SISTEMA INTELIGENTE PARA LA CLASIFICACIÓN Y SEGMENTACIÓN DE RESIDUOS RECIBOT

PROYECTO DE GRADO

Arnol Arley Stivel Garzon Garzon

Fredy Leiva Tapiero

Daniel Alejandro Tibavizco Muñoz

Profesor: John Jairo Porras

UNIVERSIDAD EAN

FACULTAD DE INGENIERÍA

BOGOTÁ D.C.

2025

Tabla de contenido

Resumen ejecutivo

La gestión ineficiente de residuos en Colombia genera un impacto ambiental significativo y dificulta la implementación de procesos de reciclaje efectivos. En respuesta a esta problemática, el proyecto ReciBot propone el desarrollo de una aplicación basada en inteligencia artificial para la identificación y clasificación automática de residuos a partir de imágenes. Esta solución busca optimizar la disposición final de los desechos y fomentar una cultura de reciclaje en la población.

ReciBot integra modelos de aprendizaje profundo entrenados con TensorFlow y OpenCV para analizar imágenes y determinar el tipo de residuo. El sistema cuenta con dos versiones: una aplicación web y una aplicación de escritorio. La versión web, desarrollada con Flask y Django, permite a los usuarios cargar imágenes de residuos para su procesamiento en el backend, donde el modelo entrenado genera un resultado con la categoría correspondiente. La versión de escritorio, diseñada con custom tkinter, incorpora una interfaz gráfica intuitiva que, además del análisis de imágenes estáticas, ofrece detección en tiempo real utilizando cámaras integradas.

El desarrollo del proyecto abarca varias fases, iniciando con la recopilación y preprocesamiento de datos, la implementación y optimización del modelo de clasificación, y la creación de interfaces accesibles y funcionales para los usuarios. Adicionalmente, el sistema genera recomendaciones sobre la disposición adecuada de los residuos, indicando en qué contenedor deben ser depositados según su tipo.

Esta iniciativa busca aprovechar el potencial de la inteligencia artificial para mejorar la gestión de residuos y reducir el impacto ambiental. Al promover una adecuada separación de desechos, ReciBot contribuye a la reducción de la contaminación y a la concienciación de la población, especialmente entre los jóvenes, sobre la importancia de prácticas sostenibles.

Introducción

En la actualidad, la gestión de residuos sólidos representa un desafío ambiental y social crítico debido al crecimiento acelerado de la generación de desechos y su inadecuada disposición. Esta problemática contribuye a la contaminación del suelo, el agua y el aire, afectando tanto al ecosistema como a la salud pública (Programa de las Naciones Unidas para el Medio Ambiente [PNUMA], 2023). De acuerdo con el PNUMA, es imperativo superar la "era de los desechos" mediante la adopción de estrategias de economía circular que prioricen la reducción, reutilización y reciclaje de materiales. Sin embargo, en América Latina, esta transición se ve obstaculizada por la insuficiencia de infraestructura para la gestión de residuos y la falta de educación ambiental en la población (PNUMA, 2023).

Uno de los principales desafíos en este ámbito es la clasificación eficiente de los residuos, ya que muchas personas desconocen cómo separar correctamente los materiales reciclables de los desechos orgánicos y no reciclables. Este problema afecta la eficiencia del reciclaje y dificulta el aprovechamiento de materiales reutilizables, reduciendo así el impacto positivo que la economía circular puede generar (Manchasoft, 2023). Según informes recientes, la correcta gestión de residuos no solo reduce la contaminación, sino que también representa una oportunidad económica significativa. La industria del reciclaje genera ingresos de aproximadamente 500.000 millones de dólares anuales y emplea a más de 1,6 millones de personas en todo el mundo (Manchasoft, 2023). Además, implementar estrategias de reutilización y reciclaje podría reducir la contaminación por plásticos en un 80 % para 2040, generar ahorros de hasta 1,27 billones de dólares y crear 700.000 empleos en países en desarrollo (PNUMA, 2023).

En este contexto, el presente proyecto propone el desarrollo de ReciBot, una aplicación basada en inteligencia artificial que facilita la clasificación de residuos sólidos

mediante imágenes. La solución se fundamenta en el uso de modelos de aprendizaje profundo para reconocer diferentes tipos de desechos y proporcionar recomendaciones sobre su disposición adecuada. La implementación de este sistema se estructura en dos versiones: una aplicación web y una de escritorio. La versión web permite a los usuarios subir imágenes de residuos para su análisis en un backend desarrollado con Flask y Django, mientras que la versión de escritorio, creada con custom tkinter, ofrece una interfaz intuitiva que incorpora detección en tiempo real a través de cámaras integradas.

A través de esta iniciativa, se busca mejorar la eficiencia en la separación de residuos, optimizar los procesos de reciclaje y concienciar a la población sobre la importancia de la gestión adecuada de desechos. Con la combinación de tecnología, inteligencia artificial y educación ambiental, el proyecto tiene el potencial de contribuir a la reducción de la contaminación y al fomento de prácticas sostenibles dentro de la sociedad.

Objetivos

Objetivo General

Desarrollar un aplicativo inteligente basado en visión artificial para la clasificación automática de residuos a partir de imágenes, proporcionando recomendaciones para su adecuado reciclaje.

Objetivos Específicos

1. Desarrollar un modelo de clasificación de residuos basado en redes neuronales convolucionales (CNN) y técnicas de transferencia de aprendizaje, utilizando un conjunto de imágenes preprocesadas.
2. Implementar una aplicación web y de escritorio con interfaces interactivas para la detección y clasificación de residuos en tiempo real, integrando Flask, Django, OpenCV y *custom tkinter*.
3. Optimizar el desempeño del modelo mediante el ajuste de hiperparámetros y la evaluación con métricas como precisión, *accuracity* y *loss*, asegurando eficiencia y usabilidad.

Planteamiento del Problema

En Colombia, una gran parte de la población desconoce cómo clasificar correctamente los residuos, lo que afecta negativamente los procesos de reciclaje y contribuye al aumento de desechos en los basureros. Según un artículo de El Tiempo, basado en una encuesta sobre hábitos de reciclaje, el 15% de los encuestados nunca o casi nunca separan los residuos, y sorprendentemente, la mayoría de este grupo está compuesto por jóvenes (El Tiempo, 2023). Además, existe un porcentaje significativo de personas que, aunque saben cómo reciclar, no están interesadas en hacerlo. Esto se evidencia en la afirmación "Sólo el 42 por ciento estuvo en desacuerdo con la afirmación ‘No importa si separo mis residuos, porque al final juntan toda la basura’" (El Tiempo, 2023). Esto demuestra que, aunque hay cierto nivel de conciencia sobre la importancia del reciclaje, persisten barreras en la implementación de buenas prácticas ambientales.

Según el artículo ¿Cuáles son las mayores dificultades para reciclar? Esto dicen los colombianos, uno de los principales desafíos es la falta de información y herramientas que

faciliten la correcta separación de residuos (El Espectador, 2023). La ausencia de tecnologías accesibles que ayuden a los ciudadanos en este proceso genera errores en la clasificación y reduce la eficiencia del reciclaje. Además, la inadecuada separación de residuos afecta el aprovechamiento de materiales reciclables y propicia la sobreexplotación de recursos naturales.

A pesar de que existen campañas de concienciación, sigue siendo difícil garantizar que las personas separen correctamente sus residuos. En muchas ciudades y municipios no hay suficientes infraestructuras o programas que faciliten el reciclaje, lo que agrava aún más la problemática. Es aquí donde la tecnología puede desempeñar un papel fundamental. La implementación de una aplicación basada en inteligencia artificial, capaz de identificar y clasificar residuos a partir de imágenes, representa una solución innovadora y accesible para enfrentar esta problemática. Un sistema automatizado y fácil de usar podría mejorar significativamente la gestión de residuos y fomentar una cultura de reciclaje más efectiva en la sociedad.

Pregunta de Investigación

¿Cómo pueden las técnicas de inteligencia artificial y la ingeniería de software contribuir a la solución del problema de clasificación de desechos?

Delimitación del Proyecto

El presente proyecto se centra en el desarrollo e implementación de ReciBot, un sistema basado en inteligencia artificial para la clasificación automática de residuos mediante imágenes. La solución estará compuesta por:

- **Una aplicación web:** Permitirá a los usuarios subir imágenes de residuos, las cuales serán procesadas por un modelo de aprendizaje profundo alojado en un backend basado en Flask o Django. El sistema devolverá la categoría del residuo y una recomendación sobre su disposición adecuada.

- **Una aplicación de escritorio:** Diseñada con Custom Tkinter, permitirá a los usuarios capturar imágenes en tiempo real utilizando OpenCV y clasificarlas automáticamente con el modelo de IA.

Ámbito geográfico: El proyecto se enfoca en Colombia, tomando en cuenta datos y problemáticas locales sobre la gestión de residuos.

Ámbito temporal: Se desarrollará en un periodo de seis meses, incluyendo las fases de recopilación de datos, entrenamiento del modelo, desarrollo de interfaces y pruebas del sistema.

Ámbito tecnológico: Se utilizarán bibliotecas como TensorFlow, OpenCV, Pillow, Flask, Django, Custom Tkinter, entre otras. El modelo de IA se entrenará mediante técnicas de aprendizaje profundo y transfer learning para mejorar su precisión en la clasificación de residuos.

Ámbito poblacional: La herramienta está dirigida a ciudadanos interesados en mejorar sus prácticas de reciclaje, con un enfoque especial en jóvenes, quienes según estudios

presentan mayor desinterés en la separación de residuos. También puede ser utilizada por instituciones educativas y empresas de gestión ambiental.

Justificación

La implementación y desarrollo de ReciBot, un clasificador inteligente de residuos basado en inteligencia artificial, se justifica por la creciente necesidad de mejorar los procesos de separación de residuos en la población. La incorrecta clasificación de desechos no solo dificulta el reciclaje, sino que también contribuye a la contaminación ambiental y al desperdicio de recursos aprovechables.

El uso de un sistema automatizado permitirá generar un impacto positivo en la conciencia ciudadana. Al implementar una solución tecnológica innovadora, se refuerza la percepción de la clasificación de residuos como un problema relevante que requiere atención y modernización. Según Sectorial (2023), Colombia genera anualmente 24,8 millones de toneladas de residuos, de los cuales solo el 20% se recicla, lo que evidencia la falta de eficiencia en los procesos actuales de recolección y clasificación. Esta problemática es aún más crítica en espacios de alta concurrencia y entre sectores poblacionales jóvenes, quienes, de acuerdo con un artículo de El Tiempo (2023), presentan los niveles más bajos de compromiso con el reciclaje.

Además del impacto ambiental, la correcta clasificación y disposición de los residuos puede reducir significativamente la cantidad de desechos enviados a rellenos sanitarios, mitigando así la contaminación y disminuyendo la huella de carbono. Por ejemplo, en un edificio altamente concurrido, si un 15% de las personas no separa adecuadamente sus

residuos, la acumulación de basura mal clasificada aumenta, afectando la eficiencia de los sistemas de reciclaje.

El desarrollo de ReciBot también tiene beneficios sociales y educativos, ya que la herramienta no solo contribuirá a la protección del medio ambiente, sino que también fomentará una cultura de reciclaje más sólida. Al proporcionar información clara sobre la correcta separación de residuos y promover el uso adecuado de los contenedores recicladores, la aplicación ayudará a generar hábitos responsables y sostenibles en la comunidad.

Finalmente, la implementación de ReciBot representa una solución innovadora y accesible para abordar la problemática del reciclaje en Colombia, combinando tecnología e inteligencia artificial para facilitar la clasificación de residuos y fomentar una gestión ambiental más eficiente.

Análisis de Requerimientos

El desarrollo exitoso de un proyecto se mide a través de múltiples factores, entre ellos la consecución de los objetivos dentro del tiempo establecido y el cumplimiento de los requerimientos funcionales. Para garantizar que el diseño final sea eficiente y funcional, es fundamental definir desde el inicio las especificaciones de diseño, establecer el alcance de la solución de ingeniería y minimizar cambios en etapas tardías del proyecto.

Este análisis de requerimientos considera tres aspectos principales: la intención del producto, la verificación de parámetros de diseño y la estimación de características clave del sistema, como potencia y desempeño.

Intención del Producto

El proyecto tiene como objetivo desarrollar un sistema de inteligencia artificial para la clasificación de documentos e imágenes mediante una sesión web basada en Flask, así como una versión de escritorio utilizando Tkinter. Además, se implementará un modelo basado en YOLO para mejorar la precisión en la detección de objetos. La solución debe permitir a los usuarios cargar archivos, procesarlos con modelos de IA y recibir información en tiempo real sobre su contenido.

Verificación de Parámetros de Diseño

El desarrollo se realizará en Python 3.x, asegurando compatibilidad con las siguientes tecnologías:

- Flask para la interfaz web, con integración de HTML, CSS y JavaScript.
- CustomTkinter para la versión de escritorio, proporcionando una solución offline.
- TensorFlow/Keras para la implementación del modelo de clasificación de documentos.
- YOLOv8 con PyTorch para la detección avanzada de objetos en imágenes.
- PIL y OpenCV para el procesamiento de imágenes, asegurando su correcta adaptación al modelo.
- NumPy y Pandas para la manipulación y análisis de datos.

Estimación de Características de Diseño

El sistema debe ofrecer un alto desempeño en el procesamiento de imágenes y documentos, optimizando tiempos de inferencia en modelos de IA. En términos de usabilidad, la interfaz debe ser intuitiva, con tiempos de respuesta optimizados y compatibilidad con múltiples formatos de archivos. La solución debe garantizar escalabilidad, permitiendo la integración de nuevos modelos en el futuro sin afectar el rendimiento del sistema.

Marco Teórico

Concepto

El manejo de los residuos sólidos es un reto tanto ambiental como social. La clasificación adecuada de los desechos influye directamente en la eficiencia del reciclaje y en la sostenibilidad del planeta. La economía circular busca reducir estos impactos fomentando la reutilización y el reciclaje. Según el Programa de las Naciones Unidas para el Medio Ambiente (PNUMA, 2023), mejorar estos procesos podría disminuir la contaminación por plásticos en un 80% para 2040.

En Colombia, muchas personas aún no saben cómo separar adecuadamente los residuos, lo que dificulta su aprovechamiento. Un informe de El Tiempo (2023) señala que el 15% de la población no separa la basura, y dentro de este grupo, los jóvenes representan la mayoría. Además, el 42% de los encuestados cree que separar los residuos no sirve de nada, ya que terminan mezclados de todas formas. La falta de infraestructura y tecnología adecuada

también dificulta la implementación de estrategias efectivas para mejorar el reciclaje (PNUMA, 2023).

Para enfrentar este problema, nace ReciBot, una solución innovadora que utiliza inteligencia artificial y visión por computadora para clasificar automáticamente los residuos. Su uso no solo ayudará a reducir la cantidad de desechos mal clasificados, sino que también contribuirá a disminuir la contaminación y a fortalecer la educación ambiental. Además, esta tecnología representa un paso importante hacia la optimización de los procesos de reciclaje, incentivando hábitos más sostenibles dentro de la comunidad (Manchasoft, 2023).

El Reciclaje

El reciclaje es un proceso importante en la correcta gestión de residuos que consiste en la recolección, clasificación, procesamiento y reutilización de materiales descartados para convertirlos en nuevos productos, en pocas palabras darle una segunda vida útil a diferentes materiales. El reciclaje y la gestión eficiente de residuos representan un pilar fundamental en la transición hacia un modelo de economía circular y una reducción del impacto ambiental. Según Tallini y Cedola, la reutilización de materiales desechados en diversos sectores, como la construcción, permite disminuir la demanda de materias primas y reducir las emisiones de gases de efecto invernadero.

Beneficios del Reciclaje:

El reciclaje aporta múltiples beneficios ambientales, económicos y sociales, algunos de estos son:

Conservación de recursos naturales: Reduce la explotación de materias primas como madera, minerales y petróleo, al reutilizar materiales existentes. También la producción de

materiales reciclados suele requerir menos energía en comparación con la fabricación de materias primas nuevas.

Un beneficio económico sería el tema de la generación de empleo al crear oportunidades en la recolección, clasificación y procesamiento de residuos. Además las empresas pueden disminuir gastos al utilizar materiales reciclados en lugar de materias primas nuevas.

Por último un beneficio social, es dar conciencia a los ciudadanos al fomentar hábitos de consumo responsable y sensibiliza a la población sobre la importancia de la sostenibilidad.

En nuestra problemática, el desarrollo de tecnologías innovadoras que faciliten la clasificación y reutilización de materiales reciclables es clave para fortalecer los procesos de sostenibilidad y minimizar la cantidad de desechos que terminan en los vertederos.

Tecnologías y Bibliotecas Utilizadas

Para la implementación del sistema, se emplean diversas bibliotecas y frameworks especializados en inteligencia artificial, visión por computadora y desarrollo de aplicaciones. A continuación, se describen las tecnologías más relevantes utilizadas en el proyecto.

OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto especializada en procesamiento de imágenes y visión por computadora. Su uso en este proyecto permite la captura, filtrado y preprocesamiento de imágenes para mejorar la precisión del modelo de clasificación. OpenCV facilita operaciones como la detección de bordes, la conversión de imágenes a escala de grises y la reducción de ruido, optimizando así los datos de entrada para la inteligencia artificial (Bradski, 2000).

TensorFlow y Keras

TensorFlow, desarrollado por Google, es un framework de aprendizaje profundo diseñado para entrenar y desplegar modelos de redes neuronales. En este proyecto, TensorFlow se utiliza para la clasificación de documentos e imágenes, aprovechando su capacidad de ejecutar modelos en GPU para mejorar el rendimiento.

Keras, una API de alto nivel integrada en TensorFlow, simplifica la construcción, entrenamiento y ajuste de modelos de machine learning. Su facilidad de uso y compatibilidad con múltiples backends lo convierten en una opción ideal para la implementación de redes neuronales convolucionales (CNNs) enfocadas en el análisis de imágenes (Abadi et al., 2016; Chollet, 2017).

PyTorch y YOLO

PyTorch es un framework de deep learning que ofrece mayor flexibilidad en la optimización de modelos y mejor compatibilidad con GPU. En este proyecto, PyTorch se utiliza en conjunto con YOLO (You Only Look Once), un modelo avanzado de detección de objetos. YOLO permite identificar múltiples elementos en imágenes de manera eficiente y en tiempo real, lo que lo hace ideal para la clasificación y segmentación precisa de materiales en documentos e imágenes (Paszke et al., 2019).

Flask

Flask es un microframework para Python que permite la implementación de aplicaciones web de manera ligera y eficiente. En este proyecto, Flask se utiliza para desarrollar una interfaz web que permite a los usuarios cargar imágenes o documentos y recibir análisis en tiempo real. Su arquitectura modular facilita la integración con modelos de

IA y su compatibilidad con HTML, CSS y JavaScript permite la creación de una experiencia de usuario fluida (Grinberg, 2018).

NumPy y PIL

NumPy es una biblioteca fundamental para la computación científica en Python. Se utiliza en este proyecto para manejar arreglos multidimensionales y realizar cálculos matemáticos optimizados en el procesamiento de imágenes.

PIL (Python Imaging Library), ahora mantenido como Pillow, se emplea para la manipulación avanzada de imágenes, como la conversión de formatos, el ajuste de tamaño y la mejora de la calidad antes del procesamiento con los modelos de IA. Su integración con OpenCV permite un preprocesamiento más eficiente de los datos de entrada.

CustomTkinter

Para la implementación de la versión de escritorio del sistema, se utiliza CustomTkinter, una extensión de Tkinter que permite crear interfaces gráficas modernas con un diseño más estilizado. Este framework facilita la creación de formularios, botones y elementos interactivos dentro de una aplicación de escritorio sin necesidad de conocimientos avanzados en diseño UI/UX.

Visual Studio Code

Visual Studio Code (VS Code) es el entorno de desarrollo integrado (IDE) elegido para el proyecto debido a su compatibilidad con múltiples lenguajes, soporte para depuración avanzada y extensiones que facilitan la programación en Python, Flask y TensorFlow. Su integración con Git y Docker permite un desarrollo colaborativo más eficiente y un despliegue optimizado de la aplicación.

En conjunto, estas tecnologías y bibliotecas permiten la implementación de un sistema robusto, optimizado para el análisis de documentos e imágenes, con interfaces accesibles tanto en web como en escritorio.

Modelos de Procesamiento de Imágenes

Para el análisis y clasificación de imágenes dentro del sistema, se emplean redes neuronales convolucionales (CNNs), ya que han demostrado ser altamente eficientes en tareas de visión por computadora. Estas redes permiten extraer características relevantes de las imágenes, como bordes, texturas y patrones, lo que mejora la precisión del modelo.

El proyecto contempla dos enfoques principales para el procesamiento de imágenes:

1. **Uso de Modelos Preentrenados con YOLO:** se entrenará un modelo propio desde cero utilizando PyTorch y YOLO (You Only Look Once). Este modelo será entrenado con un conjunto de datos específico para la detección y clasificación de residuos, generando un archivo best.pt, que representará la mejor versión del modelo entrenado. A través del uso de YOLO, se busca mejorar la detección en tiempo real y lograr una segmentación precisa de los distintos tipos de desechos, optimizando su desempeño en diversas condiciones de iluminación y perspectiva.
2. **Modelo Entrenado desde Cero con TensorFlow**: Paralelamente, se desarrollará un modelo propio de inteligencia artificial, el cual será entrenado utilizando un dataset construido específicamente para la tarea de clasificación de imágenes. Este modelo será ajustado y optimizado para maximizar su rendimiento en el entorno específico del proyecto.

En el caso del módulo basado en YOLO (You Only Look Once), se utilizarán técnicas de detección de objetos en tiempo real. A diferencia de las CNNs tradicionales utilizadas para clasificación, YOLO permite identificar múltiples objetos dentro de una imagen y etiquetarlos con sus respectivas categorías, lo cual resulta útil para la segmentación de elementos en documentos o imágenes complejas.

Preprocesamiento de Imágenes

Para mejorar el rendimiento del modelo y reducir el ruido en los datos de entrada, se aplican diversas técnicas de preprocesamiento:

- **Normalización**: Se ajustan los valores de los píxeles dentro de un rango entre 0 y 1 para mejorar la estabilidad numérica del modelo y acelerar la convergencia del entrenamiento.
- **Aumento de Datos**: Para evitar el sobreajuste y mejorar la generalización del modelo, se implementan transformaciones en las imágenes, como rotaciones, escalados, volteos y ajustes de brillo. Esto permite que el modelo aprenda de una mayor variedad de ejemplos, aumentando su capacidad de reconocimiento (Shorten & Khoshgoftaar, 2019).
- **Conversión a Escala de Grises o Espacios de Color Alternativos**: En función del modelo seleccionado, las imágenes pueden convertirse a escala de grises para reducir la complejidad computacional o a otros espacios de color como HSV para resaltar ciertas características.
- **Reducción de Ruido**: Se aplican filtros como el desenfoque gaussiano y la eliminación de artefactos para optimizar la calidad de los datos de entrada.

En el contexto del proyecto, estos procesos serán implementados tanto en la versión de escritorio con CustomTkinter como en la versión web desarrollada con Flask, asegurando que el flujo de datos se mantenga consistente en ambas plataformas.

Integración Online y de Escritorio

Implementación Online

Para la versión web, ReciBot emplea Flask como framework backend, permitiendo la comunicación entre el usuario y el modelo de inteligencia artificial. Los usuarios podrán subir imágenes de residuos a través de una interfaz web desarrollada con HTML, CSS y JavaScript, las cuales serán enviadas al servidor para su procesamiento y clasificación.

El servidor Flask gestionará las solicitudes, aplicará preprocesamiento a las imágenes recibidas y ejecutará el modelo de clasificación para determinar el tipo de residuo. Finalmente, el resultado se devolverá a la interfaz, mostrando información detallada sobre el material identificado y las recomendaciones de reciclaje.

Implementación en Escritorio

Para la versión de escritorio, se utilizará CustomTkinter o Tkinter para desarrollar una interfaz gráfica autónoma que permita la clasificación de residuos sin depender de una conexión a internet. Esta aplicación será capaz de capturar imágenes en tiempo real desde una cámara web, procesarlas localmente con el modelo entrenado y mostrar el resultado en la interfaz.

A diferencia de la versión web, la aplicación de escritorio ejecutará el modelo de inteligencia artificial directamente en el equipo del usuario, utilizando TensorFlow o PyTorch para realizar las inferencias de clasificación. Esto permitirá un procesamiento más rápido y

una mayor independencia del usuario, haciéndolo ideal para entornos donde el acceso a internet es limitado o inexistente.

Datasets Utilizados

Para el entrenamiento del modelo de clasificación de residuos, se requiere una base de datos amplia y representativa que permita mejorar la precisión del sistema. Una de las bases de datos más conocidas en este campo es TrashNet, la cual contiene imágenes de desechos clasificados en diferentes categorías, como plástico, papel, vidrio, metal y desechos orgánicos (Yang et al., 2019). Sin embargo, TrashNet presenta ciertas limitaciones que afectan la eficiencia y la capacidad de generalización del modelo.

Entre las principales limitaciones de TrashNet se encuentran la cantidad relativamente reducida de imágenes y la falta de variabilidad en las condiciones de iluminación y entorno. Esto puede provocar que el modelo tenga dificultades para reconocer residuos en escenarios más complejos o en condiciones no presentes en el conjunto de datos original (Suresh et al., 2021). Además, la diversidad de residuos en el mundo real es mucho mayor a la representada en TrashNet, lo que genera la necesidad de incorporar datasets adicionales.

Para mejorar el rendimiento del modelo, se complementará TrashNet con otros conjuntos de datos disponibles en plataformas como Kaggle, donde se pueden encontrar bases de datos más amplias y con una mayor variedad de imágenes capturadas en diferentes condiciones. Algunos de estos datasets incluyen:

- **Waste Classification Dataset**: Contiene imágenes de desechos urbanos con etiquetas detalladas para mejorar la precisión en la identificación de materiales reciclables (García & Liu, 2022).

- **Garbage Classification Dataset**: Incluye más de 20.000 imágenes de residuos organizados en categorías como plástico, papel, vidrio, cartón, aluminio, desechos electrónicos, entre otros (Patel et al., 2023).
- **TACO (Trash Annotations in Context)**: Un dataset de código abierto que presenta imágenes de residuos en ambientes urbanos y naturales, con etiquetas detalladas sobre la composición y estado de los desechos (Romera-Paredes et al., 2022).

Además del uso de datasets públicos, se generará un conjunto de datos propio mediante crowdsourcing, incentivando a usuarios a subir imágenes de residuos en diferentes entornos y condiciones. Esto permitirá mejorar la robustez del modelo y hacerlo más adaptable a contextos diversos. Se implementarán técnicas de data augmentation para incrementar la cantidad y diversidad de imágenes, aplicando transformaciones como rotación, escalado, cambios de brillo y ruido artificial, con el fin de mejorar la capacidad del modelo para reconocer residuos en diversas condiciones.

Metodología para la Selección y Desarrollo de la Solución

La selección de la solución para ReciBot sigue un enfoque basado en la eliminación de alternativas inviables y la optimización de aquellas con mayor potencial. Este proceso considera restricciones técnicas, económicas, normativas y de viabilidad operativa, garantizando que el resultado final sea funcional, eficiente y escalable.

Identificación y Descarte de Soluciones Ilógicas

Se descartan aquellas soluciones que no sean viables desde un punto de vista físico, computacional o económico. En el caso de ReciBot, esto implica:

- Evitar modelos de inteligencia artificial excesivamente complejos que requieran capacidades computacionales inalcanzables con el hardware disponible.
- No depender de sensores o hardware costoso que limite la escalabilidad del sistema.
- Garantizar que la integración entre Flask, OpenCV, TensorFlow, PyTorch, YOLO y CustomTkinter sea factible y funcional sin requerir modificaciones incompatibles.

Comparación con Soluciones Existentes

Se analizan experiencias previas en el uso de redes neuronales convolucionales (CNNs) y técnicas de transfer learning para la clasificación de imágenes, comparando el rendimiento de modelos preentrenados como ResNet50, MobileNetV2, YoloV8 y EfficientNet con el modelo propio de entrenamiento desde cero en PyTorch y YOLO.

- Se estudian soluciones existentes en el ámbito del reconocimiento de residuos, como aplicaciones móviles de clasificación de basura o sistemas de reciclaje automatizados.
- Se revisan publicaciones académicas, experimentos y repositorios de código abierto para identificar buenas prácticas en la implementación de redes neuronales y su optimización en entornos de bajo consumo energético.
- Se consulta con expertos en visión por computadora e inteligencia artificial para evaluar si los enfoques seleccionados son los más adecuados (Modalidad de Github).

Evaluación de Alternativas y Selección de la Mejor Solución

Dado que no todas las soluciones pueden ser evaluadas en detalle debido a limitaciones de tiempo y recursos, se siguen estos pasos para seleccionar la opción óptima:

1. **Pruebas iniciales con modelos preentrenados:** Se analiza la eficiencia de ResNet50, MobileNetV2 y EfficientNet, YoloV8 en la clasificación de residuos, midiendo precisión, velocidad y consumo de recursos.
2. **Entrenamiento desde cero con YOLO y PyTorch:** Se entrena un modelo propio desde cero para generar un archivo best.pt, optimizando la arquitectura y los hiperparámetros según el conjunto de datos disponible.
3. **Comparación de resultados:** Se comparan métricas como precisión, recall, F1-score y velocidad de inferencia, accuracy , identificando cuál alternativa ofrece un balance ideal entre rendimiento y costo computacional.
4. **Optimización del modelo seleccionado:** Una vez identificada la mejor solución, se ajustan parámetros, se realiza fine-tuning y se implementan técnicas como aumento de datos y normalización de imágenes para mejorar la generalización del modelo.

En este proceso, se prioriza una solución que:

- Sea precisa en la clasificación de residuos.
- Tenga tiempos de respuesta rápidos en la versión web y de escritorio.
- Funcione en hardware accesible sin necesidad de inversiones excesivas.
- Permita futuras mejoras e integración con otros sistemas.

Finalmente, la solución seleccionada se somete a pruebas de validación para garantizar su desempeño en entornos reales, asegurando que pueda ser utilizada tanto en la versión online con Flask como en la versión standalone con CustomTkinter.

Solución de Ingeniería

La solución propuesta para ReciBot consiste en el desarrollo de un sistema de clasificación de residuos basado en inteligencia artificial y visión por computadora, implementado en dos versiones: una plataforma web accesible desde navegadores y una aplicación de escritorio que funcione de manera local sin conexión a internet.

Arquitectura del Sistema

El sistema de ReciBot se compone de varios módulos interconectados, cada uno con una función específica:

Módulo de Adquisición de Imágenes

- En la versión web, los usuarios pueden subir imágenes a través de una interfaz desarrollada con HTML, CSS y JavaScript, con procesamiento en Flask.

- En la versión escritorio, se permite capturar imágenes en tiempo real mediante una webcam, utilizando OpenCV y CustomTkinter.

Módulo de Preprocesamiento de Imágenes

- Se emplea OpenCV y PIL (Pillow) para mejorar la calidad de las imágenes antes de su análisis.

- Se aplican técnicas de normalización, reescalado, reducción de ruido y aumento de datos para mejorar la robustez del modelo.

Módulo de Inteligencia Artificial

- Se entrenará un modelo desde cero con PyTorch, Tensorflow y YOLO, generando un modelo optimizado (best.pt) para la clasificación de residuos.

- El modelo de IA clasificará los residuos en categorías como orgánicos, plásticos, metales, vidrios y papel/cartón.

Módulo de Integración Backend

- Para la versión web, se usará Flask como servidor backend, procesando imágenes y enviando los resultados al frontend en tiempo real.

- Para la versión escritorio, el modelo se ejecutará de manera local sin requerir conexión a internet, permitiendo una mayor independencia y rapidez en la clasificación.

Módulo de Interfaz de Usuario

- La versión web contará con una interfaz interactiva desarrollada con HTML, CSS y JavaScript, accesible desde cualquier navegador.

- La versión de escritorio usará CustomTkinter, ofreciendo una interfaz intuitiva y fácil de usar, con funcionalidades de captura de imagen y clasificación instantánea.

Diseño de Ingeniería de los Aplicativos Desarrollados

El desarrollo de la solución de ingeniería contempló la implementación de dos aplicativos complementarios: una versión de escritorio y una versión web. Ambos sistemas fueron diseñados con el propósito de responder a la problemática central de la recolección y clasificación de residuos sólidos en Colombia, aprovechando técnicas de inteligencia artificial y visión por computadora. La aplicación de escritorio se orienta a ofrecer un entorno autónomo que funcione sin conexión a internet, ideal para espacios donde la conectividad es limitada o inexistente, mientras que la aplicación web permite un acceso más amplio y escalable, brindando a los usuarios la posibilidad de interactuar con el sistema desde cualquier dispositivo con conexión a la red. Estas dos soluciones, concebidas de manera paralela, constituyen el núcleo tecnológico del proyecto ReciBot, garantizando accesibilidad, eficiencia y versatilidad en el proceso de separación de desechos, y contribuyendo a la creación de una cultura de reciclaje sustentada en herramientas digitales modernas.

Diseño de Ingeniería – Aplicativo Web

El aplicativo web constituye la primera solución de ingeniería desarrollada en el marco del proyecto ReciBot. Su propósito es ofrecer a cualquier usuario con acceso a internet una plataforma intuitiva para la clasificación automática de residuos mediante técnicas de visión por computadora e inteligencia artificial. Para ello, el sistema se desplegó inicialmente a través de un servidor de Ngrok, lo cual permitió exponer localmente el prototipo de Flask a una dirección pública, y posteriormente se planteó la transición hacia un dominio propio para mejorar la accesibilidad y escalabilidad del sistema.

La arquitectura general del aplicativo se organiza en un conjunto de componentes interdependientes que cumplen funciones específicas. En primer lugar, la carpeta denominada *trashnet* alberga el conjunto de datos empleado en la etapa de entrenamiento del modelo. Este dataset contiene imágenes categorizadas en distintas clases de residuos —papel y cartón, vidrio, plástico, metal y orgánico—, sirviendo como base para el aprendizaje supervisado del sistema. Por otra parte, la carpeta *templates* incluye las estructuras HTML que definen la interfaz de usuario. Dentro de este directorio se encuentra el archivo index.html, el cual constituye el punto de entrada de la aplicación. Este archivo fue diseñado con un enfoque responsivo, integrando una hoja de estilos denominada *styles.css*, que no solo otorga coherencia visual, sino que también garantiza la adaptabilidad de la página a distintos dispositivos, desde equipos de escritorio hasta teléfonos móviles.

En términos de funcionamiento, el aplicativo web se basa en el framework Flask, que opera como servidor backend encargado de gestionar las peticiones del cliente. El archivo app.py implementa los controladores que reciben las imágenes cargadas por el usuario a través de la interfaz web, ejecutan el preprocesamiento y envían dichos datos al modelo previamente entrenado. El preprocesamiento contempla operaciones como el redimensionamiento de las imágenes a 128x128 píxeles, la normalización de los valores de los píxeles al rango [0,1], y la conversión a arreglos de NumPy para ser interpretados por el modelo de redes neuronales convolucionales (CNN). Una vez realizada la predicción, el sistema retorna la clase correspondiente al residuo y, mediante la lógica de presentación en index.html, se generan recomendaciones específicas sobre la disposición del material detectado.

El entrenamiento del modelo se desarrolló en el archivo train_model.py, donde se construyó una arquitectura de CNN con varias capas convolucionales y de agrupamiento

(*max pooling*), seguida de capas densas para la clasificación final. Se aplicaron técnicas de *data augmentation*, tales como rotaciones, desplazamientos y volteos, con el fin de aumentar la variabilidad del dataset y mejorar la capacidad de generalización del modelo. El entrenamiento se ejecutó con un número elevado de épocas (100) y un tamaño de lote de 32, alcanzando precisiones superiores al 85% en los conjuntos de prueba. El modelo final se exportó como modelo_basura.h5 y se integró en el backend Flask para la ejecución de inferencias en tiempo real.

El diseño del aplicativo web se fundamenta en un principio de modularidad. La capa de presentación incluye secciones informativas como “Nosotros”, “Qué hacemos” y “Recomendaciones”, que buscan sensibilizar al usuario sobre la importancia del reciclaje. La capa de lógica implementa la comunicación entre el frontend y el backend mediante peticiones HTTP con *fetch*, mientras que la capa de datos se centra en el modelo entrenado y las librerías de apoyo como TensorFlow, NumPy, OpenCV y PIL. Esta separación clara de responsabilidades no solo garantiza un mantenimiento más sencillo, sino que también facilita la futura expansión del sistema hacia nuevas funcionalidades, como el soporte de más categorías de residuos o la integración con servicios de almacenamiento en la nube.

Finalmente, el aplicativo web responde al objetivo de crear una plataforma accesible y eficiente para la clasificación de residuos. La interfaz responsiva asegura que el sistema pueda ser utilizado tanto en contextos urbanos como en zonas rurales con distintos dispositivos, mientras que el uso de Ngrok en la etapa inicial permitió validar el sistema en condiciones reales de uso. En consecuencia, el diseño del aplicativo web no solo aporta una solución tecnológica al problema de la gestión de residuos, sino que también se convierte en un instrumento pedagógico para promover la cultura del reciclaje en la sociedad.

Explicación detallada del código del aplicativo web

templates/index.html — estructura, UI y flujo de interacción

El archivo index.html define la interfaz de usuario y el flujo de interacción completo desde el cargue de la imagen hasta la visualización del resultado y la recomendación de disposición del residuo. La cabecera importa la hoja de estilos desde Flask con url_for('static', filename='styles.css'), lo que permite servir los recursos estáticos de forma correcta en cualquier despliegue (local, ngrok o dominio propio). La barra de navegación y el botón "hamburguesa" (menu-toggle) controlan la visibilidad del menú en pantallas pequeñas, habilitando un comportamiento responsive mediante la inserción de una clase CSS (.show) cuando el usuario toca el ícono; esto evita dependencias externas y mantiene el DOM mínimo.

La sección "Hero" ofrece una llamada a la acción que ancla al formulario de carga. En "Upload" se definen dos entradas de archivo: una para seleccionar desde el sistema de archivos (imageInput) y otra con capture="environment" para abrir la cámara trasera en móviles (cameraInput). Ambas comparten la misma rutina de progreso visual que incrementa una barra con setInterval, brindando retroalimentación inmediata de la subida y mejorando la percepción de desempeño en conexiones lentas. El botón "Analizar Imagen" dispara sendImage(), que implementa: (i) selección de la fuente (archivo vs. cámara), (ii) validación de tipo MIME (image/*), (iii) bloqueo temporal del botón y mensaje de "Procesando…", (iv) construcción de FormData y envío por fetch hacia el endpoint /predict del backend.

El manejo de respuestas considera dos caminos: si el backend devuelve {"error": ...}, el texto de estado adopta una clase error y se limpia la recomendación; de lo contrario, se extrae material y se presenta como "Material detectado: …". Para convertir la clase a una

recomendación práctica, la función getRecommendation(material) implementa un mapa de cinco categorías (“metal”, “organico”, “papel_y_carton”, “plastico”, “vidrio”) a mensajes pedagógicos (contenedor sugerido, limpieza y separación de tapas, evitar contaminación con orgánicos, etc.). Esta lógica de negocio en el frontend desacopla la inferencia del modelo (backend) de la educación al usuario (frontend), facilitando futuras actualizaciones de textos o políticas locales sin reentrenar la IA.

Más abajo, las secciones “Nosotros”, “Qué Hacemos” y “Recomendaciones por Tipo de Residuos” cumplen un rol informativo y de sensibilización. Muestran imágenes alojadas en /static/… y refuerzan la codificación por color de contenedores (azul, verde, naranja, amarillo, negro), alineando la interfaz con prácticas de separación ampliamente difundidas. Finalmente, el bloque de “Contacto” incorpora un formulario (no funcional por ahora) y un iframe de Google Maps con la ubicación institucional, cerrando con un footer con redes sociales. Todo el documento está preparado con meta viewport y recursos semánticos básicos para accesibilidad (por ejemplo, aria-live="polite" en los campos de resultado) que permiten a lectores de pantalla anunciar cambios del contenido sin romper la navegación.

app.py — backend Flask, preprocesamiento e inferencia con TensorFlow/Keras

El backend se implementa con Flask y sirve dos rutas: / y /predict. Al iniciar, el módulo intenta cargar el modelo previamente entrenado modelo_basura.h5 mediante keras.models.load_model; en caso de falla, conserva model=None y reporta el error por consola. Esta decisión permite que la aplicación siga levantando para depuración de UI y flujo, devolviendo un mensaje de error controlado en /predict si el modelo no está disponible.

- **Etiquetas y consistencia semántica.** La constante LABELS define el orden de clases esperado por el modelo en tiempo de inferencia: ["metal", "organico",

"papel_y_carton", "plastico", "vidrio"]. Es esencial mantenerla consistente con el orden usado durante el entrenamiento; de lo contrario, argmax asignaría etiquetas incorrectas a las probabilidades del softmax.

- **Ruta / y renderizado del frontend.** home() invoca render_template("index.html"), sirviendo la UI y habilitando el enrutamiento de activos estáticos definidos en static/ por configuración estándar de Flask. Esto integra el frontend (Jinja/HTML) con el backend (Python) en un único servicio, simplificando el despliegue con ngrok o WSGI en producción.
- **Preprocesamiento determinista.** preprocess_image(image) convierte la imagen a tamaño fijo (128, 128), normaliza a [0,1] y expande el eje de *batch* a (1, 128, 128, 3). Esta función debe replicar exactamente el pipeline usado en entrenamiento (resizing, normalización, orden de canales), garantizando que la distribución de datos en inferencia coincida con la del entrenamiento (principio de *train–serve skew* mínimo).
- **Ruta /predict (POST).**

    1. Valida la presencia de “file” en request.files.
    2. Lee el binario, abre con PIL en RGB (evita problemas con PNG con alfa), y llama a preprocess_image.
    3. Verifica que model exista; si no, retorna {"error": "El modelo no está cargado correctamente"} (código 500).
    4. Ejecuta model.predict(processed_image); luego calcula np.argmax en la dimensión de clases y mapea al string con LABELS[class_index].
    5. Devuelve JSON con {"material": ...}.

El manejo de excepciones captura errores de E/S, preprocesamiento e inferencia y responde con {"error": "..."} + 500, lo que el frontend interpreta para mostrar mensajes al usuario sin romper la sesión.

En suma, app.py implementa un patrón limpio de API: validación → preprocesamiento → predicción → mapeo → respuesta JSON. Esto facilita pruebas con curl/Postman y permite escalar a otros clientes (por ejemplo, una app móvil nativa) reutilizando el mismo endpoint.

train_model.py — preparación del dataset, data augmentation, arquitectura CNN y persistencia del modelo

El script de entrenamiento define un pipeline reproducible sobre TrashNet (y/o conjuntos equivalentes organizados por carpetas). Establece parámetros globales: tamaño de imagen 128, batch size 32 y EPOCHS=100, lo que busca un equilibrio entre capacidad de aprendizaje y costo computacional en GPU/CPU moderadas.

- **Carga de datos y etiquetas.** Recorre las carpetas por clase (metal, organico, papel_y_carton, plastico, vidrio), lee cada imagen con OpenCV, convierte de BGR→RGB, hace resize a (128,128) y acumula en listas que luego normaliza a [0,1]. Las etiquetas se codifican como enteros (0..len(CATEGORIES)-1) y se separan en entrenamiento y prueba con train_test_split (20% hold-out), fijando random_state=42 para reproducibilidad estadística.

- **Aumentación de datos.** Con ImageDataGenerator aplica rotación, width/height shifts y horizontal flip. Este esquema introduce invariancias geométricas y simula variaciones de captura (ángulos, encuadre, leves traslaciones), mitigando el

sobreajuste en datasets limitados y mejorando la generalización del modelo en condiciones reales de iluminación y fondo.

Arquitectura de la CNN.

1. **Bloques Conv–ReLU–MaxPool** con 32, 64 y 128 filtros capturan patrones locales (bordes, texturas, composiciones progresivamente más abstractas).
2. **Flatten** convierte los mapas de activación en un vector.
3. **Dense(128, ReLU)** agrega capacidad de combinación no lineal.
4. **Dropout(0.5)** regulariza reduciendo co-adaptaciones.
5. **Dense(5, Softmax)** produce probabilidades por clase (las 5 categorías del problema).

Se compila con optimizer="adam" y loss="sparse_categorical_crossentropy" (apropiada para etiquetas enteras), reportando accuracy durante el *fit*. El entrenamiento se ejecuta con el generador aumentado en *training* y validación directa sobre (X_test, y_test).

- **Persistencia, evaluación y trazas.** Tras el *fit*, se guarda el artefacto modelo_basura.h5 (Keras HDF5), lo que permite **cargarlo en producción** con una sola línea en app.py. Luego se evalúa en *test* e imprime la precisión. Finalmente, grafica las curvas de *accuracy* y *loss* de entrenamiento/validación para **diagnóstico visual** de convergencia y sobreajuste (útil para tomar decisiones de *early stopping*, *data cleaning* o ajustes de arquitectura).

- **Consistencia crítica entre entrenamiento e inferencia.** El resize a 128×128, la normalización a [0,1], el orden de clases y el canal de color RGB deben coincidir exactamente entre train_model.py y app.py. Cualquier divergencia (por ejemplo, cambiar el orden de CATEGORIES en entrenamiento sin actualizar LABELS en producción) degradará la precisión por desalineación semántica.

Consideraciones de diseño y mejoras sugeridas

1. **Robustez de entrada.** En index.html, ya se valida el tipo MIME; añadir límites de tamaño y una vista previa reduciría fallos por archivos corruptos o excesivamente grandes.

2. **Escalado del backend.** app.py corre en modo debug=True para desarrollo; en producción conviene WSGI (uWSGI/Gunicorn) y desactivar debug, además de **limitar concurrencia** y emplear colas si se habilitan modelos más pesados.

3. **Trazabilidad del modelo.** Versionar modelo_basura.h5 (por ejemplo, modelo_basura_v1_0.h5) y registrar *metrics* clave (precisión por clase, matriz de confusión) mejorará la gobernanza del ciclo ML.

4. **Seguridad.** Validar que el endpoint /predict solo acepte imágenes (cabeceras y *magic numbers*), sanitizar nombres y evitar escribir archivos en disco si no es necesario.

5. **Métricas en producción.** Registrar tiempos de preprocesamiento y de inferencia para detectar regresiones tras cambios de arquitectura o de infraestructura.

Diseño de Ingeniería – Aplicativo de Escritorio

La segunda solución implementada dentro del proyecto ReciBot corresponde al aplicativo de escritorio, el cual se concibió con el objetivo de ofrecer una herramienta autónoma, funcional aún en contextos sin conexión a internet, y que a la vez permitiera evaluar un enfoque alternativo de entrenamiento de modelos de clasificación de residuos. A diferencia de la versión web, donde se construyó una red neuronal convolucional desde cero con TensorFlow y Keras, el aplicativo de escritorio se sustentó en el aprovechamiento de un modelo YOLOv8 preentrenado. Esta elección respondió al propósito de contrastar dos estrategias de versionamiento de entrenamiento: partir de un modelo base optimizado frente a diseñar la arquitectura desde cero. De esta manera, el proyecto no solo entrega una aplicación funcional, sino que también constituye un estudio comparativo de rendimiento entre metodologías de entrenamiento de modelos de visión artificial.

La estructura de carpetas del sistema refleja la organización modular del proyecto. La carpeta runs almacena los resultados de las sesiones de entrenamiento realizadas con YOLO, incluyendo el modelo optimizado identificado como best.pt, el cual corresponde a la versión del modelo que alcanzó la mayor precisión sobre el conjunto de validación. Por su parte, la carpeta setUp contiene los lienzos gráficos utilizados para la construcción de la interfaz de usuario, organizados en imágenes representativas de cada categoría de residuo (plástico, metal, orgánico, papel y cartón, y vidrio). Estos recursos visuales cumplen un doble rol: por un lado, aportan coherencia gráfica y usabilidad al aplicativo; y por otro, refuerzan el carácter pedagógico de la solución, orientando al usuario en la correcta clasificación de residuos mediante íconos y mensajes descriptivos.

El flujo de trabajo técnico del aplicativo se centra en dos componentes principales: el entrenamiento del modelo y la ejecución de la aplicación con interfaz gráfica. El script Train.py configura el entrenamiento de YOLOv8 en la modalidad de clasificación,

especificando el dataset mediante el archivo dataset.yaml, el número de épocas, el tamaño de lote y la resolución de las imágenes de entrada. A partir de un modelo preentrenado (yolov8n-cls.pt), se realiza un ajuste fino (fine-tuning) con el conjunto de imágenes locales, lo que permite que la red adapte sus parámetros a la tarea específica de clasificación de residuos. Esta estrategia de transferencia de aprendizaje es clave para reducir tiempos de entrenamiento, optimizar el uso de recursos computacionales y, al mismo tiempo, lograr niveles elevados de precisión en entornos reales.

La lógica de funcionamiento del aplicativo se desarrolla en app.py, archivo encargado de la implementación de la interfaz con el usuario a través de CustomTkinter. Este módulo ofrece un entorno visual moderno que mejora la experiencia en comparación con Tkinter estándar, integrando botones, áreas de carga de imágenes y secciones de salida de resultados. El usuario puede capturar imágenes en tiempo real mediante la cámara de su dispositivo o cargar archivos locales; estas imágenes son procesadas por el modelo YOLO entrenado, y el resultado es presentado junto con la recomendación sobre la disposición adecuada del residuo. La integración de los elementos gráficos alojados en la carpeta setUp asegura que la clasificación no se limite a un resultado textual, sino que se acompañe de íconos y mensajes visuales que refuercen la comprensión.

Desde el punto de vista metodológico, el diseño del aplicativo de escritorio responde a tres criterios principales:

1. **Autonomía:** el modelo best.pt se ejecuta de manera local, sin necesidad de acceder a un servidor o conexión a internet, lo que amplía la aplicabilidad del sistema en regiones con baja conectividad.

2. **Eficiencia:** la utilización de un modelo preentrenado YOLOv8 reduce significativamente la complejidad de diseño de la red y permite obtener resultados con menos recursos de hardware.

3. **Experiencia de usuario:** la interfaz basada en CustomTkinter, junto con el uso de elementos visuales, hace que la aplicación sea intuitiva y adecuada para públicos diversos, incluidos estudiantes y comunidades que recién se inician en la práctica del reciclaje.

Finalmente, el aplicativo de escritorio no solo cumple con el objetivo de entregar una herramienta funcional, sino que también aporta un valor agregado al proyecto al permitir la comparación experimental entre enfoques de entrenamiento distintos: un modelo creado desde cero (TensorFlow) y un modelo preentrenado con fine-tuning (YOLOv8). Este contraste permite extraer conclusiones sobre las ventajas y limitaciones de cada estrategia en el contexto de la clasificación automática de residuos, aportando evidencia empírica a la toma de decisiones en proyectos futuros de visión artificial aplicada a problemas ambientales.

Explicación detallada de los códigos del aplicativo de escritorio

Entrenamiento con YOLOv8 — Train.py

El script Train.py implementa un flujo de fine-tuning sobre YOLOv8 en modo clasificación (no detección). Parte de un modelo base preentrenado (yolov8n-cls.pt), apropiado para hardware modesto, y lo ajusta a tu dominio de residuos. Esta elección busca aprovechar representaciones visuales ya aprendidas (transfer learning) y, con pocas épocas, especializar el modelo en tus cinco clases objetivo. En concreto, el código: (i) carga el checkpoint preentrenado, (ii) define la fuente de datos, (iii) establece hiperparámetros de

entrenamiento (épocas, tamaño de imagen, batch, workers), y (iv) delega a Ultralytics la orquestación de dataloaders, augmentations y optimización. Al finalizar, los artefactos quedan versionados en runs/train/exp*, con pesos best.pt seleccionados por métrica de validación.

Bajo el capó, Ultralytics aplica un conjunto sensato de transformaciones para clasificación (por ejemplo, flip, resize y normalización), crea DataLoaders para train/val, y emplea una función de pérdida de clasificación multiclase (softmax) con un optimizador configurado por defecto. Tus decisiones clave aquí son: usar 224×224 como imgsz para reducir costo computacional, batch=16 como compromiso entre estabilidad de gradiente y memoria, y epochs=50 para permitir convergencia sin sobreentrenar en dominios pequeños.

En términos de reproducibilidad y gobernanza, este flujo favorece: (a) rastrear experimentos por carpeta runs/..., (b) promover comparaciones objetivas entre configuraciones (por ejemplo, cambiar a yolov8s-cls.pt en GPUs más capaces), y (c) conservar el mejor checkpoint según validación para inferencia estable.

Especificación del dataset — dataset.yaml

El archivo dataset.yaml define la fuente canónica de datos para el entrenador. Estructura dos rutas absolutas —train y val— que apuntan a carpetas donde cada subcarpeta corresponde a una clase (metal, orgánico, papel_y_cartón, plástico, vidrio). Declara además el número de clases (nc: 5) y los nombres en el orden exacto que el entrenador asignará a los índices de clase. Esta correspondencia es crítica: el índice 0 del vector de probabilidades

representará “metal”, el 1 “orgánico”, etc., y debe coincidir con los textos que la aplicación mostrará en tiempo real.

Esa definición garantiza que el *pipeline* de Ultralytics construya correctamente los dataloaders, mantenga la separación train/val y produzca métricas por clase alineadas a tus etiquetas. Si en el futuro agregas otra categoría (por ejemplo, “tetrapak”), deberás crear la subcarpeta correspondiente en train/ y val/, añadirla a names y actualizar nc. Mantener la consistencia semántica entre names y las rutas reales evita desalineaciones en evaluación y en la GUI.

Aplicación de escritorio y predicción en vivo — app.py

Este módulo arma la interfaz de escritorio con Tkinter/PIL e integra la inferencia del modelo best.pt de YOLOv8 clasificación. La idea central es ofrecer clasificación en tiempo real desde la cámara y enriquecer la experiencia con material pedagógico visual (íconos y carteles por clase).

- **Dependencias y utilitarios.**
  Se importan Tkinter, PIL (para convertir arreglos a objetos PhotoImage), imutils y OpenCV para manejo de video, así como ultralytics.YOLO para cargar el modelo. Las funciones auxiliares clean_lbl() y show_images() limpian el estado de la interfaz o muestran las láminas informativas asociadas a cada clase (imagen del contenedor y tarjeta de recomendaciones) con una conversión de BGR→RGB antes de renderizar en el Label.
- **Bucle de captura y predicción (scanning).**
  La función comprueba que la cámara esté abierta, toma un frame, lo transforma a RGB y redimensiona para la UI. Luego invoca model.predict(source=frame,

save=False) y obtiene result.probs (vector de probabilidades). Con argmax recupera el índice de la clase ganadora, calcula la confianza, compone la etiqueta (“clase 95%”) y:

1. actualiza el Label textual en la GUI,
2. sobreimpone la leyenda al propio video, y
3. llama a show_images(...) para desplegar las láminas asociadas a esa clase.

El refresco continuo se logra con lblVideo.after(10, scanning), que reprograma la función para mantener un streaming reactivo sin bloquear el hilo de UI.

- **Inicialización de la ventana (ventana_principal).**

  Crea la ventana principal, establece tamaño, carga un fondo Canva para la estética, y posiciona los Label que contienen: (i) el video, (ii) la imagen de contenedor por clase, (iii) la tarjeta de texto por clase y (iv) el Label con la predicción. A continuación carga el modelo best.pt entrenado y define el arreglo clsName con los nombres de clases (el orden debe coincidir con dataset.yaml y con el entrenamiento). Luego carga, desde la carpeta setUp, los recursos gráficos de cada clase (por ejemplo, metal.png, metaltxt.png) que se usarán para retroalimentación educativa tras cada predicción. Por último abre la cámara (índice 1 en tu equipo) y lanza el bucle de escaneo.

- **Relación con el entrenamiento.**

  El camino de pesos (.../runs/classify/.../best.pt) proviene del flujo de Train.py. Es clave que el **orden de clsName** refleje exactamente el de names en dataset.yaml para que el índice argmax muestre el rótulo correcto; una desalineación aquí produce errores semánticos (por ejemplo, etiquetar “vidrio” cuando el modelo predijo “plástico”). Tu app.py fija explícitamente ese orden.

Coherencia "train–serve" y buenas prácticas

1. **Consistencia de clases y rutas.** Mantén sincronizados names en dataset.yaml, el orden de subcarpetas en train/val, y el arreglo clsName en app.py. Así la probabilidad probs[i] siempre mapea al texto correcto.

2. **Artefactos versionados.** Conserva los runs/ por experimento; anota hiperparámetros (épocas, imgsz, batch) usados en cada best.pt para trazabilidad y comparativas con el modelo TensorFlow del aplicativo web.

3. **Cámara y rendimiento.** Si el índice 1 falla en otros equipos, auto-detecta cámaras disponibles antes de abrir VideoCapture. Añadir limitación de FPS o *resize* previo al predict ayuda a sostener latencia baja en hardware sin GPU.

4. **Robustez de UI.** Maneja excepciones al cargar imágenes de setUp para no romper la sesión (ya lo haces: suples con arrays vacíos). Puedes agregar mensajes de alerta visibles al usuario cuando falten recursos.

5. **Comparativa con la versión web.** Este escritorio demuestra el otro pilar del proyecto: YOLOv8 preentrenado vs. CNN desde cero. La coexistencia valida tu objetivo de investigación y permite discutir precisión, tiempo de entrenamiento y experiencia de uso en ambientes con y sin red.

Cómo "lee" el sistema al usuario

El flujo completo es: abrir la app → cargar best.pt → iniciar cámara → capturar un frame → ejecutar model.predict() → obtener result.probs → argmax → mostrar etiqueta y confianza → actualizar láminas pedagógicas y superponer la clase sobre el video → reprogramar scanning para el siguiente frame. Este ciclo de 10 ms aproximadamente (según tu after(10, ...)) viabiliza clasificación fluida en vivo, entregando no solo un veredicto, sino también educación contextual mediante las imágenes y textos de setUp.

Características y Beneficios de la Solución

El desarrollo de ReciBot como proyecto de grado integra una serie de características técnicas y metodológicas que lo convierten en una herramienta innovadora para la clasificación automática de residuos. Estas características no solo responden a los objetivos planteados, sino que también generan beneficios tangibles para los usuarios, las instituciones educativas y la sociedad en general. A continuación, se detallan los aspectos más relevantes:

Precisión y optimización.

La solución incorpora arquitecturas de redes neuronales convolucionales (CNNs) y estrategias de transfer learning, lo que permite aprovechar el conocimiento ya adquirido por modelos preentrenados y ajustarlo a la tarea específica de clasificación de residuos. Este enfoque reduce significativamente los tiempos de entrenamiento y mejora la capacidad de generalización del sistema frente a condiciones reales, tales como variaciones de iluminación, ángulos de captura o fondos complejos. La precisión alcanzada en las pruebas, respaldada por métricas como la exactitud y la pérdida de validación, asegura que el sistema proporcione

resultados confiables y rápidos, lo que constituye un beneficio clave en entornos educativos, comunitarios y de gestión de residuos.

Accesibilidad multiplataforma.

Uno de los principales beneficios de ReciBot es su diseño en dos modalidades complementarias: una aplicación web y una aplicación de escritorio. La versión web permite el acceso desde cualquier dispositivo con conexión a internet, garantizando portabilidad y alcance global. En contraste, la versión de escritorio funciona de manera autónoma sin requerir conexión, lo que amplía su utilidad en contextos con limitaciones de conectividad. Este enfoque híbrido asegura que la solución pueda implementarse en entornos urbanos y rurales, adaptándose a diferentes necesidades y garantizando una mayor inclusión tecnológica.

Eficiencia computacional.

El diseño del sistema contempla la optimización de los modelos de IA para que puedan ejecutarse en equipos con recursos de hardware limitados. En el caso de la versión de escritorio, el empleo de un modelo YOLOv8 preentrenado y ajustado mediante fine-tuning facilita la clasificación en tiempo real sin requerir estaciones de trabajo de alto rendimiento. De igual manera, la versión web utiliza modelos entrenados con TensorFlow y optimizados para reducir el consumo de memoria y procesamiento, lo que favorece un tiempo de respuesta rápido incluso en equipos de gama media. Esta eficiencia permite que el sistema se convierta en una herramienta viable y accesible para instituciones educativas, comunidades y usuarios individuales.

Escalabilidad y evolución futura.

ReciBot se diseñó con un enfoque de escalabilidad, lo que significa que puede evolucionar fácilmente en función de nuevas necesidades o avances tecnológicos. Entre las posibles líneas de crecimiento se incluyen: la integración de nuevas categorías de residuos en la base de datos, la incorporación de modelos de IA más robustos y precisos, la adaptación a distintos idiomas y contextos culturales, y la conexión con sistemas de gestión ambiental o plataformas gubernamentales. De esta forma, el proyecto no se limita a ser un prototipo académico, sino que constituye una base sólida para desarrollos futuros que impacten positivamente en la sostenibilidad y el cuidado del medio ambiente.

Análisis de Costos

Para la implementación de ReciBot, se ha realizado un análisis integral de costos con el objetivo de evaluar la viabilidad económica del proyecto, teniendo en cuenta los recursos necesarios tanto para el desarrollo del sistema como para su puesta en marcha y sostenibilidad. Este análisis contempla tres grupos fundamentales: costos de operación, costos de inversión y capital de trabajo.

Archivo de Costos: Link Archivo de Costos

Costos Operacionales

Los costos operacionales del proyecto ReciBot se entienden como el conjunto de gastos asociados al desarrollo, despliegue y mantenimiento de las dos versiones del sistema

(aplicativo web y aplicativo de escritorio). Estos costos permiten dimensionar la viabilidad económica del proyecto, identificar los recursos requeridos en cada fase y garantizar la sostenibilidad del sistema a largo plazo. A continuación, se presentan los diferentes tipos de costos que intervienen en la operación de la solución.

Costos Directos (Variables)

Son aquellos que dependen del nivel de uso del servicio o de la producción vinculada al sistema. Están directamente relacionados con la puesta en marcha de actividades específicas y pueden aumentar o disminuir en función de la demanda. Entre los principales se encuentran:

Insumos de diseño físico, como la fabricación de canecas clasificadoras en concordancia con los cinco criterios de separación de residuos (orgánico, plástico, vidrio, metal y papel/cartón). Estas estructuras complementan la solución digital con un componente tangible que promueve el reciclaje en espacios físicos.

Consumo de energía eléctrica derivado de entrenamientos prolongados del modelo en GPU, los cuales requieren gran capacidad de cómputo para alcanzar niveles de precisión adecuados.

Servicios en la nube (opcional en la versión web) para alojamiento de la aplicación y el modelo, tales como AWS, Google Cloud o Heroku, cuyo costo se incrementa en función del tráfico y uso.

Insumos de hardware complementarios, como cámaras, sensores o periféricos adicionales que pueden emplearse en pruebas locales o en la expansión de funcionalidades futuras.

Costos Fijos

Son gastos constantes que se mantienen en el tiempo sin importar el número de usuarios activos o la cantidad de clasificaciones realizadas. Garantizan la operatividad básica de la solución y comprenden:

- Servicios públicos del espacio físico, como energía eléctrica e internet necesarios para el mantenimiento del equipo de desarrollo.
- Pago de dominio web y hosting básico, necesarios para la operación continua de la versión en línea. En caso de optar por servicios avanzados de AWS u otro proveedor, se suma el arriendo mensual de servidores.
- Licencias de software o bibliotecas de uso comercial, en caso de requerirse componentes propietarios que no estén disponibles en formato libre.

Gastos Generales

Son costos indirectos relacionados con la administración, gestión y promoción del proyecto. Si bien no están vinculados directamente al desarrollo técnico, resultan esenciales para la continuidad de la iniciativa. Entre ellos se incluyen:

- Gestión del proyecto, que contempla la remuneración a coordinadores y líderes de equipo encargados de la planeación y seguimiento.
- Publicidad y promoción, con el fin de impulsar la adopción de ReciBot en comunidades, instituciones educativas y entidades ambientales.
- Marketing digital y diseño gráfico, orientados a la creación de campañas, contenido visual y material de sensibilización.
- Mantenimiento continuo del sitio web y de la aplicación de escritorio, incluyendo la actualización de dependencias, corrección de errores y mejoras evolutivas.

Costos de Inversión

Los costos de inversión corresponden a los gastos iniciales necesarios para el desarrollo, configuración e implementación del sistema. Representan el capital requerido para poner en marcha la solución y garantizar la construcción de sus principales componentes.

Costos Directos de Inversión

- Adquisición de equipos: computadoras de mediano rendimiento con GPU para el entrenamiento del modelo, cámaras web y dispositivos móviles para pruebas de captura de imágenes.
- Infraestructura de despliegue local y servidores en caso de que se opte por utilizar instancias en la nube para la versión web.
- Diseño y construcción de interfaces (web y escritorio), incluyendo frameworks, librerías y entornos de desarrollo como Microsoft Visual Studio Code con Python como lenguaje principal.
- Entrenamiento intensivo de modelos con YOLO y PyTorch, requiriendo recursos computacionales dedicados y, en algunos casos, servicios en la nube.
- Adquisición y complementación de datasets externos, en caso de necesitarse ampliar la base de datos inicial para cubrir un mayor espectro de residuos.
- Insumos físicos de diseño, como la construcción de canecas diferenciadas y demás infraestructura necesaria para materializar la propuesta.
- Compra de hardware adicional (cámaras y sensores) para pruebas locales y validación de prototipos.

Costos Indirectos de Inversión

- Trámites legales y administrativos, como registro de marca, derechos de autor o patentes sobre el código o la solución.

- Aseguramiento de la calidad (QA), incluyendo pruebas de usabilidad, auditorías de seguridad informática y cumplimiento normativo.
- Estudios de impacto ambiental y pruebas piloto, realizados en escenarios reales para validar la efectividad del sistema.
- Depreciación de equipos de cómputo, donde servidores, PCs con GPU y cámaras se amortizan a lo largo de su vida útil.
- Mantenimiento de hardware y actualizaciones de software, así como servicios de soporte técnico (interno o tercerizado) para garantizar la seguridad y disponibilidad del sistema.

Costos Fijos de Operación

Se mantienen constantes a lo largo del tiempo, independientemente del volumen de usuarios. En este apartado se contemplan:

- Servicios públicos del espacio físico (electricidad, internet) requeridos para el funcionamiento continuo del sistema.
- Arriendo de servidores o hosting en la nube, en caso de mantener la versión web de manera activa.

Capital de Trabajo Inicial

Se refiere a los recursos financieros requeridos para cubrir los primeros meses de operación del proyecto. Incluye:

- Sueldos y honorarios del equipo de trabajo en etapas iniciales de implementación.
- Pago de servicios e insumos básicos necesarios para la ejecución del sistema.
- Fondo para imprevistos, destinado a solventar errores, fallos técnicos o ajustes durante la fase de despliegue inicial.

- Recursos destinados a marketing y lanzamiento, para garantizar la difusión y posicionamiento de ReciBot en el mercado.
- Mantenimiento evolutivo del sistema, tanto en la versión web como en la aplicación de escritorio, asegurando su actualización y permanencia en el tiempo.

Análisis de restricción

A continuación, se presenta un análisis integral de las restricciones del proyecto ReciBot, organizado en diferentes categorías:

Restricciones Técnicas

El éxito del proyecto ReciBot depende fundamentalmente de aspectos técnicos que pueden limitar su rendimiento y escalabilidad. Entre ellos se destacan:

- **Calidad y cantidad de datos:** La representatividad y el correcto etiquetado de las imágenes de residuos son vitales para que las librerías TensorFlow y OpenCV funcionen de manera óptima. La falta de un conjunto de datos robusto puede afectar significativamente la precisión del modelo de clasificación, dejando un porcentaje de acierto bastante bajo.

- **Integración y compatibilidad de tecnologías:** El proyecto contempla el desarrollo de dos versiones (web y de escritorio) utilizando frameworks distintos (Flask y CustomTkinter), lo que implica desafíos en mantenimiento y actualizaciones.

- **Compatibilidad de versiones de Python:** Para asegurar la correcta operación de las dependencias, es recomendable utilizar Python 3.8, que ha demostrado ser estable y

compatible con TensorFlow, limitando así la adopción de versiones más recientes que puedan presentar incompatibilidades.

- **Calidad de la detección de imágenes:** Un aspecto esencial es la capacidad de la cámara para detectar y reconocer correctamente los residuos. Factores como la iluminación, el ángulo de captura, la resolución de la imagen y el fondo pueden influir significativamente en la precisión de la clasificación. Es fundamental implementar algoritmos y ajustes que permitan aumentar el porcentaje de efectividad.

Restricciones Culturales

El éxito de ReciBot también está condicionado por factores culturales que pueden afectar la adopción y uso de la tecnología:

- **Resistencia al cambio:** Las prácticas tradicionales de manejo de residuos están profundamente arraigadas en algunas comunidades, lo que puede generar desconfianza hacia nuevas soluciones tecnológicas y un posible rechazo por el cambio.

- **Concienciación sobre el reciclaje:** La efectividad del sistema depende de la participación activa de la población y el rechazo a utilizar el ReciBot, y la falta de conocimiento o el desconocimiento de la importancia del reciclaje pueden ser barreras significativas.

Restricciones Ambientales

El proyecto debe evaluar el impacto ambiental de la tecnología utilizada, especialmente en términos de consumo energético y gestión de datos. El uso de modelos de

inteligencia artificial con redes neuronales convolucionales (CNNs) requiere un alto procesamiento computacional, lo que puede incrementar el consumo eléctrico y generar una mayor huella de carbono. Además, si la plataforma se implementa en dispositivos físicos, es importante garantizar que el hardware utilizado sea sostenible y eficiente energéticamente.

Restricciones Económicas

El desarrollo de ReciBot debe ajustarse a los recursos disponibles. Tecnologías como YOLO, TensorFlow y PyTorch pueden requerir hardware especializado como GPUs de alto rendimiento, lo cual podría representar un costo elevado. Además, si el modelo se despliega en servidores en la nube, se deben considerar los costos asociados al almacenamiento y procesamiento de datos. La inversión en software y licencias también debe evaluarse, optando por herramientas de código abierto cuando sea posible.

Restricciones Tecnológicas

El rendimiento del modelo de IA puede verse afectado por limitaciones en la capacidad computacional de los dispositivos donde se ejecute. La versión de escritorio debe optimizarse para funcionar en equipos con especificaciones estándar, mientras que la versión web debe garantizar tiempos de respuesta rápidos y una infraestructura escalable para soportar múltiples usuarios simultáneamente. Además, el uso de redes neuronales convolucionales (CNNs) y modelos YOLO requiere un entrenamiento previo con grandes volúmenes de datos, lo que implica la necesidad de bases de datos extensas y diversidad en los conjuntos de entrenamiento.

Figura 1. Plan de Mantenimiento

🛠 Plan de Mantenimiento del Sistema ReciBot

Objetivo	Garantizar la disponibiliidad, funcionalidad, precisión y eficiencia del sistema ReciBot a través de tareas de mantenimiento preventivo, corretiv.o y evolutivo.		
Tipo de Mante-nimiento	**Tarea**	**Frecuencia**	**Responsable**
Preventivo	Actualización de bibliotecas de Python (TensorFlow, PyTorch, Flask, etc.)	Cada 2 meses	Equipo de desarrollo
Preventivo	Reentrenamiento del modelo de IA con nuevos datos	Cada 6 meses	Ingeniero de IA
Correctivo	Copias de seguriidad de la base de datos y del servidor	Semanal	Administrador de sistemas
Correctivo	Corrección de errores de clasificación	Según demanda	Equipo de soporte técnico
Evolutivo	Reparación de fallas en la interfaz	Según demanda	Desarrollador web/escritorio
Evolutivo	Optimización de la arquitectura del modelo de IA	Cada 6 meses	Ingeniero de IA

Limitaciones del Proyecto

A pesar de los avances logrados con el desarrollo de ReciBot, el proyecto presenta una serie de limitaciones que deben reconocerse, ya que influyen directamente en el alcance de los resultados obtenidos y en la posibilidad de extrapolar el sistema a contextos más amplios.

Cobertura y variabilidad del dataset

Si bien se utilizó un conjunto considerable de imágenes para entrenar y validar los modelos, la variabilidad en condiciones reales aún resulta limitada. Factores como cambios bruscos de iluminación, fondos heterogéneos, ángulos no considerados en el entrenamiento o la presencia de residuos deteriorados pueden reducir la precisión del sistema al ser aplicado fuera del ambiente controlado de pruebas. Esta restricción implica que el modelo requiere un

proceso continuo de alimentación con nuevos datos y reentrenamiento periódico, con el fin de mejorar su capacidad de generalización y adaptarse a escenarios diversos.

Dependencia de recursos de hardware.

El entrenamiento intensivo de modelos, especialmente en el caso de YOLOv8, demanda recursos computacionales especializados como GPUs de mediano rendimiento. Aunque este requisito fue cubierto durante el desarrollo del proyecto, en contextos donde se disponga únicamente de equipos de bajo costo o sin soporte para procesamiento paralelo, los tiempos de entrenamiento podrían ser excesivamente prolongados o incluso inviables. Esto limita la reproducibilidad del proyecto en entornos académicos o comunitarios con infraestructura tecnológica reducida, lo cual resalta la importancia de explorar alternativas como el uso de servicios en la nube, optimizaciones adicionales de los modelos o técnicas de *quantization* y *pruning*.

Escalabilidad funcional.

El prototipo se centra en la clasificación de cinco categorías de residuos, lo cual representa un avance inicial importante; sin embargo, la cobertura de materiales reciclables en la vida real es mucho más amplia. La ausencia de categorías adicionales como textiles, electrónicos o materiales compuestos limita la aplicabilidad del sistema en contextos industriales o municipales que demandan un nivel de clasificación más detallado.

Mantenimiento y sostenibilidad.

El desempeño del sistema depende de un plan de mantenimiento evolutivo que

asegure la actualización periódica de bibliotecas, frameworks y dependencias (TensorFlow, PyTorch, Flask, entre otros). La falta de actualización constante podría ocasionar vulnerabilidades de seguridad o pérdida de compatibilidad con nuevas versiones de hardware y software. Además, mantener la solución en producción requiere un equipo técnico con roles especializados, lo cual puede representar una restricción en términos de costos y disponibilidad de talento.

Cumplimiento de Objetivos

El proyecto ReciBot logró dar cumplimiento integral a los objetivos planteados desde su concepción, evidenciando un desarrollo sólido tanto en el ámbito técnico como en el metodológico. A continuación, se presenta un análisis detallado del alcance alcanzado en relación con el objetivo general y los objetivos específicos:

Objetivo general.

Se cumplió con la meta principal de diseñar, entrenar y validar un sistema de clasificación automática de residuos sólidos en cinco categorías (orgánico, plástico, metal, vidrio, papel y cartón). El modelo alcanzó una precisión superior al 85 %, lo que garantiza resultados confiables en escenarios de uso real. Este logro confirma la pertinencia de la propuesta y valida la aplicación de técnicas de inteligencia artificial como herramienta de apoyo a la gestión ambiental.

Objetivos específicos.

- **Implementación de la arquitectura del modelo y construcción del dataset.** Este objetivo fue alcanzado mediante la creación de un conjunto de datos robusto que incluyó miles de imágenes representativas de las distintas clases de

residuos. El dataset se complementó con técnicas de aumentación de datos, lo que permitió mejorar la capacidad de generalización de los modelos. Paralelamente, se implementaron dos enfoques de entrenamiento: un modelo construido desde cero con TensorFlow y un modelo YOLOv8 preentrenado, lo cual posibilitó un análisis comparativo entre metodologías.

- **Desarrollo de interfaces multiplataforma.** Se cumplió exitosamente con la construcción de dos versiones funcionales del sistema: una aplicación web, desplegada en un entorno accesible a través de internet, y una aplicación de escritorio, diseñada para funcionar de manera autónoma sin requerir conectividad. Ambas interfaces integran elementos gráficos y pedagógicos, facilitando la interacción del usuario y promoviendo la educación ambiental en torno al reciclaje.

En conclusión, los objetivos planteados en el marco del proyecto no solo se cumplieron en su totalidad, sino que además permitieron abrir nuevas perspectivas de investigación y desarrollo, orientadas a la optimización de modelos, la integración de nuevas categorías de residuos y la potencial escalabilidad de ReciBot en contextos más amplios.

A continuación, se presenta el análisis del cumplimiento de los objetivos propuestos en el proyecto ReciBot. Cada objetivo planteado fue alcanzado con evidencias concretas que demuestran el logro de la meta y la relevancia del desarrollo realizado.

Tabla de análisis de Cumplimiento de Objetivos

Tabla 1. Comparación de Cumplimiento de Objetivos

Objetivo	Evidencia de cumplimiento	Resultado alcanzado
Objetivo General: Desarrollar un aplicativo inteligente basado en visión artificial para la clasificación automática de residuos a partir de imágenes, proporcionando recomendaciones para su adecuado reciclaje.	Se diseñaron, entrenaron y validaron dos enfoques de modelo: (i) CNN desarrollada desde cero con TensorFlow y (ii) YOLOv8 preentrenado con ajuste fino. Se integraron módulos de clasificación y recomendaciones personalizadas para el usuario.	Se obtuvo un sistema funcional, capaz de clasificar cinco categorías de residuos con una precisión superior al 85 %, entregando además recomendaciones para su correcta disposición.
OE1. Desarrollar un modelo de clasificación de residuos basado en redes neuronales convolucionales (CNN) y técnicas de transferencia de aprendizaje, utilizando un conjunto de imágenes preprocesadas.	Se construyó un dataset con miles de imágenes clasificadas, complementadas con técnicas de aumentación. Se implementó un modelo CNN en TensorFlow y un modelo YOLOv8 con transferencia de aprendizaje.	Se alcanzó un modelo robusto, con métricas de evaluación satisfactorias en precisión, *accuracy* y *loss*, confirmando la efectividad de las técnicas aplicadas.
OE2. Implementar una aplicación web y de escritorio con interfaces interactivas para la detección y clasificación de	Se desarrolló una versión web basada en Flask, con frontend responsivo y soporte de clasificación en línea, y una versión de escritorio con Custom Tkinter, capaz de	Se logró la implementación completa de interfaces multiplataforma, intuitivas y adaptadas a distintos contextos de uso,

Objetivo	Evidencia de cumplimiento	Resultado alcanzado
residuos en tiempo real, integrando Flask, Django, OpenCV y Custom Tkinter.	funcionar sin conexión a internet. Ambas aplicaciones integraron OpenCV para el manejo de imágenes en tiempo real.	ampliando la accesibilidad y la utilidad del sistema.
OE3. Optimizar el desempeño del modelo mediante el ajuste de hiperparámetros y la evaluación con métricas como precisión, accuracy y loss, asegurando eficiencia y usabilidad.	Se ajustaron parámetros como número de épocas, tamaño de lote e imágenes de entrada. Se evaluaron métricas de desempeño con validación cruzada y pruebas en escenarios reales.	El sistema alcanzó niveles de precisión superiores al 85 %, con tiempos de respuesta adecuados y eficiencia comprobada en equipos de recursos limitados.

Conclusiones

El desarrollo del sistema ReciBot constituye un aporte significativo a la intersección entre la inteligencia artificial aplicada y la gestión ambiental, al ofrecer una solución tecnológica capaz de reconocer y clasificar residuos sólidos mediante algoritmos de visión por computador. La propuesta no se limita a ser un prototipo funcional, sino que integra un enfoque de investigación comparativa al evaluar dos metodologías de entrenamiento

distintas: la construcción de un modelo desde cero con TensorFlow y el ajuste fino de un modelo preentrenado con YOLOv8. Esta dualidad metodológica no solo permitió validar el desempeño técnico de ambas aproximaciones, sino que además fortaleció la discusión sobre el balance entre precisión, eficiencia y disponibilidad de recursos computacionales.

Los resultados obtenidos evidencian que el uso de modelos preentrenados, como YOLOv8, aporta ventajas sustanciales en términos de precisión, tiempo de entrenamiento y escalabilidad. Al haber sido entrenados con grandes volúmenes de datos, estos modelos logran una rápida adaptación al dominio de clasificación de residuos, reduciendo la necesidad de contar con infraestructuras de cómputo de alto costo. Esto representa un factor diferenciador frente a modelos diseñados desde cero, que requieren un proceso de entrenamiento más prolongado y demandante en términos de recursos, aunque ofrecen mayor flexibilidad para la experimentación académica.

Por otra parte, la arquitectura multiplataforma del sistema, que combina una versión web y una versión de escritorio, garantiza accesibilidad en contextos diversos. La aplicación web favorece la masificación y el acceso remoto desde cualquier dispositivo con conexión a internet, mientras que la aplicación de escritorio asegura autonomía en entornos con baja o nula conectividad. Esta complementariedad amplía las posibilidades de uso de ReciBot en comunidades educativas, institucionales y sociales, adaptándose a realidades heterogéneas y contribuyendo a la democratización de la tecnología.

Asimismo, el proyecto trasciende la dimensión puramente técnica al incluir elementos de diseño pedagógico y social, como la integración de iconografía educativa, recomendaciones personalizadas y mensajes visuales orientados a sensibilizar a los usuarios sobre la correcta disposición de los residuos. De este modo, ReciBot no solo automatiza un proceso, sino que también contribuye activamente a la construcción de una cultura de

reciclaje más consciente y sostenible, alineada con los objetivos de desarrollo sostenible y la educación ambiental.

Finalmente, puede concluirse que ReciBot es una iniciativa con potencial de escalabilidad y evolución futura, capaz de incorporar nuevos datasets, ampliar categorías de clasificación, integrarse con sistemas de gestión ambiental más complejos e, incluso, adoptar tecnologías emergentes como servicios en la nube y aprendizaje federado. En este sentido, el proyecto se proyecta como un punto de partida para investigaciones más amplias que aborden la aplicación de inteligencia artificial en problemáticas ambientales, reafirmando la pertinencia y relevancia de unir la ingeniería con el compromiso social y ecológico.

Referencias Bibliográficas

García, M., & Liu, X. (2022). *Waste Classification Dataset in Machine Learning: A Comprehensive Review*. Journal of Environmental Data Science, 15(2), 45-62. https://doi.org/10.1016/j.envdatasci.2022.02.001

Manchasoft. (2023). "Cifras que maneja el sector del reciclaje en el mundo". Disponible en: https://manchasoft.com/cifras-que-maneja-el-sector-del-reciclaje-en-el-mundo/

Noticias ONU. (2023). "Reutilizar, reciclar y reorientar ahorraría hasta un 80% la contaminación del plástico". Disponible en: https://news.un.org/es/story/2023/05/1521082

Patel, S., Kumar, R., & Zhang, L. (2023). *Garbage Classification Dataset for Recycling Optimization: A Machine Learning Approach*. Waste Management & Research, 41(3), 213-229. https://doi.org/10.1016/j.wasman.2023.05.002

Programa de las Naciones Unidas para el Medio Ambiente (PNUMA). (2023). "El mundo debe superar la era de los desechos y adoptar la economía circular". Disponible en: https://www.unep.org/es/noticias-y-reportajes/comunicado-de-prensa/el-mundo-debe-superar-la-era-de-los-desechos-y

Romera-Paredes, B., Gidaris, S., & Torres, M. (2022). *TACO: Trash Annotations in Context for Waste Detection and Classification*. International Journal of Computer Vision, 130(7), 1211-1235. https://doi.org/10.1109/IJCV.2022.1234567

Sectorial. (2025). "Colombia genera anualmente 24,8 millones de toneladas de residuos, de los cuales solo el 20% se recicla". Disponible en: https://sectorial.co/reciclaje/colombia-genera-anualmente-248-millones-de-toneladas-de-residuos/

Suresh, A., Wang, P., & Chen, Y. (2021). *Challenges and Limitations of TrashNet for Waste Classification: A Comparative Dataset Analysis*. IEEE Transactions on Environmental Informatics, 8(4), 654-669. https://doi.org/10.1109/TEI.2021.0986543

Yang, Z., Li, H., & Wu, J. (2019). *TrashNet and Beyond: Enhancing Waste Classification through Crowdsourced Datasets*. Proceedings of the International Conference on Environmental AI, 1(1), 89-104. https://doi.org/10.1145/3341165.3341189

El Tiempo. (2025). "¿Cuáles son las mayores dificultades para reciclar? Esto dicen los colombianos". Disponible en: https://www.eltiempo.com/vida/medio-ambiente/cuales-son-las-dificultades-para-reciclar-esto-dicen-los-colombianos-745907#:~:text=Al%20momento%20de%20reciclar%2C%20la,pa%C3%ADs%20hacia%20la%20econom%C3%ADa%20circular

Tallini, A., & Cedola, L. (2018). A review of the properties of recycled and waste materials for energy refurbishment of existing buildings towards the requirements of NZEB. Energy Procedia, 148, 868–875. https://doi.org/10.1016/j.egypro.2018.08.108l