UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Hacia una Ciudad Sostenible: Desarrollo de una Solución Móvil para Reciclaje Inteligente

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Pérez Leizaola, Naiara

Tutor/a: Julian Inglada, Vicente Javier

Cotutor/a: Marco Detchart, Cédric

CURSO ACADÉMICO: 2023/2024

Resum

L'objectiu principal del treball és desenvolupar una aplicació mòbil intuïtiva i accessible que facilite el procés de reciclatge oferint informació sobre reciclatge, especialment dissenyada per a persones amb poca competència tecnològica o coneixements limitats sobre el tema, permetent així que tots puguen contribuir a cura del medi ambient. L'aplicació permet als usuaris classificar tipus de residus utilitzant imatges com a entrada i proporciona informació sobre el contenidor més pròxim on reciclar-los. Per a aconseguir això, es va utilitzar la plataforma *Flutter* per a desenvolupar la interfaç d'usuari, es va implementar un model de xarxes neuronals per a la classificació d'imatges i es va establir un servici web per a connectar l'aplicació amb el model i les dades dels contenidors a través d'una *API*. Els resultats mostren que la intel·ligència artificial és una ferramenta molt valuosa, i esta pot ser acostada a l'usuari final facilitant així la realització del reciclatge de manera més accessible i senzilla.

Paraules clau: Intel·ligència artificial, Smart cities, aplicacions mòbils

Resumen

El objetivo principal del trabajo es desarrollar una aplicación móvil intuitiva y accesible que facilite el proceso de reciclaje ofreciendo información sobre reciclaje, especialmente diseñada para personas con poca competencia tecnológica o conocimientos limitados sobre el tema, permitiendo así que todos puedan contribuir al cuidado del medio ambiente. La aplicación permite a los usuarios clasificar tipos de residuos utilizando imágenes como entrada y proporciona información sobre el contenedor más cercano donde reciclarlos. Para lograr esto, se utilizó la plataforma *Flutter* para desarrollar la interfaz de usuario, se implementó un modelo de redes neuronales para la clasificación de imágenes y se estableció un servicio web para conectar la aplicación con el modelo y los datos de los contenedores a través de una *API*. Los resultados muestran que la inteligencia artificial es una herramienta muy valiosa, y esta puede ser acercada al usuario final facilitando así la realización del reciclaje de manera más accesible y sencilla.

Palabras clave: Inteligencia artificial, Smart cities, aplicaciones móviles

Abstract

This paper focuses on the development of a mobile application designed to facilitate the recycling process through an intuitive and accessible interface. The main objective is to develop an intuitive and accessible mobile application that facilitates the recycling process by providing information on recycling, especially designed for people with little technological competence or limited knowledge on the subject, thus allowing everyone to contribute to the care of the environment. The application allows users to classify types of waste using images as input and provides information on the nearest recycling bin. To achieve this, the *Flutter* platform was used to develop the user interface, a neural network model was implemented for image classification and a web service was set up to connect the application with the model and bin data via an API. The results show that artificial intelligence is a very valuable tool, and can be brought closer to the final user, making recycling easier and more accessible.

Key words: Artificial intelligence, Smart cities, Mobile applications

Índice general

Apéndice

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Contexto y Motivación

Un problema de gran importancia que ha adquirido una mayor atención por parte de los gobiernos y los medios en los últimos años es el cambio climático y los daños que nuestra manera de vivir han causado al medio ambiente. Sin embargo, este problema no ha surgido recientemente y la comunidad científica lleva advirtiendo los efectos que se están viendo hoy en día desde hace décadas.

Uno de los ámbitos principales donde se han hecho mejoras para los problemas de medioambiente, es en el reciclaje. Gracias al reciclaje, se puede disminuir la gran cantidad de desechos que la humanidad produce en todo el mundo. No obstante, a pesar de ser una de las medidas para detener el cambio climático y otros problemas relacionados con el medio ambiente, aún queda mucho camino por recorrer para que se realice el reciclaje de manera correcta, tanto por parte de las instituciones como por parte de los ciudadanos.

En el caso de los ciudadanos, se ha mostrado una mejora en las nuevas generaciones debido a la educación que se ha recibido. Sin embargo, esto no ha ocurrido en todos los lugares, ya que no todas las instituciones han tenido el interés o la posibilidad de subvencionar programas para enseñar estos conceptos con propiedad. Un ejemplo claro de esto es como en los países con más dinero y donde se encuentra una mayor corriente ecologista, como en el caso de Alemania o Austria, el porcentaje de residuos reciclados es mayor; a diferencia de en países con menos recursos y menor relevancia de la rama ecologista en política, como en Bulgaria o Rumanía. Se puede observar la diferencia de las cantidades de residuos reciclados por persona en los países de la Unión Europea en el gráfico de la figura 1.1 [Eurostat, 2024].

En la observación ciudadana, se evidencia un notable desconocimiento en relación con las opciones de reciclado, situación que se ha visto incrementada con la reciente in-

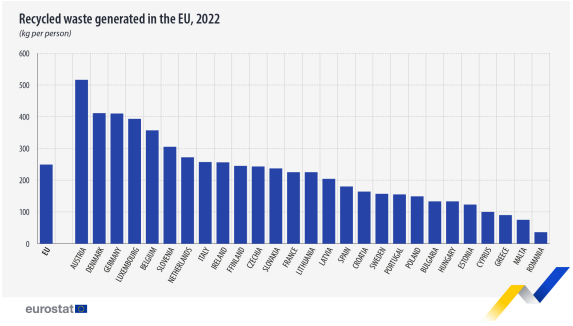

Figura 1.1: Desechos reciclados en la UE.

troducción del contenedor de orgánicos. A medida que se han diversificado las opciones de contenedores para separar los residuos, surge una dificultad adicional para los ciudadanos al momento de seleccionar el contenedor adecuado para cada tipo de desecho. Esta complejidad en la elección puede derivar en una incorrecta disposición de los residuos y, en última instancia, en un impacto negativo en los esfuerzos de reciclaje y sostenibilidad.

Es por esto, que en este proyecto se plantea la necesidad de crear una herramienta más accesible para todas las personas dispuestas a tratar de mejorar la situación del reciclaje, pero sin la capacidad de acceso a la información necesaria ni el tiempo para dedicar al aprendizaje sobre ello. Especialmente para personas de edad avanzada, tanto la dificultad de adquirir nuevos conceptos como la motivación de averiguar información sobre temas que no les apasionan es mayor. De esta forma, se planteó que era necesario para mejorar esta situación crear dicha herramienta que permitiese al usuario individual acceder a la información necesaria para que le facilitase llevar a cabo la tarea del reciclaje de manera simple y sencilla.

1.2 Objetivos

Debido a lo explicado en la sección previa, se decidió crear una aplicación que pudiese ayudar a las personas a reciclar los residuos de una manera sencilla. Para ello se decidió proponer que la aplicación permitiese al usuario añadir la imagen del residuo que requiriese, conocer donde debe reciclarlo, así como la localización del contenedor más cercano a donde se encontrase el usuario.

Por tanto, el objetivo principal del trabajo es crear esta aplicación como herramienta para que cualquier persona pueda conseguir información fácilmente como ayuda para poder reciclar. Al querer ofrecer este servicio de manera simple, se trata de tener accesibilidad para personas diferentes, que no se suelen tener en cuenta, como personas de mayor edad con menores competencias tecnológicas o menores conocimientos sobre el reciclaje. De esta manera se consigue que todo el mundo pueda aportar su pequeño grano de arena para construir un mundo mejor y tratar de evitar la destrucción de nuestro planeta tierra como lo conocemos.

Para conseguir el objetivo principal se han definido varios objetivos específicos a seguir, que se realizarán a lo largo del trabajo propuesto.

- En primer lugar, se realizará un análisis de las herramientas de las que se dispone para efectuar el trabajo, con el propósito de escoger los instrumentos que sean más adecuados para el desarrollo.

- Seguidamente, se diseñará la arquitectura del sistema, que dictará los elementos que contendrá además de la relación que tendrán entre ellos.

- A continuación, se realizará el diseño, implementación y la validación del modelo de clasificación, donde se probarán diferentes modelos hasta lograr el mejor modelo posible para clasificar las imágenes.

- Finalmente, se describirá el diseño y desarrollo de la aplicación móvil, donde además se describirán los accesos que la aplicación necesitará para cumplir correctamente todas las acciones que requiera.

1.3 Estructura de la memoria

En esta sección, se hará una descripción de la manera en la que se estructura la memoria a continuación. Esta estructura se divide en los siguientes seis capítulos:

- **Introducción**: El primer capítulo en el que nos encontramos actualmente, describe brevemente lo que se va a realizar en el trabajo, la motivación detrás de este, así como los objetivos propuestos para el trabajo.
- **Estado del arte**: El segundo capítulo analiza el estado del arte, donde se examinarán trabajos similares y se analizarán las herramientas para utilizar a continuación.
- **Arquitectura del sistema**: En el tercer capítulo se comentará la arquitectura del sistema, donde se analizarán los requisitos de los elementos propuestos, además de profundizar en la interacción entre ellos.
- **Modelo de Clasificación**: En el cuarto capítulo se analizarán diferentes tipos de modelos de redes neuronales hasta escoger uno con los mejores resultados para el caso concreto para el que se necesita.
- **Aplicación Desarrollada**: En el quinto capítulo se diseñará e implementará la aplicación, y además, se comentará como se manejan ciertos datos externos como el acceso al clasificador o a los datos de contenedores.
- **Conclusiones**: Finalmente, en el sexto capítulo se realizará una recapitulación de los objetivos que se han logrado cumplir en el trabajo respecto a los propuestos inicialmente, además de proponer ciertas mejoras futuras para el trabajo.

CAPÍTULO 2
Estado del arte

Desde la era industrial, la sociedad ha sido responsable de una decadencia importante del estado del medio ambiente. Desde los años 70, los científicos han estado advirtiendo sobre el cambio climático y no ha sido hasta recientemente que se han empezado a hacer mayores cambios respecto a todo lo que se provoca con el sistema actual que afecta a la naturaleza y al planeta. Una de las áreas donde se han empezado a efectuar cambios, es en la gestión de residuos, que gracias al reciclaje, está ayudando a reducir la necesidad de manufacturar nuevos productos con recursos naturales y aumentar los nuevos productos hechos con recursos reciclados. Es por ello que la gestión adecuada de residuos se ha convertido en un desafío crítico en la sociedad moderna. A medida que la población crece y los patrones de consumo evolucionan, la generación de residuos aumenta exponencialmente.

La identificación y clasificación precisas de los tipos de residuos han sido históricamente un desafío, con métodos tradicionales basados en la inspección visual humana y sistemas de clasificación mecánicos. Antes de la adopción generalizada de las redes neuronales, la clasificación de imágenes dependía de algoritmos y técnicas de procesamiento de imágenes convencionales. Estos incluían el procesamiento de imágenes y la extracción manual de características, máquinas de vectores de soporte (SVM), árboles de decisión, métodos basados en reglas y análisis de componentes principales (PCA). Aunque útiles, estas técnicas presentaban limitaciones en la adaptación a la complejidad de imágenes reales de residuos.

La evolución de las redes neuronales convolucionales (CNN) ha sido impresionante, marcando un hito en la clasificación de imágenes y el procesamiento visual. Inicialmente, las CNN se popularizaron en la década de 1990, pero su eficacia se vio limitada debido a las capacidades computacionales de la época y la falta de grandes conjuntos de datos para entrenar modelos complejos. Fue en la última década, con los avances significativos en hardware y la disponibilidad de grandes cantidades de datos etiquetados, cuando las CNN experimentaron una resurgencia espectacular.

La arquitectura básica de una CNN implica capas convolucionales que aplican filtros para extraer características locales y capas de agrupación que reducen la dimensionalidad. A medida que los modelos evolucionaron, se introdujeron capas adicionales, como las capas de normalización y de abandono, mejorando la estabilidad y generalización del modelo. Además, arquitecturas más complejas, como la red neuronal convolucional profunda (DCNN) y la red neuronal residual (*ResNet*), permitieron la construcción de modelos más profundos y precisos, abordando el problema de la degradación del rendimiento en redes más profundas.

La implementación de técnicas como la transferencia de aprendizaje, donde se aprovechan los conocimientos aprendidos por una CNN en una tarea para aplicarlos a otra,

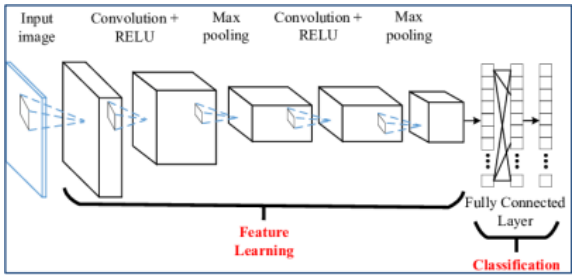

Figura 2.1: Arquitectura de la Red Neuronal Convolucional.

ha contribuido a mejorar la eficiencia del entrenamiento y la capacidad de generalización de los modelos. La evolución constante de las CNN también se refleja en la diversificación de aplicaciones, extendiéndose desde la clasificación de imágenes hasta la detección de objetos, segmentación semántica y análisis de video. En conjunto, esta evolución continua ha posicionado a las CNN como una herramienta esencial en la gestión de residuos y en diversas áreas de la visión por computadora.

2.1 Crítica al estado del arte

En los últimos años, diversos investigadores han dirigido su atención hacia la aplicación de redes neuronales para la clasificación de imágenes de residuos, contribuyendo así a la búsqueda de soluciones sostenibles para la gestión de deshechos.

Algunos de los trabajos de investigación notables en esta área incluyen el trabajo *Waste Classification using Convolutional Neural Network* [Azis et al., 2020] publicado en los procedimientos de la Conferencia Internacional de Tecnología de la Información y Comunicaciones en Computadoras (ITCC '20), aborda este desafío al explorar la aplicación de la arquitectura *Inception-v3* en la clasificación automatizada y precisa de residuos. Este estudio ofrece una perspectiva innovadora sobre cómo las Redes Neuronales Convolucionales pueden transformar la gestión ambiental, destacando el potencial de la tecnología para impulsar prácticas de reciclaje más eficientes.

El uso de Redes Neuronales Convolucionales (CNN) es una técnica ampliamente difundida para la clasificación de imágenes en aprendizaje profundo. Este enfoque extrae características del conjunto de datos de imágenes, agrupa el conjunto y, luego, utiliza esta información para clasificar imágenes desconocidas de manera apropiada. La CNN consta de varias capas, como se muestra en la Figura 2.1, que se puede encontrar en [Islam et al., 2018]: 1) capa convolucional, 2) capas de activación y agrupación, y 3) capa totalmente conectada.

La Figura 2.2, que se puede encontrar en [Serengil, 2018], proporciona una comparación de rendimiento entre diferentes modelos de *ImageNet*. El estudio de referencia realiza un análisis detallado de diversas redes neuronales profundas para el reconocimiento de imágenes, comparando su rendimiento en términos de operaciones requeridas para el procesamiento y la precisión obtenida utilizando el conjunto de datos *ImageNet*.

Para la clasificación de residuos, se selecciona el modelo *Inception-v3* debido a un compromiso entre precisión y cantidad de procesamiento. Aunque *Inception-v3* tiene 42 capas, su eficiencia computacional es solo aproximadamente 2,5 veces mayor que la de *GoogleNet*, a pesar de tener 24 millones de parámetros. Esto permite el uso de un procesador portátil y económico, Raspberry Pi 3B, tanto para entrenamiento como para pruebas, crucial para un sistema asequible de segregación de residuos.

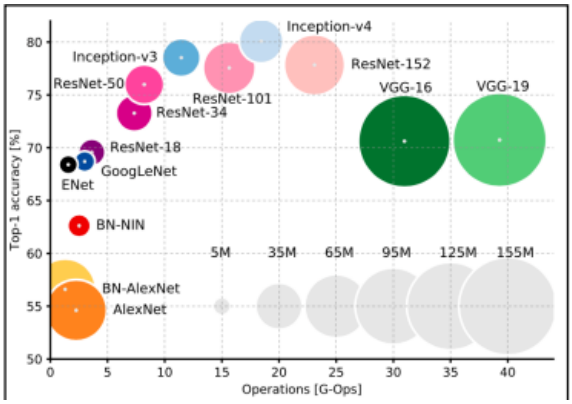

Figura 2.2: Comparación del rendimiento de diferentes modelos de ImageNet.

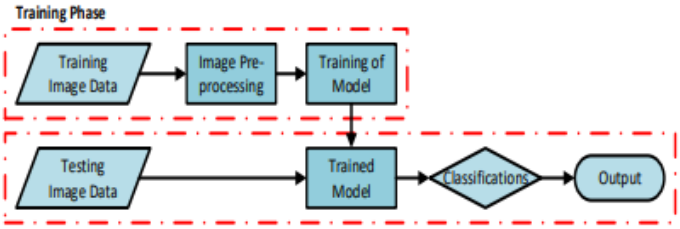

Figura 2.3: Metodología simplificada para las fases de entrenamiento y prueba.

La arquitectura de *Inception-v3* está compuesta por capas como convolución, agrupación promedio, agrupación máxima, concatenación, eliminación, totalmente conectada y activación $Softmax$. La imagen de entrada se ajusta a 512 x 384, y la salida es la probabilidad de que los datos de entrada pertenezcan a uno de los seis tipos de residuos considerados.

Para la metodología, se divide en fases de entrenamiento y prueba, como se muestra en la Figura 2.3. Durante el entrenamiento, se etiquetan los datos de imagen según los seis tipos de residuos, y estos datos se utilizan para entrenar el modelo *Inception-v3*. Una vez entrenado, se determina la precisión del modelo utilizando datos de imagen de prueba.

Se obtienen alrededor de 2.400 datos de imagen de entrada de un repositorio de modelos de detección de plástico en *GitHub* para entrenamiento y prueba. Estos datos se dividen en conjuntos de entrenamiento, validación y prueba. Se logra una precisión del 99 % y 92 % en los conjuntos de entrenamiento y validación, respectivamente. Durante la prueba, la precisión global del modelo es del 92,5 %, con precisiones más altas para metal y papel (95 %) y precisiones más bajas para plástico y otros residuos (90 %).

La matriz de confusión revela que el modelo es capaz de clasificar correctamente la mayoría de los tipos de residuos, con algunas confusiones, como plástico con vidrio. La eficiencia computacional es mínima, y los resultados son prometedores para un sistema de reciclaje automatizado y asequible.

Como resultado final, el estudio propone el uso de una CNN, específicamente el modelo *Inception-v3*, para la clasificación general de residuos. Se logra una alta precisión del 92,5 %, lo que respalda la viabilidad de un sistema automatizado y económico para la segregación de residuos. Para mejorar aún más la precisión, se sugiere considerar otros métodos de clasificación y recopilar imágenes adicionales de residuos comunes para mejorar el entrenamiento del modelo. Los resultados son alentadores para el desarrollo de sistemas de reciclaje eficaces y accesibles.

Figura 2.4: Imágenes de muestra de *OrgalidWaste*.

Un segundo trabajo notable en esta área es un estudio presentado en *IEEE Xplore*, donde se abordó la clasificación de residuos mediante técnicas avanzadas de aprendizaje profundo, proponiendo un enfoque innovador para mejorar la eficacia en la identificación y clasificación de residuos [Faria et al., 2021]. En este trabajo utilizan diferentes métodos de mejora para conseguir un mayor porcentaje de precisión en la red neuronal, que clasifica entre cuatro clases; vidrio, metal, orgánico y plástico.

En el proceso de preparación de datos, se destaca la importancia del preprocesamiento para potenciar el rendimiento de los modelos de aprendizaje automático en el conjunto de datos *OrgalidWaste* (Figura 2.4), que fue creado por los autores del trabajo, junto con algunas imágenes tomadas tanto de *Trashnet* [Yang and Thung, 2016] como de *Trash* [Ochecolo, 2020]. La técnica de aumento de datos se emplea para enriquecer la capacidad de los modelos al generar ejemplos adicionales durante el entrenamiento. Utilizando la función *ImageDataGenerator* de la biblioteca *Keras.preprocessing*, se aplican transformaciones como rotación, volteo horizontal y vertical, así como desplazamientos en ancho y alto. Las imágenes se redimensionan a 224 píxeles para mantener uniformidad.

En el ámbito de la implementación de modelos, se exploran cinco arquitecturas de redes neuronales convolucionales (CNN) con un enfoque especial en el entorno colaborativo *Google Colab*. Cuatro de estas arquitecturas, *VGG16*, *VGG19*, *ResNet50* e *Inception-V3*, se implementan mediante la técnica de transfer learning, una metodología que aprovecha conocimientos previos para resolver problemas de aprendizaje en diferentes dominios. Se profundiza en las características de cada arquitectura, desde una CNN de tres capas hasta modelos más complejos como *VGG16* e *Inception-V3*.

La fase de entrenamiento y prueba se lleva a cabo con el optimizador Adam y la función de pérdida de entropía cruzada categórica. Se ajustan diferentes épocas para cada modelo con el fin de evitar el sobreajuste y mejorar la capacidad de generalización. Los modelos se evalúan en carpetas de validación y prueba del conjunto de datos, proporcionando una visión completa del rendimiento de cada arquitectura.

Los resultados revelan el rendimiento de los modelos en términos de precisión y tiempo de ejecución. VGG16 se destaca con la mayor precisión, alcanzando un 88,42 %, y muestra una convergencia sólida durante el aprendizaje. Este enfoque se valida con la matriz de confusión, donde se evidencia la capacidad de clasificación precisa, especialmente en las categorías de metal y orgánico.

En términos generales, este trabajo aborda la problemática global de la gestión de residuos no procesados. Al introducir el conjunto de datos *OrgalidWaste* y aplicar modelos avanzados de aprendizaje automático, como *VGG16*, se demuestra el potencial de estas

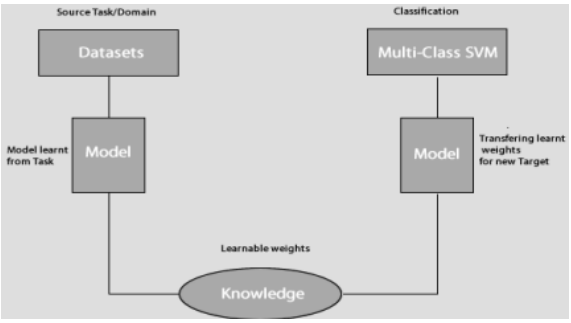

Figura 2.5: Diagrama de flujo del método propuesto.

tecnologías para mejorar la eficiencia en la clasificación de residuos. Este enfoque tiene implicaciones significativas para la gestión sostenible de residuos, mostrando cómo la ingeniería informática y las redes neuronales pueden contribuir de manera crucial a los desafíos medioambientales actuales.

Otro trabajo destacado en la clasificación de residuos mediante técnicas avanzadas de aprendizaje profundo se encuentra en un estudio publicado en ScienceDirect, el cual aborda de manera innovadora la identificación y clasificación de residuos [Adedeji and Wang, 2019]. El estudio se centra en el desarrollo de un sistema de clasificación de residuos automatizado, impulsado por la creciente preocupación por la acumulación de desechos a nivel mundial. Para ello decidieron utilizar los datos de *Trashnet* que contienen seis clases de residuos; plástico, papel, cartón, vidrio, metal y basura.

Para abordar esta problemática, los investigadores proponen un sistema basado en la combinación de una Red Neuronal Convolucional (CNN) y una Máquina de Soporte Vectorial (SVM), como se muestra en la Figura 2.5. El enfoque de la CNN, específicamente utilizando la arquitectura *ResNet-50* preentrenada, se elige para abordar las limitaciones del conjunto de datos, que es relativamente pequeño. La CNN se utiliza para extraer características de las imágenes de residuos, y luego estas características se introducen en un modelo SVM para la clasificación final.

En el experimento y resultado se utilizan las características del modelo explicadas previamente, así como el uso de descenso de gradiente estocástico con momentum(SGDM) durante el entrenamiento para actualizar los valores de los pesos y los sesgos. El dataset se dividió guardando el 20 % para prueba y utilizando el resto en el entrenamiento, con lotes pequeños de 12. Este estudio logró una precisión en el test de 87 %.

En resumen, el estudio demuestra la viabilidad de utilizar modelos de aprendizaje profundo, especialmente CNN y transfer learning, para abordar el desafío de clasificar eficientemente los residuos. La implementación práctica de estos modelos revela perspectivas sobre su rendimiento relativo y destaca la importancia del preprocesamiento de datos para mejorar la precisión del modelo. Con un énfasis en la automatización de la clasificación de residuos, el informe proporciona una base sólida para futuras mejoras y desarrollos en este campo crítico de gestión ambiental.

Tanto estos como otros estudios existentes que utilizan las redes neuronales para la clasificación de residuos muestran que el uso de estas está teniendo un auge tanto en la comunidad científica como en la sociedad en general. De la misma manera, se destaca la creciente implicación de la población para poder abordar los desafíos medioambientales, tratando de aportar algo a los avances de esta área, aunque no se pertenezca como tal a ella. Se podría destacar que estos estudios no solo son valiosos por sus hallazgos y metodologías específicas, sino que también cumplen un papel más amplio como fuente de inspiración y referencia para investigaciones futuras. Otros investigadores pueden

construir sobre estas bases, adaptar metodologías exitosas y explorar nuevas direcciones basándose en el conocimiento y la experiencia compartidos en estos trabajos.

2.2 Tecnologías relacionadas

La elección de tecnologías desempeña un papel crucial en el éxito y la eficacia de cualquier proyecto, y en el ámbito de la clasificación de residuos mediante técnicas avanzadas, esta premisa cobra una importancia aún mayor. La selección adecuada de herramientas, lenguajes de programación, marcos de trabajo y plataformas de desarrollo no solo impacta la calidad y eficiencia del trabajo, sino que también influye en la viabilidad y el rendimiento de las soluciones propuestas. En esta sección, se explorarán detalladamente las categorías de tecnologías clave utilizadas en los trabajos de clasificación de residuos revisados, abordando sus fortalezas, limitaciones y cómo su elección estratégica contribuye al logro de los objetivos específicos de cada estudio.

Para ello se ahondará en diferentes opciones para cada una de las tecnologías necesarias para poder implementar el trabajo deseado. Se comenzará por explicar las diferentes tecnologías existentes para poder desarrollar modelos basados en redes neuronales, así como concretar las razones por la que se eligió el lenguaje finalmente. El mismo proceso se llevará a cabo con los datasets que se utilizarán posteriormente para la red neuronal, los *frameworks* que se utilizarán para el desarrollo de la aplicación y los servicios web para poder añadir diferentes utilidades a la aplicación, así como para poder llamar al modelo desde la aplicación al clasificar las imágenes.

2.2.1. Modelos basados en redes neuronales

En la era de la inteligencia artificial, las redes neuronales han mostrado su gran utilidad en muchos ámbitos, emulando la complejidad del cerebro humano para abordar desafíos diversos. Compuestas por capas interconectadas de nodos, estas redes ofrecen soluciones innovadoras en campos como el reconocimiento de imágenes y el procesamiento de lenguaje natural. En este contexto, la elección de un lenguaje para implementar redes neuronales desempeña un papel crucial, ya que influye en la eficiencia y flexibilidad del modelo final.

Para la implementación de la red neuronal en el marco del trabajo de fin de grado, se evaluaron exhaustivamente tres opciones prominentes en el ámbito de la inteligencia artificial: TensorFlow[1], Keras[2] y PyTorch[3]. Cada uno de estos marcos presenta características distintivas que fueron cuidadosamente consideradas antes de tomar una decisión fundamentada, disponibles en la Tabla 2.1.

TensorFlow: TensorFlow, desarrollado por Google, es un marco de aprendizaje automático de código abierto ampliamente utilizado. Se destaca por su versatilidad y capacidad para operar en una variedad de plataformas, desde dispositivos móviles hasta sistemas distribuidos. Algunos aspectos clave de TensorFlow incluyen:

- TensorFlow es conocido por su escalabilidad, su capacidad para manejar proyectos de aprendizaje automático a gran escala, permitiendo la implementación eficiente de modelos complejos y extensos.

[1] https://www.tensorflow.org/
[2] https://keras.io/
[3] https://pytorch.org/

Tensorflow	Keras	PyTorch
-Buena escalabilidad	-Interfaz de alto nivel	-Grafo computacional dinámico
-Herramienta Tensorboard	-Integración con Tensorflow	-Fácil proceso de depuración
-Comunidad extensa	-Abstracción intuitiva	-Comunidad activa

Tabla 2.1: Comparación de lenguajes para la red neuronal.

- La comunidad de TensorFlow es extensa y activa, lo que resulta en una abundancia de recursos, tutoriales y soporte técnico. Esto facilita la resolución de problemas y la optimización de modelos.
- La herramienta TensorBoard integrada en TensorFlow proporciona visualizaciones interactivas para facilitar la comprensión y la depuración de modelos.

Keras: Keras, originalmente desarrollado como una interfaz de alto nivel para TensorFlow, se ha integrado completamente con este último y se ha convertido en un marco independiente. Algunas de las razones por las que destaca Keras son:

- Keras ofrece una interfaz de alto nivel que simplifica significativamente la construcción y el entrenamiento de modelos de redes neuronales. Esta simplicidad es especialmente valiosa para la experimentación y el prototipado rápido.
- La integración profunda de Keras con TensorFlow, un marco ampliamente reconocido y utilizado en la comunidad, proporciona estabilidad y confiabilidad en la implementación.
- Keras proporciona una abstracción intuitiva para definir capas, conexiones y entrenar modelos de redes neuronales. Esto agiliza el proceso de investigación y desarrollo al reducir la complejidad de la implementación.

PyTorch: PyTorch es otro marco de aprendizaje automático de código abierto que ha ganado popularidad debido a su naturaleza dinámica y amigable para el usuario. Algunos aspectos destacados de PyTorch incluyen:

- PyTorch utiliza un grafo computacional dinámico, lo que facilita la construcción de modelos más dinámicos y cambios en tiempo real en la estructura del modelo durante la ejecución.
- PyTorch ha sido adoptado por muchos investigadores debido a su flexibilidad y facilidad de uso en experimentos de investigación. Esto ha llevado al desarrollo de una comunidad activa en el ámbito académico.
- La estructura de PyTorch facilita el proceso de depuración, lo que es beneficioso durante las fases de desarrollo y experimentación.

Después de una evaluación minuciosa de estas opciones, la elección de Keras se justifica por su simplicidad, flexibilidad y eficacia. La interfaz de alto nivel de Keras, combinada con su integración perfecta con TensorFlow, proporciona un entorno propicio para el desarrollo rápido de prototipos y la investigación en el ámbito de las redes neuronales, cumpliendo así con los requisitos específicos de este proyecto.

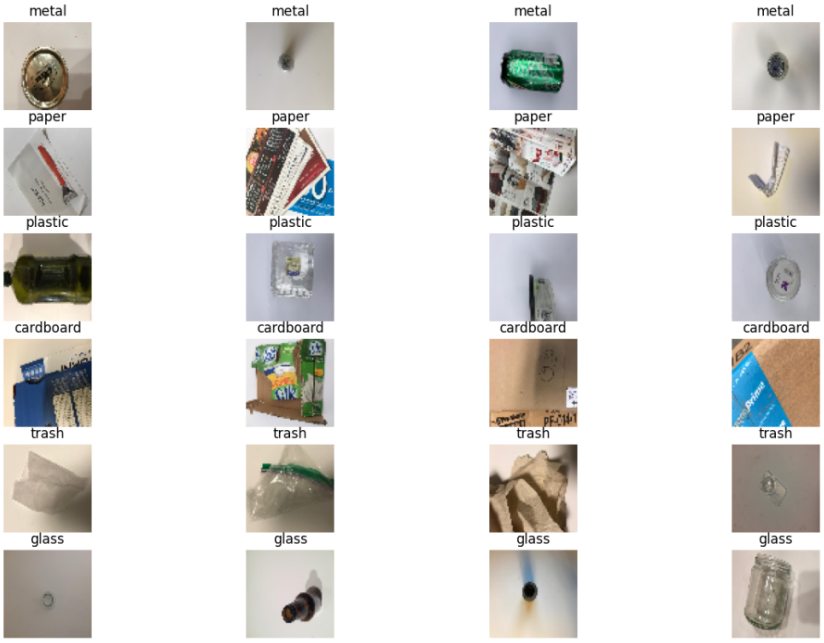

Figura 2.6: Varias imágenes de cada clase del dataset 'Trashnet'.

2.2.2. Datasets

Uno de los primeros desafíos cruciales en este proyecto fue la búsqueda de un conjunto de datos adecuado para la tarea. Era necesario encontrar un conjunto de datos que contuviera una cantidad suficiente de imágenes y categorías para permitir una clasificación precisa de los residuos en la aplicación.

Durante la búsqueda, fueron encontrados varios conjuntos de datos que se limitaban a distinguir entre residuos reciclables y no reciclables, o entre categorías como orgánicos y reciclables, como es el caso del conjunto de datos *Waste Classification Data* [Sekar, 2019].

Entre los conjuntos de datos revisados, se continuó con *RecyclePhoto* [Acosta, 2019], que se centra en la clasificación de cartón y plástico. Lo cual podría ser útil para la clasificación específica de estos dos tipos de residuos, pero no tenía el número de categorías necesario para este trabajo.

Luego, se exploró el conjunto de datos *Drinking Waste Classification* [Serezhkin, 2020], que se especializa en la clasificación de residuos asociados con bebidas, como latas de aluminio, vidrio, y envases de PET y HDPEM. Este enfoque específico lo convierte en una herramienta valiosa para abordar desafíos relacionados con los residuos generados por productos de consumo comunes. Pero al ser tan específico, no era lo que se buscaba para el reciclaje diario del usuario.

El conjunto de datos *Taco* [Proença and Simões, 2020] es extenso, abarcando más de 20 clases, lo que proporciona una amplia gama de categorías de residuos. Sin embargo, la gran cantidad de clases podría complicar el entrenamiento y la eficacia del modelo para aplicaciones específicas de reciclaje cotidiano.

Finalmente, *Trashnet* [Yang and Thung, 2016] se destaca como una opción sólida al incluir seis categorías: cartón, vidrio, metal, plástico, papel y basura. La amplitud de categorías y la calidad del conjunto de datos hacen que sea una elección adecuada para abordar la clasificación de residuos en el contexto del reciclaje diario de un usuario promedio. La diversidad de clases y la relevancia práctica hacen que *Trashnet* sea el elegido, respaldado por su idoneidad para el reciclaje cotidiano y la gestión eficiente de residuos.

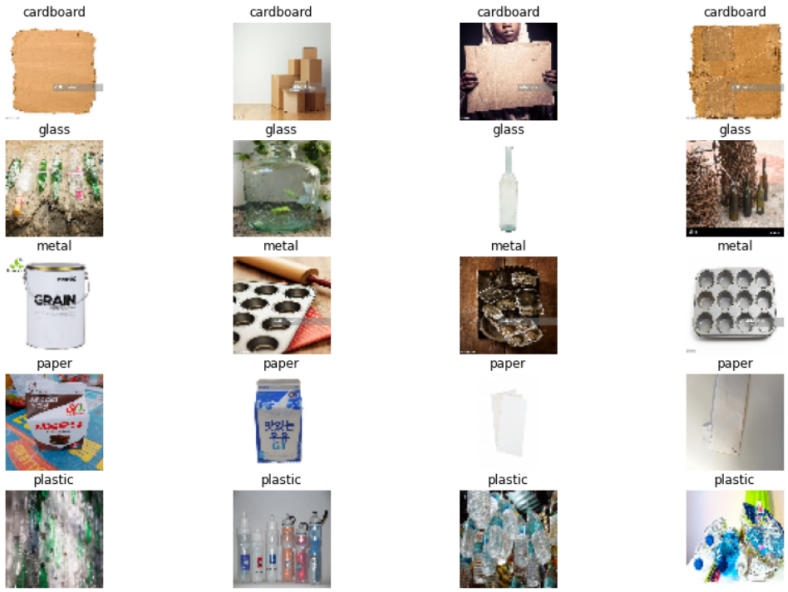

Figura 2.7: Varias imágenes de cada clase del dataset 'TrashBox'.

TrashNet, mostrado en la Figura 2.6 ha demostrado ser una elección sólida en diversas investigaciones previas, que han empleado este conjunto de datos con éxito en aplicaciones de clasificación de residuos. Entre los trabajos que han utilizado *TrashNet* se incluyen los siguientes:

- Un estudio publicado en la revista *Sustainability* investigó la identificación y clasificación de residuos sólidos urbanos. [Malik et al., 2022].
- Un proyecto de investigación de la Universidad de Stanford [Awe et al., 2017] que explora la clasificación de residuos con un enfoque de aprendizaje profundo.
- Una investigación para una tesis de doctorado de la universidad de National College of Ireland [Pote, 2022] que utiliza un modelo de aprendizaje profundo para clasificar los residuos del dataset.

La versatilidad de *TrashNet* ha respaldado con éxito una variedad de enfoques y técnicas en estos trabajos, lo que subraya su idoneidad para nuestra propia investigación en la clasificación de residuos. Posteriormente, para lograr ampliar el número de muestras, se consideró mezclar el dataset mencionado con otro para obtener mejores resultados, y para ello se utilizó *TrashBox* [Kumsetty et al., 2022], mostrado en la Figura 2.7. Este dataset incluye algunas clases más que el dataset utilizado inicialmente, y por ello fue editado para satisfacer las seis clases que se encuentran en el inicial, quitando las clases de residuos médicos y residuos electrónicos.

2.2.3. Framework

Un *framework*, en este contexto, se refiere a un conjunto estructurado de herramientas, bibliotecas y patrones de diseño que proporcionan una base sólida y coherente para construir aplicaciones móviles de manera eficiente. En lugar de abordar cada aspecto del desarrollo desde cero, los *framework* ofrecen una infraestructura predefinida que facilita la creación, implementación y mantenimiento de aplicaciones.

La elección del *framework* adecuado para el desarrollo de aplicaciones móviles representa una consideración estratégica en el ámbito tecnológico actual. Entre las opciones

más destacadas, como React Native[4], Node.js[5] y Flutter[6], se llevó a cabo una evaluación exhaustiva para determinar la elección más apropiada. A continuación, se detallan las características distintivas de cada opción, disponibles en la Tabla 2.2:

Node.js: Node.js, un entorno de ejecución de JavaScript del lado del servidor, ha ganado popularidad en el desarrollo web. Algunos aspectos clave de Node.js incluyen:

- Node.js permite utilizar JavaScript tanto en el lado del cliente como en el servidor, brindando una coherencia en el lenguaje de programación a lo largo de toda la aplicación.
- La arquitectura no bloqueante de Node.js, basada en eventos, permite manejar múltiples operaciones simultáneamente, lo que es especialmente beneficioso para aplicaciones que requieren una alta concurrencia.
- Node.js se beneficia del Node Package Manager (NPM), que facilita la gestión de dependencias y la integración de módulos externos, contribuyendo a la modularidad del desarrollo.

React Native: React Native, respaldado por Facebook, es un marco de desarrollo que permite construir aplicaciones móviles utilizando JavaScript y React. Algunas características destacadas de React Native son:

- React Native permite compartir código entre aplicaciones de iOS y Android, lo que reduce el esfuerzo necesario para el desarrollo en múltiples plataformas.
- React Native permite la incorporación de componentes nativos, lo que proporciona un rendimiento más cercano al de las aplicaciones nativas.
- La funcionalidad de *hot reload* facilita la visualización instantánea de los cambios realizados en el código, acelerando el proceso de desarrollo.

Flutter: Flutter, respaldado por Google, destaca por su capacidad para construir aplicaciones móviles con un único código base en el lenguaje de programación Dart. Algunas características notables de Flutter incluyen:

- Flutter utiliza un único código base para iOS y Android, lo que acelera el desarrollo y permite una rápida expansión a otras plataformas.
- Dart, el lenguaje de programación de Flutter, ofrece un rendimiento sólido y es fácil de aprender, contribuyendo a la eficiencia en el desarrollo.
- Flutter proporciona una amplia colección de widgets personalizables que facilitan la creación de interfaces de usuario únicas y atractivas.

La elección de Flutter se basa en diversas consideraciones. En primer lugar, la eficiencia del desarrollo se ve impulsada por la capacidad de Flutter para utilizar un único código base, permitiendo una expansión sencilla a múltiples plataformas. El lenguaje Dart también juega un papel crucial al proporcionar un rendimiento sólido y una curva de aprendizaje amigable. La alta personalización de la interfaz de usuario y la comunidad activa de Flutter complementan estas ventajas, creando un entorno propicio para la creación de aplicaciones móviles modernas y atractivas en el ámbito de este proyecto de fin de grado.

[4] https://reactnative.dev/
[5] https://nodejs.org/en
[6] https://flutter.dev/

Node.js	React Native	Flutter
-JavaScript en cliente/servidor -Arquitectura no bloqueante -Fácil gestión de dependencias	-Comparte código iOS/Android -Incorpora componentes nativos -Hot reload	-Comparte código iOS/Android -Facilidad de aprendizaje Dart -Widgets personalizables

Tabla 2.2: Comparación de *framework* para la aplicación.

Django	Flask	FastAPI
-Comunidad extensa -Amplia documentación -Funciones ya implementadas	-Sencillo y flexible -Mayor control de decisiones -Sin larga configuración	-Rápido, sencillo y flexible -Validación automática -Inyección de dependencia

Tabla 2.3: Comparación de API Python.

2.2.4. Servicio web

Una API, o Interfaz de Programación de Aplicaciones, sirve como un conjunto de reglas y herramientas que permite la comunicación y la interacción entre distintos sistemas de software. Funciona como un intermediario, permitiendo que aplicaciones o servicios se conecten y compartan datos de manera estandarizada.

Para poder lograr utilizar la red neuronal creada que analice la foto que se le manda desde la aplicación y devuelva la clase a la que esta pertenece, es necesario tener una API que conecte la aplicación de Flutter con la red neuronal que clasifica la imagen que se le pasa a esta. Para ello es necesario decidir cuál es la API más adecuada para este caso. Una API actúa como intermediario entre dos sistemas, es decir, permite a diferentes aplicaciones comunicarse entre sí, en este caso, se utiliza como medio de comunicación entre la red neuronal de Keras y la aplicación de Flutter. A continuación, se detallan las características de las tres API que fueron consideradas; Django[7], Flask[8] y FastAPI[9]; disponibles en la Tabla 2.3:

Django: Django es el *framework* más grande y utilizado en Python, por lo que tiene una comunidad y una trayectoria extensa, con grandes compañías como Instagram o Spotify utilizándolo. De entre los tres *framework* de API, Django destaca en las siguientes características:

- Lleva mucho tiempo en el mercado y tiene una comunidad extensa, lo cual hace que tenga una gran cantidad de información que se pueda conseguir para los problemas que surjan.
- Tiene una documentación muy amplia, de más de 3.000 páginas.
- Tiene una gran cantidad funciones ya implementadas, como seguridad o panel de administración y más.

[7] https://www.djangoproject.com/
[8] https://flask.palletsprojects.com/en/3.0.x/
[9] https://fastapi.tiangolo.com/

Flask: Flask es el segundo *framework* más utilizado en Python, que grandes compañías como Netflix y Reddit utilizan en sus plataformas. Flask destaca en los siguientes aspectos:

- Se centra más en ser sencillo de aprender y flexible, destacando la facilidad que tiene para que aprendan nuevos desarrolladores.
- Deja tomar más decisiones al desarrollador sobre cómo implementarlo.
- No es necesario pasar por todo el proceso de configuración que es necesario en otros como Django.

FastAPI: FastAPI es el *framework* más reciente y moderno de los tres mencionados. A pesar de esto ha crecido mucho en los últimos años y ha llegado a usarse en grandes empresas como Uber. De él destacan estas características:

- Como su nombre indica, es una API que tiene una gran velocidad, comparable con la velocidad de grandes API para otros lenguajes como *NodeJS* y *GO*.
- Al igual que *Flask*, destaca su facilidad de aprender a utilizarla, al igual que su flexibilidad.
- Cuenta con validación automática, lo cual evita la necesidad de utilizar herramientas externas para ello.
- Tiene una inyección de dependencia incorporada que asegura que las clases no tengan dependencia entre sí, gracias a esto se pueden hacer cambios en el código de manera más sencilla, aumentado de esta manera la escalabilidad.

Tras considerar todos los aspectos de cada una de las API previamente mencionadas, la que se consideró más apropiada para utilizar en este proyecto fue *FastAPI*, debido a su facilidad de aprendizaje, su flexibilidad, su velocidad, así como la comodidad de que incluya validación automática e inyección de dependencia incorporada.

2.3 Conclusiones

A lo largo del capítulo se ha podido observar como previamente se han realizado trabajos en el ámbito del reciclaje de residuos, que al igual que en el trabajo que se plantea, utilizan redes neuronales para ello. Se ha hablado de tres trabajos diferentes, específicamente, que utilizan distintos tipos de modelos de redes para clasificar los residuos, que utilizan técnicas variadas, como aprendizaje por transferencia utilizando diferentes modelos como *VGG16*, *Inception-v3*, etc., así como máquinas de soporte de vectorial y otras técnicas. Por ello, se puede hacer notar la relevancia y creciente empleo de las redes neuronales como solución a todo tipo de problemas, así como el aumento de importancia dada a la sostenibilidad y el reciclaje para poder disminuir los efectos nocivos que la sociedad está teniendo en el planeta.

Para la posible realización del trabajo se han analizado las distintas tecnologías disponibles para los diferentes ámbitos que las necesitaban. Así consiguiendo elegir la mejor opción de cada una de ellas para el problema que se quiere resolver. En el caso de los modelos basados en redes neuronales, se decidió utilizar la biblioteca de *Keras*, debido a su facilidad, flexibilidad y eficacia. Para el dataset, se escogió *Trashnet*, debido a su fiabilidad, al haber sido utilizado en numerosos trabajos y a una cantidad de clases y

clasificación adecuada de estas para el problema. Referente al *framework*, se terminó eligiendo *Flutter*, por su sencillez de expansión, su facilidad de aprendizaje, su comunidad activa y su alta capacidad de personalización. Finalmente, el servicio web elegido fue *FastApi*, principalmente, por su flexibilidad, su velocidad, su facilidad de aprendizaje, su propiedad de validación automática e inyección de dependencia.

A continuación, se comenzará a contar como se ha realizado la solución al problema propuesta de una manera más general, comentando también la estructura de esta, antes de comenzar a explicar cada uno de los aspectos de la solución con mayor profundidad.

CAPÍTULO 3
Arquitectura del sistema

El propósito de este capítulo es mostrar la interconexión de los elementos de nuestro trabajo mediante su arquitectura. Para ello se comentará sobre el propósito de cada uno de los elementos del sistema y se analizarán los requisitos para cada uno de ellos. La implementación y adecuada unión de los elementos serán vitales para la correcta solución final de la aplicación que le llegará al usuario. Es por ello que una buena arquitectura es clave en el sistema, aportando eficiencia al producto final de la aplicación móvil.

La aplicación móvil a implementar tiene como objetivo lograr clasificar los residuos y ayudar al usuario a encontrar donde poder reciclar estos. Debido a esto, sus funcionalidades principales serían las siguientes: la primera sería poder clasificar las imágenes proporcionadas por el usuario en el tipo de residuo correctamente; otra función necesaria sería facilitar el reciclado mediante un mapa con la ubicación del usuario y de los contenedores del tipo requerido para el residuo clasificado previamente utilizando los datos públicos sobre los contenedores. En este capítulo, se explicarán a rasgos generales los elementos necesarios y como se comunicarán entre ellos.

3.1 Componentes de la Arquitectura

En la realización del trabajo, se implementaron varios elementos que posteriormente necesitarían estar unidos para conseguir las tareas necesarias para el funcionamiento deseado. Antes de profundizar en su arquitectura, se describirán brevemente los elementos necesarios. Estos elementos son los que componen la arquitectura y se pueden dividir en cuatro fracciones.

El primer elemento sería la interfaz de usuario, que en el sistema del trabajo realizado se referiría a la aplicación móvil desarrollada. Esta interfaz es la capa final que se muestra al usuario. En ella se recopilan los datos necesarios por parte del usuario y se devuelven las funcionalidades proporcionadas con la información útil para el usuario. En la aplicación, se solicitará al usuario que proporcione la imagen que desea clasificar y tras realizar el proceso en otros elementos, se le devolverá la información del tipo de residuo obtenido, además se le ofrecerá, tras conseguir los permisos de ubicación del usuario, el mapa que muestre los contenedores de la zona cercana al usuario gracias a los datos que se tiene de estos.

El siguiente elemento a definir es el *backend*, donde se establece la conexión con los datos necesarios para el funcionamiento adecuado de la aplicación móvil. En este componente, se encuentran dos aspectos importantes en relación con los datos: la obtención de información de la página de datos públicos del organismo público adecuado, en este caso el ayuntamiento en cuestión, que proporciona los datos de los contenedores a

mostrar en el mapa de la aplicación; y la implementación de un componente lógico de negocio de *Flutter* encargado de lograr los permisos de ubicación del usuario. Esto permite posteriormente mostrar en el mapa los contenedores cercanos a donde se encuentra el usuario.

A continuación, se determina el elemento del modelo de la red neuronal para la clasificación de imágenes. Este elemento se encargará de realizar el proceso de clasificación de imágenes para poder establecer cual sería el tipo de contenedor adecuado para depositar el objeto encontrado en la imagen. Para ello, es necesario realizar un entrenamiento previo con un conjunto de datos fijado, que tras lograr el mejor resultado con el conjunto de datos que se tiene de antemano se guardará. Con este modelo obtenido, se realizará posteriormente la clasificación de la imagen que se reciba del usuario, devolviendo la solución del tipo de residuo al que pertenecería dicha imagen.

Finalmente, se encuentra el elemento de las API, a través del cual se integran otros elementos. En el sistema se encuentran dos API que conectan la aplicación con otros dos elementos. La primera está diseñada para conectar el modelo de clasificación de imágenes con la aplicación móvil. Cuando la aplicación recibe una imagen para clasificar, esta API la procesa y la pasa al mejor modelo disponible. Una vez que el modelo ha realizado la clasificación, la API devuelve la información obtenida a la aplicación para su visualización o uso posterior. La segunda se encarga de descargar los archivos que contienen los datos de los contenedores, fundamentalmente la posición y el tipo. La API se ejecuta automáticamente periódicamente para garantizar que los datos estén actualizados y disponibles para su uso en la aplicación.

3.2 Metodología y Requisitos del Sistema

Los elementos previamente mencionados requieren una manera de poder comunicarse entre ellos con un flujo de datos determinado. Esto es necesario para que la aplicación pueda ofrecer al usuario todos los servicios que se quieren facilitar al mismo. De esta manera, tras recoger la información otorgada por el usuario, se puede tratar y conseguir devolver la información que el usuario quiere conocer.

El usuario final solamente se va a encontrar en contacto con la aplicación, aportando y recibiendo datos de este según sea necesario para las acciones deseadas por el usuario. La funcionalidad principal del sistema, como se ha mencionado previamente, es clasificar los residuos y mostrar los contenedores cercanos a la zona donde se encuentra el usuario. Para poder realizar este proceso, el usuario tiene que transmitir ciertos datos a la aplicación. Lo primero es que para poder clasificar la imagen requerida por el usuario, el sistema va a necesitar que el usuario le transfiera la imagen que desee para que este pueda procesarla, lo cual se realiza con un paquete de *Flutter*, que permite al usuario abrir la cámara para sacar una foto o escoger una de la galería. Posteriormente, se comunica de nuevo con el usuario, devolviéndole el tipo de residuo al que pertenece la imagen tras clasificarla, mostrando en pantalla textualmente el tipo de residuo al que pertenece. El otro punto de comunicación que el usuario debe transmitir a la aplicación es debido a que para poder mostrar el mapa de la zona donde se encuentra el usuario, le debe otorgar a la aplicación permisos de localización, para que de esta manera la aplicación sea capaz de mostrarle el mapa con los contenedores de su zona al usuario.

Posteriormente, para poder conseguir mostrar al usuario las funcionalidades mencionadas, como el mapa con los contenedores y la respuesta de clasificación del residuo, la aplicación debe comunicarse con el resto de elementos. Para poder comunicarse con los datos de los contenedores y el modelo, la aplicación llama a las API respectiva para cada uno de ellos, la cual se ha definido en el punto anterior. La manera en la que la app con-

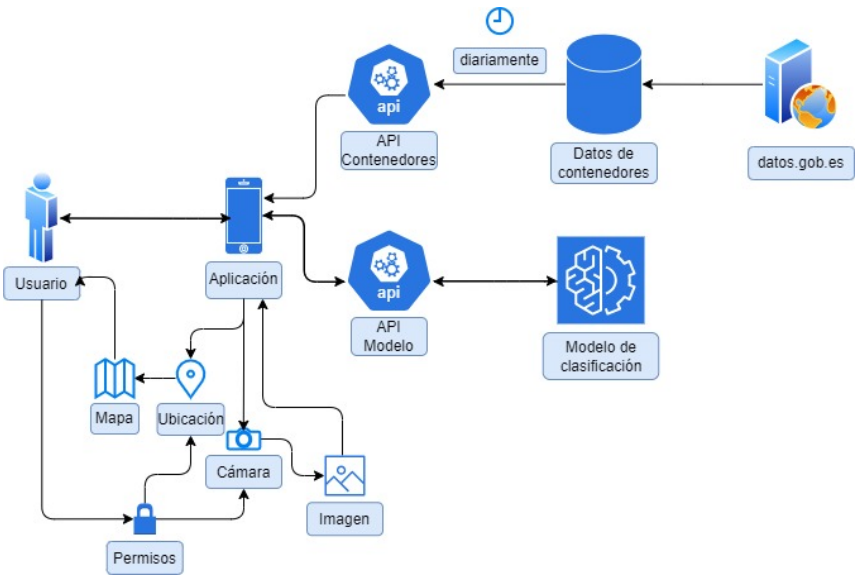

Figura 3.1: Diagrama de la arquitectura de la aplicación.

sigue llamar a la API, se realiza utilizando un paquete de *Flutter*, que permite conseguir datos desde internet para poder utilizarlos en la aplicación después. De esta manera, como se ha mencionado en el punto previo, se pueden tanto enviar los datos necesarios a través de las API a los otros elementos como recibir respuesta u otros datos desde estos.

Las API se encuentran conectadas, además de con la aplicación, con los datos de los contenedores y el modelo de redes neuronales. Estos dos elementos están conectados de manera distinta con sus respectivas API. El acceso a los datos de los contenedores se realiza utilizando un paquete de Python que permite hacer solicitudes a páginas web, de manera que permite descargar desde la página del gobierno los datos de los contenedores, y con el proceso que se realiza en la API actualizar los archivos periódicamente. En el caso del modelo, la API utiliza el archivo que se ha guardado tras el entrenamiento con el que se ha obtenido mejor resultado. En esta API se utiliza la imagen recibida desde la aplicación y se realiza el proceso de clasificación de esta utilizando el modelo de mejor resultado. Con el resultado obtenido de la clasificación se devolverán las predicciones a la aplicación. En la figura 3.1 se puede observar un diagrama de la arquitectura.

Para que la arquitectura propuesta sea próspera, es necesario que cada uno de los elementos descritos previamente cumpla con ciertos requisitos propuestos. Estos requisitos serán de ayuda para lograr los objetivos propuestos inicialmente. Se propone que para que la interfaz de usuario de la aplicación desarrollada funcione como es deseado, sea accesible para un amplio rango de usuarios, así como amigable, de modo que los usuarios no se sientan confundidos y consigan utilizar la aplicación sencillamente, sin importar sus competencias tecnológicas.

Respecto al *backend*, los datos que se manejan, tanto respecto a los contenedores como los permisos que se otorgan, se requiere transparencia para que los usuarios estén claros de los permisos que otorgan, no se utilizan sin su permiso, además de ofrecer la posibilidad de revocarlos si fuese necesario. También, se requiere mantener la información, como los datos de los contenedores, actualizada para poder ofrecer un servicio actualizado y que no se ofrezca información errónea por el cambio de información de la base de datos.

En el caso del modelo, los requisitos más importantes son la precisión y la generalización. El usuario debe tener la confianza de que el sistema le va a proporcionar la información del tipo de residuo correcta respecto a la imagen que le facilita a la aplicación. Es por ello que el modelo de red neuronal tiene que ser un modelo con la mejor precisión

posible, además de tener una capacidad de generalización para evitar la concentración en ciertos tipos de residuos y que el usuario sea capaz de recibir la correcta clasificación de los residuos que proporcione.

Finalmente, tenemos las API que conectan unos elementos con otros, por lo que es necesario que estas funcionen de la manera más eficiente posible, con una estructura en ellas que facilite la escalabilidad. Esto es debido a que necesitamos que la conexión sea rápida para que los usuarios no tengan que esperar para recibir respuestas de otros elementos por un tiempo prolongado. Además, es necesario mantener esa velocidad en el caso de que el número de usuarios aumente, por lo que es importante la escalabilidad.

3.3 Conclusiones

Durante el desarrollo de este capítulo, hemos explorado la arquitectura del sistema de nuestra aplicación móvil, comentando los componentes de esta, los requisitos de cada uno, además del funcionamiento de la arquitectura que conecta los componentes. La idea principal del trabajo realizado es crear una aplicación accesible para una amplia variedad de usuarios, a pesar de su falta de conocimientos tecnológicos, y hacerla eficiente y eficaz en su funcionalidad.

La arquitectura diseñada pretende facilitar el cumplimiento de los requisitos de accesibilidad y eficiencia propuestos inicialmente. La separación de los componentes, la interconexión entre ellos y el manejo adecuado de los datos contribuirán a una experiencia de usuario satisfactoria y a un funcionamiento óptimo de la aplicación. Sin embargo, también se ha identificado que la escalabilidad de la arquitectura podría convertirse en un desafío a medida que la aplicación crezca y se expanda, ya que podría requerir ajustes adicionales para manejar un aumento en la carga de usuarios y datos.

Por último, cabe destacar que se busca avanzar significativamente en la eficiencia y la arquitectura del sistema de la aplicación móvil. La meta es desarrollar una estructura modular y una interconexión de los componentes que sean efectivas para garantizar un rendimiento óptimo y una respuesta rápida a las solicitudes de los usuarios. Sin embargo, se es consciente de que siempre existe margen para mejorar. Por lo tanto, se priorizará la optimización continua de la arquitectura y la implementación de prácticas de desarrollo eficientes para mantener la aplicación ágil y adaptable a medida que evolucionen las necesidades del usuario y la tecnología. En resumen, si bien se aspira a alcanzar un nivel satisfactorio de eficiencia en la arquitectura del sistema, se reconoce la importancia de seguir buscando oportunidades para mejorar y optimizar el enfoque para garantizar el máximo rendimiento y escalabilidad a largo plazo.

CAPÍTULO 4
Modelo de Clasificación

En este capítulo se va a profundizar en como se ha llevado a cabo el proceso de búsqueda y elección del modelo final utilizado para la clasificación de residuos. Para ello se comenzó por buscar un conjunto de datos adecuado y posteriormente se fueron eligiendo diferentes modelos que posteriormente fueron entrenados y probados, hasta encontrar el mejor modelo posible, con la mejor precisión de todos los probados previamente.

4.1 Conjunto de Datos

El primer paso al decidir el modelo de la red neuronal, fue elegir el dataset más adecuado para nuestra red neuronal. Tras hacer una búsqueda, previamente mencionada en el estado del arte, la decisión fue elegir el dataset *Trashnet*. Este dataset fue el que se consideró más adecuado para el trabajo, por su número de imágenes y su número de clases, así como la amplia cantidad de trabajos donde fue utilizado con buenos resultados.

4.1.1. Descripción

El conjunto de datos *Trashnet* [Yang and Thung, 2016] constituye una recopilación cuidadosamente seleccionada de imágenes que abarcan seis categorías representativas de los residuos más comúnmente encontrados en el día a día: papel, cartón, plástico, vidrio, metal y restos. Esta elección se fundamentó en reflejar los desechos que las personas suelen reciclar con mayor frecuencia. Con un total de 2.527 imágenes, la distribución por clases es la siguiente: 594 de papel, 403 de cartón, 482 de plástico, 501 de vidrio, 410 de metal y 137 de restos. Las imágenes se presentan con un fondo blanco y están iluminadas tanto por la luz solar como por la ambiental de la habitación. Además, se capturaron en diversas posiciones y estados, como papel liso o arrugado, botellas rotas, etc., para garantizar que el modelo pueda clasificar con precisión, independientemente de las variaciones de posición.

En una etapa posterior, se incorporó al análisis el conjunto de datos adicional denominado *TrashBox* [Kumsetty et al., 2022], que amplía la variedad de clases a siete: vidrio, plástico, metal, papel, cartón, residuos médicos y residuos electrónicos. Este conjunto de datos cuenta con un total de 17.785 imágenes, distribuidas de manera desigual entre las clases, con representaciones que incluyen 2.528 imágenes de vidrio, 2.669 de plástico, 2.586 de metal, 2.695 de papel, 2.414 de cartón, 2.010 de residuos médicos y 2.883 de residuos electrónicos. Las imágenes de *TrashBox* fueron principalmente seleccionadas de fuentes en línea y presentan objetos específicos en cada categoría, como botellas, bolsas, latas, papeles, entre otros. Las imágenes se han capturado en diferentes fondos e iluminación para agregar diversidad al conjunto de datos.

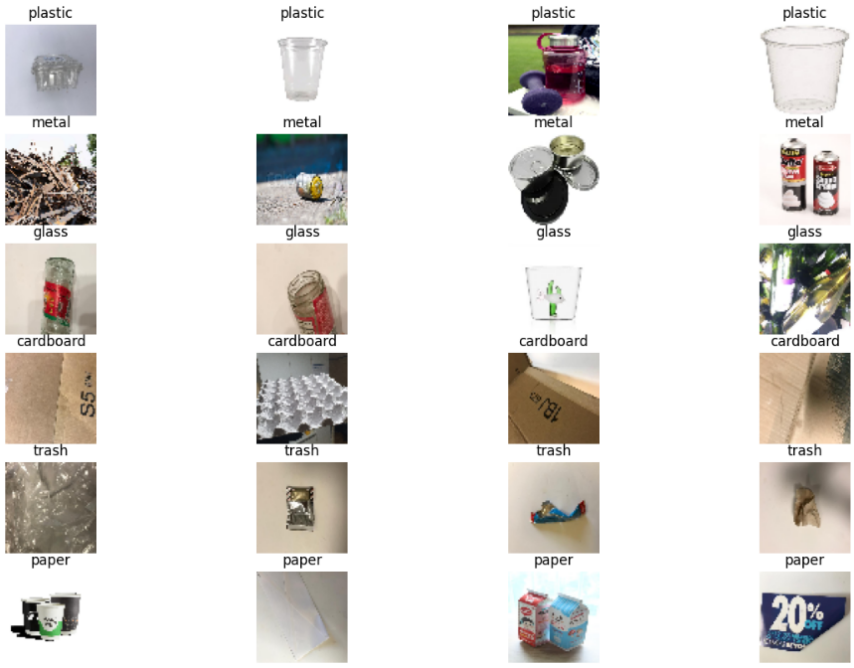

Figura 4.1: Varias imágenes de cada clase del dataset final unido.

La integración de ambos conjuntos de datos se llevó a cabo mediante la adición de imágenes específicas de las clases de vidrio, plástico, metal, papel y cartón al conjunto original *Trashnet*. Este proceso ha resultado en un conjunto de datos ampliado y más diverso, ahora compuesto por un total de 12.177 imágenes. Este enriquecimiento del conjunto de datos mejora la capacidad del modelo para generalizar y clasificar una amplia gama de residuos, incluso ante diversas condiciones y formas de presentación en las imágenes. En la Figura 4.1 se pueden observar varias imágenes de la unión de los dos conjuntos de datos.

4.1.2. Estructura y etiquetado

Al iniciar el trabajo tan solo se pensaba utilizar el conjunto de datos de *Trashnet*, por lo que el etiquetado del proyecto fue modelado por como estaba en este dataset. Por lo tanto, el conjunto de datos que se utiliza está compuesto por seis clases etiquetadas según su categoría de residuo, al igual que en el caso del dataset *Trashnet*. Estas categorías fueron escogidas, debido a que estas son los residuos que se suelen reciclar habitualmente y pueden ser los más comúnmente solicitados por los usuarios. Las categorías de los datos, fueron modelados de la misma manera que en el conjunto de datos inicial de *Trashnet*, con los nombres de las clases de los distintos residuos nombrados en inglés de la siguiente manera: *plastic* para plásticos, *glass* para vidrio, *paper* para papel, *cardboard* para cartón, *metal* para metal y *trash* para restos.

Al unir las imágenes del segundo dataset *Trashbox*, al tener este las mismas categorías que las que teníamos, pero en lugar de tener restos con una clase para residuos médicos y otra para residuos electrónicos, lo que se hizo fue simplemente añadir a su respectiva categoría las imágenes de los nuevos datos. Las clases que no tenían ninguna clase equivalente a la suya fueron desechadas, al igual que no se añadieron imágenes a la clase de restos porque el nuevo conjunto de datos no incluía imágenes de esta categoría. Hubo ciertas imágenes de la categoría de plástico del conjunto de datos *Trashbox* que se tomó la decisión de no añadir debido a que eran imágenes de cigarrillos que tenían el filtro de plástico, pero los cigarrillos no pueden ser reciclados, pese a tener el filtro de plástico.

Para clasificar las clases de residuos, se separaron las clases en distintas carpetas, las cuales se intentaron distribuir equiparadamente, pero al tener la clase de restos solamente en el conjunto de datos de *Trashnet*, esta no tiene el mismo número de imágenes. De hecho, de antemano en el conjunto de datos de *Trashnet*, tenía un menor número de imágenes que el resto de clases de residuos, con 137 imágenes, frente alrededor de 400 o 500 imágenes que se encontraban en el resto de clases que se encontraban en el dataset original.

Para poder ejecutar el modelo y realizar el entrenamiento y pruebas, el conjunto de datos fue dividido utilizando el paquete de Python *splitfolders*. Este, mediante el método *splitfolders.ratio* permite que se dividan las imágenes de las clases que se encuentran en el conjunto de datos en los porcentajes especificados en el método para las imágenes de entrenamiento, validación y pruebas. Se decidió dividir las imágenes en los porcentajes de 70 % de las imágenes para entrenamiento, 20 % para la validación y 10 % para las pruebas.

En el caso del conjunto de datos inicial no se tuvo que realizar ningún preproceso para poder utilizar las imágenes, que se pasaron directamente al generador mediante *flow from directory*, que automáticamente hizo que los datos se pudiesen utilizar en la red neuronal posteriormente. Sin embargo, al intentarlo con el conjunto de datos aumentado al unir el segundo dataset fue necesario guardar las imágenes en un array que posteriormente fue convertido en un array de *NumPy*, en el que antes de añadir las imágenes fue cerciorado que todas las imágenes añadidas tuvieran el tamaño adecuado y fuesen de tipo RGB, y a continuación se guardó en el generador con *flow*, en lugar de *flow from directory*. Los archivos del conjunto de datos son todos imágenes de tipo .jpg.

4.1.3. Análisis

En esta sección, se llevará a cabo una revisión detallada que abordará aspectos como el desequilibrio de clases, posibles ruidos en las muestras y otros factores que contribuyen a la complejidad del conjunto de datos. Este análisis crítico sienta las bases para una comprensión más profunda de los desafíos y limitaciones asociados con el conjunto de datos, lo cual es fundamental para la toma de decisiones informada durante el diseño y entrenamiento del modelo de clasificación.

Sobre el desequilibrio de clases, como se ha comentado previamente, podemos ver que en el dataset inicial ya tenemos una clase con un desequilibrio, ya que la clase de los restos contiene tan solo 137 imágenes, respecto al rango de entre 400 y 600 que tienen el resto de clases. Además, esta cantidad de imágenes se podría considerar reducida, por lo que se decidió tomar la medida de añadir un nuevo conjunto de datos al conjunto de datos inicial, con aproximadamente unas 2.500 imágenes más para cada clase. Este dataset no contenía imágenes que fueran de restos, por lo que se mantuvo con 137 imágenes a pesar de aumentar el conjunto de datos.

Esto puede perjudicar la clasificación de los residuos que son restos, ya que puede costar más encontrar el patrón de estos en una muestra más reducida. Sin embargo, debido a que el objetivo de esta aplicación es ayudar a reciclar al usuario, esto no nos preocupa realmente, ya que lo importante es poder reciclar, y al clasificar algo en los restos, esto no se va a poder reciclar, así que es de mayor interés que las clases que se puedan reciclar sean bien clasificadas. A pesar de eso, se tomó la decisión de hacer una prueba añadiendo imágenes de las clases que no se utilizaron del conjunto de datos de *Trashbox*, con residuos médicos y residuos electrónicos. Esta prueba mostró peores resultados que los resultados que se consiguieron sin equiparar la diferencia de clases, por lo que se decidió descartar y, al ser la categoría que realmente no se recicla, dejarlo como estaba al inicio.

Al unir las muestras del conjunto de datos inicial con el dataset *Trashbox*, se encontraron unas imágenes que no cuadraban con la categoría asignada, al tener imágenes de cigarrillos en la categoría de plásticos. Esto fue debido a que los creadores del dataset lo añadieron por el filtro de plástico que contienen, pero los cigarrillos no se reciclan actualmente y se deben desechar en el contenedor de restos. Es por ello que se decidió eliminar estas imágenes de la categoría de plásticos.

Otro de los problemas que se encontró en las muestras fueron las imágenes corrompidas. En el caso del conjunto de datos unificado de *Trashnet* y *Trashbox*, al tratar de utilizar las imágenes en el clasificador de redes neuronales no se podría procesar, debido a un error. Tras investigar, se logró averiguar que esto había sido debido a ciertas imágenes defectuosas que se encontraban en el conjunto de datos *Trashbox*. Para conseguir solucionar el problema, se realizó un método de preproceso de los datos que eliminaba las imágenes defectuosas, mencionado en la sección previa. Gracias a este, se logró eliminar las imágenes corrompidas y que el modelo se pudiese ejecutar correctamente.

4.2 Modelos

Una red neuronal es un modelo computacional que se inspira en la estructura y funcionamiento del cerebro humano. Su configuración básica consiste en nodos, también llamados neuronas o unidades, organizados en capas. La red consta de una capa de entrada, que representa las características de entrada del modelo, seguida por capas ocultas que realizan operaciones matemáticas en estas entradas y, finalmente, una capa de salida que produce la respuesta final del modelo. La profundidad de la red se determina por el número de capas ocultas.

El proceso de una red neuronal implica dos fases principales: entrenamiento y prueba. Durante el entrenamiento, la red ajusta los pesos de las conexiones entre nodos para minimizar la discrepancia entre sus predicciones y las salidas reales, utilizando funciones de pérdida y algoritmos de optimización. En la fase de prueba, la red utiliza los patrones aprendidos durante el entrenamiento para realizar predicciones sobre nuevos datos y evaluar su rendimiento en tareas específicas.

En los modelos de las redes neuronales, hay abundantes técnicas distintas para mejorar los resultados de la clasificación, como las técnicas de *fine-tuning*, *transfer-learning*, preprocesado de datos, o cambios de parámetros necesarios, como el número de capas, el número de neuronas, el número de épocas, el optimizador utilizado en el compilador y otros.

La clave para conseguir una buena clasificación de imágenes utilizando redes neuronales, es elegir el modelo adecuado para el problema que se tiene que resolver. Para conseguir esto, es necesario evaluar múltiples modelos distintos y tratar de encontrar los mejores parámetros y las técnicas que más se ajusten a los datos que se van a clasificar.

4.2.1. Modelos Evaluados

En este trabajo se pueden dividir en dos partes los modelos que han sido evaluados a lo largo del tiempo; los modelos que fueron probados con el conjunto de datos inicial de *Trashnet*, y los modelos que se utilizaron tras ampliar el conjunto de datos gracias al dataset de *Trashbox*.

En la preparación de los datos para el entrenamiento de la red neuronal convolucional, se empleó *ImageDataGenerator*, una herramienta eficaz que simplifica la manipulación y procesamiento de imágenes. Mediante esta técnica, se llevó a cabo la normalización de

las imágenes mediante el reescalado, dividiendo cada valor de píxel por 255. Este procedimiento es esencial para garantizar que las imágenes se ubiquen en un rango compatible con el modelo, mejorando así la convergencia durante el entrenamiento. Para las muestras de entrenamiento, además del reescalado, se decidieron utilizar diferentes maneras de realizar aumento de datos. Entre ellas se encuentran; deformaciones de cizallamiento, aumentos de zoom aleatorios, rotaciones aleatorias, traslaciones horizontales y verticales aleatorias en las imágenes; de manera que la red neuronal sea capaz de clasificar las imágenes de manera correcta a pesar de que estas tengan distintos ángulos y posiciones a las muestras originales. Se configuraron los parámetros específicos del generador de datos, como el tamaño deseado de las imágenes (150x150), el tamaño del lote (*batch_size*) establecido en 32, y el modo de clasificación (*sparse*) para manejar eficientemente etiquetas dispersas.

Como el conjunto de datos que se tenían que clasificar eran imágenes, se decidió utilizar una red neuronal convolucional (CNN) para ello. El primer modelo utilizado fue una CNN que iniciaba con una primera capa convolucional, la cual utilizaba 16 filtros con un tamaño de kernel de 3x3 y la función de activación *ReLU*. La subsiguiente capa añadida es la de pooling que reduce la dimensionalidad de la salida. Este proceso se repite otro par de veces, incrementando el número de filtros en cada repetición, a 32 y finalmente a 64, permitiendo de esta manera extraer las características jerárquicas de las imágenes.

A continuación, se añadió una capa *Flatten*, que aplana las salidas del proceso de las capas anteriores para conseguir que la salida tenga una forma unidimensional. Seguidamente, las capas del modelo añadidas, fueron las capas densas, que son dos capas, la primera de ellas, una capa oculta, tiene 64 neuronas con activación *ReLU* y la segunda, una capa de salida, tiene seis neuronas. Esto es debido a que nuestro conjunto de datos tiene seis clases que debemos clasificar. En la función de activación se eligió utilizar $Softmax$, que es el utilizado con más frecuencia para problemas de clasificación multiclase. En total, el modelo cuenta con 1.207.782 parámetros entrenables.

Para realizar la compilación en el modelo, se utilizó la función de pérdida que se describe a continuación, *sparse_categorical_crossentropy*:

$$\mathrm{L}_{SCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(p_{ij}) \tag{4.1}$$

A su vez, se define en la compilación, el optimizador que va a utilizar el modelo, que en este caso, se decidió utilizar el descenso de gradiente estocástico (SGD). Este, ajusta iterativamente los parámetros del modelo para minimizar la función de pérdida, utilizando la siguiente fórmula para definir los parámetros de la iteración t de la siguiente manera:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla L(\theta_t, x, y) \tag{4.2}$$

El método que se utiliza en la métrica de evaluación para el modelo es el de *accuracy* que utiliza la precisión del modelo durante el entrenamiento y la evaluación.

En el proceso de entrenamiento, se decidió utilizar un generador de datos con 50 épocas, utilizando como datos de validación los guardados previamente en *val_generator*, con imágenes distintas a las utilizadas para el entrenamiento. Además, se decidió utilizar la técnica de guardar el mejor modelo durante el entrenamiento, para posteriormente utilizarlo para testear mediante $ModelCheckpoint$.

Con este modelo inicial solamente, se consiguió una precisión en test del 37,9 %, que es un porcentaje muy bajo para el resultado al que se aspira en la clasificación, por lo que se empezaron a considerar otras opciones.

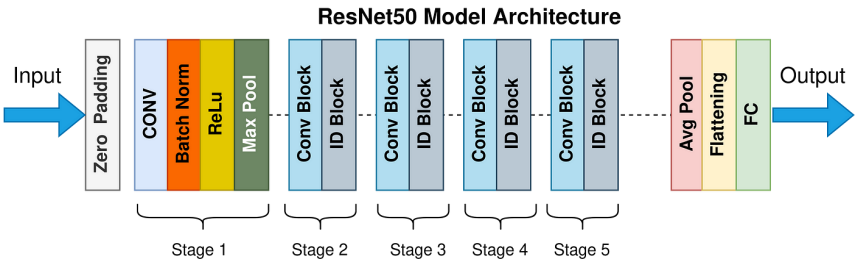

Figura 4.2: Arquitectura de red neuronal *ResNet-50*.

A continuación, debido al resultado previo con baja precisión, se probaron dos modelos, uno de ellos simplemente cambiando algunos parámetros, como el número de neuronas, que pasó de 64 a 128, y también se cambió el optimizador a Adam, que es más ampliamente utilizado y comúnmente genera mejores resultados. Este optimizador se llama *Adaptetive Moment Estimation*(Estimación Adaptativa de Momentos), y este al igual que *RMSprop*, busca mitigar los impactos adversos derivados de la acumulación global de cachés, transformándola mediante la aplicación de un promedio móvil exponencial ponderado, sin embargo, en el caso de Adam utiliza momentum para ello. Su fórmula, siendo t la iteración, g_t el gradiente de la iteración t, β_1 y β_2 coeficientes de decaimiento de los momentos, m_t y v_t los momentos de primer y segundo orden, respectivamente, y ϵ es una constante que evita la división entre 0, es la siguiente:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t, \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2, \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \\ theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon} \cdot \hat{m}_t \end{aligned} \tag{4.3}$$

En este modelo, probado con 10 *epochs*, el resultado obtenido en la precisión en test fue de 60 %, lo cual es una gran mejora respecto al modelo inicial, pero seguía quedándose corto respecto al resultado esperado. En el otro modelo, se decidió añadir una capa de *Dropout*, que desconecta un porcentaje de neuronas que se especifica al llamarlo. En este caso, se puso un porcentaje de 20 %, añadido entre la última capa de pooling y la capa de *Flatten*, lo que mejoró los resultados con una precisión de test de 66 % tras 15 *epochs*.

Posteriormente, buscando otras técnicas para utilizar, se pensó en probar un modelo con transfer learning. Esta técnica consiste en utilizar un modelo entrenado como punto inicial, de modelo base, al que posteriormente se le añaden las capas densas, etc. Para hacer transfer learning se utilizó la red neuronal de *ResNet-50*, la cual tiene 50 capas de profundidad y la estructura que podemos observar en la siguiente Figura 4.2 [Mukherjee, 2022]. Utilizando esta red base, con cuatro capas densas de 512, 256, 128 y 64, y posteriormente una final de seis neuronas igual al número de clases a clasificar, se consiguió un porcentaje de 32,8 %. También se probó a añadir capas de *Dropout* entre las capas densas, lo cual no logró mejorar el porcentaje de precisión en el modelo.

Tras probar esta técnica, se decidió ir por otro camino, pensando que el tamaño de las imágenes era demasiado grande, este decidió cambiarse a un tamaño objetivo de (64x64). Con este nuevo tamaño, se intentaron probar varias de las redes previamente exploradas, pero con esta alteración, y algún que otro retoque. Para empezar se entrenó una red con las tres iteraciones de capas convolucionales seguidas por capas de pooling, aunque en

este caso se utilizaron filtros de 48 en las dos primeras iteraciones y 32 en la última. En el resto, se utilizó las mismas tres capas densas con 128, 64 y seis neuronas, además de la función de pérdida utilizada fue la de entropía cruzada categórica. Fue compilado con el optimizador Adam y se ejecutaron 30 *epochs*. Con estas características se consiguió una precisión en el test de 70 %.

Se modificó este modelo añadiéndole capas de *Batch Normalization* entre las iteraciones de capas convolucionales y capas de pooling, así como entre las capas densas. Esta técnica, que en castellano se podría llamar normalización por lotes, se emplea para acelerar y estabilizar el entrenamiento. La manera en la que consigue hacer esto es normalizando las entradas de cada capa de red, ajustando de esta manera la distribución para conseguir que la media que tenga sea cercana a 0, y la desviación típica sea cercana a 1. Esto se consigue gracias a que utiliza la siguiente fórmula, que dada una entrada x, realiza la normalización, escalado y desplazamiento de esta de una capa específica del entrenamiento. Para ello utiliza los parámetros: γ que se aprende durante el entrenamiento para escalar la salida normalizada, β que se aprende durante el entrenamiento para desplazar la salida normalizada, μ es la media del lote actual, σ^2 es la varianza del lote actual, ϵ es una constante añadida para evitar la división por cero.

$$\textit{Batch Normalization}(x) = \gamma \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{4.4}$$

Además, se decidió añadir un *Learning Rate Scheduler*, que es una técnica que ajusta dinámicamente la tasa de aprendizaje, es decir, el tamaño de los pasos que da el modelo en el entrenamiento, durante el entrenamiento. Esta técnica se utiliza habitualmente, debido a que si se tiene una tasa de aprendizaje muy grande es posible que el modelo se salte información importante, y a su vez, si se tiene una tasa muy pequeña, es posible que se quede atascado en un máximo local. El entrenamiento se realizó con el planificador de *cosine annealing with restarts*. Esta estrategia utiliza la función de coseno para variar con suavidad la tasa de aprendizaje a través del tiempo. Comienza con una tasa mayor (lr_{max}) que disminuye gradualmente hasta una tasa mínima (lr_{min}), mediante la fórmula siguiente:

$$\mathrm{lr} = \frac{\mathrm{lr}_{\max}}{2}\left(1 + \cos\left(\frac{\pi \cdot (x \mod \frac{\mathrm{epochs}}{5})}{\frac{\mathrm{epochs}}{5}}\right)\right) \tag{4.5}$$

En este modelo se consiguió mejorar la precisión previa, logrando resultados en el test de 82,17 % al entrenar el modelo en 50 *epochs*.

Al ver las mejoras que se podían observar tras cambiar el *target size* e incrementar el número de neuronas, se decidió continuar probando la técnica de transfer learning, haciendo pruebas con otros dos modelos similares ya entrenados. Estos fueron *MobileNet* y *MobileNet-v2*. *MobileNet* es una arquitectura de red neuronal optimizada para dispositivos móviles y entornos con recursos limitados, desarrollada por Google. Destaca por su eficiencia en términos de tamaño y velocidad de inferencia. Utiliza capas separables en profundidad para reducir parámetros y operaciones, y ofrece un factor de ancho que ajusta el número de canales para equilibrar la precisión y la eficiencia computacional. Además, incorpora bloques residuales para mejorar el entrenamiento y utiliza capas de clasificación y activación lineal para mantener la simplicidad. En conjunto, *MobileNet* se ha diseñado para ofrecer modelos más ligeros y eficientes, lo que los hace ideales para ejecutarse en dispositivos móviles y otros entornos con restricciones de hardware. Se puede observar su arquitectura en la Figura 4.3 [Ratul et al., 2019]. Con esta arquitectura se consiguieron unos resultados de 82,24 % de precisión en el test con 50 *epochs* de entrenamiento.

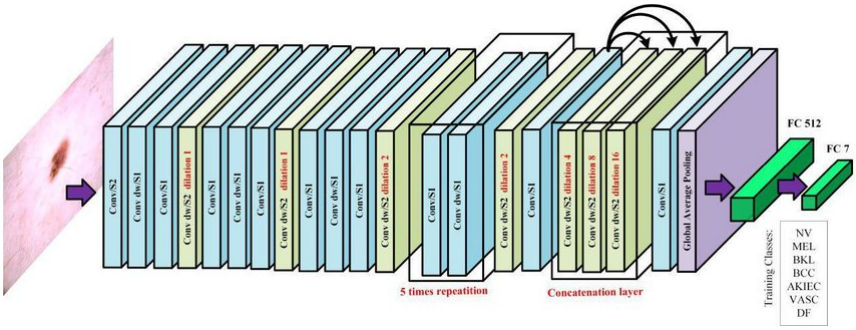

Figura 4.3: Arquitectura de red neuronal *MobileNet*.

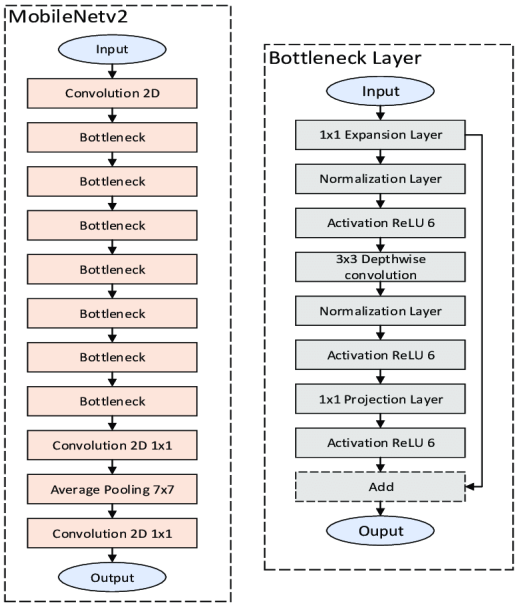

Figura 4.4: Arquitectura de red neuronal *MobileNet-v2*.

MobileNet-v2 es una evolución de la arquitectura *MobileNet* original, introduciendo mejoras significativas en términos de eficiencia y rendimiento. Una de las diferencias clave radica en la introducción de bloques residuales invertidos y expansiones lineales, lo que permite un mejor flujo de información y aprendizaje más profundo en la red. Además, *MobileNet-v2* implementa capas de activación lineales entre las capas convolucionales, lo que contribuye a una mayor no linealidad en el modelo. Estas mejoras ayudan a *MobileNet-v2* a alcanzar un equilibrio aún más efectivo entre la eficiencia computacional y la precisión, convirtiéndolo en una opción aún más avanzada para aplicaciones en dispositivos móviles y otros entornos con recursos limitados. Se puede observar la arquitectura que tiene el modelo en la Figura 4.4 [Tragoudaras et al., 2022]. A pesar de la similaritud con el modelo de *MobileNet*, con esta arquitectura se obtuvieron unos resultados de menor precisión, consiguiendo en el test con 50 *epochs*, 62,93 %.

En un nuevo escenario de evaluación, se incorporó un segundo conjunto de datos al dataset original, expandiendo así la diversidad y complejidad del conjunto de prueba. Con la adición de este segundo dataset, se procedió a poner a prueba los modelos previamente entrenados en un entorno más amplio y representativo. Esta integración de datos adicionales busca mejorar la capacidad de generalización de los modelos, permitiéndoles enfrentar una variedad más extensa de casos y escenarios. En el instante siguiente, se presenta el análisis de desempeño de los modelos ante esta configuración actualizada de datos combinados.

Al unir el nuevo dataset, fue necesario realizar un preprocesado de las imágenes, ya que de otra manera no se lograba ejecutar el modelo correctamente. Fue necesario, para el preproceso, recorrer todas las imágenes que se tenían en el conjunto de datos actual, donde se encontraban las imágenes previas y las añadidas. Al recorrer dichas imágenes se convirtieron las imágenes al tipo RGB y a su vez, se cambiaron de tamaño, al tamaño deseado de (64x64), comprobando que la imagen era correcta mediante un *try-catch* y almacenándola en una lista si ese era el caso, además de añadiendo su etiqueta a otra lista para poder utilizar posteriormente en el método *flow* donde se configura el generador de datos. Finalmente, estas listas se convirtieron en array *numpy* para poder ser utilizadas de parámetros en el método *flow*.

Teniendo en cuenta los resultados obtenidos en los entrenamientos y pruebas con un solo dataset, se decidió comenzar por ejecutar los modelos que habían obtenido los resultados con mejor precisión. Se comenzó por un modelo bastante básico a utilizar como base, con tres capas convolucionales seguidas por capa de *pooling*, con filtros de 48, 48 y 32; seguidamente una capa *Flatten*, con dos capas densas de 128 y 64 neuronas, finalizando con una capa densa de seis capas, como el número de clases. El optimizador utilizado fue *Adam* y la función de pérdida *categorical crossentropy*. Los resultados de este modelo con 30 *epochs* fue de 60,97 % en test.

A continuación, se añadió al modelo básico unas capas de *BatchNormalization* entre las capas para normalizarlas, obteniendo un resultado con 30 *epochs* de una precisión del 55,97 %. También se hicieron pruebas con *Dropout*, entre las capas densas de 128 y 64 neuronas, consiguiendo una precisión del 63,7 %. Se unieron ambas opciones en un mismo modelo, alcanzando en el resultado una precisión de 63,1 %.

Considerando los porcentajes de resultados moderados conseguidos a partir del modelo base utilizado con los conjuntos de datos unidos, se tomó la decisión de alterarlo levemente para obtener unos resultados más adecuados. A este nuevo modelo base, se le añadió una tercera capa densa oculta y se modificaron las neuronas de dichas capas ocultas a 1.024 en el caso de las tres. Adicionalmente, fue sustituido el tamaño de los lotes a 128, se volvieron a añadir capas de *Batch Normalization* entre las capas del modelo y se añadió un planificador de tasa de aprendizaje utilizando el método *ReduceLROnPlateau*, que monitorea la pérdida en el conjunto de validación y reduce la tasa de aprendizaje si no se observa mejora durante un número específico de épocas (*patience*). La reducción se realiza multiplicando la tasa de aprendizaje por un factor específico (*factor*), pero asegurándose de que no caiga por debajo del valor mínimo especificado (*min_lr*). En este caso, los valores utilizados fueron de dos épocas para *patience*, factor de 0,2 y un valor mínimo de factor de aprendizaje *min_lr* de 1e-5. Los resultados fueron similares a los obtenidos previamente, con una precisión de 60,58 %, por lo que se procedió a utilizar otras técnicas empleadas en los modelos con un dataset.

Partiendo de este nuevo modelo *base*, se decidió estudiar el efecto de utilizar el planificador de coseno con reinicios, con el cual se habían obtenido buenos resultados en el caso de un solo conjunto de datos. En este caso, solamente se consiguió una precisión del 66,33 %, que son peores resultados que lo obtenido previamente con un solo dataset, por lo que se tuvo que continuar con distintas estrategias. Sumado a ello, se propuso incorporar a las capas densas del modelo una regularización L2 a los pesos, la cual ayuda a prevenir el sobreajuste mediante la penalización de los valores extremos en los pesos de la red. La fórmula general de la regularización L2; donde λ es el parámetro de regularización, en este caso 0,01, n es el número total de pesos y w_i el peso del índice i; es la siguiente:

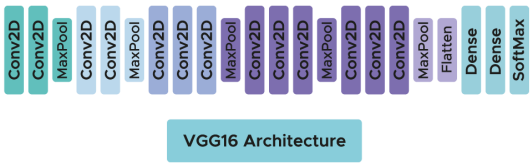

Figura 4.5: Arquitectura de red neuronal *VGG16*.

$$\text{L2 Regularization} = \lambda \sum_{i=1}^{n} w_i^2 \tag{4.6}$$

Con este procedimiento, fue posible conseguir un *accuracy* algo mayor, pero sin llegar al mejor conseguido previamente con un solo dataset, con precisión de 71,55 %.

Se continuó por aplicar el método de *transfer learning*, empezando por probar el modelo de mejores resultados con un solo conjunto de datos, *MobileNet*. Además, también se decidieron hacer pruebas con los otros modelos utilizados, *MobileNet-v2* y *ResNet-50*, y, por otra parte, se empleó el modelo *VGG16*, el cual contiene 16 capas de profundidad y su arquitectura se puede observar en la Figura 4.5 [B., 2022]. De estos modelos se observó que el que mejor resultados se había conseguido fue con el de *MobileNet*, que con el dataset inicial conseguía un mejor resultado, por lo que se probó a añadirle la técnica del *Learning Rate Scheduler*, con la función de *cosine annealing with restarts*, que había logrado buenos resultados previamente. Con esta unión de los modelos se consiguió mejorar la precisión de ambos modelos con un *accuracy* de 84,94 %. En la siguiente tabla 4.1 podemos observar los resultados obtenidos con los diferentes modelos evaluados a lo largo del capítulo, tanto con un conjunto de datos, como con la nueva unión de dos de ellos.

Después de evaluar diversos modelos y entender la importancia de las técnicas de preprocesamiento, nos adentramos ahora en la crucial etapa de seleccionar la solución más adecuada para nuestro problema. En la siguiente sección, detallaremos el proceso de elección del modelo, destacando las razones detrás de la selección final y cómo esta solución aborda de manera efectiva los desafíos identificados en la fase de evaluación de modelos.

4.3 Conclusiones

A continuación, se profundizará sobre la elección final del modelo como cierre del capítulo. Tras comprobar todos los resultados obtenidos, se pueden observar las mejoras que se han conseguido con las diferentes estrategias, así como otras que no han conseguido mejorar un modelo básico. Ciertas técnicas como utilizar una capa de *Dropout*, aumentar el número de neuronas, reducir el tamaño objetivo de las imágenes, regularización L2 o el planificador *cosine annealing with restarts* fueron técnicas que lograron mejorar los resultados obtenidos en modelos previos a esta adición. Mientras que otras estrategias como añadir capas de *BatchNormalization* al modelo entre capas no lograron mejorar los resultados previos a su integración.

En el caso de la táctica de *transfer learning*, se obtuvo una amplia variedad de resultados dependiendo de las arquitecturas escogidas para ello. En el caso de ResNet-50, en lugar de llegar a un nivel de precisión mayor, disminuyó abundantemente esta, llegando

Accuracy en test	SGD	Adam+128 neurons	Dropout	Filter of 48
Trashnet	37,9 %	60 %	66 %	70 %
Trashnet + Trashbox	- %	- %	63,7 %	60,97 %
	BatchNorm	BatchNorm + Dropout	ReduceLROnPlateau	Cosine annealing
Trashnet	- %	- %	- %	- %
Trashnet + Trashbox	55,97 %	63,1 %	60,58 %	66,33 %
	Cosine annealing + BatchNorm	Regularización L2	MobileNet	MobileNet+ Cosine annealing
Trashnet	82,17 %	- %	82,24 %	84,94 %
Trashnet + Trashbox	- %	71,55 %	77,58 %	- %
	MobileNet-v2	ResNet-50	VGG16	
Trashnet	62,93 %	32,8 %	69,88 %	
Trashnet + Trashbox	67,7 %	20,77 %	59,52 %	

Tabla 4.1: Comparación de resultados con *transfer learning*.

al nivel más bajo obtenido en los modelos evaluados. Respecto a otras de las arquitecturas de modelo utilizadas, las de *VGG16* y *MobileNet-v2*, no consiguieron realizar un gran cambio en la precisión, con resultados que a lo mejor disminuían o aumentaban ligeramente la precisión previa a la adición de estos modelos preentrenados. Sin embargo, de todos los modelos de *tranfer learning* evaluados, el que consiguió mejores resultados fue el modelo *MobileNet*, obteniendo una diferencia de un mínimo de 10 % de mejora respecto a los otros modelos preentrenados previamente mencionados.

La última estrategia que se tuvo en cuenta fue la de añadir nuevas imágenes al conjunto de datos mediante la anexión del conjunto de datos *Trashbox* al original. No obstante, luego de estudiar los modelos que se habían analizado anteriormente con el conjunto original con el nuevo conjunto de datos anexados, se puede observar que los resultados obtenidos con el conjunto de datos original conseguía unas precisiones mayores por lo general en los modelos, y sin duda no lograba superar los mejores resultados obtenidos con este. Es muy probable que no se hayan conseguido mejores resultados al añadir nuevas muestras debido a la calidad de las muestras o al desequilibrio creado en las clases al añadir una gran cantidad de imágenes a todas las clases menos a la de *trash*.

Finalmente, el modelo con el que se alcanzaron mejores resultados era un modelo utilizando el conjunto de datos original de *Trashnet*, sin utilizar preprocesado en las imágenes y el cual contenía el siguiente esquema:

- Por medio de *transfer learning*, se utilizó un modelo preentrenado inicial de *MobileNet*, con los pesos de las imágenes de *Imagenet* y descartando la primera capa de 1.000 neuronas de este

- Incorporación de una capa *GlobalAveragePooling*, que se utiliza como capa de reducción de dimensionalidad de la red neuronal transferida.

- Tres capas densas ocultas, de 1.024, 1.024 y 512 neuronas cada una, con función de activación *ReLU*.
- Una capa densa de salida, con seis neuronas, igual al número de clases que se tiene en el conjunto de datos, con la función de activación *Softmax*.
- Se declaran no entrenables las primeras 20 capas.
- Función de perdida *categorical_crossentropy* y optimizador Adam para la compilación, con métrica de precisión.
- Entrenamiento del modelo con 50 *epochs*, con el planificador de coseno con reinicios y un control para guardar el mejor modelo del entrenamiento a utilizar posteriormente.

Con este modelo, el resultado de la precisión alcanzada fue de 84,94 % en el conjunto de imágenes de prueba, con un 99,94 % de precisión en el conjunto de entrenamiento. Por lo tanto, esta es la mayor precisión obtenida en todos los modelos analizados a lo largo del trabajo.

En el próximo capítulo, se comentará la aplicación desarrollada para poder acercar al usuario a la tecnología de las redes neuronales estudiadas en este. A través de ella, el usuario será capaz de solamente con sacar una foto, recibir la información de la clase de residuo al que pertenece la imagen, así como conseguir la información de los contenedores cercanos a los que puede acercarse para reciclar su residuo.

CAPÍTULO 5
Aplicación Desarrollada

Esta aplicación es una manera de poder acercarse al usuario final, sin necesidad de que este tenga amplias competencias digitales para poder utilizarla con facilidad. A día de hoy cada vez se hacen más aplicaciones con una gran magnitud de funciones que pueden llegar a confundir a un usuario inexperto. Además, en el ámbito de la sostenibilidad hay mucha información que puede llegar a saturar a una persona común que no puede dedicar el mismo tiempo que ciertas personas más comprometidas o con menos obligaciones. Es por ello que, a través de esta aplicación, se intenta simplificar la tarea del reciclaje para cualquier persona, de modo que no requiera mucho de su tiempo y sea una tarea sencilla para poder aportar su grano de arena a mejorar el planeta.

En este capítulo, se mostrará el modo en el que ha sido desarrollada la aplicación. A lo largo de este proceso, se ha empleado la plataforma *Flutter*, que permite la creación eficiente de aplicaciones móviles multiplataforma. A lo largo de las siguientes secciones, se presenta en detalle la estructura, funcionalidades y decisiones de diseño detrás de la aplicación, destacando su contribución a los objetivos establecidos en este proyecto de investigación. Exploraremos las tecnologías utilizadas, los desafíos enfrentados durante el desarrollo y las soluciones implementadas.

5.1 Bocetos y Diseño

En esta sección, se comentará el proceso creativo para la realización de la aplicación diseñada de clasificación de imágenes de residuos. Desde la concepción inicial hasta su diseño final, se explorará a fondo la trayectoria completa de ideación y diseño que ha modelado la interfaz de la aplicación. Enfocándose en la importancia de una experiencia de usuario eficiente y amigable, detallamos la evolución de bocetos, resaltando las decisiones de diseño y las iteraciones que han resultado en una interfaz que consideramos ergonómica y visualmente atractiva.

5.1.1. Proceso Creativo e Ideación

Para esta aplicación se pensó en intentar llegar a un público con no necesariamente un amplio conocimiento de reciclaje, que es la razón por la que tendrían la necesidad de utilizar esta aplicación; además, se decidió crear una aplicación intuitiva y sencilla para poder ser de utilidad a personas de cierta edad, las cuales habitualmente tienen menos conocimientos y experiencia utilizando tecnologías actuales.

Como inspiración de las tendencias actuales en las aplicaciones, se tomó como referencia la filosofía del minimalismo. Esto es debido a la decisión de tomar la simplicidad

y la funcionalidad como elementos clave de nuestro diseño. Esta filosofía se ha traducido al diseño de aplicaciones, dando lugar a interfaces limpias, sin elementos superfluos y centradas en la experiencia del usuario. La adopción de paletas de colores reducidas, tipografías sencillas y lineas limpias ha dado lugar a interfaces más accesibles y fáciles de entender [Vernon, 2023]. La prevalencia del minimalismo en el diseño de aplicaciones no solo se debe a su estética atractiva, sino también a su capacidad para mejorar la usabilidad y la eficiencia del usuario.

A medida que miramos hacia el futuro, la tendencia minimalista en el diseño de aplicaciones parece estar lejos de disminuir. Por el contrario, se espera que continúe evolucionando y adaptándose a las demandas cambiantes de los usuarios. La simplificación de las interfaces seguirá siendo crucial para la creación de experiencias de usuario efectivas, especialmente en un mundo digital en constante expansión. La importancia del minimalismo radica en su capacidad para ofrecer claridad visual, facilitar la interacción y proporcionar una experiencia sin distracciones. Así, el minimalismo se perfila como un elemento fundamental en la evolución del diseño de aplicaciones, destacando su versatilidad y su capacidad para proporcionar soluciones efectivas y atractivas para los usuarios contemporáneos.

La decisión de adoptar el minimalismo responde a la necesidad de crear una interfaz que sea fácilmente comprensible para un público diverso, incluyendo aquellos con conocimientos limitados sobre reciclaje y tecnologías actuales. Se busca no solo estética, sino también funcionalidad, priorizando la experiencia del usuario. La idea fue concebir una aplicación con solo unas pocas páginas, cada una dedicada a acciones esenciales para el funcionamiento óptimo de la aplicación. Desde la selección de la imagen hasta la visualización del resultado de clasificación, la aplicación se centra en proporcionar un proceso directo y eficiente. Este enfoque minimalista se ha convertido en la columna vertebral de nuestro diseño, asegurando que la aplicación no solo sea visualmente atractiva, sino también accesible e intuitiva para usuarios de todas las edades y niveles de experiencia tecnológica.

La manera de idearlo fue que solamente tuviese unas pocas páginas con las acciones necesarias para el correcto funcionamiento del planteamiento inicial. De manera que solo se encuentre una sección de páginas para poder tomar o seleccionar la imagen que se quiera clasificar, con su resultado de la clasificación que indique el contenedor más cercano a tu posición actual; un mapa donde aparezcan todos los contenedores que se encuentran en tu alrededor, por si el usuario no necesita saber el tipo de residuo que necesita clasificar sino solo la ubicación del contenedor, y además una página para manejar los permisos de la aplicación y una para almacenar las imágenes ya clasificadas previamente.

5.1.2. Boceto

La idea inicial fue crear una aplicación sencilla, que contuviese todos los elementos necesarios para el correcto funcionamiento del planteamiento inicial. El planteamiento, al comienzo, era que la aplicación pudiese, cuando el usuario le proporcionaba una imagen, tanto a través de la cámara como a través de la galería, que esta la lograse clasificar mediante el modelo de red neuronal elegido en el capítulo anterior. A continuación, debería mostrar tanto el tipo de residuo obtenido mediante la clasificación como la ubicación del contenedor más cercano donde poder reciclar dicho residuo. Además, se ideó añadir acceso al mapa con los diferentes contenedores que se encuentren en la zona cercana a la ubicación del usuario, distinguibles por colores dependiendo del tipo de residuo que se reciclase en ellos.

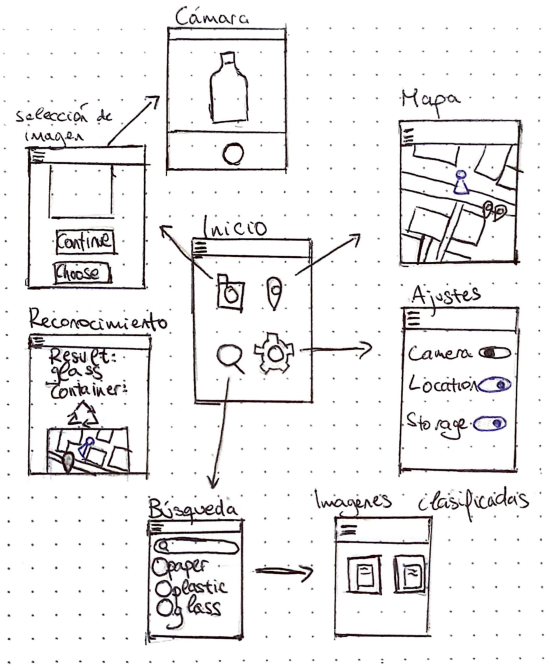

Figura 5.1: Boceto de aplicación.

Para ello, sería necesario una pantalla de inicio al comenzar la app, que pudiese llevar al usuario a las diferentes opciones. Al comienzo se planteó hacer cuatro pantallas accesibles desde el menú inicial; una de la clasificación de imágenes, otra que llevase al mapa donde aparecerían los distintos contenedores, una con una lista de imágenes de referencia de cada tipo de residuo donde se guardarían las imágenes utilizadas para la clasificación y una de ajustes para poder alterar los permisos otorgados.

Al acceder a la pantalla de la clasificación de imágenes se encontraría una imagen, que estaría vacía al inicio, con dos botones. Se podría continuar con la imagen escogida en ese momento a la clasificación o elegir una foto, donde presentaría al usuario dos opciones; la cámara, para poder capturar una foto en el momento, o la galería, para poder escoger una imagen que se encontrase en la galería. Tras escoger la imagen y continuar a la clasificación, aparecería una pantalla donde mostraría tanto el tipo de residuo devuelto por la red neuronal, como el tipo de contenedor donde se reciclaría ese residuo, además de un pequeño mapa que mostraría la ubicación del contenedor a menor distancia de la posición actual del usuario.

Otra de las opciones desde el inicio sería acceder al mapa donde se podrían observar todos los contenedores de la zona. Los contenedores estarían clasificados por colores según el tipo de residuo, acorde con los colores de los contenedores físicos, para mantener la homogeneidad de la información, de manera que el usuario sepa el contenedor a donde debería de acudir para reciclar el residuo que necesitase. Esta página sería de utilidad para todos los residuos que el usuario ya tendría claridad en el tipo al que pertenecen, ya sea por conocimiento adquirido tras la clasificación previamente de la aplicación o por conocimiento conocido por su parte.

Finalmente, se encuentran una página de ajustes donde se podrían modificar los permisos otorgados relevantes para la aplicación, de permisos de cámara, ubicación y almacenamiento. Además, se podría encontrar, a través del símbolo de búsqueda, una página donde se encontrarían las imágenes de cada categoría previamente clasificadas. Todo esto se puede observar en el boceto de la figura 5.1. Además, todas las pantallas tendrían un menú lateral para poder acceder con facilidad a cualquiera de las páginas mencionadas.

A continuación, se observará como se llevó a cabo la creación del diseño basándose en el boceto previamente mencionado, detallando más a fondo la interconexión de las pantallas y las funcionalidades que tendría cada una de ellas.

5.1.3. Diseño

Para el diseño de la aplicación previa a la implementación se decidió utilizar la herramienta de Moqups[1], que permite fácilmente crear un modelo para aplicaciones móviles, y contiene herramientas adecuadas para poder agregar iconos, botones y demás al modelo. Al comienzo, se tomó la decisión de crear un modelo para la correcta visualización de la aplicación utilizando el boceto explicado previamente como punto de partida.

Al igual que en el caso del boceto, este modelo se podría dividir en cuatro secciones partiendo del menú de inicio. Siguiendo la ideación original, el diseño del modelo ha seguido la concepción de tener un diseño sencillo e intuitivo, sin abarrotar las páginas de características que no sean realmente necesarias para su correcto funcionamiento. Una de las técnicas que se utilizó para no confundir al usuario y mantener la simplicidad fue evitar utilizar colores muy vivos y llamativos, tratando de sustituirlos en lo posible por colores pasteles o en escala de grises, además de no juntar demasiados colores en el mismo espacio.

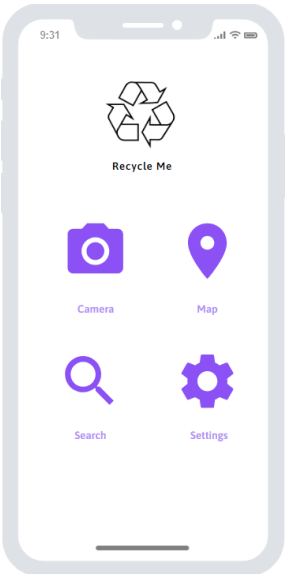

Figura 5.2: Pantalla de inicio del modelo.

El menú de inicio contaría con un icono con un logo intuitivo para cada uno de ellos que funcionaría como botón a la página indicada. Estos serían, el icono de cámara donde se accedería a la clasificación de imágenes, el icono de ubicación donde se accedería al mapa, el icono de búsqueda donde se accedería a las imágenes ya clasificadas y el icono de ajustes donde se podrían modificar los permisos otorgados. En la pantalla inicial, para poder mantener la visión de minimalismo y simplicidad, se decidió solamente añadir los iconos mencionados que llevasen al usuario a la pantalla elegida, además de un icono de reciclaje como cabezal de la página a manera de *logo* de la aplicación, mostrando la función de reciclaje para la que está diseñada la aplicación. Estos iconos se mostrarían solamente de color morado, o en escala de grises, para no saturar al usuario. La página de inicio del modelo se puede observar en la figura 5.2.

[1] moqups.com

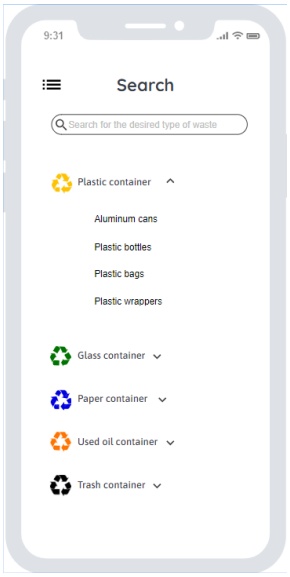

Figura 5.3: Pantalla de búsqueda.

En el modelo, la pantalla de búsqueda está compuesta por una barra de búsqueda para buscar el tipo o subtipo de residuo que se querría verificar, además de una lista de los elementos que coincidiesen con la búsqueda actual. La manera en la que estaría formada la lista, sería con una categoría para cada tipo de residuo dependiendo de los contenedores donde se deberían reciclar, con un icono de reciclaje con el color del contenedor correspondiente, que contendrían una sublista que se podría extender mediante una flecha y listaría las subcategorías con posibilidad de reciclaje en ese contenedor. Al hacer clic en una categoría o subcategoría, se podrían encontrar imágenes de los residuos pertenecientes a esta. La página de búsqueda del modelo se puede observar en la figura 5.3.

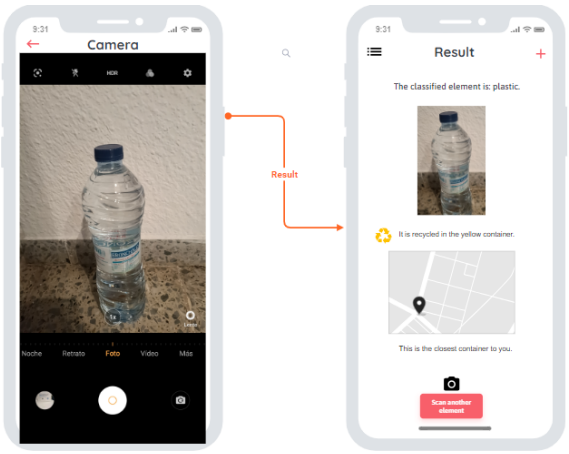

Figura 5.4: Pantallas de clasificación.

Desde el icono de cámara de la pantalla inicial se llega al conjunto de páginas donde se realiza la clasificación. Primero se encuentra una página de cámara, donde se permite sacar una foto del residuo que se quiere elegir para ser clasificado, o también se tiene la posibilidad de elegir una imagen de la galería del usuario, confirmando que la imagen es la que se desea clasificar, con posibilidad de elegir otra, por no tener suficiente calidad de imagen o algún otro problema posible.

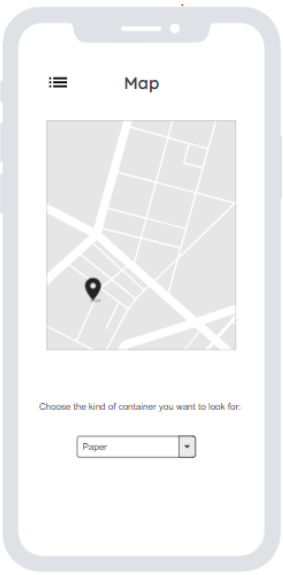

Figura 5.5: Pantalla del mapa.

A continuación, se dirige a otra página donde devuelve el resultado de la clasificación. En esta se puede ver la imagen que hemos utilizado para la clasificación, el tipo de residuo que se ha obtenido mediante la clasificación al que pertenece el residuo de la imagen, el color del contenedor donde se recicla este tipo de residuo mediante una imagen del símbolo de reciclaje de dicho color, así como un pequeño mapa donde se muestra el contenedor más cercano a la ubicación donde se encuentra el usuario. Finalmente, en la parte baja de la pantalla se encuentra un botón para poder clasificar otro residuo distinto. Este conjunto de pantallas se puede observar en la figura 5.4.

La tercera pantalla accesible desde el inicio sería la del mapa, donde se podrían ver los contenedores de los alrededores de la ubicación del usuario, para cuando este sepa de antemano el tipo de residuo. En ella se contiene el mapa donde con marcadores aparecen los contenedores del tipo de residuo elegido en el cuadro combinado que se encuentra debajo de este, de manera que solo aparezcan los contenedores que se quieren buscar y no se encuentren todos juntos, confundiendo de esa manera al usuario. Esta pantalla se puede observar en la figura 5.5.

La última pantalla sería la de los ajustes, en ella el usuario sería capaz de alterar los permisos otorgados, dando y quitando los permisos según lo deseado. La pantalla consiste de tres botones interruptores asociados a cada uno de los tres permisos necesarios para las diferentes funciones de la aplicación. Estos permisos serían los permisos de cámara y galería, necesarios para poder tomar la foto del residuo que se querría clasificar, y los permisos de localización, para poder utilizar en el mapa la ubicación actual del usuario correctamente. A pesar de tener la capacidad de activar y desactivar los permisos desde la pantalla de ajustes, se seguirán solicitando al acceder a pantallas en donde sean requeridas. Esta pantalla se puede ver en la figura 5.6.

Finalmente, en todas estas pantallas se encuentra la posibilidad de acceder el menú lateral, que permite al usuario navegar entre ellas sin la necesidad de retornar a la pantalla de inicio cada vez que quiera cambiar entre dos páginas distintas. Este menú lateral consiste de un icono junto a su nombre para cada una de las pantallas explicadas previamente, que al presionar dirigen al usuario a la pantalla indicada. La pantalla en la que se encuentra el usuario actualmente se muestra de un color distinto, para permitir distinguirla del resto. Se puede observar en la siguiente figura 5.7.

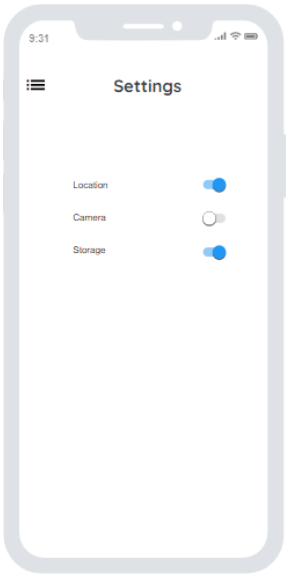

Figura 5.6: Pantalla de los ajustes.

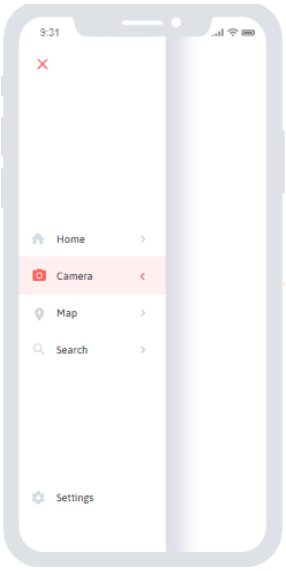

Figura 5.7: Menú lateral.

De esta manera conseguimos que todas las pantallas principales estén conectadas entre sí, simplificando la experiencia del usuario. Todas las páginas son accesibles desde cualquiera del resto en no más que un par de pasos. Es posible lograr esto, debido a que todas las páginas principales tienen una conexión a ellas, tanto en la página inicial mediante los botones que las identifican como en el menú lateral. Esta conexión entre las páginas que podría visitar el usuario se muestra por completo en el diagrama de flujo 5.8.

Posteriormente, en la implementación, se decidió realizar un cambio en el diseño. Se consideró que la pantalla de búsqueda no tenía realmente una función muy útil, y para lo que se quería conseguir con esta, que era ayudar al usuario a distinguir mejor los residuos de cada contenedor sin necesidad de tener que realizar el proceso de clasificación cada vez en caso de que no lo viesen necesario, se decidió sustituir esta pantalla por una de ayuda. En esta página, se encontraría un cuadro combinado donde se podría elegir el tipo de residuo de los contenedores, que dependiendo el tipo de residuo elegido mostraría el color del contenedor correspondiente, además de los residuos que se reciclarían en este.

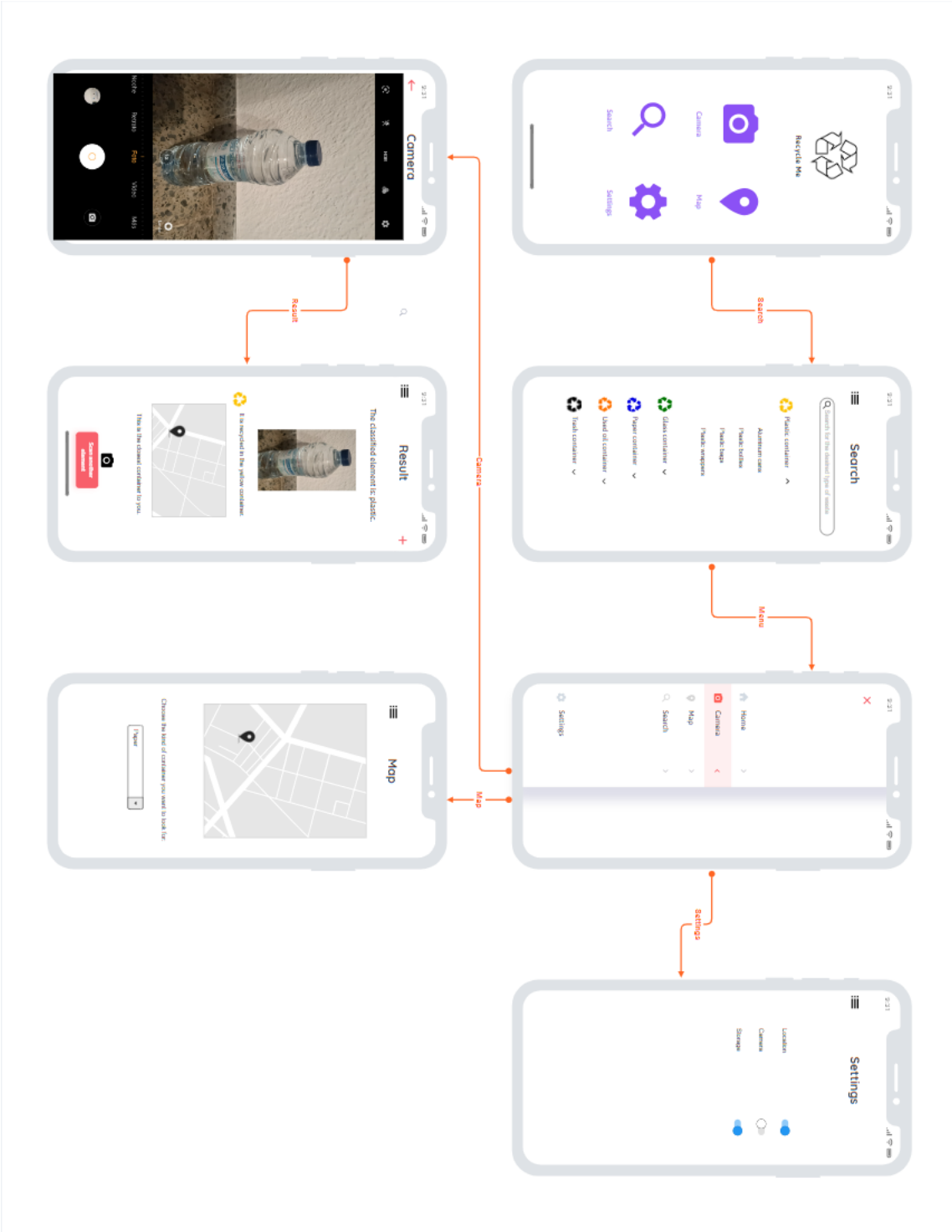

Figura 5.8: Diagrama de flujo del diseño.

5.2 Implementación

En esta sección del capítulo, se presenta el proceso de implementación del diseño de nuestra aplicación en *Flutter* utilizando el lenguaje *Dart*[2]. Después de haber realizado una exhaustiva fase de diseño y planificación, se transformarán las ideas iniciales en una aplicación funcional y visualmente atractiva para los usuarios. *Flutter*, junto con *Dart*, nos brinda las herramientas necesarias para lograr este objetivo de manera eficiente y efectiva. Se explorarán los aspectos prácticos de la implementación en *Flutter*, desde la configuración del entorno de desarrollo hasta la construcción de componentes de interfaz de usuario y la integración de funcionalidades clave. Además, se examinará cómo *Dart* simplifica la creación de aplicaciones móviles mediante su sintaxis clara y características modernas.

Flutter y *Dart* son dos tecnologías esenciales en el desarrollo moderno de aplicaciones multiplataforma. *Flutter*, un *framework* de código abierto desarrollado por *Google*, ofrece una solución eficiente para crear interfaces de usuario nativas en iOS, *Android*, web y escritorio desde un único código base.

Recientemente, ha experimentado un crecimiento significativo en la comunidad de desarrollo debido a su capacidad para crear aplicaciones multiplataforma con una sola base de código. Su enfoque en la productividad, rendimiento y diseño atractivo ha atraído a un número creciente de desarrolladores y empresas que buscan soluciones eficientes para el desarrollo de aplicaciones móviles, por lo que elegir *Flutter* es la decisión adecuada. En el gráfico que se puede observar en la figura 5.9 [Statista, 2022], se percibe claramente como ha aumentado la tendencia de los desarrolladores a utilizar *Flutter*, mientras que otras como *React* o *Cordova* han disminuido.

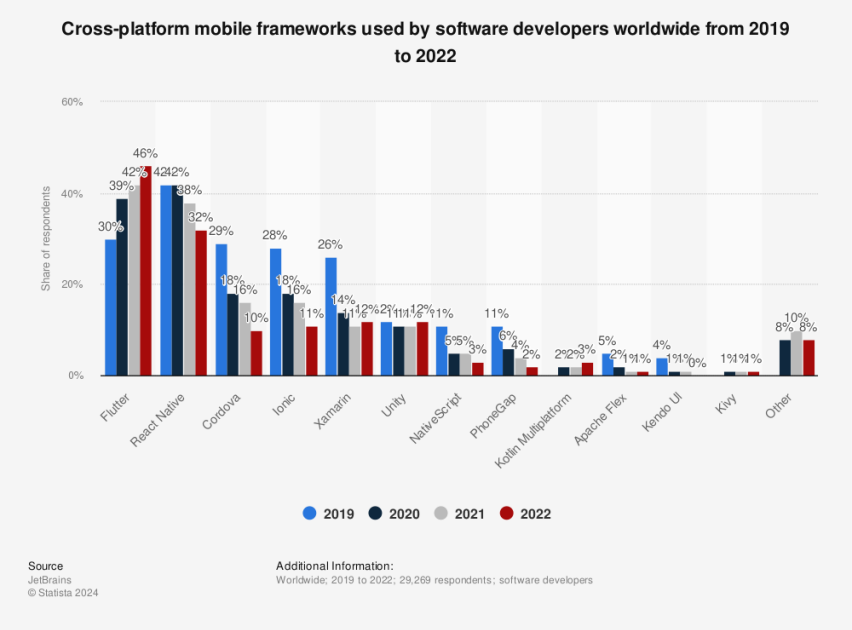

Figura 5.9: Frameworks móviles multiplataforma utilizados por desarrolladores de software en todo el mundo desde 2019 hasta 2022.

Flutter destaca por su enfoque en el desarrollo multiplataforma, permitiendo a los desarrolladores escribir una vez y ejecutar en múltiples plataformas. Además, ofrece una amplia gama de widgets personalizables y predefinidos que facilitan la creación de interfaces de usuario atractivas y coherentes en todas las plataformas. Con un rendimiento su-

[2] dart.dev

perior respaldado por el motor gráfico *Skia*[3], *Flutter* garantiza una experiencia de usuario fluida y receptiva. La función de *Hot Reload* agiliza el proceso de desarrollo al permitir a los desarrolladores ver los cambios en tiempo real, mientras que la comunidad activa y los abundantes recursos proporcionan un sólido soporte para los desarrolladores.

Por otro lado, *Dart*, el lenguaje de programación utilizado por *Flutter*, se destaca por su sintaxis clara y concisa, lo que facilita la escritura, lectura y mantenimiento del código. Con el tipado estático opcional, *Dart* ofrece una mayor seguridad y claridad en el código sin imponer restricciones innecesarias. Además, su eficiencia en la ejecución, ya sea compilando a código nativo o a *JavaScript*, lo hace adecuado para una variedad de plataformas y escenarios de ejecución. *Dart* sigue un enfoque orientado a objetos que facilita la creación de aplicaciones complejas, y ofrece características integradas para trabajar de manera eficiente con operaciones asíncronas, como *futures* y *streams*.

Para la elección del IDE se decidió utilizar *Android Studio*[4]. Un IDE (Entorno de Desarrollo Integrado) es una herramienta de software que proporciona un conjunto completo de funcionalidades para desarrollar, depurar y desplegar aplicaciones de software. Este, en particular, es un IDE específicamente diseñado para el desarrollo de aplicaciones *Android*. Ofrece una gama de ventajas que lo convierten en la opción preferida para los desarrolladores de *Android*.

Android Studio destaca por su interfaz de usuario amigable, que facilita la navegación y el uso de sus diversas herramientas sin requerir una curva de aprendizaje pronunciada. Además, proporciona herramientas de depuración avanzadas que permiten a los desarrolladores identificar y corregir errores de manera eficiente, agilizando el proceso de desarrollo.

Una de las ventajas clave de *Android Studio* es su integración con emuladores de dispositivos virtuales, lo que permite probar sus aplicaciones en una amplia gama de dispositivos *Android* simulados. Esto simplifica la detección de problemas de compatibilidad y garantiza una experiencia de usuario consistente en diferentes dispositivos.

Además, *Android Studio* se actualiza regularmente para ofrecer soporte para las últimas tecnologías y características de *Android*. Esto permite a los desarrolladores aprovechar al máximo las capacidades de la plataforma y mantenerse al día con las tendencias emergentes en el desarrollo de aplicaciones móviles.

Por último, *Android Studio* cuenta con una amplia comunidad de desarrolladores y una extensa documentación en línea, lo que facilita la resolución de problemas y la obtención de ayuda cuando sea necesario. Esta comunidad activa proporciona un entorno de colaboración que enriquece la experiencia de desarrollo y fomenta el intercambio de conocimientos entre sus miembros. En resumen, *Android Studio* ofrece un entorno de desarrollo completo y eficiente que satisface las necesidades de los desarrolladores de *Android*, permitiéndo crear aplicaciones de alta calidad de manera efectiva.

En *Android Studio* se decidieron organizar los archivos por carpetas, de manera que el código fuese más comprensible para entender. Para ello, se crearon varias carpetas con usos distintos en cada una de ellas. Se creó una carpeta de imágenes, donde se guardarían todas las imágenes necesarias para utilizar en la aplicación; se creó una de páginas, donde se guardarían los archivos para cada una de las pantallas de la aplicación. Además, se creó una para guardar los archivos que se usarían para utilizar *Flutter Bloc*. Esta es una herramienta de gestión de estado para *Flutter* respaldada por los desarrolladores de *Google*, que facilita la gestión centralizada del estado y el acceso a los datos en los proyectos.

[3] skia.org
[4] developer.android.com

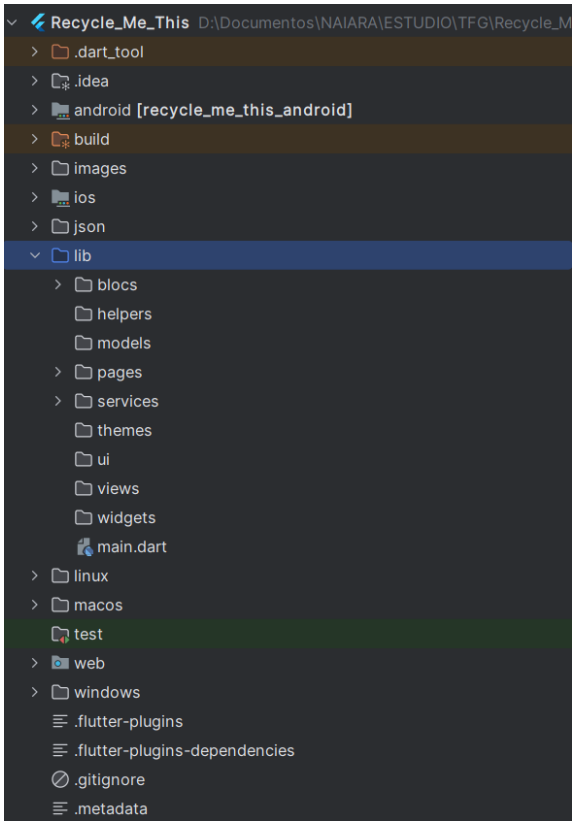

Figura 5.10: Estructura de la aplicación.

El archivo inicial al que se llama al ejecutar la aplicación, es *main.dart*. En este se declaran mediante el método *MaterialApp* todas las páginas que se van a utilizar en la aplicación con su respectiva ruta, comenzando con la ruta inicial '/' que lleva al usuario inicialmente a la página de inicio llamada 'Home'. Además, se declaran el resto de rutas para poder navegar entre ellas cuando sea necesario. Para poder manejar los permisos mediante *Flutter Bloc* como mencionado previamente, es necesario envolver este método con otro método *MultiBlocProvider*, que permite, además de proporcionarle como hijas las rutas de las páginas para la visualización de estas, proporcionarle como proveedores los blocs para poder manejar en segundo plano ciertas funciones como los permisos de ubicación.

Tanto en la carpeta de páginas como en la de blocs se ha creado un archivo para poder exportar todos los otros archivos de la carpeta, de manera que al necesitar llamar a cualquier número de páginas o de blocs solo sea necesario realizar una sola importación en el archivo. Esto se ha logrado con tan solo añadiendo cada página que se quería tener la posibilidad de utilizar en el archivo, poniendo *export* y a continuación añadiendo la ruta que tiene el archivo de la siguiente manera, 'package:nombre-de-la-aplicacion/nombre-de-la-carpeta/nombre-del-archivo'.

Para las páginas que se encuentran en el código, se ha decidido seguir en lo posible el modelo realizado en la etapa de diseño. Para ello, cuenta con, un archivo *home.dart*, que contiene un icono para cada función distinta que dirige a la página necesaria; *camera.dart*, que muestra la pantalla con la imagen elegida actualmente que permite, sustituirla por otra o pasar a realizar la clasificación; *scanning-result.dart*, que muestra el resultado del tipo de residuo que se ha decidido clasificar, así como un mapa donde se muestra el contenedor más cercano de dicho tipo; *help.dart*, que muestra información sobre cada tipo de contenedor; *loading.dart*, que se utiliza como puente para revisar si los permisos de ubicación están otorgados para directamente mostrar el mapa o mostrar una página

intermedia; *location-permissions.dart*, que muestra un mensaje mientras el usuario otorga los permisos necesarios antes de abrir el mapa; *mapa_prueba.dart*, que muestra los contenedores de alrededor de la ubicación del usuario filtrados dependiendo de su tipo, y *settings.dart*, que permite al usuario alterar los permisos otorgados. A continuación, se explicará más sobre cada una de estas páginas y sus funcionalidades.

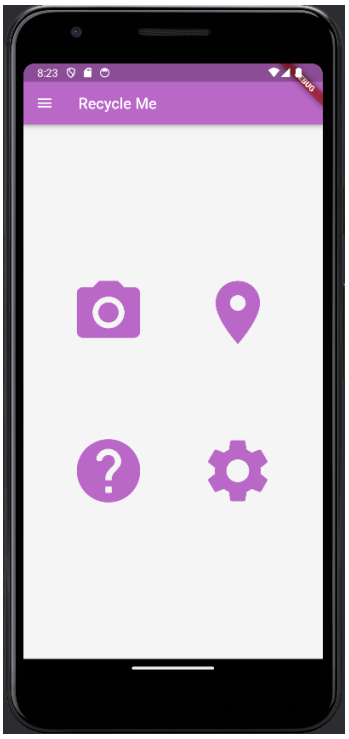

Figura 5.11: Pantalla de inicio.

En todas las pantallas, se ha utilizado el widget *Scaffold*, que en español significa andamio, debido a que está estructurado en varias partes. En la aplicación, todas las pantallas principales se han estructurado de la siguiente manera utilizando ese widget; con una cabecera de tipo *AppBar* que muestra el título de la pantalla con su color correspondiente, un *Drawer* que hace que aparezca un menú lateral con las pantallas por las que podemos navegar y un *Body*, que muestra el cuerpo de la pantalla que varía según la pantalla actual.

Para mantener el estilo minimalista y claro de la idealización previa de la aplicación, se decidió elegir unos colores menos llamativos y más claros para las diferentes pantallas, tratando de evitar los altos contrastes que puedan confundir al usuario. Para ello, en las cabeceras, menús y botones de cada pantalla se mantiene una coherencia de un solo color en cada pantalla, con una gama de colores pastel para estos. Además del color principal de cabecera de cada pantalla, solamente se utiliza la escala de grises, siempre tendiendo a utilizar grises, ya sean claros u oscuros en lugar de colores completamente blancos o negros.

En la pantalla de inicio, se tomó la decisión de alterar ligeramente el diseño y mostrar únicamente los cuatro iconos sin añadir el texto de lo que hace cada uno, debido a la facilidad de intuición que tienen los iconos. Estos iconos son a su vez botones que te redirigen a la pantalla adecuada, utilizando el método *Navigator.pushNamed*. Lo que hace este método es coger el contexto actual de la aplicación que se pasa cada vez que se vuelve a ejecutar el método constructor y le pasa la nueva ruta a elegir, dependiendo en el botón que se pulsa. Se realiza el mismo procedimiento en el menú lateral de la pantalla, excepto que al pulsar el botón de la pantalla actual, en este caso la de inicio, en lugar del método

previamente mencionado se utiliza el método *Navigator.pop*, que solamente cierra el menú lateral y devuelve el estado al de la pantalla de inicio. Se puede observar la pantalla de inicio en la figura 5.11

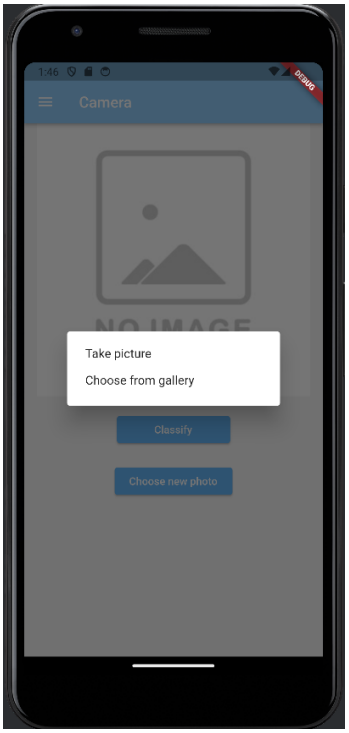

Figura 5.12: Pantalla de selección de imagen.

Mediante el botón de la pantalla de inicio con el icono de una cámara, se dirigiría a la pantalla de cámara, donde podríamos elegir la imagen que se querría clasificar. En esta, se encontraría una pequeña vista previa de la imagen elegida y se encontrarían dos botones con las acciones posibles a tomar. El primero de ellos se presionaría tras haber decidido que la imagen elegida es la adecuada para clasificar y dirigiría al usuario a la pantalla donde el residuo habría sido clasificado; el segundo, permitiría al usuario escoger una nueva imagen, mostrando una pequeña caja de diálogo con dos opciones, con posibilidad de sacar la foto en el momento o escogerla desde la galería. Para poder utilizar la cámara o la galería del dispositivo se ha utilizado el paquete *image_picker*. Usando este paquete, se ha podido al presionar el botón tanto de cámara como de galería acceder a un método que espera a que el usuario acepte los permisos necesarios, y cuando los acepte procede a abrir la cámara o la galería según le corresponda. Tras elegir la imagen de la fuente, se archiva para poder enviarla posteriormente a la clasificación en caso de continuar con el otro botón de clasificar. Se puede observar la pantalla de cámara en la figura 5.12.

Al pulsar el botón de continuar y realizar la clasificación de fondo, se muestra la pantalla con el resultado obtenido para la imagen escogida. En esta pantalla se muestra tanto el tipo de residuo que se debe clasificar como el color del contenedor en donde se reciclan ese tipo de residuos. Esto se consigue a través de la llamada a la API de clasificación mediante el paquete de *Flutter http*. En el se consigue el resultado de la clasificación de la imagen transferida y se almacena en la variable que muestra el tipo de residuo y el tipo de contenedor la de la respuesta recibida. Además, en la parte baja de la pantalla se muestra un mapa con los contenedores que se encuentran en la zona cercana al usuario, que pertenecen al mismo tipo de residuo que el necesario para el resultado obtenido. Esta pantalla se puede observar en la figura 5.13.

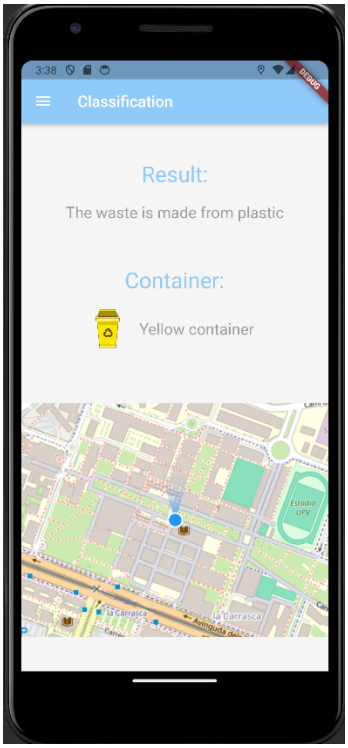

Figura 5.13: Pantalla de resultado de la imagen clasificada.

Otra de las pantallas accesibles desde la pantalla de inicio a través del botón con icono de signo de interrogación es la de ayuda de clasificación de residuos. En esta, además de los elementos habituales de cabecera y menú lateral, en el cuerpo de la pantalla encontramos cuatro elementos: un cuadro combinado con la lista de varios tipos de contenedores distintos que se pueden distinguir en la aplicación, que al cambiarse muestran diferente información en los otros tres elementos; una imagen de un contenedor, con el color correspondiente al del contenedor del residuo elegido en el cuadro combinado; un pequeño texto descriptivo sobre el tipo de residuo actual, y además, una lista realizada con *ListView.builder* que muestra las diferentes categorías de residuos que se reciclan en el contenedor con un icono al lateral de manera que se muestra tanto visualmente como textualmente. La pantalla de ayuda se puede ver en la figura 5.14.

La siguiente pantalla con posibilidad de accesibilidad desde el menú lateral o la pantalla inicial es el del mapa. Antes de acceder a esta, es necesario que el usuario tenga la ubicación del dispositivo activa, además de permitir a la aplicación la posibilidad de acceder a su ubicación. Es por ello, que esta pantalla se podría dividir en tres pantallas, y en tres archivos distintos. En el primero de ellos, *loading.dart* se comprueba si la ubicación del dispositivo está activada y el permiso otorgado de antemano. En el caso de que una de estas dos no se cumpla, se dirigiría al archivo *location-permissions.dart*, que comprobaría cuál de los dos casos previos es el que no se cumple. En el caso de que la condición que no se cumplió fuese que la ubicación del dispositivo no estuviese activada, aparecería un mensaje diciendo que es necesario activar la ubicación. En caso de que la condición sin cumplir fuese que la aplicación no tuviese aún permisos de ubicación, aparecería otra pantalla para solicitar permisos con un botón donde se pedirían los permisos a través del proceso de fondo de *Flutter Bloc*.

En el caso de que ambas condiciones se cumpliesen, dirigiría al usuario a la pantalla del mapa en el archivo *mapa_prueba.dart*, donde se podría observar una pantalla con la cabecera y el menú lateral, al igual que el resto, y en el cuerpo se encontraría el siguiente par de elementos. Se observaría un cuadro combinado donde se podría escoger el tipo

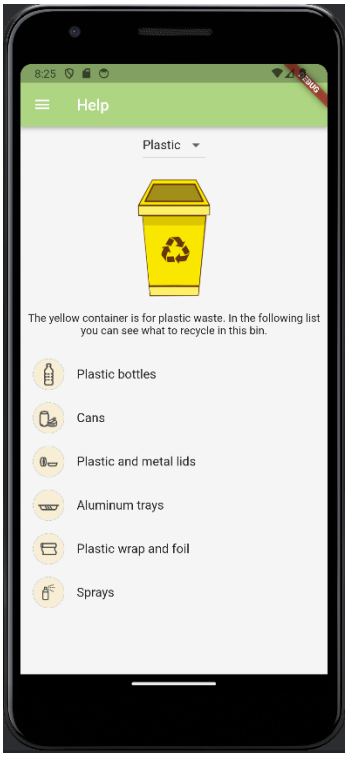

Figura 5.14: Pantalla de ayuda.

de contenedor que se querría ver, así como el mapa, con un indicador de la ubicación del usuario actual y marcadores para los contenedores de alrededor. El mapa se muestra con el paquete de *flutter_map*, que contiene en sus hijos el *widget CurrentLocationLayer*, como capa para mostrar la ubicación actual del usuario en el mapa con un marcador que seguirá al usuario según este se mueva. Otro de los hijos del mapa sería la lista de marcadores de los contenedores correspondientes, que se conseguiría creando una lista de contenedores para cada tipo distinto y decidiendo cuál utilizar dependiendo el valor elegido en el cuadro combinado mediante un método que se llamaría en la capa de los marcadores *Marker*. Se puede observar la pantalla en la siguiente figura 5.15.

Para lograr hacer las diferentes listas de contenedores se leyeron los dos archivos *json*, uno con los contenedores de plástico, papel, orgánico y restos, con el otro siendo solamente de contenedores de vidrio, de manera asíncrona. En ese método se recorrió la información, comprobando que las coordenadas del contendor actual no fuesen nulas, y posteriormente, se almacenaron las coordenadas de cada tipo de contenedor en un *Marker* en su respectiva lista a utilizar posteriormente. La manera de conseguir los datos de los contenedores en el archivo *json*, se especifica con mayor profundidad en la sección 5.4.

En la pantalla de ajustes, el propósito es que el usuario pudiese alterar los permisos otorgados para las diferentes acciones de la aplicación que los necesitan. En la aplicación solo son necesarios tres permisos: los permisos de localización para utilizar en los mapas, los permisos de cámara para sacar fotos de los residuos a clasificar o los permisos de galería para elegir las imágenes a clasificar desde la galería. En la página, se encuentran tres botones de tipo interruptor para cada uno de los permisos necesarios para las diferentes acciones, con el nombre de su respectivo permiso a su lado. Al modificar los valores de este interruptor, sus permisos se activan o desactivan. Esto se consigue con el paquete de *permission_handler*, que permite pedir al usuario los permisos. En el caso de querer desactivarlos, no es posible hacerlo utilizando directamente ese paquete para ello y redirige al usuario a la aplicación de ajustes de su teléfono, donde permite desactivar los permisos

Figura 5.15: Pantalla de mapa.

de la aplicación. El proceso de otorgar permisos se realiza de fondo utilizando *Flutter bloc*. Se puede observar la pantalla de ajustes en la figura 5.16.

El menú lateral se encuentra en todas las pantallas para poder acceder al resto de pantallas sin necesidad de volver a la página de inicio para ello. Se puede acceder a este a través del botón lateral de tres líneas, que se encuentra en el código en el elemento previamente mencionado de *Drawer*. Dentro de este se encuentra el *widget ListView*, que muestra los elementos a mostrar a modo de lista. Esta contiene una cabecera del mismo color que la cabecera de la página actual donde se encuentra el usuario con el título de *Menu*, utilizando el *widget DrawerHeader* para ello.

A continuación, se encuentran los elementos de la lista *ListTile* de los mismos cuatro elementos que se encontraban en la página inicial, mostrando un icono y un nombre que los reconozca con links a sus pantallas respectivas, que dirigen al usuario a la pantalla adecuada. La pantalla en la que se encuentra el usuario actualmente, se marca de color azul, para distinguirla del resto. Al presionar el botón de esta en lugar de editar la ruta para volver a la misma página, se utiliza el método *Navigator.pop*, que simplemente elimina de la pila de páginas la pestaña del menú y muestra de nuevo la pantalla en la que se encontraba. Para no acumular infinitas páginas en la pila al navegar en la aplicación, se decidió utilizar el método *Navigator.pushReplacementNamed*, que funciona similar al método utilizado en la página inicial, pero en lugar de simplemente apilar la página nueva a la que se va a navegar, elimina la página en la que estaba el usuario de la pila antes de añadir la siguiente página, manteniendo solamente la página de inicio y la página actual en la pila. Se puede observar la pantalla del menú en la figura 5.17.

Para poder adquirir los permisos necesarios de ubicación se utilizó *Flutter Bloc*, separando sus funciones en tres archivos: *gps_bloc*, *gps_event* y *gps_state*. Estos se guardan en una carpeta distinta a la de las páginas, ya que estos se procesan en el fondo y se utilizan en diferentes páginas. Al igual que con las páginas, tienen un archivo *bloc*, que exporta los tres archivos, facilitando de esta manera su posible importación en las páginas sin

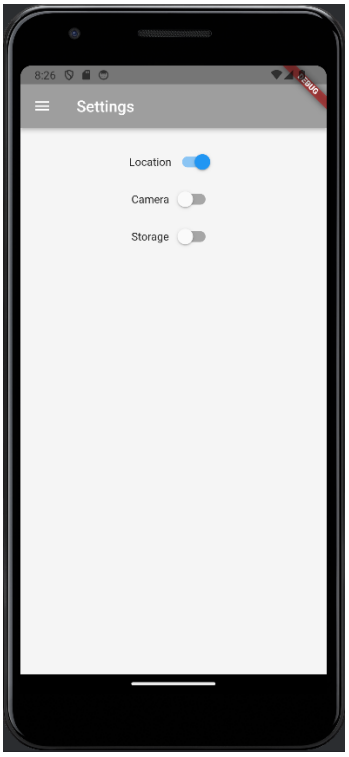

Figura 5.16: Pantalla de ajustes.

tener que llamarlos por separado. Estos archivos forman parte de un patrón de arquitectura de *Flutter* conocido como *BLoC* (*Business Logic Component*). El objetivo principal de este patrón es separar la lógica de negocio de la interfaz de usuario, lo que facilita la prueba, la reutilización y el mantenimiento del código.

A continuación, se profundizará en las tareas que desempeña cada uno de estos archivos. En el archivo *gps_bloc* se encuentra la clase *GpsBloc*, la cual define el estado inicial y cómo reacciona a los eventos. En el estado inicial, tanto la variable que comprueba si la ubicación del dispositivo como la variable que comprueba si el usuario ha otorgado los permisos de ubicación a la aplicación se establecen como falsas. Además, es aquí donde se reacciona a los eventos, alterando el valor de las variables mencionadas previamente al ser activada la ubicación u otorgados los permisos. Contiene métodos para inicializar (*_init*()), verificar el estado del GPS (*_checkGpsStatus*()), verificar los permisos (*_isPermissionGranted*()), y solicitar acceso al GPS (*askGpsAccess*()). También cancela la suscripción a los servicios de GPS en el momento de cerrar (*close*()).

El archivo *gps_event* define los eventos que puede manejar *GpsBloc*. En este caso, solo se decidió crear un evento llamado *GpsAndPermissionEvent* que contiene información sobre si el GPS está habilitado y si se otorgaron los permisos necesarios.

Finalmente, el archivo *gps_state* define el estado que puede tener el *GpsBloc*. En este caso, el estado consiste en dos booleanos: *isGpsEnabled* que comprueba si el GPS está habilitado y *isGpsPermissionGranted* que comprueba si los permisos de ubicación están concedidos. Estas son las variables que se comprueban en los otros archivos para comprobar los requisitos necesarios También define un método para copiar el estado (*copyWith*()) y un *getter* para verificar si tanto el GPS como los permisos están concedidos (*isAllGranted*). Es gracias a estos archivos que comprueban el estado de los permisos de ubicación sean correctos que posteriormente permite manejar las páginas que deben mostrarse al acceder al mapa, utilizando *BlocBuilder* en la página *loading.dart* y

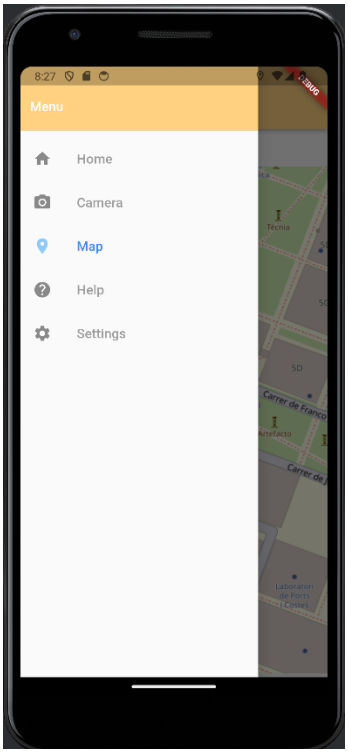

Figura 5.17: Menú lateral.

posteriormente en *location-permitions.dart*, que logra distinguir los permisos que se han conseguido.

5.3 Acceso al clasificador

Para lograr acceder al modelo de red neuronal que clasifica las imágenes en la aplicación, se decidió utilizar una *API* programada en *Python*, que permita utilizar el mejor modelo guardado previamente, para clasificar la imagen que el usuario elija desde la aplicación. Se utilizó la biblioteca de *Python FastAPI*, que permitió la creación de la *API* como intermediario entre el modelo de red neuronal y la aplicación.

Además, fueron necesarias otras librerías para lograr que esta funcionase correctamente. Se agregaron las librerías *File* y *UploadFile* que son tipos de datos que provienen de *FastAPI* y se utilizan para el manejo de la carga de archivos en una web API. *File* se usa para archivos individuales, mientras que *UploadFile* está específicamente diseñado para manejar archivos que se suben a través de una solicitud HTTP. De esta manera se simplifica la tarea de recibir archivos en una aplicación web y permite acceder fácilmente al contenido del archivo enviado.

Otra de las librerías utilizadas proporcionada por *FastAPI* es la clase *JSONResponse* para enviar respuestas JSON desde la API. Esta clase facilita la creación de respuestas JSON personalizadas, donde puedes especificar el contenido JSON que deseas enviar como respuesta a una solicitud HTTP.

Asimismo, se utilizaron los siguientes elementos: la función *load_model* desde la librería *keras.models*, que sirve para cargar un modelo de red neuronal determinado para poder utilizar el modelo sin necesidad de entrenarlo cada vez; la clase *Image* desde la biblioteca de procesamiento de imágenes de *Python PIL*, para poder manipular la imagen

recibida desde la aplicación y preprocesarla antes de utilizarla en el modelo, y *numpy* para la realización de cálculos numéricos necesarios.

Para conseguir la funcionalidad esperada utilizando las librerías mencionadas, se realiza el siguiente proceso en el archivo *Python*. Primero, se instancia la aplicación de *FastAPI* que se va a utilizar, en este caso llamada *app*. Posteriormente, se carga el mejor modelo de red neuronal explicado en el capítulo anterior, llamado *best_model.h*5 utilizando la función *load_model*. A continuación, se define la función de preproceso de imagen, que se encarga de comprobar que la imagen que recibe de entrada sea válida para su uso en el modelo; convirtiéndola a formato RGB, cambiando su tamaño de imagen a 64x64 píxeles esperado por el modelo, normalizando los valores de los píxeles dividiendo por 255 y agregando una dimensión extra para el lote. Continúa, definiendo una ruta de API utilizando el decorador *@app.post(/predict")*. Esta ruta espera una solicitud POST y permite cargar una imagen desde la aplicación. La función asociada a esta ruta se encarga de predicciones sobre la imagen cargada, esto lo consigue abriendo y preprocesando la imagen con la función mencionada anteriormente, tras lo cual se predice el tipo de residuo con el modelo y devuelve las predicciones en formato *JSON*. Toma un parámetro *file* que representa el archivo de imagen cargado.

5.4 Acceso a datos de contenedores

En el caso del acceso a los datos sobre los contenedores, se decidió comenzar por utilizar la información local, utilizando solamente los datos de los contenedores de la ciudad de Valencia, que podría ser ampliado posteriormente. Para ello, se utilizó la página de datos del gobierno[5], donde los ayuntamientos y demás publican información de datos públicos. En este caso, se utilizaron dos datasets publicados por el Ayuntamiento de València, que contienen los archivos *JSON* necesarios para poder mostrar los contenedores posteriormente en el mapa de la aplicación. Uno de ellos contiene solamente los contenedores de vidrio, mientras que el otro contiene varios tipos de contenedores distintos, de plástico, papel, restos y orgánico. Estos son los tipos de contenedores que concuerdan con las categorías clasificadas en el modelo, por lo que no fue necesario agregar más *datasets*.

Para tener los datos de los contenedores de la ciudad actualizados, se decidió, en lugar de solamente descargar los archivos *JSON* una única vez, actualizarlos diariamente para mantenerlos al día. Esto se consigue al igual que la manera de unir el clasificador de *Python* con la aplicación, con un API. En este caso, también se programó en *Python* con la biblioteca *FastAPI*. Se utilizaron algunas librerías y clases distintas a las previamente mencionadas.

De las proporcionadas por *FastAPI*, se utilizó *BackgroundTasks*, que se utiliza, como su nombre indica, para manejar tareas en segundo plano, ejecutando funciones de manera asíncrona mientras la aplicación continúa respondiendo a diferentes solicitudes. El resto de bibliotecas utilizadas fueron; *requests*, que facilita el envío de solicitudes HTTP como POST, GET, PUT, DELETE y demás, así como trabajar con las respuestas de los datos en formato JSON, bytes, etc.; *os*, que proporciona funciones para interactuar con el sistema operativo como realizar operaciones relacionadas con los archivos, y *time*, que proporciona funciones relacionadas con el tiempo.

En el programa, se comienza por instanciar la aplicación de *FastAPI* al igual que en el caso del modelo, seguido de la definición de tanto las *url* de donde se van a descargar los archivos *JSON* como el directorio donde se van a almacenar. A continuación, se define la función *download_json* que hace el proceso de descargar los *JSON* de la página web

[5] datos.gob.es

del *url*. La función toma una *url* como argumento y realiza la descarga del archivo *JSON* desde ella. Primero, verifica con *response.status_code* si la solicitud HTTP devuelve un código de estado 200, lo que indica que la solicitud fue exitosa. En ese caso, el archivo lo guarda en el directorio de descargas con el nombre extraído de la URL. Antes de guardar el nuevo archivo, verifica si ya existe un archivo con el mismo nombre en el directorio de descargas, en cuyo caso se elimina antes de guardar el nuevo archivo. La función termina imprimiendo un mensaje indicando si la descarga y el reemplazo fueron exitosos. Seguidamente, se define la función *download_json_periodically* que se ejecuta periódicamente, de modo que llama a la función definida previamente de *download_json* cada 24 horas, utilizando la función *time.sleep* para programar su periodicidad.

Finalmente, se define la ruta de la API en *start_download* y acepta solicitudes GET, ejecutando la función asíncrona *start_download* al acceder a esta ruta. La función toma un parámetro *background_tasks* que representa la instancia de *BackgroundTasks* proporcionada por *FastAPI*. Dentro de la función, se agrega la tarea *download_json_periodically* a *background_tasks* utilizando el método *add_task*. Este método toma como argumento la función *download_json_periodically* que se ejecutará en segundo plano. La función devuelve un diccionario con un mensaje indicando que la tarea de descarga se ha iniciado correctamente.

5.5 Conclusiones

La idea inicial del desarrollo de la aplicación de gestión de residuos, era la de realizar una aplicación simple y accesible para un amplio rango de personas, incluidas personas con pocas competencias tecnológicas. Durante este capítulo se ha profundizado sobre el proceso que se ha llevado a cabo, desde la idea inicial para la aplicación hasta su implementación final en *Flutter*, comentando también las maneras en las que se ha accedido a otros elementos necesarios para la aplicación como los modelos o la información de los contenedores.

La intención era desarrollar una aplicación eficiente, rápida y eficaz, con un diseño fácil de entender sin estar demasiado cargado de información para no saturar y confundir al usuario. Para lo que se desarrolló una aplicación de estilo minimalista, con pocas funcionalidades claras mientras se realizasen las tareas posibles en el fondo, en paralelo con el resto.

Desde la fase inicial de diseño, se ha puesto énfasis en crear una interfaz intuitiva y fácil de usar para garantizar que personas con diversos niveles de habilidades tecnológicas puedan utilizar la aplicación sin dificultades. A pesar de los esfuerzos por hacer la aplicación accesible para una amplia variedad de personas, aún puede excluir a ciertos grupos, como personas con discapacidades visuales. La falta de compatibilidad con tecnologías de asistencia, como lectores de pantalla, puede limitar la accesibilidad de la aplicación para estas personas, por lo que no se cumpliría completamente el objetivo de inclusión. Además, al simplificar la aplicación y no ofrecer maneras en las que personalizar más la aplicación para no confundir al usuario con competencias tecnológicas, es posible que algunos usuarios más experimentados se sientan frustrados al querer ajustes más avanzados.

La elección de tecnologías como *Flutter* y *Dart* para el desarrollo de la aplicación móvil ha permitido crear una experiencia de usuario uniforme y accesible en múltiples plataformas, maximizando la accesibilidad para un público más amplio. La aplicación se ha diseñado para cumplir con su propósito principal de ayudar a los usuarios a clasificar residuos y acceder a información sobre contenedores de manera rápida y sencilla, lo que contribuye a su eficacia y utilidad general.

En conclusión, se ha desarrollado la aplicación siguiendo las ideas generales que se tenían al inicio sobre esta. Se han implementado las funcionalidades principales pensadas, a pesar de tener ciertos cambios respecto al diseño inicial, algunos de mayor calibre como el cambio de la página de búsqueda por la página de ayuda debido a una utilidad mayor para el usuario final. Además, se han logrado solucionar ciertos problemas, como la desactualización de los datos de los contenedores mediante la implementación de la API que lo reinstale periódicamente.

CAPÍTULO 6
Conclusiones

Tras haber cumplido con los objetivos establecidos, las conclusiones de este trabajo de fin de grado son claras. La creación de una aplicación destinada a facilitar el proceso de reciclaje de residuos se presenta como una solución viable y relevante en el contexto actual. La decisión de desarrollar una plataforma que permita a los usuarios agregar imágenes de los residuos que deseen reciclar, así como brindar información sobre los lugares de reciclaje y la ubicación de los contenedores más cercanos, responde a la necesidad de simplificar este proceso para un público amplio y diverso.

El objetivo principal del trabajo ha sido el diseño y desarrollo de esta aplicación como una herramienta accesible para cualquier persona interesada en contribuir al reciclaje. Al buscar ofrecer este servicio de manera simple, se ha priorizado la accesibilidad para diferentes grupos de personas, incluidos aquellos con menor familiaridad tecnológica o conocimiento sobre reciclaje. La inclusión de estas personas en el proceso de reciclaje es fundamental para fomentar una participación más amplia y efectiva en la preservación del medio ambiente.

Para alcanzar este objetivo principal, se han definido objetivos específicos que han sido abordados a lo largo del trabajo. Desde el análisis de las herramientas disponibles hasta el diseño y desarrollo de la aplicación móvil, se ha prestado atención a cada fase del proceso para alcanzar los objetivos del proyecto. A continuación, se especificará como se han logrado cada uno de los subobjetivos:

- Para el funcionamiento del trabajo a realizar, se analizaron las tecnologías a utilizar para las diferentes funciones que se desarrollaron posteriormente, analizando las herramientas más populares y escogiendo las más adecuadas para el desarrollo del trabajo.
- En la arquitectura del sistema, se plantearon las funciones de cada uno de los elementos del trabajo, así como la manera de interactuar entre ellos. En este proceso se marcaron los requisitos que cada uno de los elementos necesitaría para el correcto funcionamiento final de la arquitectura.
- Se realizó un modelo de clasificación, tras analizar, implementar y evaluar un amplio número de diferentes técnicas y modelos, se eligió el de mejores resultados.
- Finalmente, se diseñó y desarrolló la aplicación móvil. En esta se enfatizó el objetivo principal de lograr la sencillez para el usuario con pocas competencias tecnológicas, desarrollándola con una interfaz sencilla e intuitiva para el usuario, gracias al diseño influenciado por el estilo minimalista.

Mediante este trabajo se permite acercar al usuario final a una herramienta práctica y accesible para fomentar el reciclaje y la conciencia ambiental. Gracias al uso de la tec-

nología, se permite facilitar el reciclaje a la persona común que carece de conocimientos al respecto. Aportando de esta manera lo posible por parte del individuo para evitar el agravamiento del cambio climático.

6.1 Trabajos futuros

A pesar de lo conseguido en el trabajo que se ha realizado y definido a lo largo de la memoria, se han pensado en otras maneras en las que se podría lograr mejorar la aplicación en trabajos futuros. Los siguientes tres puntos describen mejoras que se podrían llevar a cabo en el trabajo:

- Mejorar el modelo de redes neuronales para la clasificación de residuos: Este trabajo futuro implicaría investigar y probar diferentes arquitecturas de redes neuronales, así como nuevas técnicas de entrenamiento no utilizadas previamente en el trabajo. Se podrían explorar conjuntos de datos más grandes y variados para el entrenamiento del modelo, incluyendo imágenes de residuos en diferentes condiciones de iluminación, ángulos y fondos. Además, se podría considerar el uso de técnicas avanzadas de preprocesamiento de imágenes, como la segmentación semántica o la eliminación de ruido, para mejorar la calidad de los datos de entrada y la precisión del modelo.
- Crear un asistente virtual para responder preguntas de los usuarios sobre el reciclaje: Este trabajo futuro requeriría desarrollar un sistema de procesamiento del lenguaje natural capaz de comprender y responder a preguntas sobre el reciclaje en lenguaje natural. Se podría utilizar un enfoque basado en modelos de lenguaje preentrenados, como *BERT* (*Bidirectional Encoder Representations from Transformers*), y adaptarlo al dominio específico del reciclaje. Se necesitaría construir una base de conocimientos sólida sobre el reciclaje, que incluya información sobre los diferentes tipos de residuos, los materiales reciclables y los métodos de reciclaje disponibles. El asistente virtual podría integrarse en la aplicación móvil como una interfaz de chat o un sistema de preguntas y respuestas, proporcionando a los usuarios información útil y contextualizada sobre el reciclaje.
- Ampliar la base de datos de contenedores de reciclaje utilizando datos de otras ciudades/municipios: Para este trabajo futuro, se requeriría recopilar y procesar datos sobre la ubicación y características de los contenedores de reciclaje de otras ciudades y municipios. Se podría establecer colaboraciones con autoridades locales y organizaciones ambientales para obtener acceso a sus bases de datos de contenedores de reciclaje. Se debería desarrollar un sistema de integración de datos que permita agregar y actualizar automáticamente la información de nuevos contenedores de reciclaje en la base de datos de la aplicación móvil. Además, se necesitaría implementar funcionalidades de búsqueda y filtrado avanzadas en la aplicación para permitir a los usuarios encontrar contenedores de reciclaje en diferentes ubicaciones geográficas de manera rápida y eficiente.

6.2 Relación del trabajo desarrollado con los estudios cursados

En el transcurso de este trabajo, se han explorado y aplicado una variedad de conceptos y habilidades fundamentales que se han adquirido a lo largo del programa de estudio. En este apartado final, nos proponemos examinar detalladamente la relación entre el trabajo realizado y las asignaturas cursadas en el programa académico. Al comprender

cómo los conocimientos teóricos adquiridos en el aula se han traducido en prácticas concretas durante el desarrollo de la aplicación móvil, se podrá apreciar mejor la integración entre la teoría y la práctica en la formación como profesional de la informática.

La relación entre el trabajo de exploración de modelos de redes neuronales para la clasificación de residuos y la asignatura de Aprendizaje Automático se fundamenta en el uso de técnicas avanzadas de aprendizaje automático para abordar un problema específico. En la asignatura de Aprendizaje Automático se estudian las principales técnicas que sustentan las distintas tecnologías de esta área, incluyendo el reconocimiento de imágenes, que es precisamente el enfoque utilizado en el trabajo para clasificar los residuos.

La asignatura de Aprendizaje Automático proporciona una base sólida en técnicas estadísticas avanzadas que son aplicables a problemas complejos de reconocimiento de formas, lo cual es fundamental para el desarrollo de modelos de clasificación de residuos basados en imágenes. Además, la relación con asignaturas relacionadas como Percepción, Agentes Inteligentes y Técnicas, entornos y aplicaciones de IA, proporciona un contexto amplio para comprender cómo las técnicas de aprendizaje automático se integran en sistemas más grandes de inteligencia artificial.

El trabajo se beneficia de los conocimientos adquiridos en la asignatura de Aprendizaje Automático al aplicar técnicas avanzadas de esta área para resolver un problema específico de clasificación de residuos a través del análisis de imágenes. Esto demuestra la relevancia y aplicabilidad práctica de los conceptos estudiados en la asignatura en un contexto real de desarrollo de aplicaciones de inteligencia artificial.

La relación entre el trabajo de creación de un boceto, diseño e implementación de la aplicación móvil y la asignatura de Interacción Persona-Computador se basa en el enfoque en la usabilidad y el diseño de interfaces de usuario efectivas. En la asignatura de Interacción Persona-Computador se estudia cómo diseñar interfaces que sean fáciles de usar y cómo desarrollar programas que tengan una apariencia profesional. Esto se alinea directamente con el proceso que se siguió en el trabajo, donde se comenzó con la creación de un boceto para establecer la estructura y la navegación de la aplicación, seguido por el diseño de la interfaz de usuario y finalmente la implementación de la aplicación móvil.

El objetivo principal de la asignatura es que el alumno sea capaz de desarrollar aplicaciones con interfaces gráficas de usuario modernas, utilizando herramientas actuales. Esto se traduce en la habilidad para crear aplicaciones móviles intuitivas y atractivas, lo cual es fundamental para el éxito del trabajo. El ciclo de vida de desarrollo de interfaces de usuario, que se estudia en la asignatura, proporciona el marco metodológico para guiar el proceso de diseño y desarrollo de la aplicación móvil.

Además, la asignatura también aborda técnicas de prototipado y evaluación de interfaces, aspectos clave que se consideraron durante la fase de diseño de la aplicación. El prototipado permite iterar sobre diferentes diseños y funcionalidades antes de la implementación final, lo cual es fundamental para garantizar una experiencia de usuario óptima. Asimismo, la evaluación de interfaces proporciona herramientas para medir la eficacia y la satisfacción del usuario, lo cual es fundamental para mejorar continuamente la aplicación.

En resumen, se ha observado que las asignaturas cursadas en el grado han aportado una gran ayuda para realizar este trabajo de fin de grado, demostrando así la utilidad que estas tienen en aplicaciones reales. Tanto la asignatura de Aprendizaje Automático, para el desarrollo de modelos de clasificación de residuos basados en imágenes; como la asignatura de Interacción Persona-Computador donde se refleja el enfoque en la usabilidad y el diseño de interfaces de usuario efectivas, han sido cruciales para la realización del trabajo.

Bibliografía

[Acosta, 2019] Acosta, D. (2019). Recyclephoto dataset. https://www.kaggle.com/code/danielaco/recyclephoto.

[Adedeji and Wang, 2019] Adedeji, O. and Wang, Z. (2019). Intelligent waste classification system using deep learning convolutional neural network. *Procedia Manufacturing*, 35:607–612. The 2nd International Conference on Sustainable Materials Processing and Manufacturing, SMPM 2019, 8-10 March 2019, Sun City, South Africa.

[Awe et al., 2017] Awe, O., Mengistu, R., and Sreedhar, V. (2017). Final report smart trash net waste localization and classification. *preprint*.

[Azis et al., 2020] Azis, F. A., Suhaimi, H., and Abas, E. (2020). Waste classification using convolutional neural network. In *Proceedings of the 2020 2nd international conference on information technology and computer communications*, pages 9–13.

[B., 2022] B., D. (2022). Vgg: ¿qué es este modelo? *Data Scientist*.

[Eurostat, 2024] Eurostat (2024). Municipal waste by waste management operations. https://ec.europa.eu/eurostat/databrowser/view/env_wasmun__custom_9634214/default/table?lang=en.

[Faria et al., 2021] Faria, R., Ahmed, F., Das, A., and Dey, A. (2021). Classification of organic and solid waste using deep convolutional neural networks. In *2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 01–06.

[Islam et al., 2018] Islam, M. S., Foysal, F. A., Neehal, N., Karim, E., and Hossain, S. A. (2018). Inceptb: a cnn based classification approach for recognizing traditional bengali games. *Procedia computer science*, 143:595–602.

[Kumsetty et al., 2022] Kumsetty, N. V., Nekkare, A. B., Kamath, S., et al. (2022). Trashbox: Trash detection and classification using quantum transfer learning. In *2022 31st Conference of Open Innovations Association (FRUCT)*, pages 125–130. IEEE.

[Malik et al., 2022] Malik, M., Sharma, S., Uddin, M., Chen, C.-L., Wu, C.-M., Soni, P., and Chaudhary, S. (2022). Waste classification for sustainable development using image recognition with deep learning neural network models. *Sustainability*, 14(12):7222.

[Mukherjee, 2022] Mukherjee, S. (2022). The annotated resnet-50. *Medium*.

[Ochecolo, 2020] Ochecolo, D. I. (2020). Trash dataset. https://www.kaggle.com/datasets/dimonisochecolo/trash-dataset.

[Pote, 2022] Pote, S. S. (2022). *Waste Classification by Using Deep Learning and Transfer Learning*. PhD thesis, Dublin, National College of Ireland.

[Proença and Simões, 2020] Proença, P. and Simões, P. (2020). Taco dataset. http://tacodataset.org/.

[Ratul et al., 2019] Ratul, M. A. R., Mozaffari, M. H., Lee, W.-S., and Parimbelli, E. (2019). Skin lesions classification using deep learning based on dilated convolution. *BioRxiv*.

[Sekar, 2019] Sekar, S. (2019). Waste classification data. https://www.kaggle.com/datasets/techsash/waste-classification-data.

[Serengil, 2018] Serengil, S. (2018). Transfer learning in keras using inception v3. https://sefiks.com/2017/12/10/transfer-learning-in-keras-using-inception-v3/.

[Serezhkin, 2020] Serezhkin, A. (2020). Drinking waste classification dataset. https://www.kaggle.com/datasets/arkadiyhacks/drinking-waste-classification.

[Statista, 2022] Statista (2022). Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

[Tragoudaras et al., 2022] Tragoudaras, A., Stoikos, P., Fanaras, K., Tziouvaras, A., Floros, G., Dimitriou, G., Kolomvatsos, K., and Stamoulis, G. (2022). Design space exploration of a sparse mobilenetv2 using high-level synthesis and sparse matrix techniques on fpgas. *Sensors*, 22:4318.

[Vernon, 2023] Vernon, B. (2023). Exploring the principles of minimalism in ui/ux design. *Medium*.

[Yang and Thung, 2016] Yang, M. and Thung, G. (2016). Classification of trash for recyclability status. *CS229 project report*, 2016(1):3.

APÉNDICE A
OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**		X		
ODS 11. **Ciudades y comunidades sostenibles.**	X			
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**	X			
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Los objetivos de desarrollo sostenible son una manera de mostrar lo beneficiosos que pueden ser los trabajos realizados para la sociedad, por lo que su importancia ha aumentado. En el trabajo se ha intentado cubrir varios de ellos para el beneficio de la sociedad mediante el uso de la solución móvil desarrollada.

Uno de los principales aspectos destacados es la relación entre este proyecto y el ODS 11: Ciudades y comunidades sostenibles. En las comunidades urbanas, la gestión adecuada de los residuos es fundamental para mantener un entorno limpio y saludable. La aplicación proporciona a los usuarios información sobre los contenedores de reciclaje cercanos, lo que facilita el proceso de reciclaje y promueve prácticas más sostenibles en las ciudades. Al fomentar una mayor participación en el reciclaje, la aplicación contribuye directamente a la creación de comunidades más sostenibles y resistentes.

Además, el proyecto aborda el ODS 9: Industria, innovación e infraestructuras. La incorporación de tecnologías innovadoras, como redes neuronales para la clasificación de residuos, muestra un enfoque hacia la innovación en el campo de la gestión de residuos y la tecnología accesible. Esta aplicación no solo mejora la eficiencia del proceso de reciclaje, sino que también fomenta el desarrollo de infraestructuras relacionadas con el reciclaje y la gestión de residuos. Esta innovación en la industria del reciclaje es crucial para impulsar una transición hacia una economía más circular y sostenible.

Otro aspecto relevante es la contribución al ODS 13: Acción por el clima. El reciclaje de residuos es una estrategia efectiva para reducir las emisiones de gases de efecto invernadero y mitigar el cambio climático. Al promover el reciclaje a través de la aplicación, se fomenta un uso más eficiente de los recursos naturales y se reduce la dependencia de los vertederos y la incineración, que son fuentes significativas de emisiones de gases de efecto invernadero. Esta acción directa tiene un impacto positivo en la lucha contra el cambio climático y contribuye a la resiliencia climática a largo plazo.

Además de los ODS mencionados anteriormente, el proyecto también tiene implicaciones para otros objetivos. Por ejemplo, al promover el reciclaje y la gestión adecuada de residuos, contribuye indirectamente al ODS 3: Salud y bienestar. La reducción de la cantidad de residuos que se envían a vertederos o se incineran puede disminuir la exposición a sustancias tóxicas y contaminantes en el medio ambiente, lo que beneficia la salud pública y el bienestar de las comunidades.

Asimismo, la aplicación puede tener un impacto en el ODS 12: Producción y consumo responsables. Al proporcionar información sobre cómo reciclar adecuadamente los residuos y promover un comportamiento de consumo más responsable, se fomenta un enfoque más consciente hacia el consumo de recursos y la generación de residuos. Esto puede ayudar a reducir la generación de residuos y promover prácticas de consumo más sostenibles a largo plazo.

Al hacer que la tecnología sea más accesible y fácil de usar para un grupo más amplio de personas, incluidas aquellas con habilidades tecnológicas limitadas, el trabajo realizado ayuda a cerrar la brecha digital y reduce las desigualdades en el acceso a la información y las oportunidades. Esto es especialmente relevante en un mundo donde el acceso a la tecnología y la alfabetización digital son cada vez más importantes para la participación en la sociedad y la economía.

Al eliminar barreras de acceso y hacer que la información sobre el reciclaje y la gestión de residuos sea más fácilmente disponible, la aplicación desarrollada puede empoderar a comunidades marginadas o con menos recursos, brindándoles herramientas para participar activamente en la protección del medio ambiente y mejorar su calidad de vida.

En este sentido, aunque la relación con el ODS 10 puede considerarse en menor medida en comparación con otros objetivos, sigue siendo una contribución significativa a la reducción de las desigualdades y la promoción de la inclusión social.

En conclusión, el desarrollo de una aplicación accesible para el reciclaje de residuos es un ejemplo claro de cómo la innovación y la tecnología pueden contribuir significativamente a la consecución de múltiples Objetivos de Desarrollo Sostenible.