Inicializa la comunicación I2C y el driver de la pantalla OLED, comprobando las direcciones de memoria (0x3C/0x3D). Retorna true si la pantalla es detectada.
Controlador de mensajes mostrarMensajeTemporal(...)	Despliega un mensaje específico en la pantalla OLED, lo mantiene visible durante el tiempo definido y luego limpia el display .
Notificación inicial enviarNotificacionDeteccion()	Envía una petición POST inicial al servidor con el evento "objeto identificado" para alertar que el proceso de captura va a comenzar.
Captura y transformación de bytes enviarImagenComoBytes()	Captura la imagen, la serializa en un array JSON, la envía al servidor, procesa la respuesta (ok/reintentar) y libera el frame buffer.
Manejador de mensajes enviarDeteccionYClasificacion()	Función orquestadora que llama secuencialmente a enviarNotificacionDeteccion() y luego a enviarImagenComoBytes().
Procesamiento de la clasificación procesarComando(cmd)	Traduce el carácter recibido del servidor ('B' o 'N') a una acción mecánica, llamando a la función de movimiento del servo y actualizando la OLED.

Control suave de servo motor moverServoSuave(...)	Mueve el servomotor de su posición actual al ánguloObjetivo de forma gradual (incremental) para asegurar la suavidad del movimiento.
Regreso a la posición del servomotor moverYVolverSuave(...)	Ejecuta la secuencia mecánica completa: va al ángulo de clasificación, espera un momento y retorna a la posición neutra (100°).

Tabla 8 Funcionalidad de Esp32ToPythonOledHttpCAM

4.3.2 Módulo Python: appHTTP.py.

Este módulo implementa un servidor web utilizando el framework Flask para exponer el sistema de clasificación de imágenes y permitir su monitoreo en tiempo real.

El código integra la funcionalidad de la clase Manejador para procesar las peticiones de inferencia del modelo de inteligencia artificial (cargado desde un archivo .h5) y presentar los resultados en una interfaz gráfica. Está diseñado para gestionar la concurrencia y utiliza técnicas de Long Polling, garantizando que la visualización de los datos clasificados se actualice dinámicamente y al instante en el navegador del usuario sin necesidad de recargas manuales.

Paqueterías

Flask: Para el despliegue del servidor web, la gestión de solicitudes HTTP (GET/POST), la renderización de plantillas HTML y la estructuración de respuestas en formato JSON.

Time: Para el control de la ejecución temporal, gestión de timestamps y manejo de intervalos de espera en la lógica de actualización en tiempo real (Long Polling).

Módulo externo ManejadorHTTP: Interfaz personalizada que encapsula la lógica de TensorFlow/Keras, permitiendo la carga del modelo .h5 y la gestión de las predicciones fuera del hilo principal del servidor.

Variables predefinidas y su significado

Variable / Constante	Tipo	Propósito
app = Flask(__name__)	Instancia de Objeto Flask	Inicializar la aplicación web, configurar el servidor y gestionar el enrutamiento de las peticiones HTTP (GET/POST).
alpha = Manejador(modelo="")	Objeto de clase	Gestionar la lógica de IA, cargando el modelo .h5 y procesando las solicitudes de clasificación de imágenes recibidas.
ultima_actualizacion = 0	Int (luego Float)	Controlar la sincronización en tiempo real; actúa como una bandera temporal para que el cliente sepa cuándo actualizar la interfaz (Long Polling).

Tabla 9 Definición de variables iniciales appHTTP

Métodos Principales

Index(): Despliega la interfaz de usuario principal. Valida el estado actual del diccionario de identificación y renderiza la plantilla HTML (index.html), inyectando los resultados de la clasificación o un estado de espera inicial ("Esperando detección...") si no hay datos previos.

clasificar_objeto(): Procesa las solicitudes HTTP POST entrantes para la inferencia. Coordina la comunicación con el objeto **alpha** para recibir la imagen y ejecutar la predicción, actualizando posteriormente la marca de tiempo global (ultima_actualizacion) para sincronizar a los clientes conectados.

verificar_actualizacion(): Gestiona la comunicación en tiempo real mediante la técnica de Long Polling. Mantiene la conexión activa en un bucle controlado (con timeout de 30s) comparando timestamps, y retorna una respuesta JSON solo cuando detecta que el modelo ha generado una nueva clasificación.

if `__name__ == '__main__'`: Inicializa el servidor Flask en el puerto 5000. Obtiene la dirección IP dinámica mediante `alpha.getIpServidor()` y habilita el soporte para múltiples hilos (`threaded=True`), permitiendo manejar peticiones de clasificación y actualización simultáneamente.

Lógica de `Index()`:

Estado: "Inicial o Sin Detección"

Valida si el atributo `alpha.diccionarioIdentificacion` es `None` (aún no se ha procesado ninguna imagen).

Construye un diccionario local (`resultado_default`) estableciendo valores de espera: clase "Esperando detección...", probabilidad 0.0 y la ruta a una imagen de marcador de posición (`placeholder.jpg`).

Retorna la plantilla HTML renderizada (`index.html`) inyectando este diccionario por defecto para mostrar una interfaz de estado 'en espera'.

Estado: "Con Resultados Previos"

Se ejecuta implícitamente si `alpha.diccionarioIdentificacion` contiene datos.

Retorna la plantilla HTML (`index.html`), pasando directamente el objeto `alpha.diccionarioIdentificacion` al contexto de la vista para desplegar la imagen real procesada, la clase detectada y su porcentaje de certeza."

Lógica de `clasificar_objeto()`:

Acceso Global: Se declara la variable `ultima_actualizacion` como global para garantizar que su valor pueda ser modificado, habilitando el mecanismo de notificación de nuevos datos.

Recepción de Datos: Ejecuta el método `alpha.recepcionMensaje()` para parsear y preparar los datos de la imagen recibidos en el cuerpo de la petición POST del cliente.

Ejecución de Clasificación: Llama a `alpha.enviarMensaje()`, que dispara la lógica de inferencia del modelo de IA (contenida en la clase `ManejadorHTTP`) y obtiene el resultado de la clasificación.

Sincronización: Actualiza la variable `ultima_actualizacion` con el timestamp actual (`time.time()`), señalizando a los clientes que usan Long Polling que hay un nuevo resultado disponible.

Retorno: Devuelve la respuesta JSON generada por el manejador (que contiene el resultado de la clasificación o el estado del proceso) junto con el código de estado HTTP correspondiente.

Lógica de verificar_actualizacion():

Inicialización y Recepción de Parámetros

Obtiene el valor de la marca de tiempo del cliente (timestamp_cliente) a partir de los argumentos de la URL (request.args.get('timestamp')). Este valor representa el momento de la última actualización que el cliente posee.

Establece un límite de tiempo máximo de espera (timeout = 30 segundos) para evitar que la conexión del navegador se cuelgue o expire.

Almacena el tiempo actual en la variable inicio (time.time()) para monitorizar la duración del ciclo de espera.

Ciclo de Espera (Long Polling Bucle while)

El servidor entra en un bucle infinito que comprueba continuamente si la marca de tiempo global del servidor (ultima_actualizacion) es mayor que el timestamp_cliente recibido.

Si hay nuevos datos (el servidor predijo algo después de la última vez que el cliente preguntó), se cumple el bloque:

Confirma que **alpha.diccionarioidentificacion** contenga un resultado real (no el placeholder "Esperando detección...").

Retorna inmediatamente una respuesta JSON indicando 'actualizado': True, incluyendo el nuevo timestamp y los datos de clasificación (**alpha.diccionarioidentificacion**). El ciclo termina.

Condición de Timeout

Comprueba si el tiempo transcurrido desde el inicio ha superado el timeout (30 segundos).

Si Supera el Timeout Retorna una respuesta JSON indicando 'actualizado': False. Esto libera la conexión del navegador y le permite realizar una nueva petición de Long Polling inmediatamente después. El ciclo termina.

Control del Bucle

Si ninguna de las condiciones anteriores se cumple, el método induce una pausa mínima (time.sleep(0.5)) para reducir la carga de la CPU antes de volver a verificar las condiciones, manteniendo la conexión en espera.

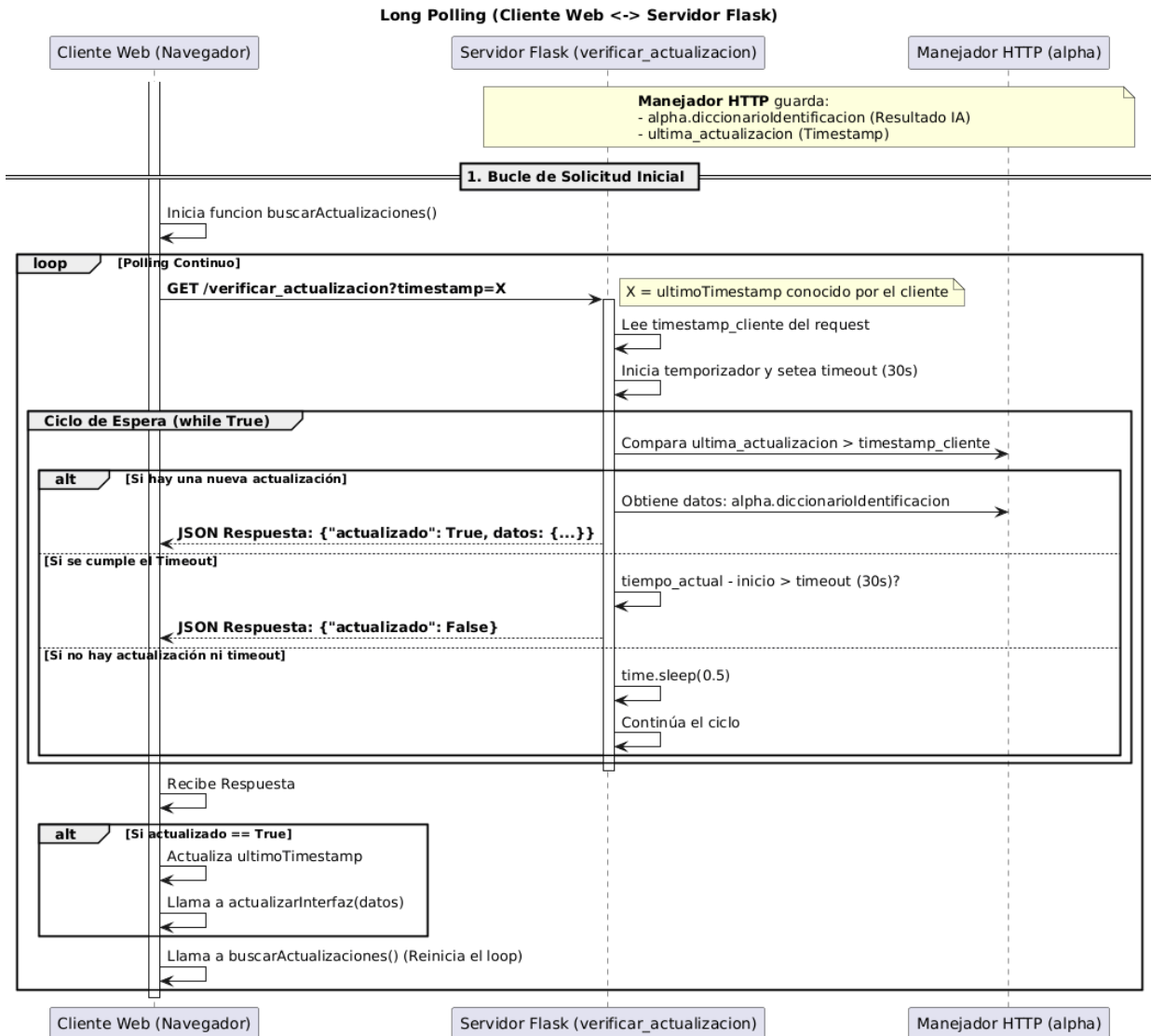


Ilustración 11 Diagrama secuencia Long Polling

Lógica de main():

Implementación de Manejo de Errores (try-except):

Intenta iniciar el servidor Flask para el servicio de inferencia.

Si el servidor no puede iniciarse (por ejemplo, si el puerto está ocupado o hay un problema de red), captura la excepción y ejecuta el código dentro del bloque.

Configuración e Inicio del Servidor:

Llama a `app.run()`, usando `alpha.getIpServidor()` para determinar la dirección IP de host y el puerto 5000 para la escucha.

`debug=False`: Deshabilita el modo de depuración de Flask, lo cual es apropiado para entornos de producción/despliegue.

`threaded=True`: Habilita el manejo multi-hilo, permitiendo que el servidor procese múltiples peticiones de clientes de manera concurrente (por ejemplo, atendiendo una clasificación POST mientras maneja una petición de Long Polling GET).

Manejo de Excepciones:

Si ocurre un error al iniciar el servidor (e.g., `PortAlreadyInUseError`), el bloque `except` lo captura.

Imprime un mensaje en la consola (`print(f"Error al iniciar el servidor: {e}")`) indicando que el servidor no pudo arrancar, mostrando el error específico que ocurrió.

Resumen de funcionalidad

Funcionalidad / Método	Descripción Técnica
Página principal <code>index()</code>	Muestra la página web principal (<code>index.html</code>). Si hay resultados de clasificación disponibles, los muestra; si no, muestra un mensaje de espera.
Manejador de respuestas del backend <code>clasificar_objeto()</code>	Recibe una solicitud y la procesa para realizar una clasificación o identificación. Luego, envía una respuesta con el resultado.

Confirmación de nuevo objeto verificar_actualizacion()	Permite que el cliente (navegador) compruebe si hay nuevos resultados de clasificación. Utiliza un bucle de espera (Long Polling) para enviar el resultado tan pronto como esté disponible, o después de un tiempo de espera.
Iniciador de servidor if __name__ == '__main__':	Es el punto de entrada principal del programa. Inicia el servidor web Flask para que la aplicación esté disponible en la red.

Tabla 10 Resumen funcionalidad main

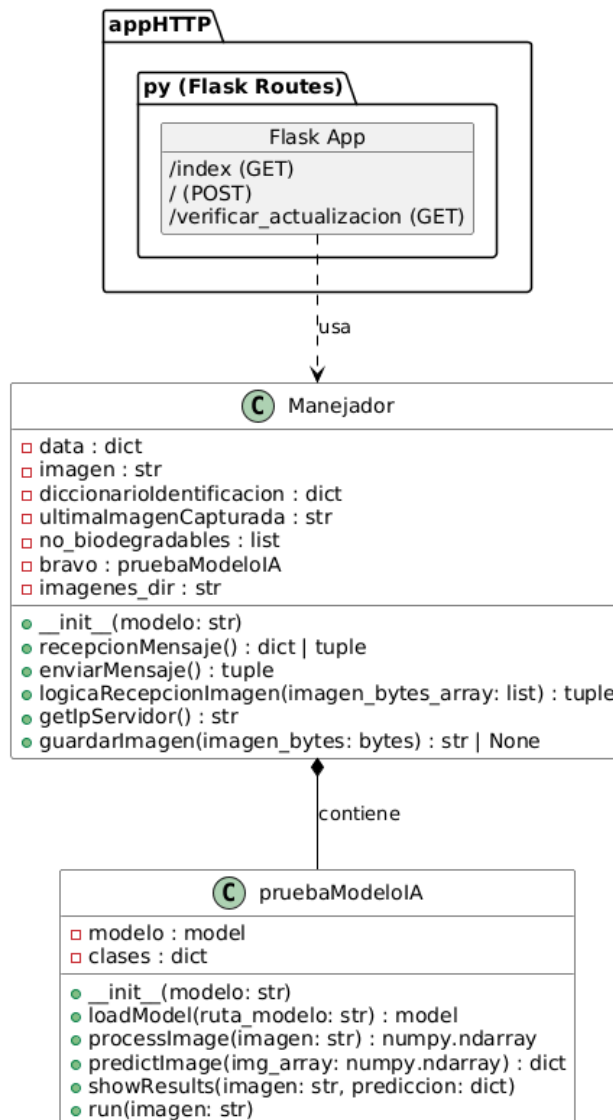


Ilustración 12 Diagrama de clases python

4.3.3 Módulo Python: pruebaModelosTF.py.

Este módulo permite realizar la clasificación automática de imágenes en dos categorías: biodegradable y no biodegradable, utilizando un modelo de inteligencia artificial previamente entrenado en TensorFlow/Keras.

El código encapsula la funcionalidad en una clase orientada a objetos (**pruebaModeloIA**) que gestiona la carga del modelo, el procesamiento de imágenes, la predicción y la visualización de resultados. Está diseñado para ser modular, fácil de usar y adaptable a distintos modelos de red neuronal preentrenados (ej. ResNet, MobileNet, etc.).

Paqueterías

TensorFlow / Keras : Para cargar y ejecutar el modelo entrenado.

NumPy : Manipulación de arreglos numéricos.

Matplotlib : Visualización de resultados.

OS : Manejo de rutas de archivos.

Clase: pruebaModeloIA

La clase pruebaModeloIA actúa como un módulo de procesamiento autónomo encargado de gestionar todo el flujo de clasificación de imágenes, desde la carga del modelo de Inteligencia Artificial hasta la visualización de los resultados. Su diseño orientado a objetos encapsula y organiza de forma secuencial las tareas esenciales del pipeline de la Red Neuronal Convolucional (RNC).

Inicialización (__init__)

Atributo	Tipo	Descripción
Self.modelo	Modelo de tensorflow	Guardar el modelo en una variable global lista para usarse en el sistema
Self.clases	array	Un diccionario que contiene las clases, sirve para mapear los resultados dados por el modelo de IA

Tabla 11 Atributos del constructo de "pruebaModelosIA"

Métodos Principales

loadModel(ruta_modelo): Carga el modelo de inteligencia artificial desde un archivo .h5 para su uso en las predicciones.

processImage(imagen): Preprocesa la imagen de entrada, adaptándola al formato y tamaño requeridos por el modelo.

predictImage(img_array): Realiza la predicción sobre la imagen procesada y determina la clase más probable junto con su nivel de confianza.

showResults(imagen, prediccion): Muestra la imagen original con el resultado de la clasificación superpuesto en pantalla.

run(imagen): Ejecuta el flujo completo de clasificación (procesamiento, predicción y visualización) en una sola llamada.

Lógica de loadModel (ruta_modelo)

Carga un modelo previamente entrenado en formato .h5 desde el almacenamiento local.

Usa la librería `tensorflow.keras.models.load_model()` para reconstruir la arquitectura, pesos y configuración del modelo.

Verifica que la carga sea exitosa antes de continuar.

Devuelve el modelo ya preparado para hacer inferencias.

Lógica de processImage (imagen)

Recibe la ruta de una imagen como entrada.

Abre la imagen y la redimensiona al tamaño de entrada del modelo.

Convierte la imagen a un arreglo NumPy y aplica normalización de valores.

Expande las dimensiones del tensor (de 3D a 4D) para que el modelo lo interprete como un lote de una sola imagen.

Devuelve la imagen procesada lista para ser clasificada.

Lógica de predictImage (img_array)

Recibe una imagen ya procesada por processImage().

Usa el modelo cargado para hacer una predicción (model.predict()).

Obtiene las probabilidades de cada clase (por ejemplo, biodegradable o no biodegradable).

Selecciona la clase con mayor probabilidad como resultado final.

Devuelve el nombre de la clase detectada y el porcentaje de confianza.

Lógica de showResults (imagen, prediccion)

Recibe la imagen original y la etiqueta predicha.

Usa OpenCV (cv2.putText) para escribir en la imagen el nombre de la clase y el nivel de confianza.

Muestra la imagen en una ventana gráfica (cv2.imshow).

Espera la interacción del usuario para cerrar la ventana.

Lógica de run (imagen)

Ejecuta de forma secuencial los métodos anteriores (`processImage()`, `predictImage()`, `showResults()`).

Controla el flujo completo desde la entrada de una imagen hasta la visualización del resultado.

Puede utilizarse como punto de entrada para automatizar pruebas o ejecuciones rápidas.

Resumen de funcionalidad

Funcionalidad / Método	Descripción Técnica
Carga del Modelo <code>loadModel(ruta_modelo)</code>	Carga el modelo de Inteligencia Artificial desde un archivo .h5 almacenado localmente. Utiliza <code>tensorflow.keras.models.load_model()</code> para reconstruir la arquitectura, pesos y configuración necesaria para realizar inferencias.
Preprocesamiento <code>processImage(imagen)</code>	Adapta la imagen de entrada a los requisitos del modelo. Esto incluye redimensionar la imagen, convertirla a un arreglo NumPy, normalizar los valores y expandir las dimensiones del tensor (de 3D a 4D) para simular un lote de una sola imagen.
Inferencia y Predicción <code>predictImage(img_array)</code>	Realiza la predicción utilizando la imagen procesada mediante <code>model.predict()</code> . Calcula las probabilidades de cada clase, selecciona la mayor y retorna tanto la etiqueta de la clase detectada como el porcentaje de confianza.
Visualización <code>showResults(imagen, prediccion)</code>	Muestra la imagen original con el resultado de la clasificación superpuesto. Utiliza OpenCV (<code>cv2.putText</code> y <code>cv2.imshow</code>) para escribir la clase y la confianza sobre la imagen y presentarla en una ventana gráfica.
Ejecución de Flujo <code>run(imagen)</code>	Actúa como orquestador, ejecutando secuencialmente el preprocesamiento, la predicción y la visualización en una sola llamada para automatizar el flujo completo.

Tabla 12 Funcionalidad de pruebaModelosTF

4.3.4 Módulo Python: ManejadorHTTP.py.

Este módulo forma parte del backend del sistema de clasificación inteligente de residuos. Es responsable de la comunicación entre el microcontrolador ESP32-CAM y el modelo de inteligencia artificial alojado en un servidor Flask.

Se ejecuta de manera local creando un endpoint para que se pueda conectar el microcontrolador ESP32, procesando peticiones HTTP y devolviendo respuestas con la clasificación del objeto detectado.

Paqueterías:

Datetime: Módulo estándar de Python para manipular fechas y horas.

io (BytesIO): Módulo para trabajar con flujos de E/S (Input/Output) en memoria (como si fueran archivos), pero usando cadenas o bytes.

Uuid: Genera Identificadores Únicos Universales (UUIDs), que son números aleatorios de 128 bits.

Os: para manejo de archivos y directorios.

Flask: Un framework liviano para construir aplicaciones web en Python.

request (de flask): Objeto que proporciona acceso a los datos entrantes de la petición HTTP, como el cuerpo del mensaje y el formato.

jsonify (de flask): Ayuda a crear respuestas JSON para las peticiones HTTP.

Módulo externo static.pruebaModelos: Contiene la clase **pruebaModeloIA**, encargada del preprocesamiento de imágenes y de la inferencia con el modelo entrenado.

processImage(imagen): Convierte bytes a formato compatible con TensorFlow.

predictImage(imagen_procesada): Retorna un diccionario con las claves "clase" y "probabilidad".

socket: Módulo estándar de Python que proporciona acceso a la interfaz de sockets de red.

Clase: Manejador

Esta clase gestiona la recepción de datos JSON del ESP32, coordina la inferencia del modelo de IA, maneja la persistencia de imágenes capturadas y formula la respuesta que se envía de vuelta al dispositivo.

Inicialización (__init__)

Atributo	Tipo	Descripción
modelo (Argumento)	str	Ruta al archivo del modelo de Keras/TensorFlow a cargar.
self.bravo.pruebaModeloIA	Objeto de clase	Instancia de la clase pruebaModeloIA que maneja la lógica de la IA.
self.data	dict o None	Contenido JSON de la última solicitud recibida.
self.diccionarioIdentificacion	dict o None	Último resultado completo de la predicción de la IA.
self.ultimaimagenCapturada	str o None	Ruta del sistema de archivos donde se guardó la última imagen.
self.imagenes_dir	str	Ruta de la carpeta donde se guardan las imágenes (por defecto: 'Programacion/static/imagenes').
self.imagen	str o None	Almacena la ruta temporal de la imagen actual que se está procesando.
self.no_biodegradables	list	Lista de etiquetas que se clasifican como 'N' ("No biodegradable", "plastico", "metal", "vidrio").

Tabla 13 Atributos del constructo de "manejadorHTTP"

Métodos Principales

recepcionMensaje(): Recepción y Validación. Verifica si la solicitud entrante es de formato JSON. Si es válido, almacena el contenido en self.data y lo devuelve. En caso contrario, retorna un error HTTP 400.

getIpServidor(): Utilidad de Red. Obtiene la dirección IP local del servidor de forma programática.

guardarImagen(imagen_bytes): Persistencia de Datos. Crea un nombre de archivo único utilizando un timestamp y un UUID, y guarda los bytes de la imagen JPEG recibida en el directorio configurado (self.imagenes_dir). Retorna la ruta completa del archivo guardado.

enviarMensaje(): Este método representa la Lógica Central del servidor Python. Procesa el contenido JSON que ha sido previamente almacenado en self.data por recepcionMensaje(), utilizando la clave "evento" para determinar el flujo de trabajo a seguir.

Flujo de errores y respuestas HTTP

Código HTTP	Descripción	Caso
200	Éxito	Clasificación completada o reintento solicitado
400	Solicitud inválida	Datos incompletos o formato JSON incorrecto
500	Error interno del servidor	Falla al procesar o guardar imagen

Tabla 14 Significado códigos http

Lógica de enviarMensaje()

El método enviarMensaje maneja dos flujos de trabajo principales, correspondientes a las dos peticiones HTTP que envía el ESP32:

Evento: "objeto identificado"

Imprime un mensaje en consola indicando que el proceso de detección ha comenzado.

Retorna {"status": "ok", "message": "Notificación recibida"} con un código de estado 200.

Evento: "imagen bytes"

Valida que el array de bytes ("imagen_bytes") exista en el JSON recibido; si no, retorna error 400.

Delega el procesamiento complejo llamando al método **self.logicaRecepcionImagen(imagen_bytes_array)**.

Implementa un bloque try-except para capturar errores imprevistos durante el procesamiento de imagen (retornando error 500 si falla).

Evento Desconocido

Si el evento no coincide con los anteriores, retorna un mensaje de error indicando "Evento no reconocido" (código 400).

Lógica de logicaRecepcionImagen (imagen_bytes_array)

Conversión de Datos:

Recibe el array de enteros proveniente de la solicitud JSON.

Convierte este array en un objeto de bytes puro para su manipulación.

Persistencia:

Invoca a `self.guardarImagen()` para almacenar el archivo físicamente y actualiza `self.ultimaImagenCapturada`.

Interacción con Modelo IA:

Llama a `self.bravo.processImage()` pasando la ruta de la imagen guardada para adaptarla a los tensores del modelo.

Ejecuta `self.bravo.predictImage()` para obtener el diccionario con la clase y la probabilidad.

Evaluación de Confianza (Umbral ≤ 0.60):

Si la probabilidad detectada es menor o igual al 60%, se considera insuficiente.

Actualiza `self.diccionarioIdentificacion` indicando "Objeto no identificado".

Retorna un JSON con estado "reintentar" y la confianza actual.

Clasificación Final (Umbral > 0.60):

Si la confianza es suficiente, verifica si la etiqueta detectada (ej. "vidrio", "plastico") se encuentra en la lista `self.no_biodegradables`.

Asigna "N" (No Biodegradable) si hay coincidencia, o "B" (Biodegradable) en caso contrario.

Retorna un JSON con estado "ok", la clasificación ("B" o "N") y la clase específica detectada.

Lógica de `recepcionMensaje()`

Validación JSON

Verifica si la solicitud entrante (`request`) está en formato JSON utilizando `request.is_json`.

Manejo de Error

Si no es JSON, imprime un mensaje de error en consola y retorna una respuesta JSON de error ("Se espera JSON") con el código de estado HTTP 400 (Bad Request).

Extracción de Datos

Si es JSON válido, extrae los datos del cuerpo de la solicitud utilizando `request.get_json()`.

Almacenamiento

Almacena los datos JSON extraídos en el atributo de instancia `self.data`.

Retorno Exitoso

Retorna los datos JSON almacenados (`self.data`).

Lógica de `getIpServidor()`

Creación del Socket

Intenta crear un socket de Internet (AF_INET) de datagrama (SOCK_DGRAM).

Conexión Simulación

Se conecta al servidor DNS de Google ("8.8.8.8", puerto 80). Nota: Esta conexión no envía datos, solo establece la ruta de red necesaria para que el sistema operativo asigne una IP de origen.

Obtención de IP

Recupera la dirección IP local utilizada para la conexión a través de `s.getsockname()[0]`.

Cierre del Socket

Cierra la conexión del socket (`s.close()`).

Retorno Exitoso

Retorna la dirección IP local obtenida.

Manejo de Error

Si ocurre cualquier excepción (ej. sin conexión de red), retorna la cadena "0.0.0.0"

Lógica de guardarImagen(imagen_bytes)

Verificación de Directorio

Verifica si el directorio de imágenes (`self.imagenes_dir`) existe.

Creación de Directorio

Si no existe, lo crea recursivamente (`os.makedirs`).

Generación de Nombre Único

Genera un nombre de archivo único componiendo:

Un timestamp de la hora actual (`YYYYMMDD_HHMMSS`).

Un identificador único de 8 caracteres (`uuid.uuid4()[8]`).

El prefijo "waste_" y la extensión ".jpg".

Construcción de Ruta

Combina el directorio (`self.imagenes_dir`) y el nombre de archivo (`filename`) para obtener la ruta completa del archivo (`filepath`).

Escritura del Archivo

Abre el archivo en modo de escritura binaria ('wb') y escribe los `imagen_bytes` proporcionados en el archivo.

Retorno Exitoso

Retorna la ruta completa (filepath) donde se guardó la imagen.

Manejo de Error

Si ocurre una excepción durante el proceso (ej. error de I/O), imprime el error en consola y retorna None.

Resumen de Funcionalidad

Funcionalidad / Método	Descripción Técnica
Inicialización __init__(modelo)	Configura el manejador instanciando la clase pruebaModeloIA con el modelo especificado. Define las rutas de almacenamiento y establece la lista de categorías consideradas "No biodegradables" (plástico, metal, vidrio).
Validación de Solicitud recepcionMensaje()	Verifica que la solicitud HTTP entrante tenga el formato correcto (application/json). Si es válida, extrae y almacena los datos; de lo contrario, rechaza la petición con un código de estado 400.
Enrutamiento de Eventos enviarMensaje()	Actúa como el controlador principal. Analiza la clave "evento" del JSON recibido para determinar si se trata de una simple notificación ("objeto identificado") o de un procesamiento de imagen ("imagen bytes"), delegando la lógica o respondiendo errores según corresponda.
Lógica de Clasificación logicaRecepcionImagen(bytes)	Ejecuta el flujo crítico: convierte el array de bytes a imagen, la guarda en disco, invoca al modelo de IA y aplica la lógica de negocio. Determina si el residuo es "B" o "N" basándose en la clase detectada y el umbral de confianza (> 0.60).
Persistencia de Archivos guardarImagen(imagen_bytes)	Genera un nombre de archivo único utilizando un timestamp y un UUID para evitar colisiones. Escribe los datos binarios de la imagen en el

	directorio static/imagenes para su posterior auditoría o reentrenamiento.
Utilidad de Red getIpServidor()	Determina la dirección IP local del servidor abriendo una conexión socket temporal (sin envío de datos reales) hacia un servidor DNS externo (8.8.8.8). Permite identificar dinámicamente dónde está alojado el servicio.

Tabla 15 Funcionalidad: ManejadorHTTP

4.4 Diseño del Modelo de Inteligencia Artificial

A lo largo del desarrollo del prototipo se crearon dos interfaces principales de inteligencia artificial, las cuales son: ResNet50 y MobileNetV2.

4.4.1 Elección de arquitectura de inteligencia artificial.

La fase inicial del desarrollo se centró en la implementación de un modelo de inteligencia artificial (IA) basado en la arquitectura ResNet50. Se partió de la hipótesis de que la reconocida capacidad de generalización y la profundidad de capas de esta red neuronal convolucional (CNN) proporcionarían la precisión predictiva necesaria para la tarea de procesamiento de imágenes del prototipo.

No obstante, las pruebas experimentales revelaron limitaciones significativas en la viabilidad operativa de ResNet50 para este sistema específico. Se observó una desproporción entre el costo computacional y la ganancia en precisión, lo que resultaba en tiempos de inferencia elevados que afectaban la latencia de respuesta hacia el microcontrolador ESP32-S3.

Ante este escenario, se determinó migrar hacia una arquitectura optimizada para la eficiencia sin sacrificar significativamente la precisión: MobileNetV2. La transición a este modelo demostró una mejora sustancial en el rendimiento del sistema, logrando un equilibrio óptimo entre velocidad de procesamiento en el servidor y exactitud en la clasificación. En consecuencia, se estableció

MobileNetV2 como la arquitectura definitiva para la etapa de inferencia, procediendo posteriormente a la selección y curación del conjunto de datos (dataset) funcional.

4.4.2 Selección y Evolución del Dataset

El rendimiento de un modelo de aprendizaje profundo es intrínsecamente dependiente de la calidad y representatividad de los datos con los que se entrena. Para este proyecto, la conformación del conjunto de datos (dataset) no fue estática, sino que siguió un proceso iterativo de cuatro etapas, evolucionando desde datos genéricos hasta una adaptación específica al dominio del prototipo.

Primera Iteración: Datos Sintéticos y de Stock

En una primera instancia, la fase de entrenamiento se fundamentó en el conjunto de datos "Non and Biodegradable Material Dataset", curado por Rayhan Zamzamy. Este dataset se utilizó inicialmente para entrenar el modelo preliminar (basado en ResNet50). Sin embargo, las pruebas de campo evidenciaron una deficiencia crítica: el modelo sufría de una alta tasa de error al procesar imágenes capturadas por el prototipo. Se determinó que la causa raíz era la naturaleza de las imágenes del dataset: fotografías de stock con iluminación perfecta y fondos limpios, aunque eran un gran número de imágenes aun presentaban una discrepancia significativa respecto a las condiciones reales de operación del sistema.

Segunda Iteración: Transición a Imágenes Reales (TrashNet)

Reconociendo la debilidad de los datos sintéticos, se procedió a sustituir la fuente de entrenamiento por el dataset "TrashNet" de Feyza Ozkefe. Este conjunto representó una mejora cualitativa al contener fotografías de residuos reales en lugar de imágenes de stock. A pesar de utilizar objetos físicos reales, el modelo resultante aún presentaba brechas importantes en la clasificación. El análisis sugirió que, aunque los objetos eran correctos, el entorno (fondo, ángulo y distorsión de la cámara del ESP32-S3) difería demasiado del entorno controlado en el que se capturó TrashNet.

Tercera Iteración: Fusión de Datos

Con el objetivo de mejorar la capacidad de generalización del modelo, se implementó una estrategia de fusión de datos. Se integró el dataset "RealWaste Image Classification" de Joakim Arvidsson junto con el conjunto "TrashNet" previamente utilizado. Esta fusión incrementó el volumen de datos y la

variabilidad de las muestras, logrando un mejor rendimiento teórico en las métricas de validación. No obstante, al desplegar el modelo en el prototipo final, la identificación de los tipos de objetos continuaba fallando en escenarios específicos. El sistema carecía de la capacidad para interpretar las particularidades visuales únicas del contenedor y la cámara del dispositivo.

Cuarta Iteración: Adaptación de Dominio y Fine-Tuning

Finalmente, para la cuarta y definitiva iteración, se optó por una estrategia de Adaptación de Dominio Iterativa (Iterative Domain Technique). Esta metodología consistió en la recolección manual de un subconjunto de datos personalizado ("custom dataset"), capturando 200 imágenes por categoría directamente desde la cámara del ESP32-S3 en su entorno de operación real.

Utilizando este nuevo conjunto de datos, se aplicó una técnica de Ajuste Fino (Fine-Tuning) sobre la arquitectura MobileNetV2 previamente seleccionada. Este proceso permitió:

- **Retener el conocimiento previo:** Se conservaron los pesos y características aprendidas de los datasets anteriores (TrashNet + RealWaste).
- **Adaptación al entorno:** La red neuronal ajustó sus capas finales para reconocer los patrones específicos de iluminación, perspectiva y ruido del sensor del prototipo.

El resultado fue un modelo capaz de reconocer patrones del mundo real con alta precisión, validando la hipótesis de que la especificidad de los datos es superior al volumen de los mismos

4.4.3 Estrategia y Procesamiento del Conjunto de Datos

La gestión de los datasets iniciales se vio influenciada por la curva de aprendizaje en el equipo, mientras que las etapas finales aplicaron técnicas de ingeniería de datos más rigurosas.

Primera Iteración: Procesamiento Nulo (ResNet50)

El conjunto de datos inicial, utilizado para la arquitectura ResNet50, se integró directamente al flujo de entrenamiento sin modificaciones de preprocesamiento o redimensionamiento. Esta decisión se basó en la premisa errónea de que las plataformas de frameworks de deep learning (como TensorFlow o PyTorch) realizarían el redimensionamiento necesario de forma automática o que un conjunto de datos extremadamente grande sería suficiente para que pudiera interpretar el mundo como se debe sin embargo el preprocesamiento no se realizó debido a la idea que requeriría un tiempo de procesamiento excesivamente largo.

El único paso realizado fue la creación de un dataframe para mapear la ruta de las imágenes a sus etiquetas. Esta ausencia de preprocesamiento contribuyó a los resultados subóptimos observados, ya que el modelo se vio obligado a procesar imágenes de dimensiones variables e inconsistentes, afectando la estabilidad de los gradientes durante el entrenamiento.

Segunda Iteración: Procesamiento Nulo (TrashNet)

El segundo conjunto de datos, TrashNet, se incorporó al modelo con la misma omisión de preprocesamiento que en la etapa inicial. La persistencia en esta estrategia, fundamentada en la falta de experiencia técnica en ese momento, resultó en un rendimiento deficiente.

Se comprobó que ignorar la estandarización de las dimensiones de entrada impedía que la red neuronal convergiera eficientemente, confirmando que la consistencia dimensional es un requisito fundamental para el entrenamiento efectivo de CNNs.

Tercera Iteración: Desarrollo del Script de Fusión y Redimensionamiento

La deficiencia en las iteraciones previas condujo al desarrollo de un script dedicado de preprocesamiento de datos. Este script se diseñó para la fusión y estandarización de los conjuntos TrashNet y RealWaste Image Classification, buscando corregir dos problemas principales:

- **Uniformidad Categórica y Reclasificación:** El dataset RealWaste Image Classification contenía originalmente nueve categorías, mientras que TrashNet contenía seis. Dado que el objetivo del proyecto era la clasificación binaria (Biodegradable (B) o No Biodegradable (N)), fue necesario implementar un esquema de mapeo categórico para reubicar cada imagen en las dos carpetas específicas finales (B o N).
- **Redimensionamiento y Consistencia Dimensional:** Se determinó que las dimensiones de entrada requeridas para la arquitectura MobileNetV2, con el objetivo de optimizar la inferencia y el balance de memoria, eran de 224×224 píxeles. El script automatizó el proceso de redimensionamiento de todas las imágenes.

Este dataset fusionado y preprocesado se dividió en ≈ 2000 imágenes para entrenamiento y ≈ 2000 para prueba, con una estructura de nombres de archivo estandarizada (e.g., biodegradable0001.jpg), facilitando la correcta ingesta de datos por parte de las funciones de carga de la plataforma de entrenamiento.

Cuarta Iteración: Preprocesamiento de Datos Custom

Para la cuarta y última iteración (Adaptación de Dominio), el dataset custom recolectado se integró por separado para el fine-tuning. La organización categórica se mantuvo desde la captura fotográfica, con las imágenes siendo clasificadas manualmente por el equipo de desarrollo directamente en sus carpetas B o N.

Aunque este dataset se mantuvo separado del conjunto fusionado de la tercera iteración para asegurar una prueba de validación de dominio más limpia, también fue sometido al mismo script de preprocesamiento para redimensionar las imágenes a 224×224 , manteniendo la consistencia dimensional con los pesos pre-entrenados del modelo MobileNetV2.

Justificación de la Redimensión 224×224

La elección de la dimensión 224×224 no fue arbitraria, sino que se basa en el requisito estándar de la mayoría de las arquitecturas de CNN pre-entrenadas, incluyendo MobileNetV2.

- **Requisito de Modelo Base:** La arquitectura MobileNetV2 fue originalmente entrenada en el dataset ImageNet utilizando 224×224 como dimensión de entrada. Utilizar este mismo tamaño en el fine-tuning asegura que las capas convolucionales puedan interpretar y procesar las características de las imágenes de entrada de manera consistente con los pesos pre-entrenados, maximizando el beneficio del Aprendizaje por Transferencia.
- **Eficiencia Computacional:** Dimensiones mayores aumentarían la carga computacional ($O(N^2)$) incrementando el consumo de memoria RAM y el tiempo de entrenamiento, mientras que dimensiones menores comprometerían la calidad de la información visual.

4.4.4 Scripts de procesamiento

El procesamiento y la curación del conjunto de datos fusionado (TrashNet + RealWaste Image Classification) se gestionaron mediante el desarrollo de dos scripts personalizados en Python, diseñados para automatizar la estandarización categórica y la división en conjuntos de entrenamiento y prueba. Estos scripts aseguraron la estructura homogénea requerida para el proceso de fine-tuning con MobileNetV2.

AcomodoDataset.py (Reclasificación y Reorganización Categórica)

Este script fue el responsable de la etapa crítica de adaptación de dominio, realizando las siguientes funciones:

- **Mapeo Categórico:** Se implementó un mapeo que relacionó las categorías originales de los datasets con las dos categorías binarias del proyecto: B (Biodegradable) y N (No Biodegradable).
- **Movimiento de Archivos:** El script iteró sobre la ubicación original de todas las imágenes. Con base en el mapeo definido, movió físicamente cada archivo a su carpeta de destino (/B o /N), asegurando que la estructura de directorios reflejara con precisión las etiquetas binarias del problema.
- **Redimensionamiento:** En esta etapa, las imágenes también fueron redimensionadas a las dimensiones estandarizadas de 224×224 píxeles (como se detalló en la sección anterior), garantizando la consistencia dimensional para la entrada al modelo MobileNetV2.

Diagrama de Flujo del Script AcomodoDataset.py

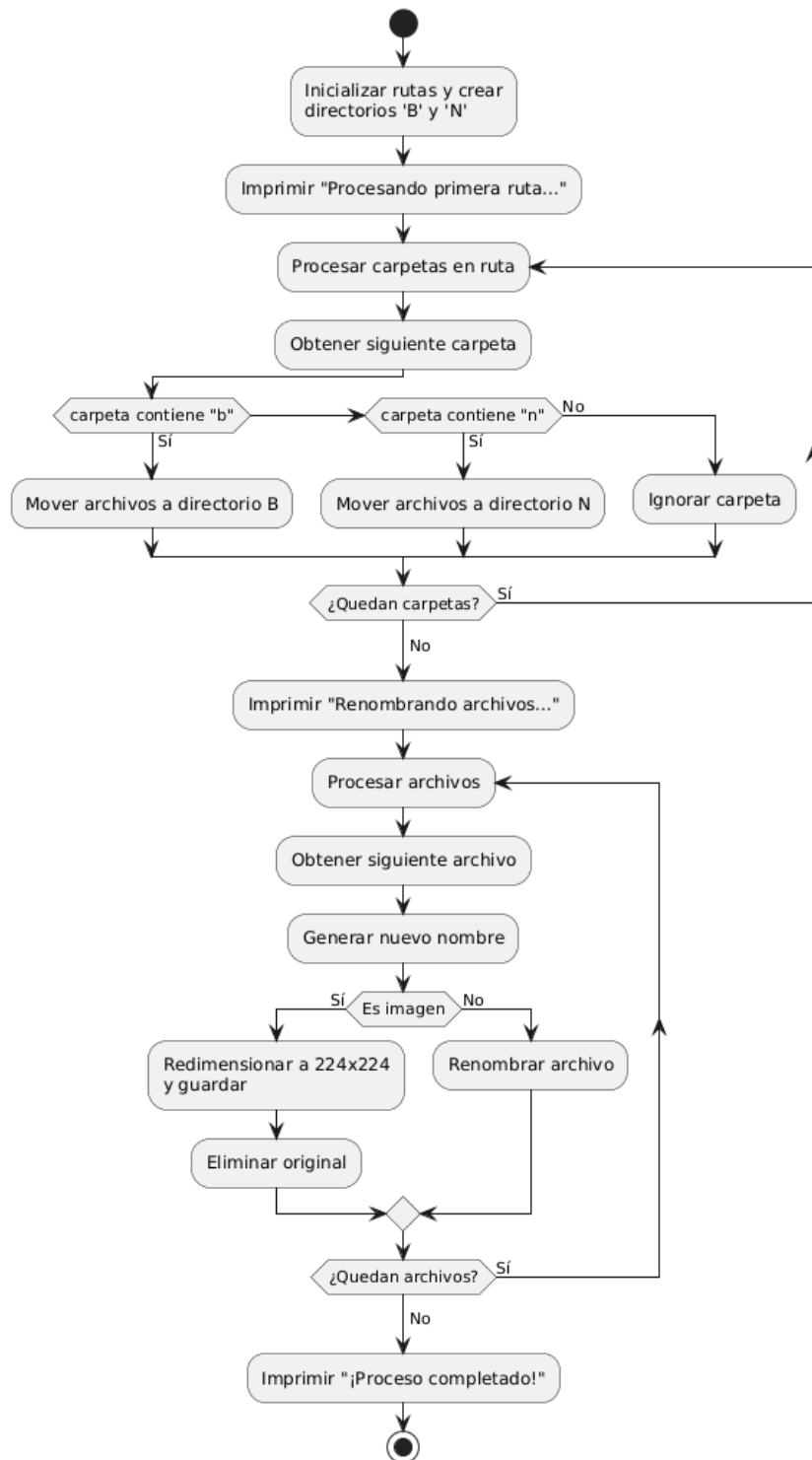


Ilustración 13 Diagrama de flujo AcomodoDataset

AcomodoDatasetTest.py (División de Entrenamiento y Prueba)

Una vez que todas las imágenes estaban correctamente clasificadas en las carpetas /B y /N, este script se encargó de la división controlada del conjunto de datos, asegurando una distribución equitativa para evitar sesgos de clase:

- **Asignación de Entrenamiento:** De manera sistemática, el script seleccionó 2000 imágenes de la carpeta /B y 2000 imágenes de la carpeta /N para ser asignadas al conjunto de Entrenamiento. Esta cantidad se consideró suficiente para adaptar los pesos del modelo pre-entrenado.
- **Asignación de Prueba:** Las imágenes restantes de ambas categorías (aquellas no seleccionadas para entrenamiento) se destinaron automáticamente al conjunto de Prueba (Test). Este conjunto final se utilizó para evaluar el rendimiento del modelo en datos no vistos, garantizando una métrica de generalización objetiva.

La separación estricta y cuantificada de los conjuntos de entrenamiento y prueba fue esencial para validar la capacidad predictiva del modelo en condiciones reales y evitar el sobreajuste (overfitting).

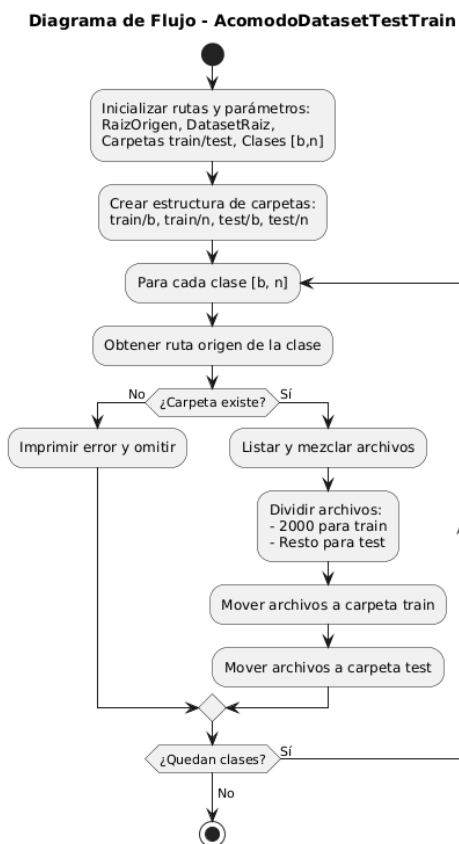


Ilustración 14 Diagrama de flujo AcomodoDatasetTestTrain

4.4.5 Scripts de entrenamiento

Una vez completado el procesamiento y la curación final del conjunto de datos, la siguiente fase crítica fue seleccionar la infraestructura adecuada para el entrenamiento del modelo MobileNetV2. Se optó por utilizar una plataforma de computación en la nube para garantizar la eficiencia, estabilidad y acceso a recursos de hardware especializado.

Elección de la Plataforma de Entrenamiento

Se seleccionó Google Colaboratory (Google Colab) como el entorno principal para el entrenamiento del modelo de inteligencia artificial. Google Colab es un servicio de notebooks Jupyter basado en la nube que proporciona acceso gratuito a Unidades de Procesamiento Gráfico (GPU) y Unidades de Procesamiento de Tensor (TPU).

La decisión de utilizar esta infraestructura de nube se justificó por las siguientes consideraciones:

- **Acceso Optimizado a Recursos de Cómputo:** El entrenamiento de redes neuronales convolucionales profundas como MobileNetV2 requiere una alta capacidad de paralelización, la cual es proporcionada de manera eficiente por las GPU virtuales de Colab.
- **Minimización de Configuración Local:** El uso de Google Colab eliminó la necesidad de complejas configuraciones de software, dependencias de frameworks y la gestión de drivers en equipos locales.

Es importante notar que el registro detallado del proceso de entrenamiento en la plataforma Google Colab abarca solo tres de las cuatro iteraciones de datos descritas previamente, debido a las limitaciones de registro iniciales. No obstante, las métricas y conclusiones de las cuatro iteraciones son conocidas y se presentarán detalladamente en el Capítulo 5 (**Pruebas y resultados**).

El enfoque del siguiente texto se centrará únicamente en los scripts y el código fuente desarrollado para la definición, configuración y ejecución del entrenamiento del modelo MobileNetV2 y ResNet50.

Paqueterías

tensorflow y tensorflow.keras: Son el framework fundamental para el aprendizaje automático y el aprendizaje profundo. TensorFlow gestiona la computación de gráficos de flujo de datos a gran escala, mientras que Keras es su API de alto nivel, utilizada para definir, configurar, entrenar y evaluar modelos de redes neuronales de manera eficiente y modular.

Modelos Preentrenados (ResNet50, MobileNetV2): Se importan estas arquitecturas preentrenadas en el dataset ImageNet para aplicar la técnica de Aprendizaje por Transferencia (Transfer Learning). Estas bases convolucionales permiten reutilizar características de bajo y medio nivel aprendidas de millones de imágenes, acelerando la convergencia del modelo para la tarea específica de clasificación de residuos.

Capas y Modelos (Dense, Dropout, GlobalAveragePooling2D, Model): Los módulos layers y models se utilizan para personalizar las arquitecturas preentrenadas. Dense define las capas de salida, Dropout previene el sobreajuste (overfitting) al apagar neuronas aleatoriamente, y GlobalAveragePooling2D reduce las dimensiones espaciales de los mapas de características. Model permite construir y ensamblar el modelo final a partir de estas capas.

Preprocesamiento de Imágenes (preprocess_input_resnet, preprocess_input_mobilenet, ImageDataGenerator): Estas funciones son esenciales para normalizar los píxeles de las imágenes de entrada según las especificaciones de los modelos ResNet50 y MobileNetV2. ImageDataGenerator es una herramienta crucial para el Aumento de Datos (Data Augmentation), aplicando transformaciones en tiempo real (rotaciones, zoom, inversión) para aumentar la variabilidad del conjunto de entrenamiento y robustecer el modelo.

Entrenamiento y Control (Adam, Callbacks): Adam es el optimizador utilizado para ajustar los pesos del modelo durante el entrenamiento. Los Callbacks (ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, CSVLogger) son mecanismos que permiten automatizar acciones durante el entrenamiento, tales como guardar el mejor modelo, ajustar la tasa de aprendizaje y detener el proceso anticipadamente si no hay mejora, asegurando una gestión eficiente del workflow.

pandas (pd) y numpy (np): NumPy es el estándar para la computación científica en Python, permitiendo operaciones eficientes con matrices y vectores. Pandas construye sobre NumPy, proporcionando estructuras de datos de alto nivel como DataFrames, esenciales para la carga, limpieza y gestión estructurada de los metadatos del conjunto de imágenes.

matplotlib.pyplot (plt) y seaborn (sns): Son las librerías primarias de visualización. Matplotlib es la base para la creación de gráficos estáticos, mientras que Seaborn ofrece una interfaz de alto nivel

para generar visualizaciones estadísticas atractivas y complejas, como los gráficos de las curvas de entrenamiento o las matrices de confusión.

Métricas de Clasificación (`precision_score`, `recall_score`, `f1_score`, etc.): Estas funciones se emplean para calcular las métricas de rendimiento más relevantes en problemas de clasificación binaria, ofreciendo una visión completa de la capacidad predictiva del modelo más allá de la precisión simple (`accuracy`). La `confusion_matrix` y el `classification_report` son fundamentales para analizar los tipos de errores (falsos positivos y falsos negativos).

Validación y Balanceo (`train_test_split`, `class_weight`, `resample`): `train_test_split` permite dividir el conjunto de datos de manera adecuada para el entrenamiento y la prueba. Los módulos de `sklearn.utils` (`class_weight`, `resample`) son utilizados para gestionar el desbalance de clases, asegurando que el modelo no esté sesgado hacia la clase mayoritaria.

os, shutil: Estos módulos proveen herramientas para interactuar con el sistema operativo y el sistema de archivos. `os` se utiliza para la gestión de rutas y directorios, mientras que `shutil` se emplea para operaciones de alto nivel con archivos y directorios, como la copia y eliminación de estructuras de archivos.

kagglehub: Este módulo facilita la descarga directa del conjunto de datos desde la plataforma Kaggle, permitiendo un acceso y una replicación del entorno de trabajo más eficientes.

Creación del dataframe ResNet50

Adquisición y Centralización del Dataset

Ejecuta la extracción remota mediante la librería `kagglehub`, descargando el conjunto de datos "non-and-biodegradable-waste-dataset" a una ubicación temporal en el sistema.

Gestiona la estructura de directorios validando y creando el directorio de destino local (`dataset`). Itera sobre el contenido descargado y, discriminando entre directorios y archivos individuales, traslada la totalidad de la información a la ruta de trabajo permanente.

Depura el entorno eliminando recursivamente la ruta de descarga original (`shutil.rmtree`) para liberar espacio y evitar redundancia de datos.

Confirma la operación imprimiendo en consola la ruta final donde se han consolidado los archivos, asegurando que el entorno está listo para el procesamiento.

Indexación y Construcción del dataframe de Entrenamiento

Define el espacio de búsqueda estableciendo las rutas base y generando dinámicamente una lista de directorios de entrenamiento (train_dirs) que comprende desde TRAIN.1 hasta TRAIN.4.

Instancia la lógica de recolección mediante la función create_dataframe, la cual inicializa listas vacías para almacenar las rutas de las imágenes y sus correspondientes etiquetas.

Itera y valida jerárquicamente explorando cada directorio de entrenamiento y subdirectorios de clase ('B' para Biodegradable y 'N' para No Biodegradable). Aplica un filtro de extensión para asegurar que solo se procesen archivos de imagen válidos (.png, .jpg, .jpeg), los cuales son imágenes sin preprocesar correctamente para el modelo de IA, como se mencionó anteriormente.

Retorna un objeto estructurado (pd.DataFrame) que vincula cada ruta de archivo (filename) con su etiqueta de clasificación (class), consolidando los metadatos necesarios para la fase de entrenamiento del modelo.

Entrenamiento de ResNet50

Preparación de Datos y Balanceo de Clases

Se calcula automáticamente los pesos de las clases.

Se aplica la estrategia 'balanced', que asigna a cada clase un peso inversamente proporcional a su frecuencia de aparición en el dataset de entrenamiento (train_df['class']).

Finalmente, se convierte el array de pesos a un diccionario, donde las claves son los índices de clase (0, 1, ...) y los valores son los pesos calculados, para su uso en la fase de entrenamiento.

Instanciación del Generador de Imágenes (train_datagen)

Se define un objeto ImageDataGenerator que establece una serie de técnicas de Aumento de Datos (Data Augmentation) para el conjunto de entrenamiento.

Esto incluye transformaciones geométricas (volteo horizontal, rotación, desplazamiento de ancho y alto, zoom) y ajustes de brillo.

Creación del Generador de Entrenamiento y de validación (train_gen, val_gen)

Se configuran parámetros clave: `target_size=(224, 224)` (tamaño requerido por ResNet50), `batch_size=32`, modo de clasificación 'categorical' y se activa la mezcla aleatoria (`shuffle=True`).

Se selecciona el subconjunto de entrenamiento (`subset='training'`).

Se configura de manera similar a `train_gen`, pero seleccionando el subconjunto de validación (`subset='validation'`).

Carga del Modelo Base (ResNet50)

La función `build_resnet()` inicia cargando la arquitectura ResNet50, pre-entrenada con el dataset ImageNet (`weights='imagenet'`).

Se omite la capa de clasificación superior original (`include_top=False`) para poder adaptar el modelo a la nueva tarea de 2 clases.

Se añade una capa `GlobalAveragePooling2D` para reducir las dimensiones de la salida de la ResNet50.

La capa final de salida es una capa `Dense` con 2 neuronas (para las 2 clases).

Configuración de Callbacks

`ModelCheckpoint`: Guarda automáticamente el modelo (`predictWaste12.h5`) únicamente cuando se alcanza la mínima pérdida de validación (`monitor='val_loss'`, `save_best_only=True`).

`ReduceLROnPlateau`: Reduce la tasa de aprendizaje a la mitad (`factor=0.5`) si la pérdida de validación no mejora después de 2 épocas (`patience=2`), ayudando a converger en el mínimo.

`EarlyStopping`: Detiene el entrenamiento si la pérdida de validación no mejora después de 3 épocas (`patience=3`) y restaura los mejores pesos guardados por el `ModelCheckpoint`.

Lanzamiento del Entrenamiento (`model.fit`)

El modelo es entrenado por un máximo de 25 épocas.

Se alimentan los datos mediante los generadores `train_gen` y `val_gen`.

Se especifican `steps_per_epoch` y `validation_steps` utilizando la longitud de los generadores.

Se utilizan los callbacks definidos para la gestión dinámica del entrenamiento.

```

# ===== DATA PREPARATION =====
# Balanceo con pesos
class_weights = class_weight.compute_class_weight('balanced',
                                                    classes=np.unique(train_df['class']),
                                                    y=train_df['class'])

class_weights = dict(enumerate(class_weights))

# ===== DATA GENERATORS =====
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input_resnet,
    validation_split=0.2,
    horizontal_flip=True,
    rotation_range=25,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.2,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

train_gen = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='filename',
    y_col='class',
    target_size=(224, 224),
    color_mode='rgb',
    batch_size=32,
    class_mode='categorical',
    shuffle=True,
    seed=42,
    subset='training'
)

val_gen = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='filename',
    y_col='class',
    target_size=(224, 224),
    color_mode='rgb',
    batch_size=32,
    class_mode='categorical',
    shuffle=False,
    seed=42,
    subset='validation'
)

# ===== MODEL ARCHITECTURE =====
def build_resnet():
    base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    for layer in base.layers[:140]:
        layer.trainable = False
    for layer in base.layers[140:]:
        layer.trainable = True

```

Ilustración 15 Código entrenamiento ResNet-50

Creación del dataframe MobileNetV2

Indexación y Construcción del dataframe de Entrenamiento

Se inicializa la función que acepta un único argumento, `base_dir`, que es la ruta principal donde se encuentra el dataset.

Se inicia un bucle (`for class_name in ['B', 'N']`) para procesar las dos clases objetivo: 'B' (Biodegradable) y 'N' (No Biodegradable).

Para cada clase, se construye la ruta completa (`class_dir = os.path.join(base_dir, class_name)`) y se verifica si el directorio de la clase existe. Si no existe, se imprime una advertencia y se salta a la siguiente clase.

Se aplica un filtro de extensión (`if file.lower().endswith(('.png', '.jpg', '.jpeg'))`) para asegurar que solo se procesen archivos que sean imágenes válidas (PNG, JPG, JPEG).

Una vez completada la iteración sobre todas las clases y archivos, la función retorna un objeto estructurado (un `pd.DataFrame`).

Este `DataFrame` tiene dos columnas: 'filename' (conteniendo todas las rutas de las imágenes) y 'class' (conteniendo sus etiquetas de clasificación correspondientes).

Entrenamiento de MobileNetV2

Preparación de Datos y Balanceo de Clases

Se aplicó la misma estrategia utilizada en el proceso de entrenamiento previo, enfocada en el balanceo de clases con el objetivo de equilibrar la distribución de los datos.

Generación de Datos (ImageDataGenerator)

Se definió un generador de imágenes para el conjunto de entrenamiento, incorporando:

- Función de preprocesamiento propia de MobileNetV2.

- División interna del dataset (80% entrenamiento, 20% validación).

- Aumento de datos mediante volteo horizontal y zoom, con el fin de incrementar la variabilidad del conjunto.

Utilizando este generador, se creó el conjunto de entrenamiento (train_gen) y el de validación (val_gen), especificando:

- Tamaño de entrada de 224×224 píxeles.

- Modo de clasificación categórica.

- Mezcla aleatoria únicamente para el conjunto de entrenamiento.

Construcción del Modelo (MobileNetV2)

La función build_mobilenetv2() inicia cargando la arquitectura MobileNetV2, pre-entrenada con el dataset ImageNet (weights='imagenet').

Se omite la capa de clasificación superior original (include_top=False) para poder adaptar el modelo a la nueva tarea de 2 clases.

Las capas del modelo base permanecieron congeladas para evitar alterar los pesos preentrenados.

Configuración de Callbacks

ModelCheckpoint: Guarda automáticamente el modelo únicamente cuando la pérdida de validación alcanza su mínimo.

ReduceLROnPlateau: Reduce la tasa de aprendizaje a la mitad cuando la pérdida de validación deja de mejorar por dos épocas.

EarlyStopping: Detiene el entrenamiento si no se observa mejora en la pérdida de validación durante 10 épocas consecutivas y restaura los mejores pesos registrados.

Proceso de Entrenamiento

El modelo fue entrenado durante un máximo de 35 épocas utilizando los generadores de entrenamiento y validación.

El cálculo de los pasos por época se obtuvo directamente del tamaño de cada generador.

Se incorporaron los class weights durante el entrenamiento para compensar el desbalance entre clases.

Los callbacks configurados permitieron gestionar dinámicamente la tasa de aprendizaje, el almacenamiento del mejor modelo y la interrupción temprana del proceso.

```
# ===== DATA PREPARATION =====
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(
class_weights = dict(enumerate(class_weights))

# ===== DATA GENERATORS =====
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input_mobilenet,
    validation_split=0.2,
    horizontal_flip=True,
    rotation_range=25,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.2,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

train_gen = train_datagen.flow_from_dataframe(
    dataframe=df_balanced,
    x_col='filename',
    y_col='class',
    target_size=(224, 224),
    color_mode='rgb',
    batch_size=32,
    class_mode='categorical',
    shuffle=True,
    seed=42,
    subset='training'
)

val_gen = train_datagen.flow_from_dataframe(
    dataframe=df_balanced,
    x_col='filename',
    y_col='class',
    target_size=(224, 224),
    color_mode='rgb',
    batch_size=32,
    class_mode='categorical',
    shuffle=False,
    seed=42,
    subset='validation'
)

# ===== MODEL ARCHITECTURE =====
def build_mobilenetv2():
    base = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    base.trainable = False
    #for layer in base.layers[:30]: # Congelar todas excepto las últimas 30 capas
    #    layer.trainable = False

    x = base.output
```

Ilustración 16 Código entrenamiento MobileNetV2

Creación del dataframe con Iterative Domain Technique para MobileNetV2

Función constructora de dataframes

Se inicializa una función llamada `creadorDataFrame` que acepta un único argumento, `base_dir`, que es la ruta principal donde se encuentra el dataset.

Se inicializan dos listas vacías: `image_paths` (para almacenar las rutas completas de las imágenes) y `labels` (para almacenar las etiquetas de clase), que se llenarán durante el recorrido.

Se inicia un bucle (`for class_name in ['B', 'N']`) para procesar las dos clases objetivo: 'B' (Biodegradable) y 'N' (No Biodegradable).

Se utiliza `os.walk(class_dir)` para recorrer recursivamente el directorio de la clase y todos sus subdirectorios. Esto es crucial porque permite encontrar imágenes anidadas en cualquier subcarpeta, no solo en la carpeta principal de la clase.

Para cada imagen válida, se construye su ruta completa (`img_path`) y se añade a la lista `image_paths`.

Simultáneamente, la etiqueta de la clase actual (`class_name`) se añade a la lista `labels`.

Una vez completada la iteración sobre todas las clases y archivos, la función retorna un objeto estructurado (`pd.DataFrame`).

Este DataFrame final tiene dos columnas: 'filename' (conteniendo todas las rutas de las imágenes recopiladas) y 'class' (conteniendo sus etiquetas de clasificación correspondientes).

Creación del DataFrame de Entrenamiento

Se define la ruta base para el conjunto de datos de entrenamiento en la variable `train_base_dir`.

Se llama a la función `creadorDataFrame` con esta ruta para construir y poblar el primer DataFrame, `df_train`, que contiene las rutas y etiquetas del conjunto de entrenamiento.

Creación del DataFrame de "Real Data"

Se define la ruta base para la carpeta de "real data" en la variable `real_data_base_dir`.

Se llama a la función `creadorDataFrame` nuevamente con esta ruta para construir el segundo DataFrame, `df_real_data`, indexando las imágenes de una fuente de datos adicional.

Combinación de DataFrames

Ambos DataFrames (df_train y df_real_data) se combinan verticalmente en un único DataFrame llamado df_combined mediante la función pd.concat.

Entrenamiento de MobileNetV2 con fine tuning

Inicialización del Generador de Datos

Realizar el preprocesamiento usando una función de TensorFlow que escala y normaliza los píxeles exactamente igual que en el entrenamiento original de MobileNetV2, garantizando compatibilidad y rendimiento óptimo.

Dividir el dataframe de forma que el 20% de las imágenes se reserven para validación y el 80% restante para entrenamiento.

Aplicar aumentos de datos para mejorar la generalización sobre objetos similares; en este caso, permitir volteos horizontales aleatorios (horizontal_flip=True) y rellenar píxeles vacíos con interpolación cercana (fill_mode='nearest').

Creación del Generador de Entrenamiento y validación

Generar lotes de imágenes con datagen.flow_from_dataframe a partir de df_combined, usando subset="training" para acceder al 80% asignado al entrenamiento.

Redimensionar todas las imágenes a 224×224 px, tamaño requerido por MobileNetV2.

Formatear las etiquetas en representación one-hot mediante class_mode="categorical".

Establecer un batch_size de 32 para definir cuántas imágenes se procesan por paso.

Crear el generador de validación siguiendo la misma configuración, usando subset="validation" y manteniendo shuffle=False para no alterar el orden durante la evaluación.

Carga del Modelo

Definir la ruta del archivo .h5 que contiene el modelo previamente entrenado.

Cargar la arquitectura, los pesos y la configuración del optimizador mediante load_model, obteniendo un MobileNetV2 listo para ajuste fino.

Preparación para Ajuste Fino (Fine-Tuning)

Congelar las capas inferiores iterando sobre la mayoría de las capas del modelo y estableciendo `layer.trainable = False`; esto preserva las características básicas aprendidas en ImageNet.

Descongelar las últimas 30 capas habilitando `trainable=True`, permitiendo que aprendan patrones propios de los residuos captados.

Recompilar el modelo tras modificar las capas entrenables, usando el optimizador Adam con una tasa de aprendizaje muy baja ($1e-5$) para realizar ajustes graduales.

Configuración de Callbacks

EarlyStopping:

Monitorear `val_loss` y detener el entrenamiento si no mejora después de 3 épocas.

Restaurar automáticamente los mejores pesos alcanzados.

ReduceLROnPlateau:

Vigilar también `val_loss` y reducir la tasa de aprendizaje a la mitad cuando no mejore durante 2 épocas consecutivas.

Entrenamiento (Fine-Tuning)

Iniciar el reentrenamiento mediante `model.fit` usando los generadores de entrenamiento y validación.

Ejecutar hasta 10 épocas como máximo, aprovechando que el modelo ya cuenta con conocimiento previo y solo requiere ajuste fino.

```
Found 3520 validated image filenames belonging to 2 classes.
Found 890 validated image filenames belonging to 2 classes.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate.
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your 'PyDataset' class should call 'self._warn_if_super_not_called()'
  self._warn_if_super_not_called()
Modelo cargado: /content/drive/MyDrive/Modelos/predictWaste_mobilenetv2_v2.h5
Epoch 1/10
110/110 ----- 46s 245ms/step - accuracy: 0.6196 - loss: 0.8970 - val_accuracy: 0.8057 - val_loss: 0.4367 - learning_rate: 1.0000e-05
Epoch 2/10
110/110 ----- 19s 170ms/step - accuracy: 0.7095 - loss: 0.6594 - val_accuracy: 0.7648 - val_loss: 0.4893 - learning_rate: 1.0000e-05
Epoch 3/10
110/110 ----- 0s 136ms/step - accuracy: 0.7246 - loss: 0.6351
Epoch 3: ReduceLROnPlateau reducing learning rate to 4.999999873689276e-06.
110/110 ----- 19s 170ms/step - accuracy: 0.7248 - loss: 0.6346 - val_accuracy: 0.7534 - val_loss: 0.5287 - learning_rate: 1.0000e-05
Epoch 4/10
110/110 ----- 20s 163ms/step - accuracy: 0.7671 - loss: 0.5481 - val_accuracy: 0.7602 - val_loss: 0.5292 - learning_rate: 5.0000e-06
WARNING:absl:You are saving your model as an HDF5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy.
Modelo actualizado guardado como model_retrained_REALDATA.h5
```

Ilustración 17 Consola de entrenamiento FT

4.5 Ensamblaje e integración del sistema embebido.

El ensamblaje del sistema embebido se realizó a partir de los módulos descritos en el apartado 4.1.2, integrando en un solo sistema funcional al ESP32-S3 CAM junto con el sensor ultrasónico HC-SR04, el servomotor MG90S, la pantalla OLED I2C y el módulo de cámara incorporado. Este capítulo detalla la arquitectura eléctrica, la disposición física de los componentes y las decisiones de diseño que permitieron obtener un prototipo estable, accesible y funcional. Adicionalmente, se describen las consideraciones técnicas empleadas para garantizar la operación adecuada del sistema bajo las limitaciones físicas del prototipo MVP.

4.5.1 Conexión eléctrica y lógica del sistema

La conexión eléctrica del sistema se diseñó para garantizar la estabilidad y el correcto funcionamiento de cada módulo. El ESP32-S3 CAM actúa como unidad de control principal, proporcionando tanto la alimentación como las señales de comunicación necesarias para el sensor ultrasónico HC-SR04, el servomotor SG90, la pantalla OLED I2C y el módulo de cámara integrado.

Módulo	Componente Específico	Pin del Módulo	Pin del ESP32 S3 CAM	Función
Sensor Ultrasónico	HC-SR04	VCC	5V	Alimentación del sensor
		GND	GND	Tierra común
		Trig	GPI01	Envío de señal de disparo
		Echo	GPI02	Recepción del pulso (medición por interrupción)
Pantalla OLED	128x64 i2c	VCC	5V	Alimentación
		GND	GND	Tierra común
		SDA	GPI42	Línea de datos I2C por software

		SCL	GPI41	Línea de reloj I2C por software
Servomotor	mg90s	VCC	5V	Alimentación
		GND	GND	Tierra común
		Señal	GPI13	Control PWM del ángulo

Tabla 16 Tabla de Conexiones

La Tabla 16 y el diagrama electrónico de la ilustración 18 permiten visualizar de manera clara la distribución de señales y líneas de alimentación dentro del sistema. La reasignación de pines, especialmente en el caso de la interfaz I2C por software implementada en los GPIO 41 y 42, garantizó la compatibilidad con la cámara integrada del ESP32-S3 CAM sin comprometer la funcionalidad del resto de los módulos.

La elección de pines capaces de manejar interrupciones para el sensor ultrasónico, así como la configuración PWM del servomotor, contribuyeron a un funcionamiento estable y coherente con los requerimientos del prototipo. Con esta arquitectura eléctrica y lógica, el sistema embebido queda plenamente integrado y preparado para su montaje físico, descrito en los apartados posteriores.

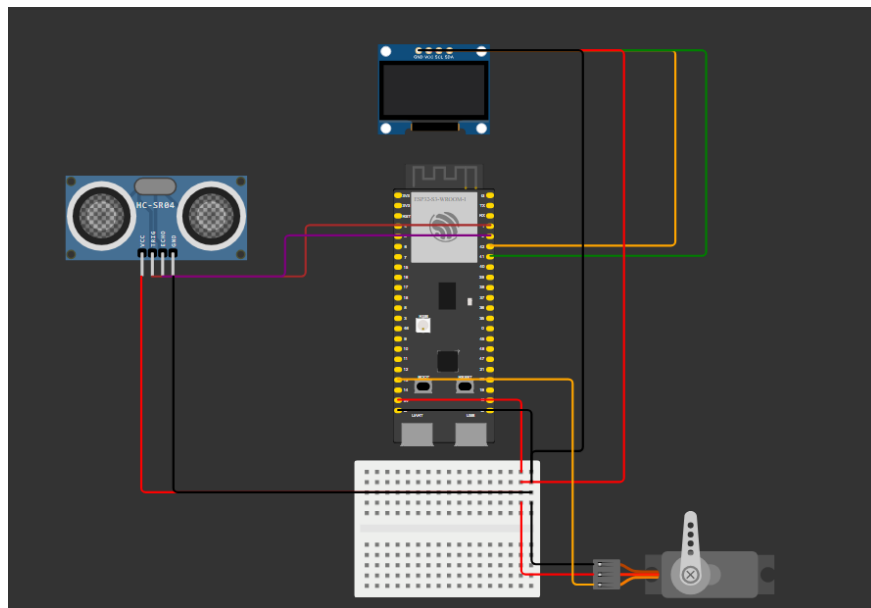


Ilustración 18 Diagrama de electrónico

4.5.2 Distribución física de los componentes

La distribución física de los componentes se realizó considerando criterios de funcionalidad, accesibilidad, estabilidad estructural y aprovechamiento del espacio disponible dentro del prototipo. Para este fin se utilizó un contenedor plástico tipo bote de basura con dimensiones aproximadas de 10.8 cm de ancho, 16 cm de alto y 12.4 cm de profundidad, el cual sirvió como chasis para integrar todos los elementos electrónicos y mecánicos del sistema.

Antes del montaje, se establecieron los siguientes criterios:

- Mantener la línea de visión libre para la cámara.
- Minimizar vibraciones en el servomotor.
- Reducir interferencias eléctricas entre líneas de señal y alimentación.
- Garantizar accesibilidad para mantenimiento y pruebas.
- Optimizar el recorrido del cableado interno.
- Aprovechar al máximo el volumen interno sin comprometer la estabilidad.

El ESP32-S3 CAM se ubicó en la parte superior interna del contenedor con el objetivo de proporcionar a la cámara integrada un campo visual amplio y despejado. Para fijarlo, se diseñó una base artesanal compuesta por una placa de cartón rígido doblada y reforzada mediante palos de madera de 2 cm de grosor, y posteriormente adherida con silicón caliente. Esta estructura permitió mantener el microcontrolador estable, reducir vibraciones y garantizar que la cámara permaneciera en una posición adecuada durante la operación.

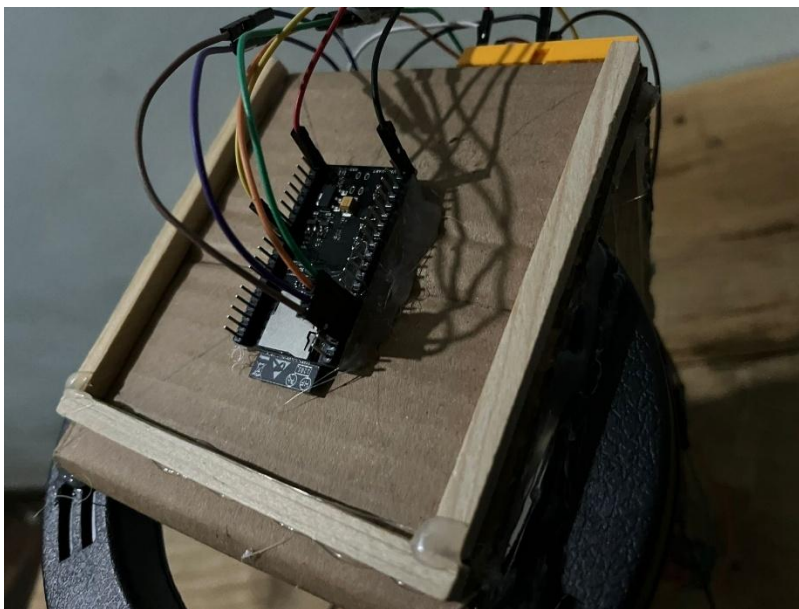


Ilustración 19 ESP32 S3 Cam ubicación

Con el fin de garantizar condiciones de iluminación constantes un factor crucial para mejorar la calidad de las imágenes capturadas por la cámara se integró un aro de luz LED de 6 pulgadas (aproximadamente 15 cm de diámetro). Este aro se colocó en la parte inferior de la base donde se sostiene el ESP32-S3 CAM, generando un entorno luminoso uniforme dentro del contenedor. Este aro genera una iluminación uniforme, reduce sombras y mejora la detección el modelo de IA en condiciones de baja luz.

Para su alimentación, se perforó discretamente la misma base, permitiendo el paso del cable USB hacia la parte posterior de la estructura. Finalmente, el aro fue fijado con silicón caliente, asegurando su estabilidad y alineación con el campo visual de la cámara. Esta adaptación mejora significativamente el desempeño del sistema en condiciones de baja iluminación y reduce sombras que podrían interferir con la identificación visual.



Ilustración 20 Iluminación prototipo

El cableado interno del prototipo se realizó mediante la unión de múltiples cables de conexión (jumpers) para alcanzar la longitud necesaria y permitir una manipulación externa más cómoda de los módulos. En algunos casos fue necesario enlazar tres o hasta cuatro jumpers de datos, garantizando así que las señales pudieran extenderse sin comprometer la movilidad o la accesibilidad durante las pruebas.

Estos conjuntos de cables fueron fijados cuidadosamente con silicón caliente en la parte posterior del prototipo, específicamente en la zona donde se localiza la base estructural elaborada con cartón. Esta fijación evitó desplazamientos indeseados y contribuyó a mantener un orden adecuado dentro del sistema.

Para la conexión con el microcontrolador se emplearon jumpers hembra, mientras que para los módulos se utilizaron jumpers macho, asegurando compatibilidad con los pines correspondientes y facilitando el montaje y desmontaje durante el proceso de integración.

Para optimizar la distribución de energía y reducir la carga directa sobre el ESP32-S3 CAM, se incorporó una protoboard mini como módulo auxiliar de alimentación. En esta se organizaron los puertos de entrada y salida de los distintos componentes, permitiendo suministrar de manera ordenada los 5 VCC requeridos por los módulos previamente descritos. La protoboard actúa como punto central de distribución eléctrica, asegurando que cada dispositivo reciba la energía necesaria sin sobrecargar el regulador del microcontrolador y facilitando además la gestión del cableado interno del prototipo.

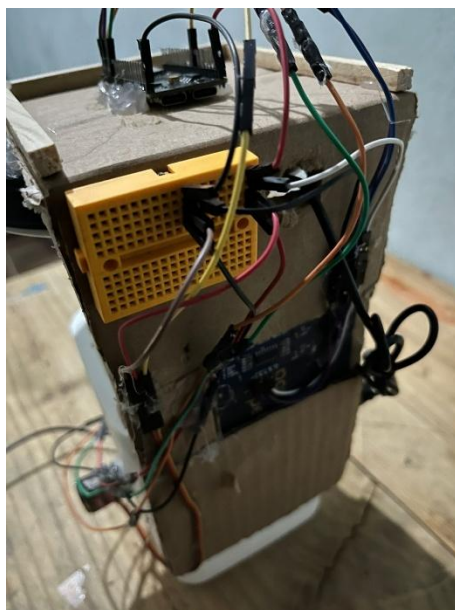


Ilustración 21 Cableado y protoboard prototipo

El sensor ultrasónico HC-SR04 se colocó en la parte frontal del prototipo, sobre la misma plataforma que sostiene al ESP32-S3 CAM, realizando dos cavidades circulares en la superficie para alojar los transductores del sensor. Esta ubicación frontal asegura una línea de detección directa hacia el exterior, evitando obstrucciones y permitiendo mediciones más precisas de distancia.



Ilustración 22 Sensor ultrasónico frente al prototipo

La pantalla OLED se instaló también en la parte frontal del contenedor, en una posición visible para el usuario. Esta ubicación permite consultar fácilmente los valores y mensajes generados por el sistema, facilitando que el usuario pueda interpretar las instrucciones de clasificación de residuos o mensajes operativos del dispositivo.



Ilustración 23 Pantalla Oled en el prototipo

Por su parte, el servomotor MG90S se montó en la zona inferior derecha del contenedor, para lo cual se realizó una perforación con una broca de $\frac{1}{4}$ de pulgada, permitiendo el paso del eje del servomotor hacia el mecanismo de apertura. La conexión mecánica entre el servomotor y la tapa del contenedor se efectuó mediante cables UTP reforzados y fijados a través de un orificio adicional perforado con

una broca fina. Esta configuración garantiza que la tapa pueda moverse de forma controlada conforme a las posiciones establecidas por el sistema.



Ilustración 24 Servomotor en el prototipo y conexión a la tapa

La disposición final de los componentes facilita un flujo operativo lógico:

Detección → Procesamiento → Visualización → Acción Mecánica

Permitiendo que el prototipo funcione de manera integrada y eficiente. Además, esta organización interna optimiza el espacio reducido del contenedor sin comprometer la accesibilidad para el mantenimiento y ajustes posteriores.

4.5.3 Render en 3D del producto final para producción

El desarrollo del render 3D tuvo como propósito ofrecer una representación visual clara, detallada y estéticamente refinada del producto final al que se pretende llegar en caso de contar con el financiamiento necesario. Dado que el prototipo construido corresponde a un MVP (Producto Mínimo Viable). Este tipo de visualización también facilita anticipar la integración de los componentes internos, prever posibles mejoras estructurales y presentar el concepto en un contexto realista para potenciales inversionistas o evaluadores técnicos.

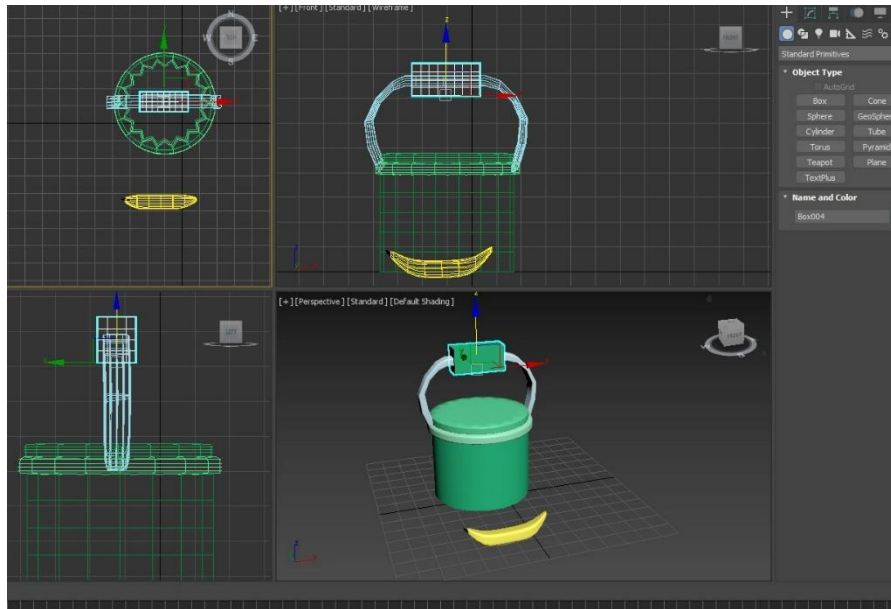


Ilustración 25 Modelo 3D construcción inicial

Para la elaboración del modelo tridimensional se empleó Autodesk 3ds Max, en lugar de SolidWorks, debido a la experiencia previa con este software y a su versatilidad para el modelado y renderizado. Su motor de render nativo permitió generar imágenes con materiales realistas, iluminación controlada y superficies mejor definidas, proporcionando una proyección visual precisa del producto final deseado. Asimismo, la flexibilidad del software facilitó ajustes iterativos en la geometría, proporciones y disposición interna del diseño.

La ilustración 26 muestra el modelo 3D en tres diferentes posiciones frontal, lateral y superior con el objetivo de mostrar de manera clara la estructura general del dispositivo. Estas vistas permiten identificar la posición proyectada del sensor ultrasónico, la pantalla OLED, el servomotor y la ruta del cableado, mostrando una distribución más limpia, compacta y profesional en comparación con el prototipo físico. Gracias a ello, se facilita la comprensión espacial del diseño final y se evidencian las mejoras previstas respecto al modelo MVP.

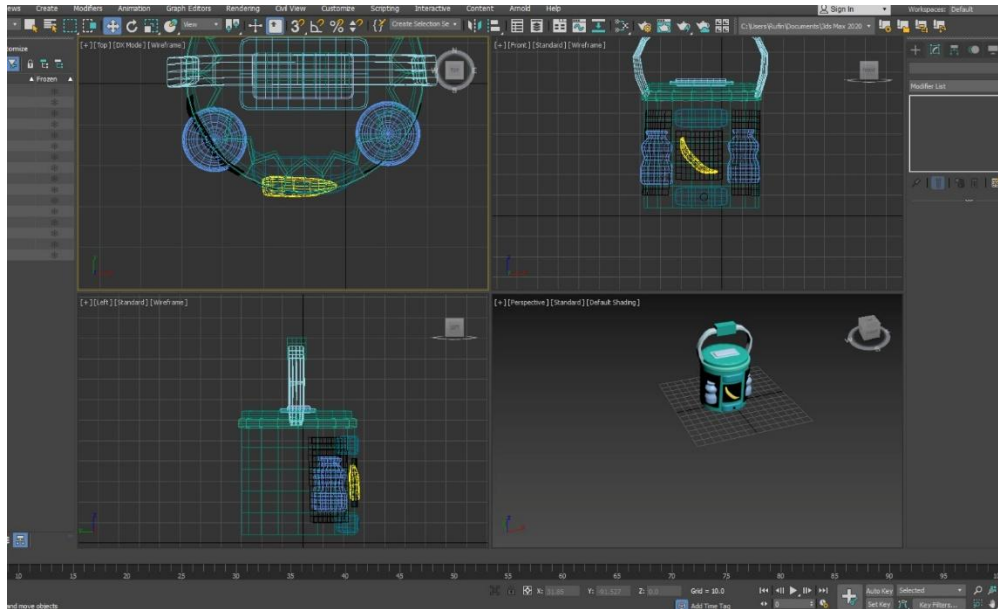


Ilustración 26 Modelado en tres posiciones

Es importante destacar que existen diferencias significativas entre el prototipo físico y el render 3D. Mientras que el prototipo se construyó con materiales accesibles y técnicas manuales debido a limitaciones de recursos, el render propone una versión optimizada: superficies más uniformes, un chasis rediseñado para una integración más eficiente de los módulos, una estructura interna más estable y acabados visuales propios de un producto final manufacturable. Estas modificaciones buscan reflejar de forma fiel la intención del diseño original, pero elevando su presentación a un estándar adecuado para producción.

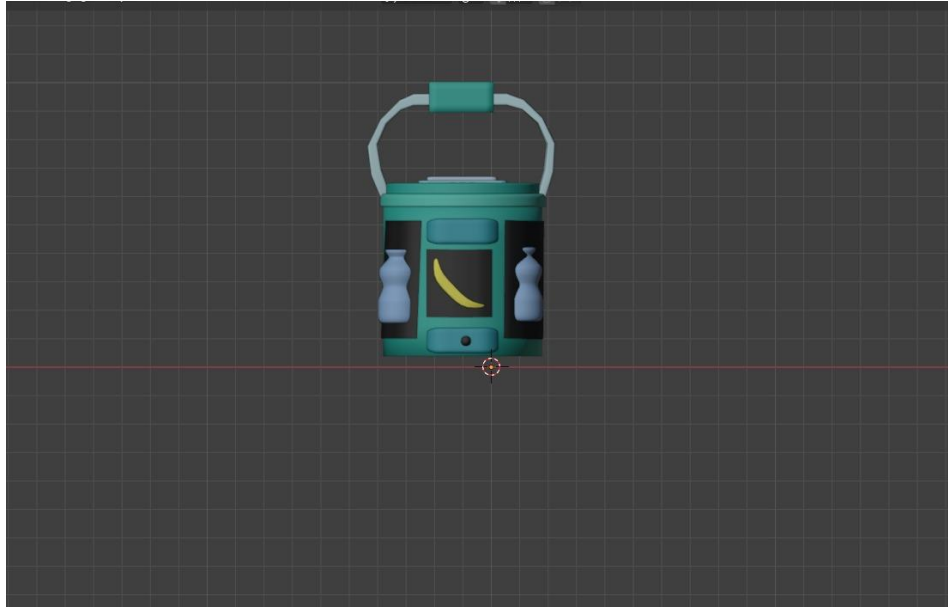


Ilustración 27 Modelo 3D finalizado versión manufacturable

La Ilustración 27 muestra la apariencia final del producto, destacando los acabados uniformes y la integración de los componentes externos. Para complementar esta visión, la Ilustración 28 presenta una vista del modelo. Esta vista interna es esencial para la evaluación técnica, ya que confirma las dimensiones internas, la gestión de espacio y la estabilidad estructural necesaria para la manufactura en serie, demostrando la asignación precisa del volumen para cada módulo.

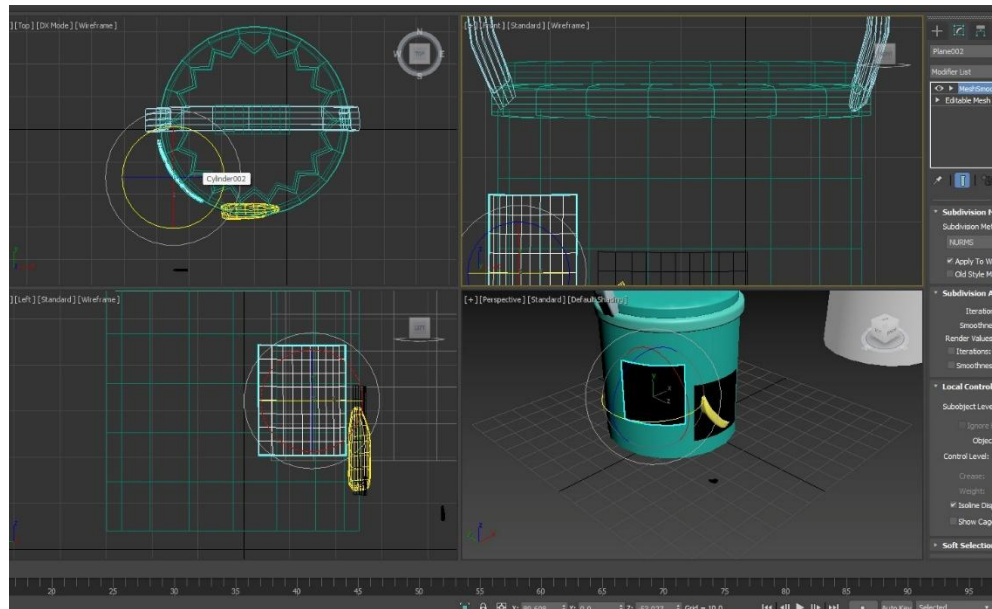


Ilustración 28 Visión interna del modelo 3D

En base a lo anterior presentado el render 3D no busca replicar el prototipo artesanal, sino representar la versión final deseada del dispositivo, utilizando materiales adecuados, una estructura interna optimizada y un acabado estético propio de un producto comercial. Mientras el prototipo MVP demuestra la funcionalidad básica del sistema, el render comunica de forma clara la visión final del proyecto, facilitando su evaluación técnica, apoyando la toma de decisiones y proyectando las mejoras previstas para una futura manufactura. En otras palabras, el modelo tridimensional funciona como una herramienta esencial para mostrar el potencial del proyecto más allá del prototipo inicial, evidenciando cómo podría evolucionar hacia una versión completamente manufacturable y lista para su implementación real.

Capítulo 5

Pruebas y resultados

En este capítulo se presentan las pruebas realizadas tanto al sistema embebido (hardware) como al servidor y al modelo de inteligencia artificial desarrollado (software), así como los resultados obtenidos de su evaluación. El objetivo principal es validar el funcionamiento individual de cada uno de los módulos que conforman el sistema. En el apartado de hardware, esto incluye el servomotor, el sensor ultrasónico, la pantalla OLED y la cámara ESP32-S3; mientras que en el apartado de software se busca comprobar el funcionamiento general del servidor, las rutas implementadas, el backend principal de la aplicación y el desempeño del modelo de inteligencia artificial, basándose en las métricas obtenidas durante las pruebas. Posteriormente, se analiza el rendimiento del sistema cuando todos estos componentes operan de manera integrada bajo el control del microcontrolador y el servidor creado.

Para lograrlo, el capítulo se estructura en tres partes.

- En primer lugar, se describen las pruebas diseñadas y ejecutadas para evaluar la precisión, estabilidad, confiabilidad y tiempos de respuesta de cada módulo, tanto a nivel de hardware como de software (Sección 5.1).
- En segundo lugar, se presentan los datos obtenidos junto con un análisis crítico que permite identificar comportamientos relevantes del sistema (Sección 5.2).
- Finalmente, se exponen las conclusiones generales sobre el rendimiento global, el grado de cumplimiento de los requisitos funcionales establecidos al inicio del proyecto y las limitaciones encontradas durante las pruebas (Sección 5.3).

Este conjunto de evaluaciones permite determinar de manera objetiva si el sistema propuesto cumple con los objetivos funcionales y operativos definidos en las etapas iniciales del proyecto, los cuales abarcan áreas como diseño electrónico, diseño mecánico, arquitectura del servidor, estructura del modelo de IA e integración completa del sistema. Además, los resultados obtenidos permiten valorar si el prototipo es adecuado para su aplicación final y para su posterior entrega como producto funcional dentro del alcance planteado.

5.1 Pruebas Realizadas

Con el fin de validar el desempeño total del sistema propuesto, se diseñaron y ejecutaron diversas pruebas destinadas a evaluar tanto los componentes de hardware que conforman el sistema embebido como los elementos de software. Estas pruebas permiten identificar el nivel de precisión, estabilidad, eficiencia y confiabilidad de cada componente.

5.1.1 Pruebas Unitarias por Módulo: Hardware

Las pruebas de hardware se enfocaron en confirmar que los módulos físicos del sistema embebido operaran correctamente de manera individual. Los componentes evaluados incluyen **el servomotor, el sensor ultrasónico, la pantalla OLED y la cámara ESP32-S3**.

Servomotor

Se realizaron pruebas de movilidad para validar la precisión y repetibilidad del movimiento. El servomotor fue programado para desplazarse entre diversos ángulos predeterminados (0°, 100° y 180°), verificando que alcanzara la posición indicada, que se mantuviera estable y que respondiera dentro de un tiempo aceptable.

Sensor Ultrasónico

El sensor fue evaluado para determinar su precisión al medir distancias. Se posicionó un objeto a distancias conocidas y fijas, repitiendo mediciones para obtener un promedio y valorar tanto el error absoluto como la variación entre lecturas consecutivas.

Pantalla OLED

Se evaluó su capacidad para mostrar información en tiempo real proveniente del microcontrolador. Las pruebas consistieron en mostrar datos dinámicos y la consistencia de la información procesada. También se evaluó si existían parpadeos, retrasos visibles o cuelgues durante la operación continua.

Cámara ESP32-S3 CAM

Se realizaron pruebas de captura y transmisión de imágenes. Se evaluó su latencia y el tiempo requerido para enviar los datos a través de Wi-Fi hacia el servidor. Además, se

probaron configuraciones de parámetros que modificaban el comportamiento del sensor de iluminación para lograr el mejor desempeño del sensor óptico.

5.1.2 Pruebas Unitarias por Módulo: Software

Por la parte del software, se evaluaron tres elementos críticos: el servidor, las rutas API implementadas y el modelo de inteligencia artificial reentrenado para la clasificación o detección correspondiente. El propósito fue asegurar que el sistema de software funcionara de manera estable, rápida y precisa.

Servidor y Arquitectura Backend

Se realizaron pruebas para verificar la correcta inicialización del servidor, la estabilidad durante periodos de operación continua y la capacidad para recibir, procesar y almacenar información enviada desde el microcontrolador. También se evaluaron los tiempos de respuesta del servidor.

Pruebas de Rutas (API REST)

Se validaron las rutas diseñadas para recibir imágenes, procesarlas, enviarlas al modelo de IA, devolver una predicción y enviar respuestas de estado al ESP32. Las pruebas incluyeron:

- Verificación del formato correcto de los datos recibidos.
- Manejo de errores ante solicitudes incompletas o corruptas.
- Verificación del envío adecuado de respuestas hacia el microcontrolador.

Estas pruebas permitieron garantizar que la comunicación entre el hardware y el software fuera estable, coherente y eficiente.

Modelo de Inteligencia Artificial

Se evaluó el desempeño de los cuatro modelos entrenados mediante métricas estándar como precisión (accuracy), matriz de confusión, recall, f1 score y curva roc. Además, se realizaron pruebas utilizando imágenes reales capturadas por la cámara del sistema embebido para asegurar que el modelo reconociera correctamente los objetos en condiciones del mundo real.

5.2 Evaluación de resultados

5.2.1 Prueba Hardware

Servomotor

```
#include <ESF32servo.h>

#define SERVO_PIN 3

Servo servoMotor;

const byte angulos[2] = {0, 180};
const char comandos[2] = {'b', 'n'};
const int velocidad = 5;

void setup() {
  Serial.begin(115200);
  servoMotor.attach(SERVO_PIN);
  servoMotor.write(90);
}

void moverServoSuave(int anguloObjetivo) {
  int posicionActual = servoMotor.read();
  int incremento = (anguloObjetivo > posicionActual) ? 1 : -1;

  while (posicionActual != anguloObjetivo) {
    posicionActual += incremento;
    if ((incremento == 1 && posicionActual > anguloObjetivo) ||
        (incremento == -1 && posicionActual < anguloObjetivo)) {
      posicionActual = anguloObjetivo;
    }
    servoMotor.write(posicionActual);
    delay(velocidad);
  }
}
```

Ilustración 29 Prueba servomotor

Prueba realizada	Parámetros evaluados	Resultado obtenido	Observaciones
Movilidad por ángulos predeterminados	0°, 100°, 180°	Movimientos completados con precisión	Sin vibraciones excesivas; operación estable
Repetibilidad de movimiento	10 cambios de rango repetitivo entre B y N, variación por ángulos	Variación máxima: $\pm 1.5^\circ$	El servomotor mantiene consistencia en posiciones repetidas
Tiempo de respuesta	Tiempo entre comando y posición final	0.18 s promedio	Dentro del rango funcional para el sistema
Estabilidad en posición	Tiempo de sostenimiento de la posición sin oscilaciones	sin desviaciones perceptibles	Adecuado para operación coordinada con otros módulos

Tabla 17 Resultados servomotor

Sensor Ultrasónico

```
#include <NewPing.h>

#define TRIG_PIN 1
#define ECHO_PIN 2
#define MAX_DISTANCE 200 // Distancia máxima a medir (en cm)

NewPing sonar(TRIG_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
  Serial.begin(9600);
  Serial.println("Iniciando prueba del sensor ultrasónico");
  Serial.println("Mostrando lecturas de distancia...");
}

void loop() {
  static unsigned long ultimo_tiempo = 0;
  const unsigned intervalo = 500; // Intervalo entre lecturas (ms)

  if (millis() - ultimo_tiempo >= intervalo) {
    ultimo_tiempo = millis();

    // Obtener distancia en centímetros
    unsigned int distancia = sonar.ping_cm();

    // Mostrar resultado
    if (distancia == 0) {
      Serial.println("Fuera de rango");
    } else {
      Serial.print("Distancia: ");
    }
  }
}
```

Ilustración 30 Prueba sensor

Prueba realizada	Distancia real	Lectura promedio	Error absoluto	Observaciones
Medición 1	2 cm	2.6 cm	+0.6 cm	Error pequeño, consistente
Medición 2	5 cm	5.4 cm	−0.6 cm	Ligera variación, dentro de tolerancia
Medición 3	9 cm	10.2 cm	+1.2 cm	Se mantuvo estable entre lecturas
Variación entre lecturas		±0.8 cm entre muestras		Comportamiento aceptable para detección de proximidad

Tabla 18 Resultados sensor

Pantalla OLED

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>

// Definiciones para la pantalla OLED
#define SCREEN_WIDTH 128 // Ancho de la pantalla en píxeles
#define SCREEN_HEIGHT 64 // Alto de la pantalla en píxeles
#define OLED_RESET -1 // Pin de reset (se usa -1 para ESP32)

// Definir los pines de I2C para ESP32 S2 Mini
#define I2C_SDA_PIN 42
#define I2C_SCL_PIN 41

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(115200);

  // Inicializa la comunicación I2C con los pines correctos antes de la pantalla.
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);

  // Inicializa la pantalla OLED
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Dirección I2C: 0x3C
    Serial.println(F("Error en la inicialización de la pantalla OLED"));
    for(;;); // Detiene el programa
  }

  // Limpia el buffer de la pantalla y la apaga
  display.clearDisplay();
  display.display();

  // Configura el texto a mostrar // Tamaño del texto
  display.setTextSize(1); // Color del texto (blanco)
  display.setTextColor(SSD1306_WHITE);
}
```

Ilustración 31 Prueba pantalla oled

Prueba realizada	Parámetros evaluados	Resultado obtenido	Observaciones
Actualización en tiempo real	1–5 actualizaciones/segundo	Sin parpadeos visibles	Flujo de datos estable
Consistencia de datos mostrados	Lecturas dinámicas del microcontrolador	Información mostrada correctamente	No se observaron errores de refresco
Operación continua	30 min de funcionamiento	Sin cuelgues	Comportamiento óptimo bajo pruebas prolongadas
Legibilidad	Contraste y brillo	Correctos sin ajuste adicional	Pantalla adecuada para operación del sistema

Tabla 19 Resultados pantalla oled

Cámara ESP32-S3 CAM

```
const char* servidor = "http://192.168.0.121:5000/";
//
#define CAM_PIN_D0 11
#define CAM_PIN_D1 9
#define CAM_PIN_D2 8
#define CAM_PIN_D3 10
#define CAM_PIN_D4 12
#define CAM_PIN_D5 18
#define CAM_PIN_D6 17
#define CAM_PIN_D7 16

#define CAM_PIN_XCLK 15
#define CAM_PIN_PCLK 13
#define CAM_PIN_VSYNC 6

#define CAM_PIN_HREF 7
#define CAM_PIN_SDA 4
#define CAM_PIN_SCL 5

#define CAM_PIN_PWDN -1
#define CAM_PIN_RESET -1

//
/**
 * @brief Inicializa la cámara.
 */
bool init_camera() {
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;

    config.pin_d0 = CAM_PIN_D0;
```

Ilustración 32 Prueba cámara

Prueba realizada	Parámetros evaluados	Resultado obtenido	Observaciones
Captura de imagen	Resolución para el modelo de inteligencia artificial (240×240)	Imágenes nítidas	Buen desempeño en interiores
Latencia de transmisión	Tiempo desde captura hasta recepción	220–310 ms	Aceptable para el sistema
Envío vía Wi-Fi	Estabilidad de enlace	Conexión sostenida sin interrupciones	Buen rendimiento a 2.4 GHz
Ajuste de exposición e iluminación	ajustes manuales aplicados al sensor: saturación = 0, brillo = -2, contraste = 1.	Mejóro claridad en condiciones de baja luz	La cámara se adapta bien a variaciones ambientales

Tabla 20 Resultados cámara

5.2.2 Pruebas del Servidor y Arquitectura Backend

Prueba de Inicialización del Servidor

Prueba realizada:

Se arrancó el servidor desde un entorno local usando Flask server

Se monitoreó la correcta carga de dependencias, inicialización de variables de entorno y activación del puerto definido.

Se verificó mediante herramientas que el servidor respondiera en la ruta base (/index).

Resultado:

El servidor se inicia sin errores, haciéndolo disponibilidad de forma inmediata.

Pruebas de Estabilidad en Operación Continua

Prueba realizada:

El servidor se dejó activo por 4 horas mientras recibía peticiones periódicas.

Se usó un script automatizado que enviaba solicitudes POST, simulando la actividad real del microcontrolador.

Resultado:

Durante las pruebas de estabilidad se observó un incremento moderado en el uso de memoria del servidor al mantener activa la actualización dinámica en la ruta index. Este comportamiento se debe a la renovación constante de la imagen mostrada, la cual se actualizaba de forma continua durante varias horas de prueba.

Sin embargo, este aumento no representa un fallo ni afecta la funcionalidad del sistema, ya que dicho escenario no corresponde a condiciones reales de operación. En un entorno real, el flujo de imágenes no sería constante durante largos periodos, por ejemplo, no se estarían depositando residuos de manera **continua** durante 4 horas, por lo que la carga de trabajo sería significativamente menor.

Prueba de Recepción y Almacenamiento de Datos

Prueba realizada:

Desde el microcontrolador se enviaron imágenes en intervalos de 15 segundos durante 2 minutos.

El servidor almacenó cada solicitud en un diccionario de datos temporal.

Se verificó:

Integridad del archivo recibido.

Estructura correcta del JSON enviado.

Resultado:

Durante las pruebas se observó que, aunque la mayoría de las imágenes eran procesadas sin inconvenientes, existieron casos aislados en los que el servidor tardó demasiado en responder, provocando un read timeout en el cliente. Este comportamiento ocurrió aproximadamente en un 10 de cada 100 imágenes enviadas.

El origen del problema no está relacionado con fallos en el servidor, sino con latencia en la red durante el envío de datos. Aunque el servidor alcanzaba a generar una respuesta inicial ("objeto detectado"), la imagen enviada se incluía dentro de un archivo JSON con un peso considerable. Debido a este tamaño excesivo en la carga útil, la transmisión se veía afectada y, en situaciones de latencia elevada, parte del contenido se perdía o no alcanzaba a recibirse de forma completa.

5.2.3 Pruebas de Rutas (API REST)

Prueba de Recepción de Mensajes

Ruta evaluada: POST "/"

Prueba realizada:

Se capturo un frame por el microcontrolador.

Se envió una imagen en formato JPG proveniente del microcontrolador.

Se verificó que el servidor le llegara la imagen correspondiente.

Se revisó que los datos llegaran completos y que el tamaño del archivo se mantuviera conforme al original.

Resultado:

La transformación de los datos en formato bytes a una imagen dentro del servidor se realizó correctamente, conservando el mismo tamaño y calidad que la imagen enviada originalmente desde el microcontrolador. El servidor procesó la información sin alteraciones, completó la conversión y respondió de manera adecuada, devolviendo un código http 200, lo que confirma que la operación se efectuó satisfactoriamente.

Prueba De Procesamiento y Envío al Modelo de IA

Ruta evaluada: POST "/"

Prueba realizada:

El backend tomó la imagen recibida y la envió al modelo de IA.

El modelo devolvió la clasificación del objeto y el nivel de certeza.

Se verificó que la respuesta fuera consistente, que el proceso no excediera el tiempo permitido y que no se generaran errores inesperados.

Resultado:

La prueba destinada a verificar que el backend enviara la predicción dada en el formato JSON correspondiente se consideró un éxito. La predicción devuelta por el servidor coincidió con el formato esperado por el microcontrolador.

Los tiempos de respuesta se mantuvieron por debajo de los 2 segundos, lo cual es posible debido a que únicamente se envía la letra correspondiente a la clase detectada, con un tamaño aproximado de 1 byte.

Tanto en el servidor como en el cliente se implementó un manejo adecuado de excepciones, lo que permitió validar y procesar correctamente diferentes tipos de JSON recibidos.

Prueba De Carga Dinámica Del Frontend

Ruta evaluada: GET "/verificar_actualizacion"

Prueba realizada:

El microcontrolador envió una solicitud GET al servidor junto con el comando de predicción y la imagen correspondiente en formato bytes.

El backend recibió la solicitud, procesó los datos y almacenó la imagen en la carpeta asignada dentro del servidor.

Una función interna del backend actualizó el estado global del sistema, registrando la nueva información disponible para el cliente.

El frontend, mediante AJAX, ejecutó solicitudes periódicas cada 30 segundos para verificar si existía una actualización pendiente.

Cuando se detectó un cambio, la interfaz gráfica se actualizó automáticamente mostrando la nueva imagen, la clase predicha y la probabilidad asociada.

Resultado:

El servidor detectó correctamente la nueva actualización, transformó la imagen enviada desde el microcontrolador manteniendo su integridad y tamaño original, y devolvió un código HTTP 200 indicando una respuesta válida. Asimismo, el frontend recibió la actualización sin errores y reflejó de inmediato los nuevos datos gracias al mecanismo de sondeo implementado con AJAX. Esto confirma que el flujo de comunicación backend–frontend funciona de manera estable, eficiente y coherente.

Prueba de Renderizado del Estado del Sistema

Ruta evaluada: GET “/index”

Prueba realizada:

Se accedió a la ruta /index desde el navegador con el sistema recién iniciado, sin que el microcontrolador hubiera enviado aún una predicción.

El servidor evaluó el estado de la variable `alpha.diccionarioIdentificacion`, la cual se encontraba en `None`, indicando ausencia de datos.

Ante esta condición, el backend generó un objeto JSON por defecto con los valores:

Clase: "Esperando detección..."

Probabilidad: 0.0

Imagen: placeholder.jpg

El servidor renderizó la plantilla `index.html` junto con la información por defecto, cargando correctamente la interfaz inicial.

Posteriormente, se realizó una prueba adicional enviando una predicción real desde el microcontrolador. Tras actualizarse `alpha.diccionarioIdentificacion`, se volvió a solicitar la ruta /index.

El servidor cargó la interfaz nuevamente, esta vez incorporando la información real enviada por el ESP32-S3 CAM (clase detectada, probabilidad e imagen asociada).

Resultado:

La ruta /index mostró un comportamiento correcto tanto en estado inicial como en estado actualizado. Cuando no existían detecciones previas, el servidor devolvió un contenido por defecto coherente y funcional. Una vez que el microcontrolador envió datos reales, la interfaz se actualizó adecuadamente con la predicción y la imagen procesada. En ambos casos, el

servidor devolvió un código HTTP 200, confirmando que el renderizado y la entrega de información al frontend operan de manera estable y sin errores.

5.2.4 Prueba Modelo de Inteligencia Artificial

Para evaluar el desempeño de los modelos de clasificación de residuos se utilizaron métricas estándar de visión computacional, incluyendo exactitud (accuracy), precisión, recall y F1-score.

Modelo	Accuracy (%)	Precisión (%)	Recall (%)	F1-Score (%)	Tamaño (MB)	Observaciones
ResNet50 Dataset masivo imágenes de stock	92.49%	93%	92.5%	92.5%	218 MB	Buen desempeño, pero baja interpretación de objetos en la vida real
MobileNetV2 Dataset crudo	72.97%	79%	73%	71.5%	18.2 MB	Buen rendimiento, pero limitado debido al conjunto de datos
MobileNetV2 Dataset fusionados	75.51%	79%	75.5%	75%	18.2 MB	Comprensión real del entorno en la vida real, pero no entiende la visión del prototipo
MobileNetV2 Dataset fusionados con técnica de dominio iterativo	82.01%	82%	82%	82%	27.4 MB	La adaptación del dominio del dataset, mejoro su desempeño real sin sacrificar el entendimiento anterior de IA

Tabla 21 Métricas de los modelos de IA

ResNet50

ResNet50 es una arquitectura de redes neuronales convolucionales profunda que incorpora bloques residuales para facilitar el entrenamiento de modelos de gran profundidad. Su capacidad para mantener gradientes estables y extraer características de alto nivel la ha convertido en una de las arquitecturas más utilizadas en tareas de clasificación de imágenes.

En este proyecto se seleccionó ResNet50 debido a la idea errónea de que un modelo más complejo y potente necesariamente ofrecería mejores resultados. Sin embargo, las pruebas demostraron que, a pesar de su alta capacidad, el desempeño obtenido no se traduce en una utilidad real para aplicaciones prácticas.

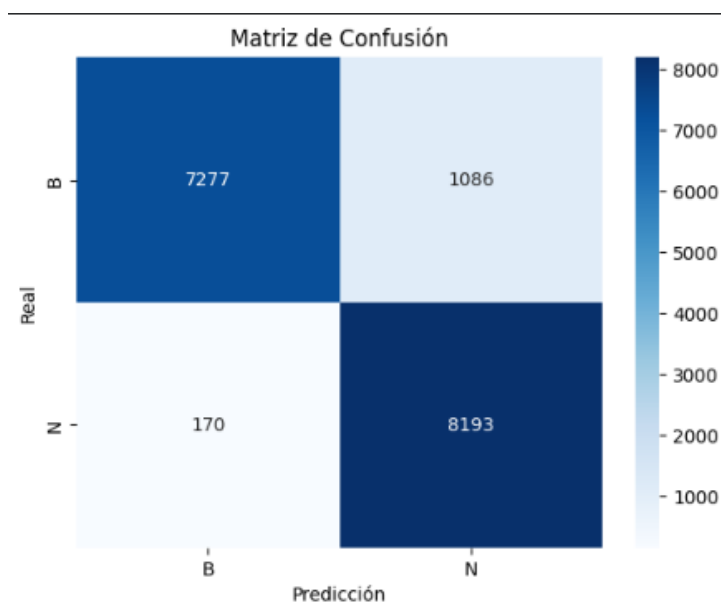


Ilustración 33 Matriz de confusión de ResNet50.

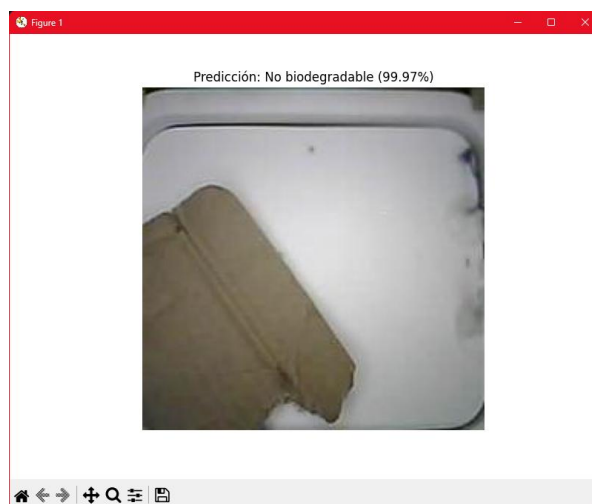


Ilustración 34 Prueba real ResNet50.

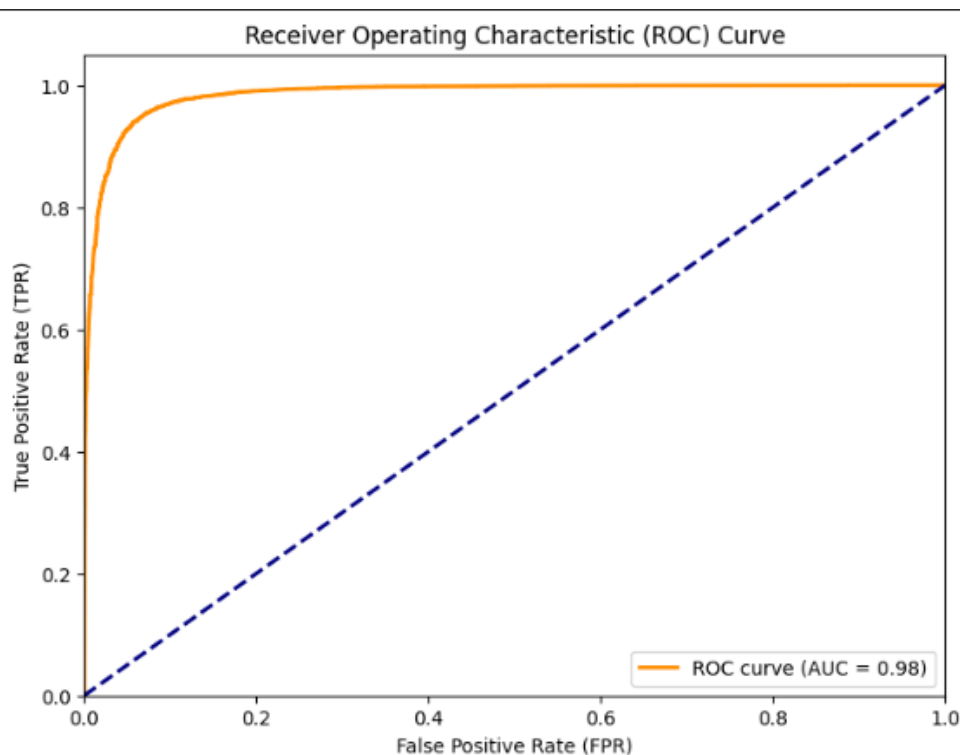


Ilustración 35 Curva ROC de ResNet50.

Aunque ResNet50 alcanzó niveles de confianza superiores al 90% durante las pruebas, estos resultados deben interpretarse con cautela. El rendimiento observado sugiere la presencia de sesgo en el modelo, principalmente debido a dos factores:

Falta de preprocesamiento adecuado del conjunto de datos

Las imágenes utilizadas para el entrenamiento no fueron normalizadas ni sometidas a técnicas de limpieza o equilibrio. Esto provoca que el modelo aprenda patrones ruidosos o irrelevantes, lo que incrementa la probabilidad de sobreajuste.

Dependencia exclusiva de imágenes de stock

El modelo se entrenó únicamente con imágenes obtenidas de internet, las cuales presentan condiciones ideales: buena iluminación, fondos limpios y objetos bien centrados.

En consecuencia, ResNet50 desarrolla una fuerte dependencia de estas condiciones controladas, lo que limita su capacidad para generalizar en entornos reales donde las imágenes presentan variaciones en iluminación, orientación, distancias y ruido visual.

Debido a estos factores, a pesar de los altos valores de confianza, el modelo resulta poco efectivo para escenarios reales. Su uso práctico queda restringido a contextos similares a los datos de entrenamiento.

MobileNetV2 Dataset Único

MobileNetV2 es una arquitectura optimizada para dispositivos con recursos limitados, diseñada a partir de bloques “inverted residual” y convoluciones separables en profundidad. Estas características reducen significativamente el número de parámetros y el costo computacional sin sacrificar en exceso la precisión del modelo.

Fue seleccionada como candidata principal para este proyecto debido a su bajo peso, rápida capacidad de inferencia y eficiencia energética, cualidades indispensables para sistemas embebidos y otros microcontroladores con recursos reducidos.

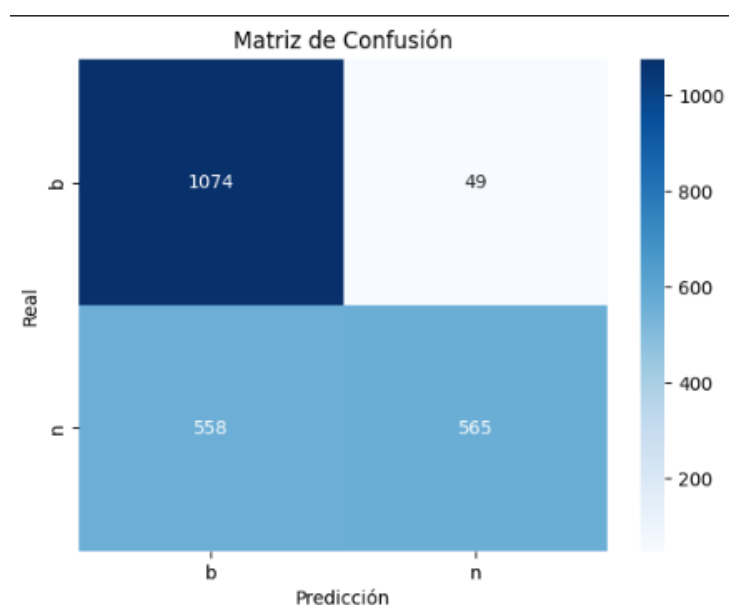


Ilustración 36 Matriz de confusión de MobileNetV2 primera iteración.



Ilustración 37 Prueba real MobileNetV2 primera iteración.

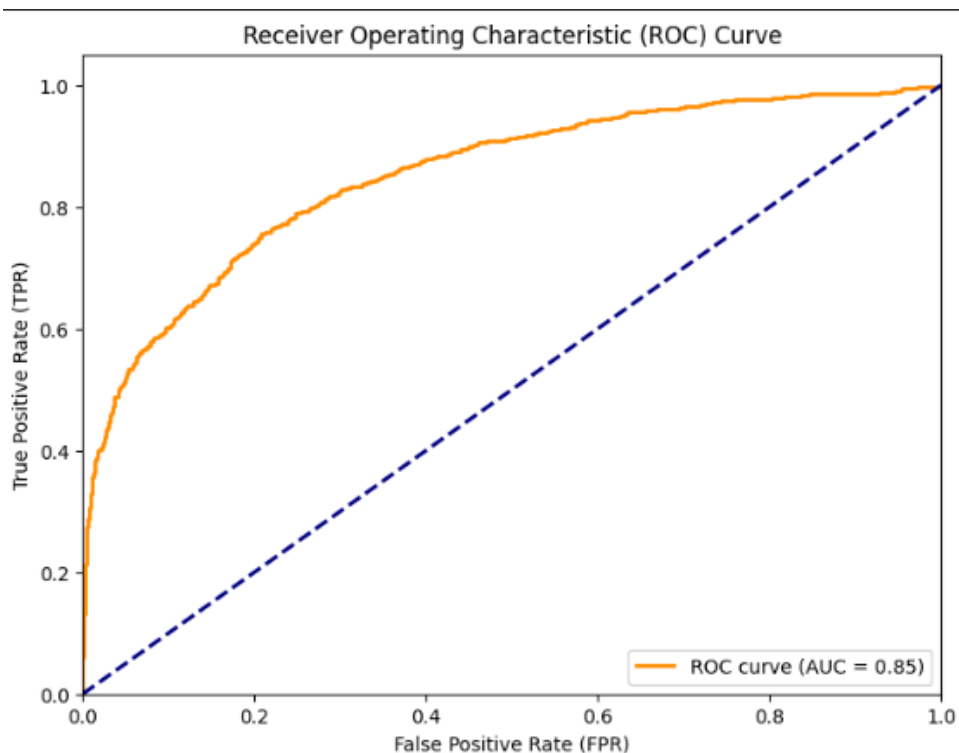


Ilustración 38 Curva Roc MobileNetV2 primera iteración.

MobileNetV2 se considera, en teoría, el modelo ideal para sistemas embebidos debido a sus tiempos de inferencia reducidos y a la facilidad con la que se puede transferir y ejecutar en hardware con capacidades limitadas. Sin embargo, durante esta primera iteración, su desempeño mostró limitaciones importantes derivadas principalmente del conjunto de datos utilizado:

Entrenamiento con un único dataset sin preprocesamiento adecuado

El modelo se entrenó exclusivamente con un conjunto de imágenes que no fue sometido a procesos de limpieza, balanceo o aumento de datos. Esto provoca que MobileNetV2 aprenda características inconsistentes y poco representativas, afectando directamente la calidad de la predicción.

Presencia significativa de sesgos

Debido a la falta de diversidad en el dataset y a la ausencia de técnicas de normalización o estandarización, el modelo desarrolló un sesgo marcado hacia los patrones presentes en las imágenes de entrenamiento.

A pesar de estas limitaciones, MobileNetV2 continúa siendo la arquitectura más adecuada para un despliegue en sistemas embebidos.

MobileNetV2 Fusión Dataset

En esta segunda iteración del modelo MobileNetV2 se realizó un proceso de mejora enfocado en aumentar la capacidad de generalización del sistema. Para ello, se combinaron dos conjuntos de datos provenientes de imágenes capturadas en entornos reales, con el objetivo de acercar el entrenamiento del modelo a las condiciones del prototipo físico. Esta fusión permitió diversificar los patrones visuales y reducir parte de los sesgos presentes en la primera versión del modelo, donde únicamente se utilizaba un dataset pequeño.

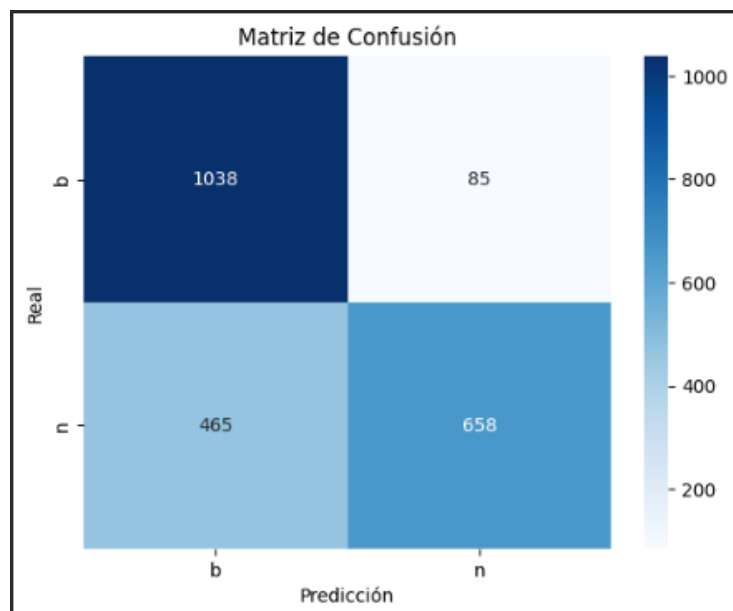


Ilustración 39 Matriz de confusión de MobileNetV2 segunda iteración.



Ilustración 40 Prueba real MobileNetV2 segunda iteración.

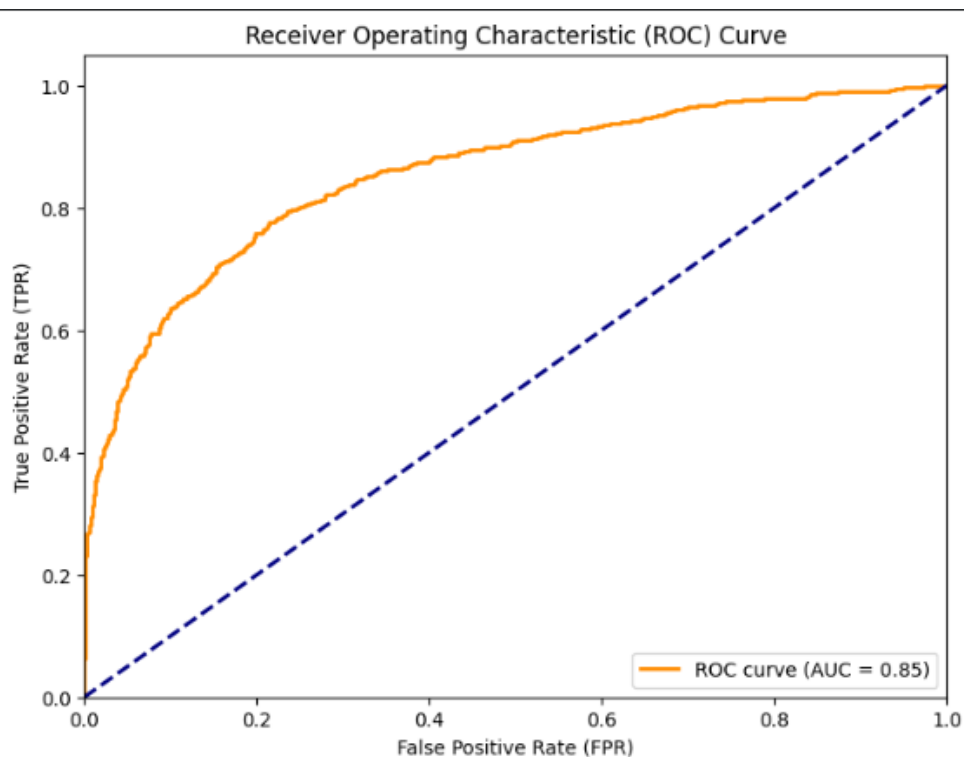


Ilustración 41 Curva Roc MobileNetV2 segunda iteración.

El desempeño del modelo MobileNetV2 en esta segunda iteración mostró una mejora considerable con respecto a la versión anterior. El método planteado hizo que el modelo adquiriera representaciones visuales más fieles a las condiciones de trabajo del prototipo, lo cual se reflejó en métricas superiores y una reducción de sesgos durante el proceso de clasificación.

A pesar de estos avances, el modelo aún no logró identificar correctamente los objetos dentro del prototipo. Este problema se explica principalmente por las diferencias persistentes entre las imágenes del dataset y las imágenes capturadas directamente por el microcontrolador, incluyendo iluminación, resolución, ruido visual y variaciones no contempladas en el entrenamiento.

Aunque todavía incapaz de clasificar los objetos en tiempo real dentro del sistema embebido, MobileNetV2 mostró un avance notable respecto a la iteración previa y estableció una base más sólida para futuras mejoras. La siguiente etapa requerirá integrar imágenes capturadas directamente desde el prototipo para cerrar la brecha entre el entorno de entrenamiento y el entorno operativo final.

MobileNetV2 Adaptación de Dominio y Fine-Tuning

En la tercera iteración del modelo MobileNetV2 se aplicó una mejora significativa basada en técnicas de datos, utilizando como base el modelo obtenido durante la segunda interacción. El objetivo fue reducir la brecha entre las condiciones del dataset y las condiciones reales del prototipo, integrando imágenes capturadas directamente desde el sistema con el fin de adaptar el modelo al entorno operativo final.

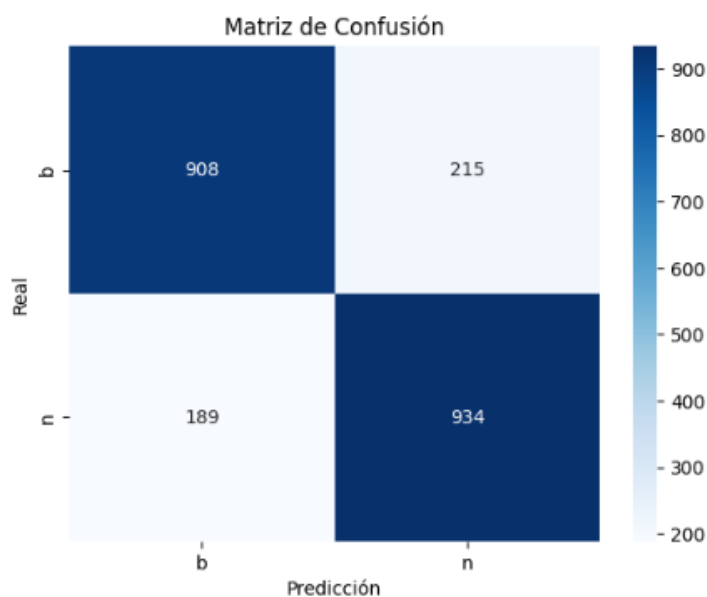


Ilustración 42 Matriz de confusión de MobileNetV2 tercera iteración.



Ilustración 43 Prueba real MobileNetV2 tercera iteración.

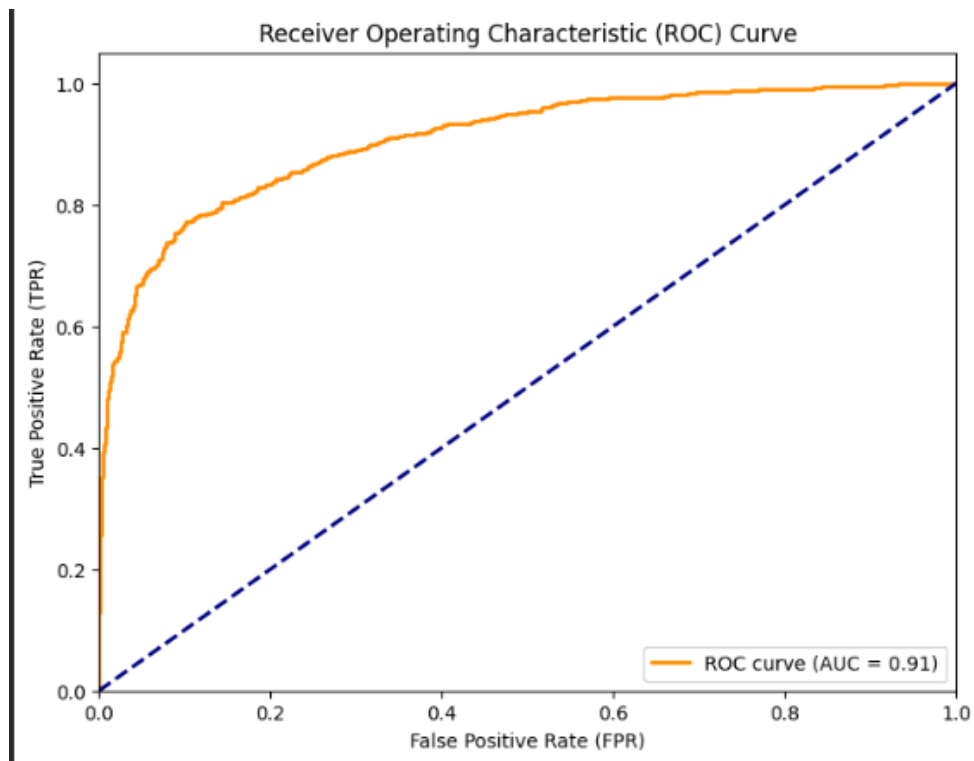


Ilustración 44 Curva Roc MobileNetV2 tercera iteración.

La tercera iteración del modelo demostró un desempeño notablemente superior respecto a las versiones anteriores. La aplicación de dominio iterativo, sumado al fine-tuning sobre el modelo previamente entrenado, permitió que MobileNetV2 adaptara sus representaciones internas a las condiciones reales capturadas por la cámara del prototipo.

Este proceso reveló un hallazgo clave: el problema principal no era la cámara ni la arquitectura del modelo, sino la calidad y pertinencia de los datos utilizados en las primeras etapas. Al incorporar imágenes reales del entorno operativo, el modelo logró comprender de manera mucho más precisa los patrones visuales reales, reduciendo drásticamente los errores de clasificación.

Los resultados obtenidos superaron incluso a modelos previos que alcanzaban más del 90% de confianza en pruebas artificiales, evidenciando que la mejora no provino de un aumento en la complejidad del modelo, sino de la correcta alineación entre los datos de entrenamiento y el dominio real.

Para finalizar esta iteración posiciona a MobileNetV2 como el modelo más adecuado y robusto para el sistema propuesto, demostrando una capacidad de adaptación destacable y un rendimiento altamente consistente bajo condiciones reales.

5.3 Resultados finales

Los resultados obtenidos durante las pruebas permiten evaluar el nivel de cumplimiento de los requerimientos funcionales y no funcionales establecidos en las primeras etapas del proyecto. Con base en este análisis, se confirma que el sistema logró implementar de manera efectiva el flujo completo de captura, procesamiento y clasificación automatizada de residuos, cumpliendo la mayoría de los objetivos propuestos.

El sistema fue capaz de capturar imágenes automáticamente mediante la cámara ESP32-S3 CAM, integrándose adecuadamente con el microcontrolador y el servidor. Esta captura se realizó de forma estable y en tiempos adecuados para el proceso de clasificación.

El modelo de inteligencia artificial, especialmente en su tercera iteración con MobileNetV2, mostró un avance significativo en su capacidad para clasificar residuos entre orgánicos e inorgánicos en condiciones reales, acercándose a un comportamiento apto para operación en prototipo.

El servidor procesó las imágenes y devolvió respuestas de manera inmediata, cumpliendo con el flujo diseñado para una clasificación continua. Asimismo, se integró correctamente un mecanismo físico controlado por el servomotor, encargado de iniciar el proceso de separación del residuo una vez obtenida la predicción del modelo.

Por último, la interfaz desarrollada permitió mostrar el estado de la clasificación en tiempo real mediante la actualización dinámica de la imagen y la información del objeto detectado, cumpliendo el requisito de ofrecer retroalimentación clara al usuario.

Sin mencionar que el sistema mantuvo un bajo consumo energético al utilizar hardware compacto y eficiente, como el ESP32-S3 y el servomotor SG90. El tiempo de inferencia del servidor — considerando únicamente la predicción del modelo— se mantuvo por debajo de los 2 segundos, cumpliendo el límite establecido para garantizar un funcionamiento fluido del sistema.

Adicionalmente, toda la infraestructura operó mediante WiFi local, permitiendo la comunicación entre el prototipo y el servidor sin necesidad de conexiones externas. Se emplearon componentes económicos y fáciles de reemplazar, cumpliendo con el objetivo de accesibilidad del sistema. Finalmente, el prototipo demostró una tolerancia básica a condiciones ambientales variables gracias a las mejoras realizadas en iluminación, parámetros de la cámara y estabilidad del modelo.

El sistema propuesto cumplió satisfactoriamente con los requerimientos funcionales y no funcionales planteados. Si bien se identificaron áreas de mejora, especialmente relacionadas con la variabilidad del entorno, la necesidad de ampliar y refinar los datos de entrenamiento, los resultados obtenidos validan la viabilidad del prototipo así establecen una base sólida para futuras iteraciones más robustas y eficientes.

Glosario

- **Actuadores:** Dispositivos que reciben señales eléctricas y las convierten en acciones físicas, como movimiento, luz o sonido. Ejemplos comunes son motores, relés y servomotores.
- **Arduino:** Plataforma de hardware y software de código abierto basada en microcontroladores, ampliamente utilizada en prototipado electrónico y proyectos educativos por su facilidad de uso y amplia comunidad de desarrollo.
- **Arquitectura DenseNet121:** Red neuronal convolucional profunda caracterizada por la conexión directa de cada capa con todas las anteriores. Este diseño reduce la redundancia, mejora el flujo de gradientes y permite un uso eficiente de parámetros.
- **Arquitectura InceptionResNetV2:** Modelo de red neuronal que combina bloques Inception con conexiones residuales, lo que incrementa la capacidad de aprendizaje y mejora la precisión en tareas de clasificación de imágenes a gran escala.
- **Arquitectura del software:** Diseño estructural de un sistema de software que define la organización de los módulos, la interacción entre ellos y los principios que guían su desarrollo, garantizando escalabilidad y mantenibilidad.
- **Aprendizaje Automático (Machine Learning):** Rama de la IA que permite a las máquinas aprender patrones a partir de datos y mejorar su desempeño en una tarea sin ser programadas explícitamente para cada caso.
- **Aprendizaje Profundo (Deep Learning):** Subcampo del aprendizaje automático basado en redes neuronales artificiales con múltiples capas, que permite resolver tareas complejas como la visión por computadora o el reconocimiento de voz.
- **Automatización:** Uso de tecnologías que permiten ejecutar procesos sin intervención humana directa, con el fin de mejorar la eficiencia y reducir errores.
- **Basura o residuo:** Material desechado después de haber cumplido su función principal, que puede clasificarse como orgánico o inorgánico.
- **Clasificación de residuos:** Proceso de separar los desechos en diferentes categorías (orgánico, plástico, papel, metal, vidrio, entre otros) para su adecuado reciclaje o disposición final.
- **ConvoWaste:** Proyecto de investigación que aplica redes neuronales convolucionales (CNN) para la clasificación inteligente de residuos, demostrando la viabilidad de aplicar aprendizaje profundo en sistemas de reciclaje automatizado.
- **Dataset (Conjunto de Datos):** Colección estructurada de datos utilizada para entrenar, validar o probar algoritmos de aprendizaje automático e inteligencia artificial.

- **Dataset WasteNet:** Conjunto de datos desarrollado por Recycleye que reúne más de 2.5 millones de imágenes de residuos en condiciones reales de reciclaje, considerado uno de los más grandes y completos en este campo.
- **Dispositivos Edge Computing:** Equipos diseñados para procesar datos cerca de la fuente de generación (como cámaras o sensores), reduciendo la necesidad de enviar toda la información a servidores remotos. Mejoran la latencia y optimizan el uso de la red.
- **Economía Circular:** Modelo económico que busca minimizar residuos y aprovechar al máximo los recursos a través de la reutilización, reciclaje y reducción de desechos.
- **ESP32-CAM:** Variante del ESP32 que incluye una cámara integrada, lo que permite capturar imágenes o video y transmitirlos para su procesamiento.
- **Hardware:** Conjunto de componentes físicos y tangibles de un sistema informático, como procesadores, placas electrónicas, sensores y dispositivos de almacenamiento.
- **ImageNet:** Dataset de referencia en visión por computadora que contiene más de 14 millones de imágenes etiquetadas en más de 20 000 categorías. Ha sido utilizado ampliamente en el entrenamiento y evaluación de modelos de aprendizaje profundo.
- **Microcontrolador:** Dispositivo electrónico programable que integra en un solo chip una unidad de procesamiento, memoria y puertos de entrada/salida. Se utiliza para controlar sistemas embebidos y ejecutar tareas específicas de automatización.
- **MobileNetV2:** Arquitectura de red neuronal ligera optimizada para dispositivos móviles y sistemas embebidos. Utiliza capas de convolución separables en profundidad para reducir el consumo de memoria y mantener una alta precisión.
- **Modelo ContamiNet:** Arquitectura de red neuronal enfocada en detectar contaminantes en residuos reciclables. Se utiliza para mejorar la calidad del material recuperado al identificar objetos no deseados en flujos de reciclaje.
- **Pantalla TFT ILI9341:** Pantalla gráfica que se conecta a microcontroladores como el ESP32-CAM, utilizada para mostrar información visual o textual en proyectos embebidos.
- **Red Neuronal Convolucional (CNN):** Tipo de red neuronal utilizada principalmente en el procesamiento y clasificación de imágenes, capaz de extraer características visuales como bordes, texturas y formas.
- **Residuos inorgánicos:** Desechos no biodegradables, como plásticos, vidrios, metales o empaques, que requieren procesos industriales para su reciclaje.
- **Residuos orgánicos:** Desechos de origen biológico, principalmente restos de comida o materiales biodegradables.
- **Sensor ultrasónico:** Dispositivo que mide distancias mediante la emisión y recepción de ondas sonoras de alta frecuencia. Se usa comúnmente en robótica y sistemas de detección de objetos.

- **Sensores:** Dispositivos capaces de captar variaciones físicas o químicas del entorno (como luz, temperatura, movimiento o proximidad) y transformarlas en señales eléctricas procesables por un sistema electrónico.
- **Servidor Flask:** Framework minimalista de Python para el desarrollo de aplicaciones web, que en este proyecto se utiliza para recibir imágenes, procesarlas con IA y devolver los resultados de la clasificación.
- **Servomotores:** Motores eléctricos que permiten controlar con precisión la posición, velocidad y aceleración de un eje, gracias a un sistema de realimentación interna. Son esenciales en robótica y automatización.
- **Software:** Conjunto de programas, instrucciones y datos que permiten al hardware ejecutar tareas específicas, desde sistemas operativos hasta aplicaciones de usuario.
- **TensorFlow:** Biblioteca de código abierto desarrollada por Google, utilizada para crear y entrenar modelos de aprendizaje automático y profundo.
- **Transfer Learning (Aprendizaje por Transferencia):** Técnica de entrenamiento en IA que aprovecha modelos previamente entrenados en grandes bases de datos y los adapta a un nuevo problema con menos recursos.
- **TrashNet:** Dataset de imágenes creado en 2016 que contiene fotografías de distintos tipos de residuos (plástico, papel, metal, vidrio, cartón y desechos generales). Se utiliza ampliamente para entrenar modelos de clasificación automática de basura mediante visión por computadora.
- **Visión por Computadora:** Área de la inteligencia artificial que permite a los sistemas interpretar y procesar información visual proveniente de imágenes o videos.
- **ESP32:** Microcontrolador de bajo costo y bajo consumo de energía, con conectividad WiFi y Bluetooth, utilizado en proyectos de IoT, automatización y control de dispositivos.
- **SCRUM:** Metodología ágil de desarrollo que organiza el trabajo en ciclos iterativos llamados sprints, promoviendo la colaboración, la adaptación a cambios y la entrega continua de resultados funcionales.

Capítulo 6

Conclusiones

Este trabajo abordó de manera integral el desarrollo de un sistema inteligente para la clasificación automatizada de residuos, integrando hardware embebido, un servidor backend, un modelo de inteligencia artificial y una interfaz visual de monitoreo. Se estableció un enfoque centrado en la captura automática de imágenes, el procesamiento en tiempo real y la capacidad del sistema para operar en condiciones en un entorno real.

El proceso de desarrollo se llevó a cabo mediante ciclos iterativos que permitieron identificar limitaciones, corregir fallas y optimizar componentes tanto de hardware como de software. Siendo un punto relevante la importancia de la calidad del conjunto de datos para el desempeño del modelo de IA.

Las pruebas realizadas permitieron validar el funcionamiento del sistema en sus diferentes niveles: capturas estables por parte de la cámara, envío y recepción de imágenes mediante WiFi, procesamiento inmediato del servidor, despliegue de información en la interfaz y activación del mecanismo físico. Aunque se identificaron desafíos relacionados con la variabilidad de iluminación, latencia de red y la todavía limitada generalización del modelo ante condiciones muy específicas, los resultados finales mostraron que el sistema cumple con los requerimientos planteados, tanto funcionales como no funcionales.

En el ámbito personal, el desarrollo de este proyecto representó un proceso de crecimiento técnico y profesional. La interacción con áreas como redes neuronales, diseño de hardware, optimización de software y pruebas experimentales permitió fortalecer habilidades de análisis, resolución de problemas y pensamiento sistémico, así como adquirir experiencia real en el desarrollo de soluciones tecnológicas integradas.

Finalmente, se plantean diversas líneas de mejora futura, tales como la ampliación del dataset con capturas directas del prototipo, la optimización del flujo de transmisión de imágenes, la reducción de latencia y la exploración de técnicas avanzadas de visión artificial. Estas recomendaciones permitirán robustecer el prototipo y avanzar hacia un sistema más preciso, confiable.

De este modo, el presente trabajo ofrece una visión completa del desarrollo del sistema propuesto, sus fundamentos, iteraciones y resultados, constituyendo una base sólida para futuros proyectos y una referencia útil para el diseño de soluciones inteligentes orientadas a la gestión y clasificación de residuos.

Referencias

- [1] United Nations. (s.f.). Objetivos de Desarrollo Sostenible. <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>
- [2] Secretaría de Medio Ambiente y Recursos Naturales [SEMARNAT]. (2025). Residuos sólidos urbanos y de manejo especial. Gobierno de México. <https://www.gob.mx/semarnat/acciones-y-programas/residuos-solidos-urbanos-y-de-manejo-especial>
- [3] Secretaría del Medio Ambiente [SEDEMA]. (2025). Programa de Residuos Sólidos. Gobierno de la Ciudad de México. <https://sedema.cdmx.gob.mx/programas/programa/residuos-solidos>
- [4] Jefatura de Gobierno de la Ciudad de México. (2025). Instala Clara Brugada órgano de gobierno de la Agencia de Gestión Integral de Residuos y anuncia campaña 'Transforma tu ciudad, cada basura en su lugar'. <https://jefaturadegobierno.cdmx.gob.mx/comunicacion/nota/instala-clara-brugada-organo-de-gobierno-de-la-agencia-de-gestion-integral-de-residuos-y-anuncia-campana-transforma-tu-ciudad-cada-basura-en-su-lugar>
- [5] Sánchez Yáñez, J. M., & Márquez Benavides, L. (2024). Gestión de residuos sólidos y la inteligencia artificial en el contexto mexicano. Ciencia Nicolaita, (90). <https://doi.org/10.35830/cn.vi90.722>
- [6] AMCS Group. (2025). AMCS Vision AI. <https://www.amcsgroup.com/es/soluciones/amcs-vision-ai>
- [7] Recycleye. (2025). La IA y el reconocimiento de residuos. <https://recycleye.com/es/la-ia-y-el-reconocimiento-de-residuos>
- [8] Remeo y Zen Robotics. (2025). Robots de reciclaje. Reduce Reutiliza Recicla. <https://reducereutilizarecicla.org/robots-reciclaje>
- [9] Hablando en Vidrio. (2025). Contenedores inteligentes y gestión de residuos. <https://hablandoenvidrio.com/inteligencia-artificial-mejor-gestion-de-residuos>
- [10] Banco Interamericano de Desarrollo. (2025). Inteligencia artificial y economía circular en residuos. IADB Blogs. <https://blogs.iadb.org/agua/es/inteligencia-artificial-hacia-una-economia-circular-y-la-eficiencia-en-la-gestion-de-residuos>
- [11] Domínguez Sánchez, M. A., Maldonado Juárez, M. M., Hernández Cadena, F. A., & Alanís Carranza, L. E. (2024). Algoritmo para la clasificación de residuos reciclables utilizando redes neuronales convolucionales [Ponencia]. Congreso Estudiantil de Inteligencia Artificial Aplicada a la Ingeniería y Tecnología, UNAM-FESC, Estado de México, México.

[12] Alanís Carranza, L. E., Altamirano Arroyo, A. I., Jiménez Hernández, E. R., Plasencia Gonzáles, I. D., Fuentes Luna, J. A., Pérez Luna, I. A., & Sánchez Hernández, E. (2024). Detección y clasificación de desperdicios humanos a través de YOLOv8 y una cámara web. *Pistas Educativas*, 45(147), 203–218.

[13] Robledo, R. (2024, 13 de agosto). Gana premio en NL estudiante del Tec con robot para basura con IA. *La Jornada*. <https://www.jornada.com.mx/noticia/2024/08/13/estados/gana-premio-en-nl-estudiante-del-tec-con-robot-para-basura-con-ia-5070>

[14] PortalAmbiental.com.mx. (2023, 20 de febrero). Tomra presentará soluciones para la industria mexicana del reciclaje en Residuos Expo. *PortalAmbiental*. <https://www.portalambiental.com.mx/empresas/20230220/tomra-presentara-soluciones-para-la-industria-mexicana-del-reciclaje-en-residuos>