TÍTULO
RECONOCIMIENTO Y GESTIÓN DE IMÁGENES EN EL ÁMBITO INDUSTRIAL BAJO EL PARADIGMA IOT

TRABAJO FIN DE MÁSTER
CURSO 2020-2021

AUTOR
IONUT ANDREI VADUVA

DIRECTOR
GONZALO PAJARES MARTINSANZ

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TÍTULO

RECONOCIMIENTO Y GESTIÓN DE IMÁGENES EN EL ÁMBITO INDUSTRIAL BAJO EL PARADIGMA IOT

TRABAJO DE FIN DE MÁSTER EN INTERNET OF THINGS
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

AUTOR
IONUT ANDREI VADUVA

DIRECTOR
GONZALO PAJARES MARTINSANZ

CONVOCATORIA: JUNIO 2021

CALIFICACIÓN: 10

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

25 DE JUNIO DE 2020

AGRADECIMIENTOS

A Gonzalo, por su tiempo y ayuda a lo largo del desarrollo de este proyecto.

RESUMEN

RECONOCIMIENTO Y GESTIÓN DE IMÁGENES EN EL ÁMBITO INDUSTRIAL BAJO EL PARADIGMA IOT

Este trabajo tiene como principal objetivo el proporcionar una solución conceptual inteligente bajo el paradigma de Internet de las Cosas (Internet of Things, IoT) para su aplicación en el contexto de la automatización de la clasificación de residuos urbanos para su reciclaje y con vistas a su posible implantación bajo el paraguas de desarrollos en lo que se conoce como industria 4.0.

La parte inteligente radica en el desarrollo de un modelo de red neuronal convolucional con base en el modelo preestablecido de MobileNetV2 convenientemente adaptado para la aplicación que se plantea.

En primer lugar, se analizan algunas de las principales plataformas de servicios orientados a IoT para el propósito previsto. Haciendo uso de estas plataformas se desarrolla una aplicación web que permite a los usuarios clasificar los residuos de forma correcta antes de que lleguen a la planta de clasificación.

En segundo lugar, se plantea el primer escenario donde haciendo uso del dispositivo ESP-EYE se toman imágenes que se envían a la nube para su procesamiento y clasificación, obteniendo de esta forma un panel de control con los resultados y estadísticas generadas.

Debido a los costes e implicaciones del primer escenario, se plantea un segundo escenario donde la clasificación de los residuos se realiza en el propio ESP-EYE que envía el resultado al mismo panel de control que en el escenario anterior para su interpretación.

En ambos escenarios se analizan los resultados a nivel de desempeño y tiempos de ejecución, para llegar a las conclusiones pertinentes tras la valoración de los resultados.

Palabras clave: IoT, Azure, IBM Cloud, IBM Watson, Node-red, ESP-EYE, Redes neuronales convolucionales, Inteligencia artificial, MQTT

ABSTRACT

INDUSTRIAL IMAGE RECOGNITION AND MANAGEMENT UNDER THE IOT PARADIGM

The main objective of this work is to provide an intelligent conceptual solution under the Internet of Things (IoT) paradigm for its application in the context of the automation of urban waste sorting for recycling and with a view to its possible implementation under the umbrella of developments in what is known as Industry 4.0.

The intelligent part lies in the development of a convolutional neural network model based on the pre-established model of MobileNetV2 conveniently adapted for the proposed application.

First, some of the main IoT-oriented service platforms are analyzed for the intended purpose. Making use of these platforms, a web application is developed that allows users to sort waste correctly before it reaches the sorting plant.

Secondly, the first scenario is proposed where making use of the ESP-EYE device images are taken and sent to the cloud for processing and classification, thus obtaining a control panel with the results and statistics generated.

Due to the costs and implications of the first scenario, a second scenario is proposed where the classification of the waste is performed in the ESP-EYE itself, which sends the result to the same control panel as in the previous scenario for interpretation.

In both scenarios, the results are analyzed in terms of performance and execution times, in order to reach the pertinent conclusions after evaluating the results.

Keywords

IoT, Azure, IBM Cloud, IBM Watson, Node-red, ESP-EYE, Redes neuronales, Inteligencia artificial, MQTT

Índice de contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

Cada día se producen nuevos avances tecnológicos que cambian poco a poco el día a día de las personas. A estos avances se les considera parte de lo que se conoce como Carta de Revolución Industrial, o técnicamente Industria 4.0.

En esta nueva industria destacan sobre todo las tecnologías digitales que tratan de profundizar en la búsqueda de técnicas avanzadas de producción, operaciones automatizadas inteligentes, gestión de las personas y seguridad entre otras. Destacan especialmente los campos de la robótica, el análisis de datos masivos, la inteligencia artificial y sobre todo el Internet de las Cosas, por sus siglas en inglés IoT (*Internet of Things*).

En este trabajo se hará uso de la inteligencia artificial e IoT para mejorar los procesos, la eficiencia y los beneficios aplicando soluciones inteligentes en la clasificación de residuos urbanos en plantas de clasificación de residuos urbanos.

Para lograr las mejoras de los procesos comentadas anteriormente se van a aplicar técnicas y algoritmos de reconocimiento de imágenes en partes concretas del proceso de reciclaje mediante el uso de dispositivos y plataformas en la nube.

1.1 Motivación

La tecnología está en constante evolución para satisfacer cada vez mejor las necesidades del ser humano. Al avanzar esta, se producen progresos paralelos en diversos ámbitos en la sociedad, en la sanidad y en la industria en general. Desde comienzos del siglo XX la población mundial ha crecido exponencialmente en gran medida gracias a avances tecnológicos, sanitarios y sociales, entre otros. Con el incremento de la población aumentan las necesidades por lo que en el ámbito industrial comienza toda una revolución para satisfacer esas necesidades. El objetivo de esta revolución industrial es mejorar los procesos para generar más productos a menor coste e incrementar los beneficios, así como satisfacer las necesidades del ser humano.

Al aumentar la población, las necesidades y en definitiva el consumo, se generan gran cantidad de residuos que darán lugar a la creación de vertederos. A mediados del siglo

XX los residuos comienzan a suponer un problema medioambiental ya que a diferencia de los siglos anteriores donde la mayoría de los residuos generados eran orgánicos o su tiempo de desintegración era corto, a mediados del siglo XX empiezan a aparecer residuos con tiempos de desintegración mucho más largos y, como se ha comentado previamente, en gran medida debido al aumento de la población, el consumo y la industrialización. Estos residuos en gran parte están formados por restos inorgánicos, tales como papel, cartón, aluminio, vidrio y sobre todo plástico, que son algunos de los elementos considerados en este trabajo.

Esto da lugar a la necesidad de gestionar los residuos de forma eficiente y sostenible para reducir la contaminación a todos los niveles, tales como atmosférica, aguas, suelos, etc.

Debido a este problema medioambiental generado, aparecen diferentes soluciones para mitigarlo, entre ellas las plantas de reciclaje, las plantas de clasificación, la división de los residuos y el establecimiento de puntos específicos para la recogida de residuos (papeleras, contenedores, etc.).

A lo largo de los años, los residuos se han clasificado de diversas formas según su estado físico, en función de su origen o de su estructura química. Este trabajo se centra en la clasificación de residuos sólidos inorgánicos como el cartón, metal, vidrio y plástico generados en los núcleos urbanos y provenientes de actividades domésticas, restaurantes o lugares públicos.

Como ya se ha comentado anteriormente, la gestión de estos residuos es complicada ya que cada día se generan grandes cantidades de materiales residuales, por lo que este sería el punto de partida para el tratamiento de los mismos. Una vez generados se depositan en contenedores y/o papeleras dentro de los núcleos urbanos donde no siempre se realiza una clasificación correcta de los mismos, por ello, una fase de la gestión de estos residuos es la recolección, que puede ser general (no se tiene en cuenta ningún tipo de clasificación) o selectiva (contenedores específicos que identifican los residuos por las mismas características).

En este punto del proceso de gestión de los residuos sólidos urbanos es donde ya se podrían aplicar soluciones basadas en ecosistemas IoT para facilitar su clasificación,

antes incluso de su clasificación en una planta específica de tratamiento de residuos. De esta forma se podría disponer de contenedores inteligentes capaces de clasificar los residuos de forma óptima y sin errores, permitiendo una gestión de estos más sencilla y eficiente, pero aumentando el coste de esta gestión al necesitar contenedores inteligentes.

Siguiendo con el proceso de gestión de los residuos una vez clasificados de una manera u otra han de transportarse a instalaciones específicas (vertederos, plantas de tratamiento de residuos, punto limpio, etc.).

Para solventar el problema que podría suponer el coste de los contenedores hoy en día, ya que la industria 4.0 está en su inicio, en este trabajo se plantea el desarrollo de una aplicación web multiplataforma que ayudará a las personas a realizar una clasificación de los residuos correcta en caso de duda, además del desarrollo de un ecosistema IoT hipotético, y por tanto de naturaleza conceptual, que se aplicará en las plantas de clasificación de residuos, con lo que se plantea el aumento de la eficiencia y por tanto, la mejora del tratamiento y la gestión de los residuos urbanos.

1.2 Análisis de soluciones existentes

Antes de comenzar con el desarrollo se ha llevado a cabo una investigación sobre proyectos existentes que podrían resultar interesantes en el contexto de una planta de clasificación de residuos.

Entre los proyectos analizados destacan:

- *DustBot*: Es un proyecto desarrollado por un grupo de investigadores italianos de una duración de treinta y seis meses que comenzó en 01/12/2006 y acabó en 30/11/2009. Los objetivos de este proyecto son la creación de robots capaces de recolectar basura en ambientes urbanos, calles, parques, residencias mediante el uso del aprendizaje automático y diferentes sensores. El robot, también sería capaz de recolectar información sobre la calidad del aire en núcleos urbanos saturados para posteriormente enviarla a una plataforma capaz de almacenarla y procesarla en forma de gráficos para facilitar la toma de decisiones. El proyecto también propone que los robots envíen información con el fin de mejorar el aprendizaje del propio robot.

- *Robot-based Autonomous Refuse (ROAR)*: Es un proyecto desarrollado por treinta y cinco estudiantes de tres universidades en el año 2017. El objetivo de este proyecto es la recolección de los cubos de basura de los núcleos urbanos. Para lograr este objetivo utilizan un camión de la basura mejorado gracias al IoT, haciendo uso de una cámara montada en la parte trasera del camión para permitir al conductor ver el proceso de vaciado del contenedor, un robot capaz de recibir unas coordenadas a las que dirigirse sorteando obstáculos para recoger el cubo y llevarlo al camión además de un dron que irá montado encima del camión de forma que cuando hay que recoger un cubo primero el dron toma altura para buscar la ubicación del cubo y después enviarla al robot recolector para que este se dirija al cubo y lo lleve al camión para su vaciado.

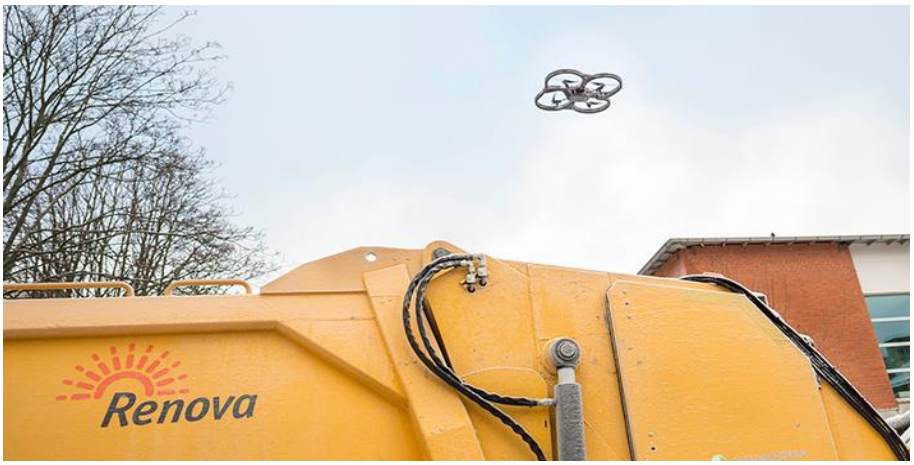


Figura 1-1. ROAR - Dron busca la ubicación de los contenedores de basura

Figura 1-2. ROAR - El robot recolector recibe las coordenadas del cubo a recoger

Figura 1-3. ROAR - El robot deposita el cubo en el camión para su vaciado

- Seabin: Este proyecto nace en el año 2014 con los australianos Andrew Turton y Pete Ceglinski siendo una empresa emergente (*startup*). La idea principal proviene de los cubos de basura que se utilizan en tierra extrapolando la misma idea en el agua. El proyecto comienza con la compra de una fábrica de muebles en desuso en Palma de Mallorca en el año 2014 y gracias a campañas de micro financiación (*crowdfunding*) y la obtención de asociados como *Poralu Marine* (empresa que ofrece soluciones de alta gama en entornos marinos) y *La Grande Motte* puerto deportivo francés, consiguen impulsar el proyecto y crear el primer prototipo en el año 2016 que se instala en Mallorca. En el año 2017 siguen impulsando el proyecto a través de plataformas online como *booking.com*, consiguiendo así más asociados para que finalmente se dé comienzo a su uso comercial y distribución a gran escala. Hoy en día cuentan con alrededor de 860 cubos desplegados que capturan cada día 3612 kg de basura. En el aspecto técnico no se han encontrado manuales o especificaciones técnicas detalladas sobre la construcción y el funcionamiento de estos contenedores, simplemente algunas características como el bajo consumo de energía (500 vatios) capaz de capturar micro plásticos con un tamaño superior a 2 milímetros, peso del contenedor con el soporte de 55 kg, 6 metros de cable, dimensiones 500 x 500 mm x 1800 mm.
- *River Trash Collector System (RTCS)*: Este proyecto es un prototipo desarrollado por varias facultades de la *Universidad Teknikal Malaysia Melaka* (por sus siglas UTeM).

Este proyecto parte de la idea del proyecto Seabin y se concibe como una posible mejora/adaptación para ríos. Algunas mejoras son conceptuales como por ejemplo el uso del agua para mejorar la flotabilidad del contenedor en vez de utilizar un lastre de hierro o piedra situado debajo del contenedor, proponen abrir cuatro válvulas que llenan las paredes del contenedor de agua situándolo al nivel del rio. También proponen la utilización de materiales en diferentes proporciones al proyecto original como por ejemplo 10 kg de metal para la parte de la bomba de agua, recubrimiento de PVC 4 kg, una capa inferior de madera 15 kg y otros componentes en menor medida obteniendo como resultado un peso del contenedor de 30 kg (20 kg menor que el Seabin v5) además se propone que el contenedor flote libremente por el rio mientras que el contenedor del proyecto Seabin permanece anclado.

1.3 Objetivos

Las soluciones descritas en el apartado anterior se centran en el problema de la recolección de los residuos de forma automatizada o semi automatizada. La solución que propone en este trabajo enfoca el problema desde otra perspectiva: la clasificación de los residuos ya recogidos y transportados a una planta de clasificación, aunque como se verá en el apartado de futuras mejoras parte del ecosistema se podría trasladar a diferentes entornos.

Para automatizar el proceso de clasificación en dichas plantas se propone utilizar diferentes cámaras, figura 1-4, colocadas en varias ubicaciones. Estas cámaras tomaran imágenes que se van a procesar de dos formas diferentes, en primer lugar, se verá el procesamiento en la nube analizando los tiempos y respuestas para su comparación con el procesamiento en el propio dispositivo, donde se hará uso de dos modelos de redes neuronales convolucionales, dentro del paradigma IoT, para clasificar dicha imagen y devolver el resultado para realizar las acciones correspondientes.

Figura 1-4. ESP-EYE capaz de tomar una imagen y enviarla para su procesamiento

Además, se dispone de una aplicación móvil, figura 1-5, que permite tomar una imagen que se envía a la nube para realizar la detección de un residuo concreto para facilitar a las personas el depósito de los residuos en el contenedor correcto en caso de duda. Las imágenes tomadas se almacenan en la nube con el fin de obtener un histórico de residuos que se podría emplear para mejorar el entrenamiento de la red neuronal que se utiliza para clasificar un determinado residuo, obtener estadísticas sobre residuos no convencionales que no se pueden procesar en una planta de clasificación de residuos urbanos, por ejemplo, residuos que contengan sustancias peligrosas o que estén formados por varios tipos de materiales y no se puede asociar a una clase concreta como podrían ser los electrodomésticos, baterías, pilas, aerosoles, pinturas, termómetros, capsulas de café, etc.

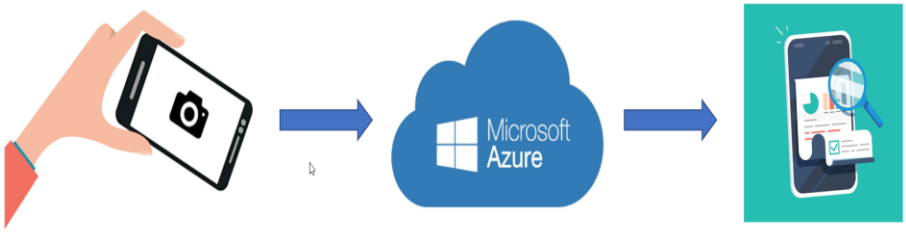


Figura 1-5. Captura de imagen y procesamiento en la nube

A continuación, se enumeran los objetivos identificados en este trabajo teniendo en cuenta las premisas anteriores:

- Crear y entrenar un modelo basado redes neuronales convolucionales (dentro del paradigma de aprendizaje profundo) capaz de clasificar residuos en cuatro categorías: cartón, metal, vidrio y plástico.
- Desarrollar un firmware capaz de tomar una imagen y enviarla para su procesamiento.
- Desarrollar un firmware capaz de tomar una imagen y clasificarla en el propio dispositivo, enviando mediante MQTT el resultado de la clasificación a un panel de control.
- Disponer de una infraestructura en la nube para la comunicación bidireccional con el dispositivo (IBM Watson IoT).

- Disponer de una aplicación que permita tomar imágenes y clasificarlas, mostrando el resultado de dicha clasificación.
- Disponer de servicio en la nube que puede ser consultado desde cualquier plataforma vía HTTP.
- Disponer de un panel de control con fines de monitorización para representar diferentes estadísticas, así como poder comunicarse con el dispositivo de forma amigable.

1.4 Plan de trabajo, contribuciones y organización de la memoria

A continuación, se describe el plan de trabajo seguido para llevar a cabo el diseño y la implementación de este proyecto.

En primer lugar, se han analizado diferentes proyectos y propuestas ya existentes en cuanto a la gestión eficiente de residuos utilizando herramientas y tecnologías en el ámbito IoT. Al no haber encontrado una solución ya existente que aplique IoT e inteligencia artificial para mejorar la eficiencia de las plantas de clasificación, se han propuesto las soluciones cuyos objetivos se mencionan en el apartado anterior.

Antes de comenzar con el desarrollo, en el capítulo 2 se ha realizado un profundo análisis de algunas de las plataformas en la nube más importantes que ofrecen diferentes tipos de servicios, algunos de ellos relacionados con IoT directamente como es el caso de IBM Cloud, Bosh IoT Suite o Azure IoT que puedan ser utilizadas en este contexto, además de buscar y analizar un modelo de redes neuronales ya entrenado que aporte un conocimiento de entrada en la clasificación de residuos.

Una vez analizadas las soluciones existentes, las plataformas y planteado los objetivos se comienza con el diseño y la toma de decisiones. Estos aspectos aparecen detallados en el capítulo 3, donde se define el ecosistema IoT planteado y el desarrollo de la aplicación web, además de dar respuesta a las preguntas:

- ¿Dónde se va a realizar la clasificación de los residuos?
- ¿Se van a almacenar las imágenes que tomen las cámaras?
- ¿Cuál va a ser el rendimiento de la solución si se decide realizar la clasificación en la nube? ¿Y en el propio dispositivo?

- ¿Se utilizará algún servicio que proporcione la infraestructura de comunicación entre la plataforma y el dispositivo?
- ¿Cómo se ha entrenado el modelo para realizar la clasificación?
- ¿Qué tecnologías se han utilizado para el desarrollo de la aplicación web?

Por último, en el capítulo 4 se describen las conclusiones obtenidas y algunas posibles mejoras y evoluciones de este trabajo.

En cuanto a las aportaciones que realiza este trabajo, se pueden destacar los siguientes aspectos:

- Creación, entrenamiento y análisis de un modelo de redes neuronales con un "accuracy" del 93% que permite clasificar cuatro tipos de residuos: cartón, metal, vidrio y plástico.
- Almacenamiento de las imágenes de los residuos en la nube con el fin de disponer de un histórico y además de poder generar conjuntos de datos mejores para mejorar la eficiencia de los modelos empleados.
- Disponibilidad de una aplicación web multiplataforma que permite a cualquier usuario con un dispositivo móvil realizar la clasificación de un residuo.
- Disponibilidad de un servicio en la nube de Azure, público, que, al recibir una imagen de cualquier tamaño, es capaz de realizar una clasificación y devolver el resultado en formato JSON, es decir, la generación de un servicio inteligente de clasificación de residuos que cualquiera podría integrar en su aplicación.
- Desarrollo de un hipotético ecosistema IoT que mediante el firmware desarrollado para el dispositivo ESP-EYE toma imágenes (con una frecuencia configurable) y las envía vía MQTT a la plataforma IBM Cloud donde se procesa mediante una aplicación Node-RED que se comunica con el servicio desarrollado en Azure para realizar la clasificación y mostrar la imagen, el resultado de la clasificación, así como otras estadísticas relevantes en un panel de control.
- Análisis del tiempo de respuesta de la clasificación en la nube y de los resultados obtenidos.

- Análisis de los resultados y tiempo de respuesta de la clasificación en el propio dispositivo.
- Planteamiento de mejoras y evoluciones de este trabajo.

Capítulo 2 - Estado de la cuestión

Antes de comenzar con el desarrollo del sistema de clasificación de residuos, se ha realizado un análisis de las diferentes plataformas digitales existentes con el fin de elegir la que mejor se adapta a las necesidades del ecosistema IoT a desarrollar. En general todas permiten a los desarrolladores la creación, ejecución, soporte y mantenimiento de servicios y aplicaciones. A continuación, vamos a ver algunas características y servicios de cada una de ellas.

2.1 Análisis de plataformas

En este apartado se va a describir algunas de las plataformas más relevantes hoy en día relacionadas con el procesamiento en la nube e IoT para ver cuál de ellas se ajusta mejor a las necesidades y objetivos descritos en capítulos anteriores.

2.1.1 IBM Cloud

IBM Cloud es una plataforma que ofrece la combinación de una plataforma como servicio (PaaS) e infraestructuras como servicio (IaaS). Permite la posibilidad de utilizar servicios y tecnologías como IA y aprendizaje automático, análisis de datos, blockchain, contenedores, bases de datos, gestión de redes, servicios IoT, etc.

Entre estas características de los servicios IoT proporcionados destacan los protocolos MQTT y HTTP disponibles para transmitir la información hacia y desde la nube a los diferentes dispositivos.

La plataforma proporciona una gran cantidad de servicios no solo orientados a IoT, por lo que este análisis se va a centrar en aquellos servicios con aplicaciones a IoT. El eje central de los servicios enfocados a IoT de *IBM Cloud* es *IBM Watson IoT Platform*.

Watson ofrece una amplia gama tanto de soluciones como de servicios inteligentes. Entre ellos destacan:

- *Watson Assistant*: Permite a los usuarios realizar búsquedas rápidas en todo tipo de orígenes, conversaciones, aplicaciones, etc.

- *Watson Discovery*: Además de realizar búsquedas permite análisis de textos proporcionando estadísticas utilizando procesamiento de lenguaje natural en documentos, páginas web, resultados de búsqueda, etc.
- *IBM Máximo Application Suite*: Es un conjunto de soluciones que permiten realizar mantenimiento proactivo en vez de reactivo, es decir, utilizando técnicas de aprendizaje automático y análisis de datos, es capaz de predecir potenciales fallos e incidencias reduciendo costes de mantenimiento y reparación. Estas predicciones se realizan utilizando datos de sensores, operaciones, IT y sistemas EAM para generar y desplegar modelos utilizando *Watson Machine Learning* junto a otros servicios de monitorización, estadísticas y visualización.
- *Watson Visual Recognition*: Es un servicio que utiliza algoritmos de "D*eep Learning*" para analizar imágenes, objectos, caras, etc. Permite tanto clasificación de imágenes como detección de objetos además de dar la opción a los usuarios de crear sus propios clasificadores.

2.1.2 Bosch IoT Suite

Bosch IoT Suite es una PaaS que ofrece un conjunto de servicios y paquetes en la nube orientados a aplicaciones IoT. Proporciona facilidades de gestión de software, dispositivos y datos además de la representación digital de dispositivos mediante APIs basadas en *Eclipse Vorto* (Lenguaje de programación basado en otros lenguajes de programación como Java) para realizar la conexión y comunicación con dichos dispositivos. Entre dichos servicios destacan:

- *Bosch IoT Edge*: Sistema de herramientas y servicios integrados para realizar la conexión y comunicación entre dispositivos y la nube. A su vez está formado por dos componentes:
    - *Bosch IoT Edge Agent*: Proporciona conexión con la nube, mensajes locales, actualizaciones de software y contenedores de mantenimiento. Este componente está orientado a dispositivos por lo que es una aplicación modular que se utiliza en pequeños microcontroladores.

- *Bosch IoT Edge Services*: Proporciona servicios como estadísticas, historial además de las herramientas necesarias para desplegar los servicios necesarios para el dispositivo.

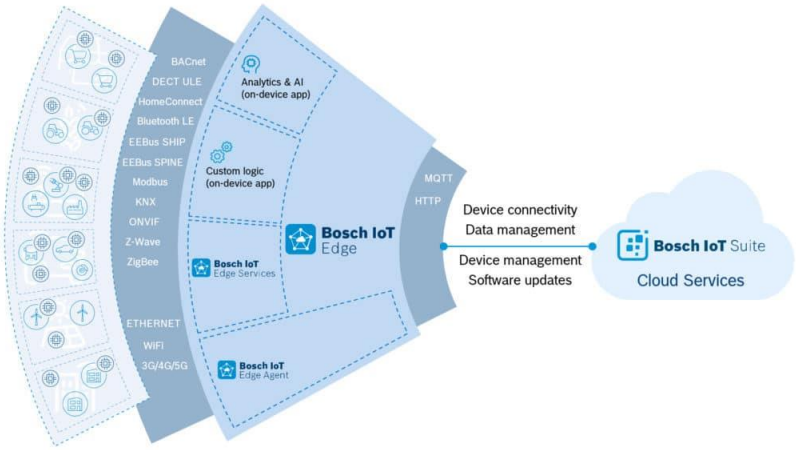


Figura 2-1. Bosh IoT Edge Components

En este sistema destaca el uso de protocolos de como MQTT y HTTP para la conexión con la nube y otros como *Z-Wave*, *Zigbee*, *IP cameras*, *Bluetooth LE*, etc. para la conexión de diferentes dispositivos IoT además permite realizar actualizaciones del software utilizando las herramientas nativas (*out of the box*), análisis de datos y servicios de inteligencia artificial.

- *Bosch IoT Hub*: Facilita la gestión de aspectos como la seguridad tanto en el transporte de la información como en la autenticación de los dispositivos permitiendo a los desarrolladores centrarse en el desarrollo de la lógica de la aplicación. La interacción con los dispositivos se produce mediante protocolos como MQTT, HTTP, LoRaWAN o CoAP además de utilizar *Bosch IoT Gateway Software* (edge-computing middleware multiplataforma) que se puede desplegar en distintos tipos de dispositivos.

  De forma gráfica podemos ver los diferentes componentes que forman *Bosch IoT Hub* y la integración con *Bosch IoT Gateway*:

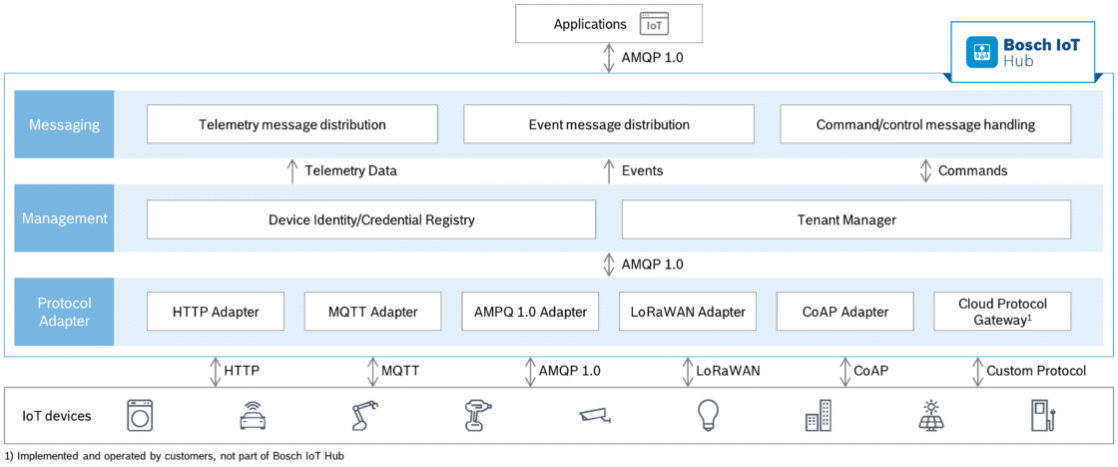


Figura 2-2. Bosch IoT Hub

En medio de la comunicación entre los dispositivos y *Bosch IoT Hub* estaría Bosch IoT Gateway:

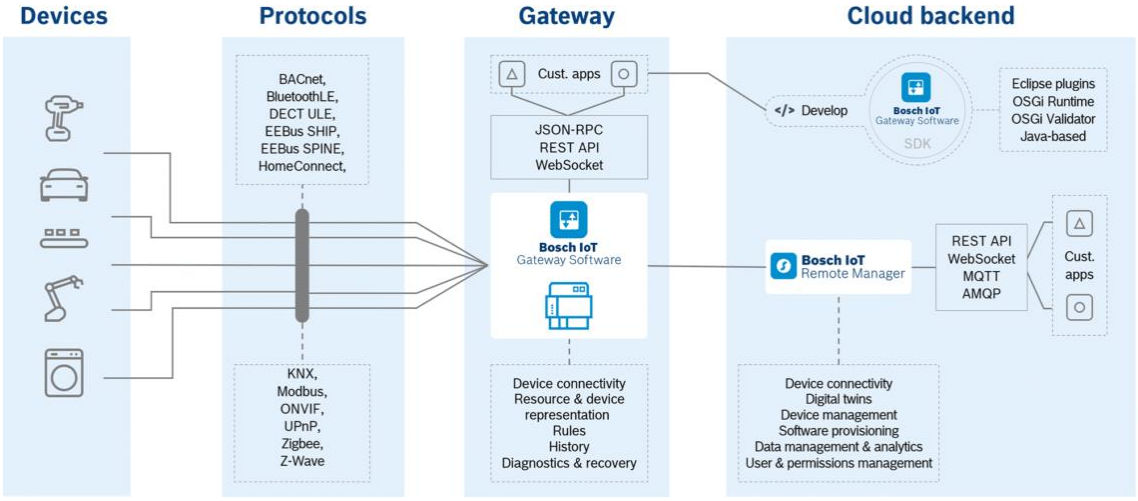


Figura 2-3. Bosch IoT Gateway

- *Bosch IoT Things*: permite gestionar la representación digital de los dispositivos registrados además de realizar lecturas y actualizaciones mediante diferentes APIs, gestión de permisos para que diferentes usuarios y aplicaciones puedan ver solo la información asociada a un rol o proceso, es decir una gestión de permisos de grano fino además de estar disponible en Bosch IoT Cloud también lo está en Microsoft Azure y en Amazon Web Services.

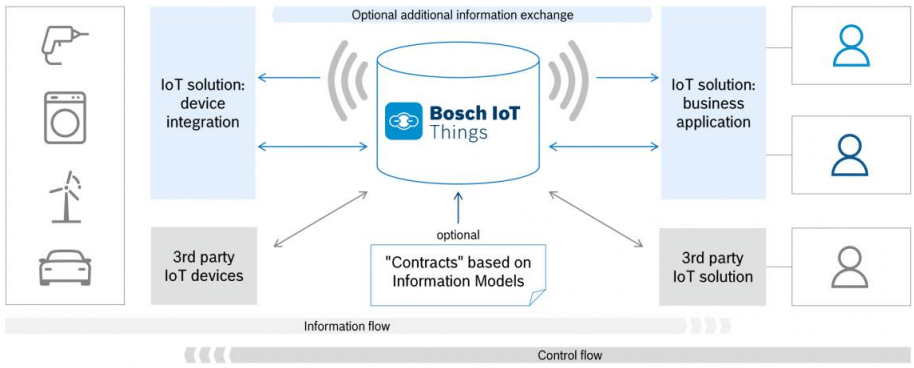


Figura 2-4. Bosh IoT Things

Como podemos comprobar Bosch IoT Suite está formada por tres componentes principales totalmente conectadas entre ellas como podemos ver en la siguiente figura.

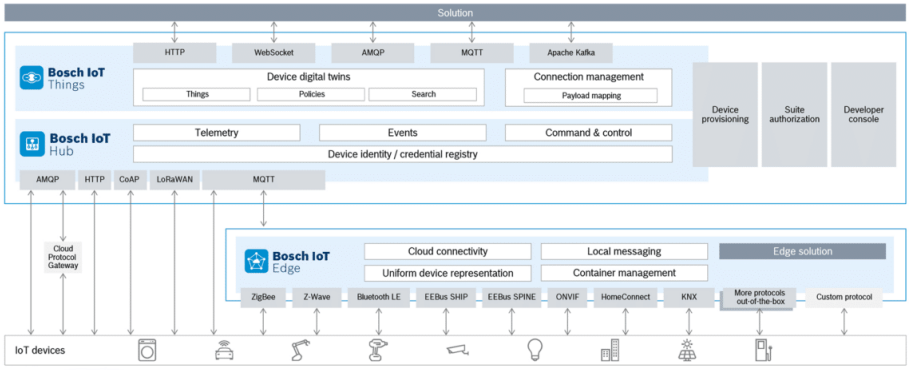


Figura 2-5. Bosh IoT Suite

2.1.3 Microsoft Azure

Microsoft Azure es una de las plataformas de servicios y productos en la nube más importantes junto con IBM Cloud, Amazon Web Services y Google Cloud. Cuenta con productos y servicios como IA, análisis de datos, contenedores, bases de datos, redes, proveedor de identidad (AAD) además de una parte de la plataforma orientada a IoT, Microsoft Azure IoT donde destacan los siguientes componentes:

- *Azure IoT Hub*: Una vez creado este recurso en la plataforma, nos ofrece una interfaz sencilla con varias características interesantes como por ejemplo la posibilidad de registrar dispositivos para obtener claves de conexión a dichos dispositivos que podrán utilizarse para obtener información de los mismos y facilitar la integración con otros servicios proporcionados por la misma

plataforma. Además, permite una simulación de costes, gestión de redes, posibilidad de crear eventos automatizados, proveedor de identidad, métricas y servicios como IoT Edge.

- *Azure IoT Edge*: permite trasladar el análisis y la lógica de negocio de la nube a dispositivos que podrán ser supervisados desde la nube lo que mejora el rendimiento ya que la información se puede procesar antes de ser enviada a la nube y por tanto filtrar de forma que se enviará solamente la información necesaria para ser analizada lo que implica una reducción de costes al reducir el procesamiento en la nube. *Azure IoT Edge* está formado por tres componentes: los contenedores que ejecutan servicios de Azure, terceros o del usuario, el entorno en tiempo de ejecución que se ejecuta en los dispositivos y la interfaz basada en la nube que permite supervisar y administrar los dispositivos de forma remota.

  En las siguientes figuras (2-6, 2-7) podemos observar el flujo de información desde los diferentes dispositivos que pasan por aplicaciones y servicios de Azure IoT Edge para finalmente enviar la información a *Azure IoT Hub* que puede consultar visualizar y administrar diferentes aspectos de los dispositivos de forma remota.

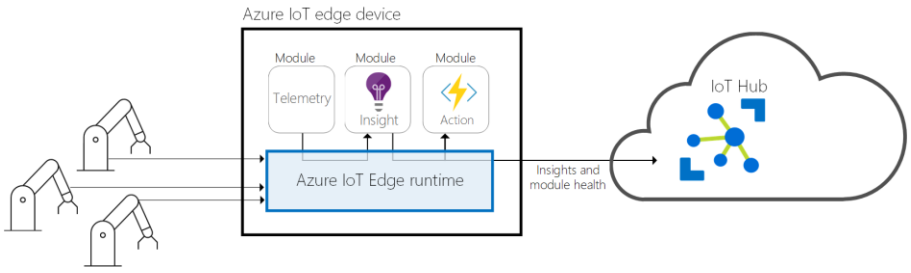


Figura 2-6. Azure IoT Edge - Hub

- *Azure IoT Central*: Es una plataforma de aplicaciones que permite conectar dispositivos y utilizar diferentes plantillas para la creación de soluciones IoT de forma sencilla simplificando la configuración y administración.

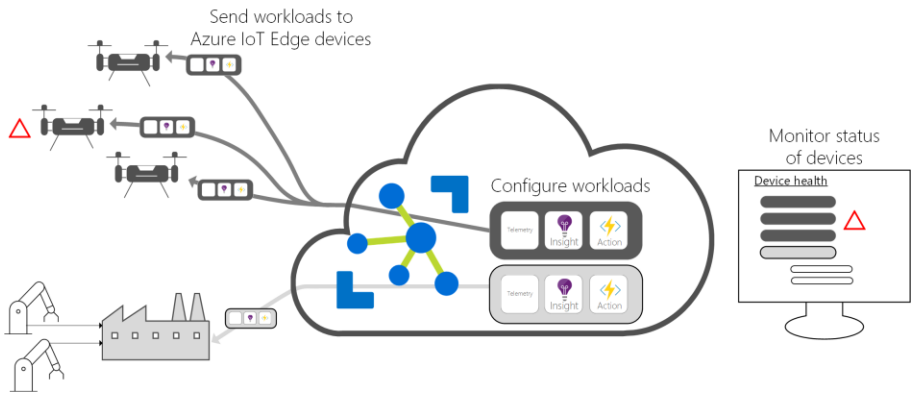


Figura 2-7. Device - Azure IoT Hub - device

Las aplicaciones disponibles se clasifican en varios grupos en función de a qué tipo de industria están orientadas: venta al por menor, energía, administración pública o salud. Entre todas ellas destaca la aplicación "Análisis de vídeo: detección de objetos y movimiento" que utiliza YOLO v3 para crear una aplicación de reconocimiento de objetos y movimientos, por ejemplo, en un pequeño comercio identificando eventos de interés mediante la supervisión de las cámaras de seguridad.

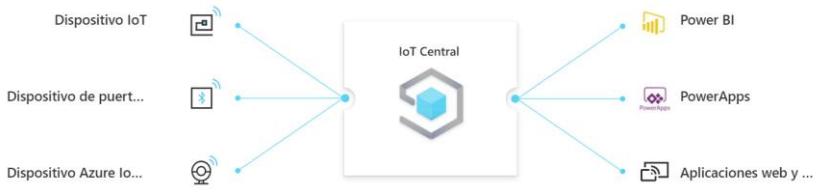


Figura 2-8. Azure IoT Central

Como se ha comentado en apartados anteriores, Azure dispone de una gran cantidad de servicios en la nube no solo en el ámbito del IoT. Entre ellos podemos destacar *Azure Function* y *Azure Static Web*:

- *Azure Function*: Es un servicio en la nube que proporciona infraestructura y recursos actualizados para desarrollar aplicaciones sin servidor basadas en eventos. Se puede utilizar para crear APIs, responder a cambios en las bases de datos, administrar colas, etc. además de permitir utilizar una amplia gama de lenguajes y entornos de programación como .NET, NodeJs, Java, Powershell, Python, etc. Dos de los aspectos más relevantes de este servicio son la duración

de la ejecución de una petición que debe durar entre cinco y diez minutos como máximo y los diferentes disparadores disponibles que facilitan la integración, de esta forma la función desarrollada puede comenzar a ejecutarse cuando se recibe alguno de estos eventos: una llamada HTTP, se ha añadido un nuevo elemento en una cola de Azure, se ha configurado la ejecución temporizada (ejecución en un día concreto de la semana, mes, cada día a una hora específica, etc.), responder a los eventos enviados desde una secuencia de eventos del centro de eventos, etc. Además, este servicio permite la fácil configuración de autenticación utilizando AAD, configuración de variables de entorno, configuraciones de seguridad utilizando TLS y SSL, gestión de redes, CORS, estadísticas mediante *Azure Insights* y control de acceso.

- *Azure Static Web*: Es un servicio orientado al desarrollo e implementación de aplicaciones web permitiendo la utilización de lenguajes de programación como JavaScript, TypeScript, Python y C# lo que implica la posibilidad de utilizar frameworks populares como React, Angular o Vue.js que destacan en el marco del desarrollo de aplicaciones web. En definitiva, proporciona hospedaje web para contenido estático, compatibilidad con la API integrada proporcionada por *Azure Function*, integración con GitHub permitiendo la creación y configuración de ciclos de integración y entrega continua de forma que al aceptar un nuevo cambio en una determinada rama del repositorio de GitHub se desencadena de forma automática la compilación del código de esa rama y el despliegue al entorno de producción, sin intervención humana, integración con AAD, Facebook, Google y Twitter, definición de roles de autorización personalizables, reglas de enrutamiento, etc. En la siguiente figura (2-9) podemos observar el flujo de las acciones configuradas para la integración y despliegue continuos desde la realización de un cambio hasta el despliegue final. Como se ha comentado anteriormente, este proceso está totalmente automatizado.

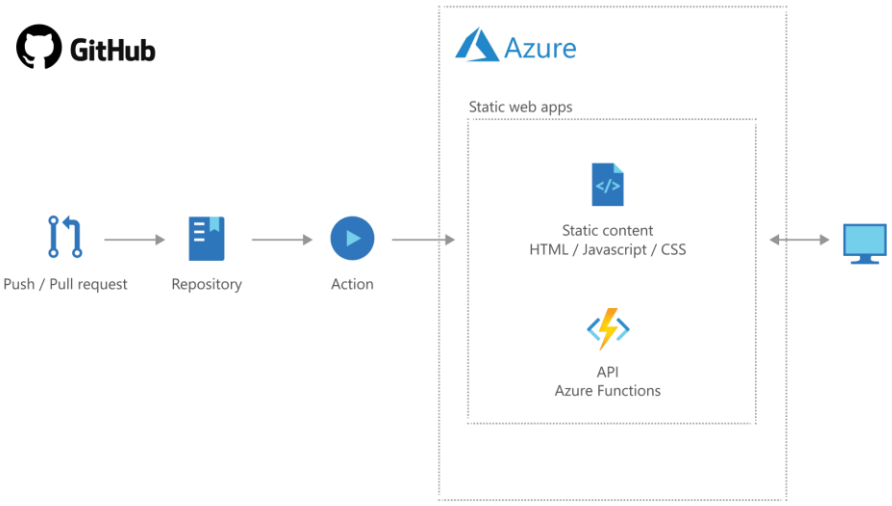


Figura 2-9. CI/CD GitHub - Azure Static Web

- Azure Blob Storage: Es un servicio que forma parte de Azure Storage justo a Queue Storage, Table Storage, y File Storage que permite el almacenamiento de objetos en la nube de Microsoft como se muestra en la figura 2-10. Este servicio está optimizado para almacenar cantidades masivas de datos no estructurados, además está diseñado para visualizar imágenes o documentos directamente en un explorador, almacenamiento distribuido, facilidades para generar copias de seguridad y restauración de archivos. Permite el acceso a los objetos metiante HTTP y HTTPS mediante la API REST de Azure Storage, Azure Powershell, la CLI de Azure o cualquier biblioteca cliente de lenguajes como .NET, Java, NodeJs, Python, etc. Como se puede observar en la figura 2-10, se necesita una cuenta de almacenamiento en la nube de Azure con la que se va a crear un contenedor donde se alojarán los archivos y sus correspondientes metadatos.

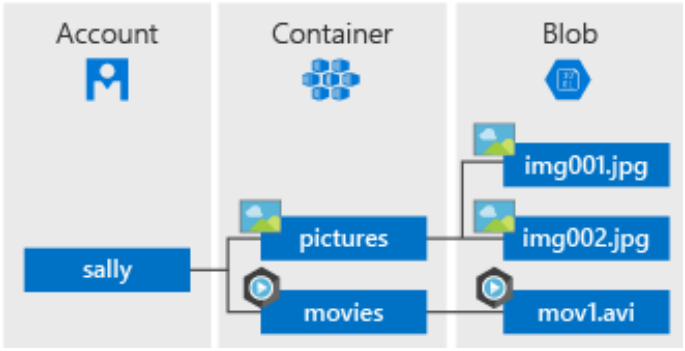


Figura 2-10. Azure Blob Storage

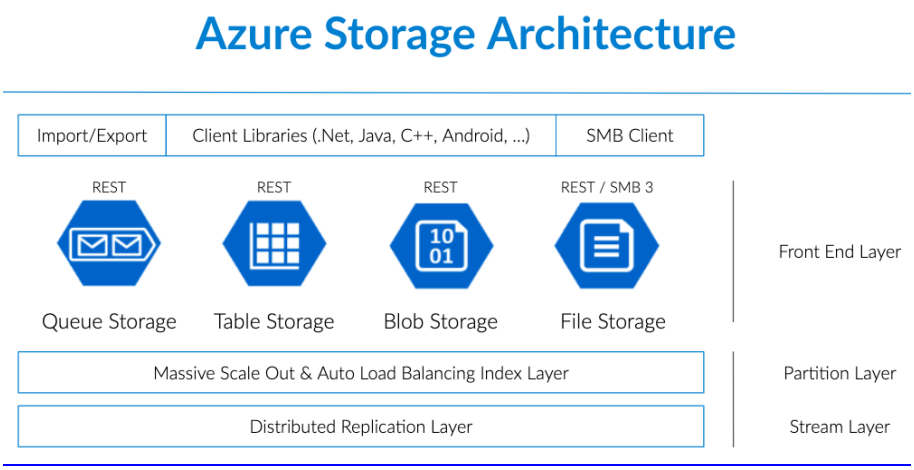


Figura 2-11. Arquitectura Azure Storage

Como se puede comprobar, estos servicios serían de gran utilizad dentro del sistema IoT a desarrollar para automatizar la clasificación de residuos en una planta de clasificación, utilizando por ejemplo *Watson Assistant* y *Watson Discovery* para permitir a los usuarios realizar consultas de forma ágil mediante comandos de voz a la vez que recabamos estadísticas tanto de las búsquedas de los empleados como de los sensores situados en la planta que transmitirán los datos mediante el protocolo MQTT a la nube de IBM para que otros servicios y soluciones como *Visual Recognition* e *IBM Maximo* realicen entrenamientos de nuevos modelos o mejorando los antiguos para que se obtengan mejores resultados por ejemplo en la predicción de futuras incidencias o mejorar las predicciones en la clasificación de residuos reduciendo los errores.

Para llevar a cabo el desarrollo de la aplicación web y del servicio que permite la clasificación de residuos se utilizará Microsoft Azure debido a los costes de licenciamiento favorables para estudiantes, así como por la infraestructura y los servicios disponibles como Azure Function, Azure Static Web Apps y Azure Blob Storage además de la fácil integración con GitHub para establecer la integración y entrega continua (CI/CD) del software desarrollado.

Para la implementación del primer escenario propuesto al comienzo de este documento se va a utilizar también IBM Cloud, concretamente se desplegará una aplicación de tipo Node-RED y se hará uso de IBM Watson Platform IoT para la comunicación bidireccional entre el dispositivo y la plataforma, así como la utilización de llamadas HTTP desde la aplicación de Node-RED al servicio de Azure.

Finalmente, para el segundo escenario planteado solo se utilizará IBM Cloud con la aplicación de Node-RED e IBM Watson.

El uso de estas dos plataformas se debe en gran medida a la facilidad con la que permiten crear y desplegar servicios con planes de estudiante o de pruebas y a su utilización en otros ámbitos fuera de este proyecto.

2.2 Análisis de datasets

Antes de comenzar con el desarrollo del sistema de clasificación de residuos aparece la necesidad de una gran cantidad de imágenes para poder crear y entrenar el modelo de redes neuronales que realizará la clasificación de los diferentes tipos de residuos. Para ello se han empleado herramientas como "*Dataset Search*" o "*Google Research*" que permiten realizar búsquedas por palabras clave o tipo de Dataset.

Gracias a estas dos herramientas se han analizado algunos datasets que podrían ser relevantes a la hora de entrenar un modelo de clasificación de residuos. Entre ellos destacan:

- *Open Images Dataset v6*: Cuenta con alrededor de nueve millones de imágenes que cuentan con etiquetas, object bounding boxes, object segmentation masks, entre otras características. Este conjunto de datos presenta una amplia diversidad de imágenes que podemos explorar de forma manual además de ofrecer gráficos para comprender la distribución de las imágenes como podemos ver en las siguientes figuras (2-10,2-11).

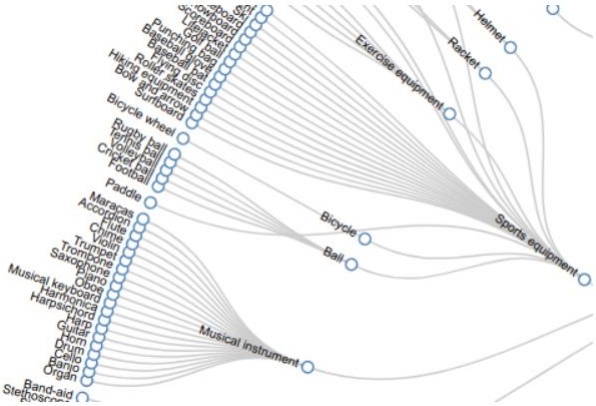


Figura 2-12. Jerarquía de clases

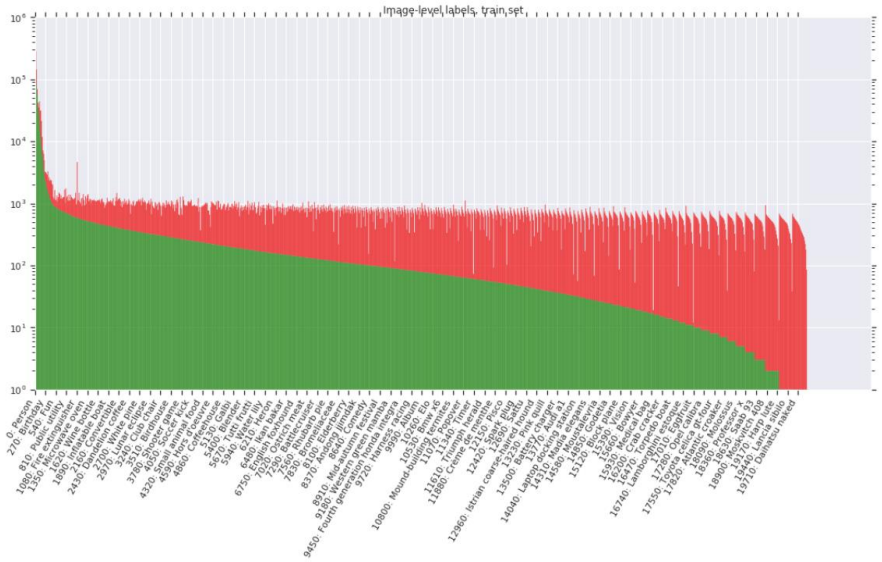


Figura 2-13. Distribución de las anotaciones de etiquetas de la versión 5 del Dataset

Hay que tener en cuenta que las anotaciones del conjunto de datos han sido licenciadas por Google LLC bajo CC BY 4.0 y las imágenes cuentan con una CC BY 2.0 además no se asegura que todas las imágenes cuenten con una licencia CC delegando ese trabajo en los usuarios o entidades que las utilicen.

- *Kaggle Datasets*: Es una plataforma gratuita que pone a disposición de los usuarios problemas a resolver en el ámbito de la inteligencia artificial, ciencia de datos, etc. en el contexto de una competición donde el creador de dicha competición pone a disposición de los usuarios (concursantes) un conjunto de datos y una descripción de un problema. Los concursantes por su parte utilizan el conjunto de datos proporcionado para resolver de la mejor forma el problema detallado. Como resultado de estas competiciones en la plataforma se van generando diferentes tipos de datasets con diferentes licencias. Entre los datasets encontrados en esta plataforma destacan:
    - *Waste classification data*: Proporciona un conjunto de datos formado por 22.564 imágenes de entrenamiento y 2513 imágenes de prueba que están divididas en dos clases, residuos orgánicos y residuos inorgánicos.
    - *Waste classification data v2*: Versión mejorada de Waste classification data que cuenta con aproximadamente 22900 imágenes de

entrenamiento y 2910 imágenes de prueba divididas en tres clases residuos orgánicos, plástico y papel.

- *Drinking Waste classification*: Esta formado por dos partes: "YOLO_imgs" que contiene alrededor de 9640 imágenes sin clasificar en subcarpetas y "rawimgs" que contiene a su vez alrededor de 4700 imágenes divididas en cuatro clases: latas de aluminio, botellas de cristal, botellas de plástico PET y botellas de plástico HDPE.
- Garbage Classification: Dataset formado por 2527 imágenes divididas en 6 clases, cartón, vidrio, metal, papel, plástico y basura.

Finalmente, para el desarrollo del sistema propuesto en capítulos anteriores se va a crear un nuevo conjunto de datos de forma manual que encaja mejor con las necesidades del problema expuesto ya que los diferentes datasets analizados no tenían una licencia CC o bien no se ajustaban a las necesidades descritas, aunque se han realizado diferentes pruebas con el dataset propio y el dataset Garbage Classification.

2.3 Modelo de Red

A continuación, se describe el modelo de red utilizado en la aplicación, junto con las unidades y operaciones básicas fundamentales que las definen. En la figura 2-14, se muestra a alto nivel el esquema general de la red utilizada.

La imagen de entrada conteniendo los objetos a clasificar es recibida en lo que se puede denominar Módulo Base, consistente en un modelo específico basado en la arquitectura Mobilenet V2, con la particularidad de que sus pesos han sido previamente entrenados con el conjunto de datos descrito en Imagenet (2021), tratándose de un conjunto de datos que ha sido ampliamente utilizado para el entrenamiento de distintos modelos tal como en AlexNet descrito en Krizhevsky y col. (2012). Las imágenes de entrada son de dimensión 224×224×3.

La salida en forma del correspondiente mapa de características producida por el módulo Base se envía a una capa de convolución seguida de una operación ReLU. A su vez, el mapa de características de esta capa se envía a una segunda unidad con las mismas operaciones que la anterior. Pasando a continuación por una unidad Dropout, seguida de una operación de agrupamiento o pooling, en este caso de tipo

promediado (average), para finalizar con el paso por la función softmax para determinar la probabilidad de pertenencia de la imagen de entrada a una de las clases establecidas, tal y como se define más adelante. A continuación, se describen los módulos y estructuras involucradas en el modelo. No obstante, dado que la red Mobilenet V2 contiene en sí misma distintas operaciones tales como Convolución, ReLU, pooling se comienza por describir las operaciones básicas de esta naturaleza, dejando para el final la descripción del modelo Mobilenet V2 (Pajares y col., 2021).

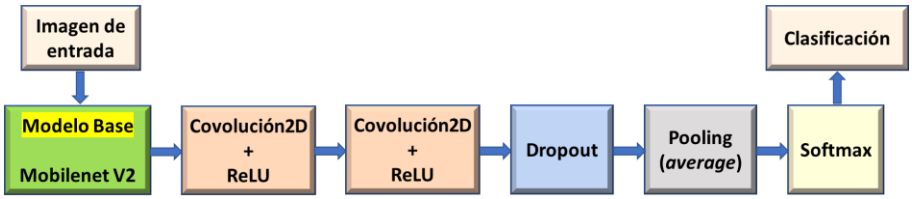


Figura 2-14. Esquema general del modelo de red

2.4 Operaciones en redes neuronales convolucionales

2.4.1 Convolución

La convolución es una operación que se define como sigue para el procesamiento de imágenes, que como se sabe son estructuras bidimensionales 2-D,

$$C(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \qquad (2\text{-}1)$$

El resultado de la convolución es C(i,j) para un determinado píxel (i,j) cuando la entrada es una imagen "I" o bien un elemento de un tensor cuando se trata de capas más internas. En la ecuación anterior "K" hace referencia a lo que se denomina núcleo de convolución. La aplicación de esta expresión al contexto de las imágenes resulta en el esquema gráfico mostrado en la figura 2-15. El núcleo de convolución K con sus correspondientes valores, se posiciona sobre la cuadrícula superior izquierda para obtener el resultado P que se posiciona en la cuadrícula superior izquierda.

A continuación, se desplaza el núcleo una posición hacia la derecha para obtener de forma similar el resultado Q y así sucesivamente se desplaza el núcleo de izquierda a derecha y de arriba hacia abajo hasta completar los valores de las celdas en la matriz resultante.

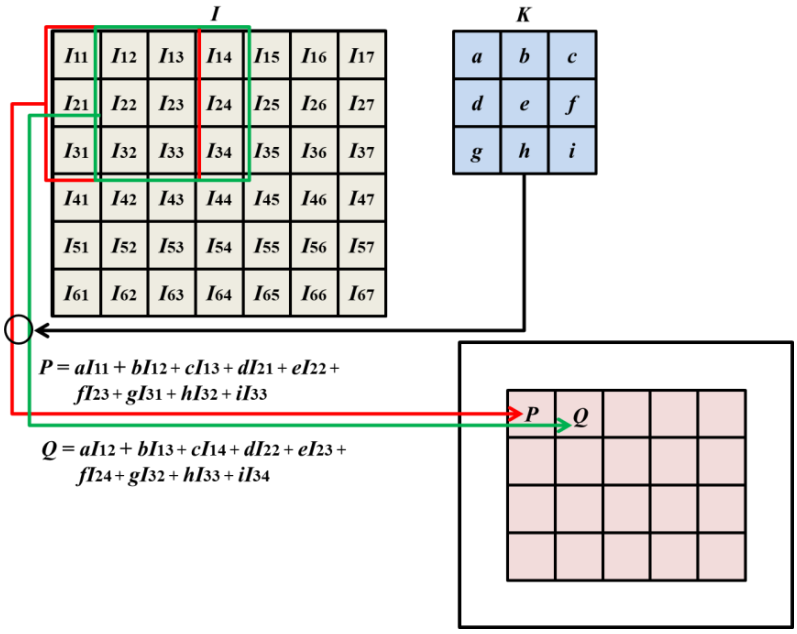


Figura 2-15. Ejemplo de convolución 2-D

No obstante, si este núcleo en lugar de desplazarse una única celda se desplaza más de una, es necesario establecer lo que se conoce como stride (s) que puede tomar valores mayores que la unidad. Además, como el núcleo puede desbordar la imagen o tensor de entrada cuando la celda correspondiente al valor e del núcleo K se sitúa en las filas o columnas más exteriores, existen valores del núcleo que no se utilizan, en cuyo caso estas filas o columnas quedan sin procesar. Por esta razón, si se desea que el resultado tenga las mismas dimensiones de la imagen, es necesario añadir filas y columnas rellenas de ceros, a esta operación se le conoce como padding (p). En función de los valores s y p para una imagen o mapa de características de entrada con dimensión i y para valores dados de p y s se obtiene la correspondiente dimensión de salida o según las siguientes relaciones. En estas expresiones k es la dimensión del núcleo de convolución, que por ejemplo en el caso del ejemplo anterior, k = 3.

$$\textit{Relación 5: } o = \left[\frac{i-k}{s}\right] + 1 \qquad \textit{Relación 6: } o = \left[\frac{i+2p-k}{s}\right] + 1 \qquad (2\text{-}2)$$

El modelo de red Mobilenet, tanto en su versión V1 como V2 aplican un tipo especial de convoluciones: convolución en profundidad (*depth-wise*) y convolución punto a punto (*point-wise*).

En la figura 2-16 se muestra el esquema de la convolución del tipo *depth-wise* donde la convolución se aplica a un único canal a la vez. En relación con esta figura el mapa de características sobre el que se aplica la operación de convolución está constituido por un volumen de dimensiones H×W×C, donde H y W son el alto y ancho respectivamente y C se refiere a la profundidad, que en definitiva identifica lo que se conoce como número de canales. Los núcleos de los filtros son de dimensión hk×wk×1 y dado que existen C canales, entonces se requieren C filtros. El mapa de características de salida será de dimensión Ho×Wo×C dependiendo del tipo de relación aplicada según la ecuación (2-2).

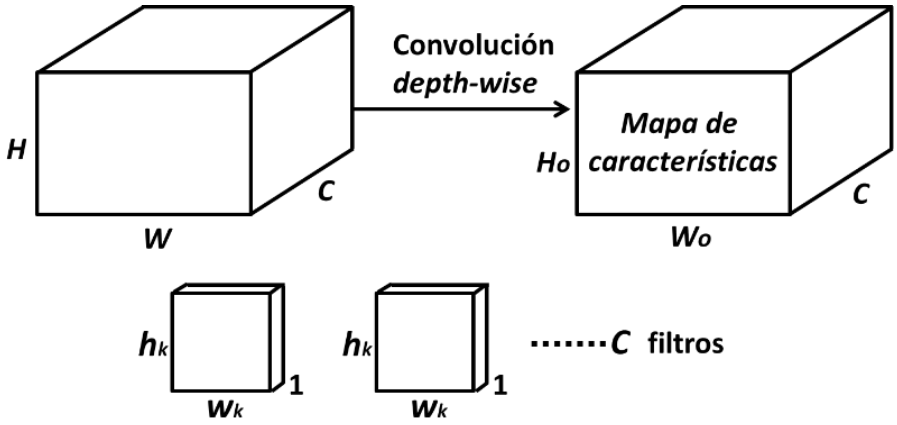


Figura 2-16. Convolución deph-wise

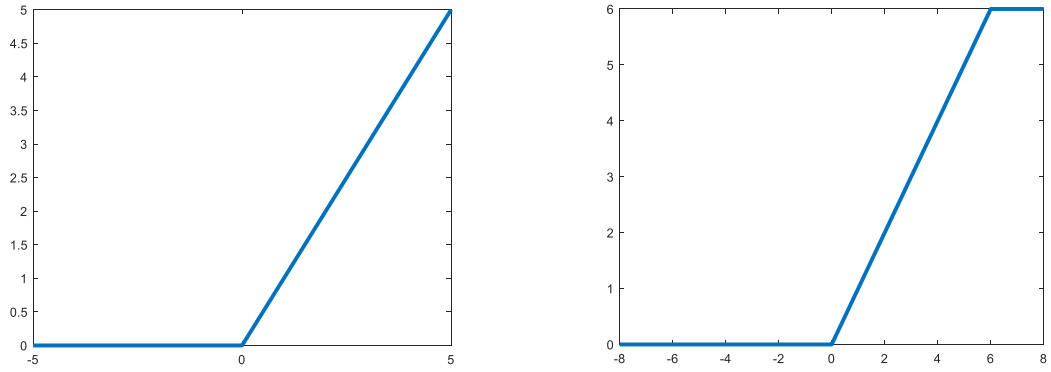
En la operación de convolución *point-wise* se aplica una operación de convolución 1×1 en los C canales. Por tanto, el tamaño de cada filtro para esta operación será de dimensión 1×1×C, de forma que, si se utilizan N filtros de este tipo, el tamaño de salida resulta ser Ho×Wo×N. En definitiva, cuando se aplica a las imágenes, una convolución de este tipo proyecta un píxel a lo largo de todos los canales a un único valor de salida sin tener en cuenta su vecindad. Se utiliza para aumentar o reducir el número de canales gracias al número de filtros N, de forma que si N < C se produce una reducción y si N > C, una expansión.

2.4.2 ReLu

Esta función conocida como Unidad Lineal Rectificada (ReLU, Rectified Linear Unit) se define como f(x) = max(0, x), cuya representación es la que se muestra en la figura 2-17(a), cuyas características y su utilidad se centran en lo siguiente:

- Evitar la saturación del gradiente durante el proceso de optimización por el método del gradiente descendente.
- Disminuir la complejidad computacional por el hecho de reducir valores negativos a cero.

En el caso del modelo de red MobilnetV2 se utiliza una variante de ReLU, concretamente ReLU6 definida como: $ReLU6(x) \equiv f(x) = min(max(x,0),6)$ cuya representación se muestra en la figura 2-17 (b).



(a) (b)

Figura 2-17. Representación de las funciones (a) ReLU; (b) ReLU6

2.4.3 Normalización

La operación de normalización se aplica con el fin de acelerar la convergencia durante el proceso de aprendizaje. En la expresión siguiente se define una función de normalización por lotes,

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^i_{x,y} \right)^2 \right)^{\beta} \qquad (2\text{-}3)$$

donde N es el número de filtros de la capa dada, $a^i_{x,y}$ es la actividad de la neurona y *k*, α, n, β son hiperparametros. En Krizhevsky (2012) se proponen como más apropiados los siguientes valores para los parámetros *k* = 2, *n* = 5, $\alpha = 10^{-4}$, β = 0.75.

2.4.4 Pooling

Se trata de una función cuyo objeto es agrupar valores en las capas de características, de tal forma que dada una pequeña traslación en la entrada no afecte a los valores de las salidas. En la figura 2-18 se muestra un ejemplo gráfico de agrupamiento de tipo promedio (average), es decir seleccionando el valor medio en cada una de las ventanas de agrupamiento, en este caso de dimensión 2×2 y aproximando el resultado final al entero más próximo.

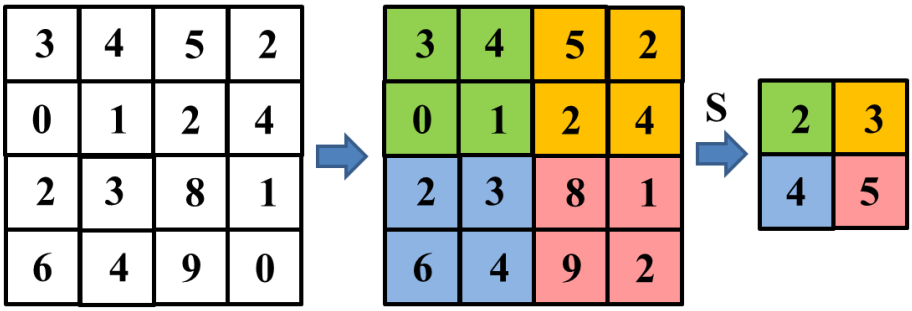


Figura 2-18. Average pooling

2.4.5 Dropout

Es un mecanismo para evitar sobresaturaciones en la red neuronal. Consiste en anular determinado tipo de neuronas de forma aleatoria mediante un hiperparámetro que indique la probabilidad de supervivencia de cada neurona.

2.4.6 Softmax y entropía cruzada

Consistente en proyectar los valores de la salida en la capa final al rango [0,1], proporcionando así una especie de probabilidad de pertenencia a cada una de las clases para una imagen de entrada dada. Su función se define según la siguiente expresión.

$$softmax(\boldsymbol{x})_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)} \quad (2\text{-}4)$$

Relacionado con este concepto, se puede definir la *función de pérdida* definida como *entropía cruzada* (*cross entropy*), que tiene en cuenta dos distribuciones de probabilidad. Una relacionada con los valores predichos $\boldsymbol{q} \equiv \{q_1, q_2, ..., q_n\}$ de forma que $\sum_i^n q_i = 1$, y la otra con los verdaderos (*ground truth*) $\boldsymbol{p} \equiv \{p_1, p_2, ..., p_n\}$ cumpliéndose igualmente $\sum_i^n p_i = 1$. De esta forma, la entropía cruzada resulta ser $H(\boldsymbol{p}, \boldsymbol{q}) = -\frac{1}{n}\sum_i^n p_i \log(q_i)$.

2.5 Modelo de red MobileNetV2

Sandler y col. (2018) proponen el modelo de red denominado MobileNetV2, cuyo objetivo principal, como en la versión v1, pero con mayor énfasis, consiste en reducir las necesidades computacionales y de memoria para su utilización en dispositivos con capacidades limitadas, dentro de los que se incluyen, como es lógico, los dispositivos móviles. Para ello diseñan una estructura integrada por varias operaciones que denominan capa residual invertida de cuello de botella lineal (Inverted Residual and Linear Bottleneck Layer). La premisa consiste en considerar que es posible codificar los mapas de características en subespacios de baja dimensión y que las activaciones no lineales conllevan una cierta pérdida de información, a pesar de su capacidad para aumentar la complejidad de representación.

Los bloques de cuello de botella siguen la filosofía del bloque residual que aparece en los modelos del tipo ResNet (He y col., 2016), inspirados por la intuición de que los cuellos de botella realmente contienen toda la información necesaria. La implementación básica de un bloque residual cuello de botella se muestra en la figura 2-19, junto con la tabla 2-1 de transformación correspondiente, de manera que una entrada con k canales se transforma en k' canales con stride s y factor de expansión t.

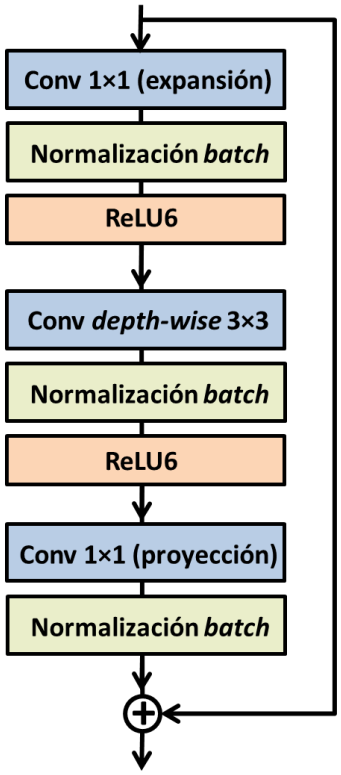


Figura 2-19. Bloque residual cuello de botella

Entrada	Operador	Salida
$h \times w \times k$	Conv 1×1, ReLU6	$h \times w \times tk$
$h \times w \times k$	Conv 3×3, depth-wise, s = a, ReLU6	$h/s \times w/s \times tk$
$h/s \times w/s \times tk$	Conv 1×1 lineal	$h/s \times w/s \times k'$

Tabla 2-1. Tabla de transformación del bloque residual cuello de botella

Como puede observarse el bloque residual consta de tres capas. Las dos últimas son ya conocidas, concretamente una capa de convolución *depth-wise* que filtra las entradas, seguida de una capa de convolución *point-wise* de dimensión 1×1. Esta última capa tiene ahora el cometido especial de reducir el número de canales. Este es el motivo por el cual esta capa se conoce como *capa de proyección*, ya que proyecta los datos con un alto número de dimensiones (canales) en un tensor con un número menor de dimensiones. Por ejemplo, la capa *depth-wise* puede operar sobre un tensor con 144 canales, para proyectarlos a 24. Así, esta capa se conoce como *capa cuello de botella* debido a que reduce la cantidad de datos que fluye a través de la red, de ahí el nombre del bloque. La primera capa consta de una convolución de dimensión 1×1, cuya

finalidad consiste en expandir el número de canales en los datos con respecto a los de entrada, antes de enviarlos a la convolución *depth-wise*, de aquí el nombre de *capa de expansión*, produciendo justo el efecto contrario a la de proyección. El factor de expansión determina la magnitud del resultado, habiendo sido fijado a 6 en Sandler y col. (2018) tras diversos experimentos. Por ejemplo, considerando los parámetros de la figura 2-20 y la tabla 2-1, para un bloque de cuello de botella que toma un tensor de entrada de $k = 64$ canales, con el valor indicado de dicho factor produce una expansión $tk = 64 \times 6 = 384$; luego, la convolución *depth-wise* aplica un filtro a los 384 canales del tensor para obtener a la salida de nuevo los 384 canales (tk); finalmente, la capa de proyección genera una salida con un número menor de canales (k'), por ejemplo 64 otra vez, de forma que puede o no producir a la salida el mismo número de canales que a la entrada.

Con relación al coste computacional y teniendo en cuenta el mismo criterio para los módulos del tipo separables *depth-wise*, este resulta ser $H \cdot W \cdot C \cdot (k_h \cdot k_w + N)$ tal y como se indicó previamente, donde H y W son las dimensiones alto y ancho del bloque con C canales con dimensiones de los filtros k_h y k_w (alto y ancho), siendo N el número de filtros en la convolución *depth-wise*. Para un bloque de dimensiones H, W con C_i canales de entrada y tamaños de los filtros k_h y k_w y para un factor de expansión t, generando finalmente C_o canales de salida se tiene $H \cdot W \cdot C_i \cdot t \cdot (C_i + k_h \cdot k_w + C_o)$, que comparada con el cómputo anterior es ligeramente superior debido al término extra relativo a la convolución de dimensión 1×1; si bien como indican Sandler y col. (2018) dada la naturaleza de la red, que permite utilizar dimensiones de entrada-salida mucho más pequeñas, este coste computacional extra queda compensado.

Por otra parte, una propiedad interesante de la arquitectura de bloques propuesta es que proporciona una separación natural entre los dominios de entrada y salida de los bloques cuello de botella y la capa de transformación final, que es una función no lineal que convierte la entrada en la salida. Lo primero puede verse como la *capacidad* de la red en cada capa, mientras que esto último se considera la *expresividad*. Esto contrasta en cierto modo con los bloques convolucionales tradicionales, tanto convencionales como separables, donde tanto la expresividad como la capacidad están entremezcladas y son funciones de la profundidad de la capa de salida.

En particular, en el caso analizado en Sandler y col. (2018), cuando la profundidad de la capa interna es 0, la convolución subyacente es la función identidad gracias a la conexión de acceso directo. Cuando la razón de expansión es menor que 1, se trata de una capa de convolución residual clásica. Si bien, en general, lo más apropiado es utilizar razones de expansión mayores que 1. La interpretación anterior permite estudiar la expresividad y la capacidad de la red por separado, lo que conlleva un mejor análisis de sus propiedades.

La arquitectura de red MobileNetV2 propuesta por Sandler y col. (2018) se muestra en la tabla 2-2. Cada línea describe una secuencia de 1 o más capas idénticas (módulo *stride*), repetidas *n* veces. Todas las capas en la misma secuencia tienen el mismo número *c* de canales de salida. La primera capa de cada secuencia tiene un *stride* (*s* = *a*) y todas las demás de la secuencia poseen un *stride*, *s* = 1. Así, teniendo en cuenta que cada línea es una secuencia, fijándose por ejemplo la fila 3 se observa que posee 2 secuencias 112×112×16, por lo que para la primera *s* es igual a 2 y para la segunda, 1. Todas las convoluciones espaciales usan núcleos de dimensión 3×3. El factor de expansión *t* siempre se aplica al tamaño de entrada como se indica en la tabla 2-2.

#	**Entrada**	**Operador**	***t***	***c***	***n***	***s***
1	224×224×3	Convolución	-	32	1	2
2	112×112×32	Cuello botella	1	16	1	1
3	112×112×16	Cuello botella	6	24	2	2
4	56×56×24	Cuello botella	6	32	3	2
5	28×28×32	Cuello botella	6	64	4	2
6	14×14×64	Cuello botella	6	96	3	1
7	14×14×96	Cuello botella	6	160	3	2
8	7×7×160	Cuello botella	6	320	1	1
9	7×7×320	Convolución 1×1	-	1280	1	1
10	7×7×1280	*Pooling* (*average*) 7×7	-	-	1	-
11	1×1×1280	Convolución 1×1	-	*k*	-	-

Tabla 2-2. Arquitectura del modelo MobileNetV2

2.6 Entrenamiento del modelo

El objetivo final en este modelo consiste en asignar cada imagen de entrada una etiqueta dada en función de la probabilidad obtenida mediante la función softmax. A partir de la probabilidad calculada en relación con la probabilidad esperada para cada píxel, dado que el entrenamiento está basado en un conjunto de imágenes etiquetadas, se define lo que se conoce como función de pérdida (loss), tal y como se ha definido previamente, con la que se determina el error cometido en cada caso. Este error es el que se propaga hacia atrás en la red (retropropagación), permitiendo actualizar los pesos involucrados en las distintas capas del modelo. En esta actualización interviene también lo que se denomina razón de aprendizaje (learning rate), que puede ser fija a lo largo de todo el proceso o irse decrementando a medida que se avanza en las iteraciones. Dicha actualización se lleva a cabo mediante el correspondiente proceso de optimización, durante el cual las imágenes se procesan por lotes (batch). A continuación, se definen brevemente los conceptos mencionados (Pajares y col., 2021) en relación con la fase de entrenamiento del modelo.

2.6.1 Optimización

El gradiente descendente, que es una técnica de minimización utilizada para el ajuste de los pesos involucrados en los modelos de las redes durante el proceso de retropropagación. La función que optimizar se conoce como función *objetivo* y cuando se está minimizando, se le denomina también *loss function* o *error function*. Se trata de minimizar una función objetivo que tiene la forma,

$$J(\boldsymbol{w})=\frac{1}{n}\sum_{i=1}^{n}J_i(\boldsymbol{w}) \qquad (2\text{-}5)$$

donde el parámetro w que minimiza $J(w)$ debe estimarse, constituyendo el objetivo principal del aprendizaje. J_i se asocia con la i-ésima observación en el conjunto de datos utilizados para el entrenamiento (ajuste). El gradiente descendente se usa para minimizar esta función de forma iterativa, con iteraciones t,

$$\boldsymbol{w}(t+1)=\boldsymbol{w}(t)-\varepsilon\frac{1}{n}\sum_{i=1}^{n}\nabla J_i(\boldsymbol{w}) \qquad (2\text{-}6)$$

De esta forma iterativa el método recorre el conjunto de entrenamiento y realiza la actualización anterior para cada muestra de entrenamiento. Se pueden realizar varios pasos sobre el conjunto de entrenamiento hasta que el algoritmo converja. Estas muestras así seleccionadas constituyen lo que se denomina ***batch*** como se describe más adelante a la hora de seleccionar los parámetros de entrenamiento. En este sentido, es habitual utilizar exactamente la técnica conocida como Gradiente Descendente Estocástico (SGD, *Stochastic Gradient Descent*) (Bishop, 2006; Ruder, 2017).

Las implementaciones típicas pueden usar una razón de aprendizaje adaptativa para que el algoritmo converja. En pseudocódigo, el método de gradiente descendente estocástico es como sigue,

- Elegir un vector inicial de parámetros *w* (puede ser aleatoriamente) y razón de aprendizaje ε .
- Repetir hasta que se consigue un mínimo aproximado:
    - Seleccionar aleatoriamente ejemplos en el conjunto de entrenamiento
    - Para i = 1, 2, … ,n, hacer

Existen diversas variantes, extensiones y mejoras del algoritmo. Resulta de particular interés el hecho de fijar la razón de aprendizaje, ya que valores altos tienden hacia la divergencia de los datos, mientras que valores demasiado pequeños hacen que la convergencia sea lenta. Una forma de abordar este problema es que dicha razón sea variable, disminuyendo a medida que se incorporan nuevas muestras (mayor aprendizaje). Esto se consigue haciéndola dependiente del número de datos o iteraciones. Una de tales extensiones es la del método del *momentum* o *momento* (Rumelhart y col., 1986; Murphy, 2012). Este método recuerda la actualización $\Delta \boldsymbol{w}$ en cada iteración, determinando la siguiente actualización como una combinación lineal del gradiente y la actualización previa (Sutskever y col., 2013):

$$\begin{aligned} \Delta \boldsymbol{w}(t) &= \alpha \Delta \boldsymbol{w}(t-1) - \varepsilon \nabla J_i(\boldsymbol{w}(t-1)) \\ \boldsymbol{w}(t) &= \boldsymbol{w}(t-1) + \Delta \boldsymbol{w}(t-1) \end{aligned} \qquad (2\text{-}7)$$

La anterior expresión conduce a:

$$\boldsymbol{w}(t) = \boldsymbol{w}(t-1) + \alpha \Delta \boldsymbol{w}(t-1) - \varepsilon \nabla J_i(\boldsymbol{w}(t-1)) \qquad (2\text{-}8)$$

donde es la razón de aprendizaje, es el momento constante que controla la velocidad de actualización de $\Delta \boldsymbol{w}(t-1)$. El vector de parámetros, w, que minimiza J(w) es realmente el que se estima en el proceso de optimización. El nombre de momento proviene del concepto de momento en física de forma que el vector de pesos w, visto como una partícula viajando a través del espacio de parámetros, adquiere una aceleración a partir de la fuerza, tendiendo a mantenerse en la misma dirección, evitando así oscilaciones.

Adam, cuyo nombre deriva de *Adaptive Moment Estimation* (Kingma y Ba, 2015). Mantiene una actualización de la media móvil, elemento a elemento, tanto de los gradientes de los parámetros como de sus valores al cuadrado.

$$m(t) = \beta_1 m(t-1) + (1-\beta_1)\nabla J_i(\boldsymbol{w}(t-1)$$

$$v(t) = \beta_2 v(t-1) + (1-\beta_2)\left(\nabla J_i(\boldsymbol{w}(t-1)\right)^2$$

$$\boldsymbol{w}(t) = \boldsymbol{w}(t-1) - \frac{\alpha m(t-1)}{\sqrt{v(t-1)} + \varepsilon} \qquad (2\text{-}9)$$

2.6.2 Razón de aprendizaje (learning rate)

Como se ha indicado previamente, determina la rapidez con la que la red actualiza o ajusta los pesos involucrados. En el diseño propuesto se establece una razón de aprendizaje variable, de forma que sobre la razón fijada inicialmente cada cierto número de epochs se aplica un determinado factor de reducción.

2.6.3 Batch Size

Durante el proceso de actualización de los pesos de la red, mediante el cálculo del error y su retropropagación mediante la técnica del gradiente descendente se dispone de N imágenes de entrenamiento en un proceso iterativo. Esto significa que al buscar el mínimo se realizan una serie de pasos, siendo el objetivo de cada paso ajustar lo mejor posible los pesos. La dimensión de un lote (batch) define el número de imágenes utilizadas antes de llevar a cabo una actualización de los pesos de la red en el modelo.

2.6.4 Epochs e iteraciones

Define el número de veces que el conjunto total de muestras son procesadas por el método de optimización. Esto significa que cada muestra del conjunto N ha tenido una oportunidad de actualizar los pesos en cada epoch. También se puede fijar un número máximo de epochs para detener el proceso de entrenamiento cuando se alcanza dicho número, lo cual puede ser útil en el caso de que no se llegue a alcanzar la convergencia, esto es, que el error no sea aceptable.

Capítulo 3 - Diseño del modelo IoT

Como se ha comentado en apartados anteriores la planta de clasificación de residuos contaría en su despliegue real, con varias cámaras situadas en diferentes ubicaciones del proceso de clasificación además de una aplicación en la nube que permitirá a las personas realizar clasificaciones de imágenes tomadas al instante, que se almacenarán y estarán siempre accesibles a todo el mundo.

En este capítulo se describirá los aspectos tecnológicos que hacen posible tanto la aplicación de clasificación de residuos como las cámaras que se dispondrán en la planta. En la figura 3-1 se puede observar la interconexión entre las diferentes soluciones propuestas:

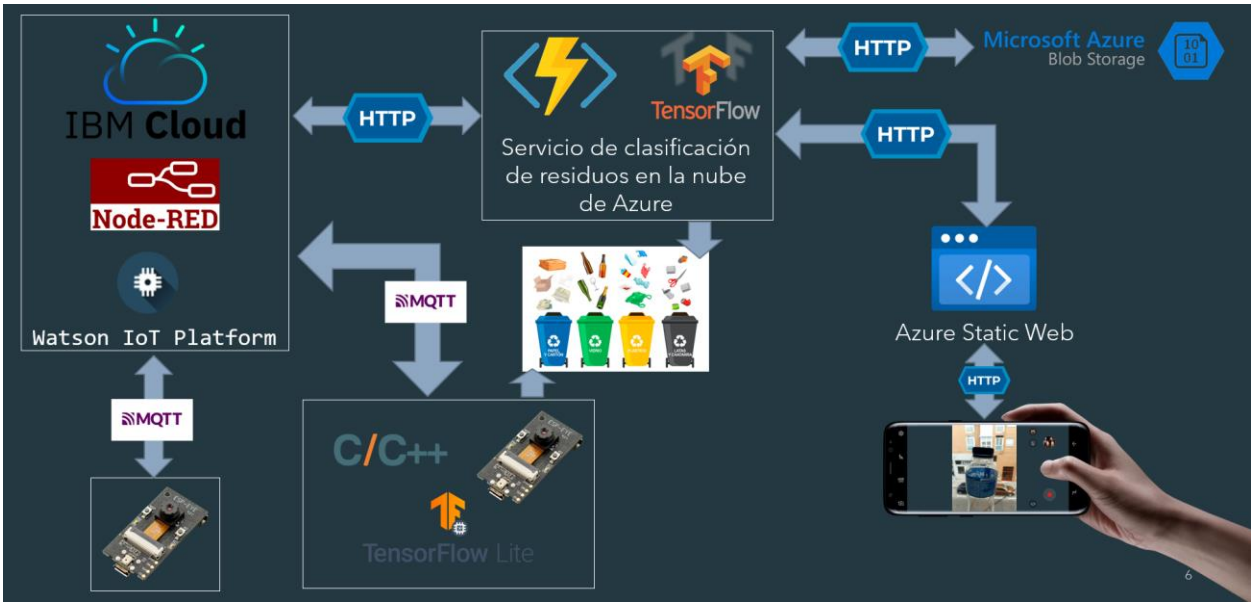


Figura 3-1. Diseño conceptual completo de la solución e integración de las aplicaciones

El listado de módulos que componen la aplicación en su conjunto aparece en el apéndice, junto con su ubicación de acceso.

3.1 Creación y entrenamiento del modelo

Para poder decidir qué tipo de residuo está capturando la cámara, el sensor deberá realizar un procesamiento de la imagen capturada y realizar la clasificación. Para realizar esta clasificación se utilizará un modelo de redes neuronales creado y

entrenado previamente. A continuación, se describen los procesos que se han realizado para conseguir el modelo final que utilizará la aplicación, así como los datos empleados para tal fin.

3.1.1 Pruebas realizadas antes de conseguir el modelo final

Como primer paso para la creación y entrenamiento del modelo se necesita un entorno de trabajo que pueda realizar todo el procesamiento necesario. Para ello se han utilizado dos entornos en la nube que han aportado una gran capacidad de procesamiento. Estos entornos son:

- **Google Colaboratory** (también conocido como Google Colab): Permite la creación y entrenamiento de modelos con TensorFlow de forma nativa, es decir, el entorno cuenta con todas las dependencias ya instaladas tanto para procesamiento con CPU, GPU e incluso TPU. Una de las grandes ventajas de este entorno de trabajo es la facilidad de compartir el código desarrollado a otros usuarios o colaboradores de forma segura. Otra ventaja es la fácil conexión con Google Drive que destaca por su gran capacidad de almacenamiento, tanto de pago como gratuita (15 GB por usuario gratis), esto permite disponer de los conjuntos de entrenamiento alojados en Google Drive y acceder a ellos desde el entorno como a una carpeta más directamente integrada en el entorno de desarrollo.

  Además de estas ventajas, se proporciona a los usuarios un determinado tiempo de ejecución con GPU y TPU (este tiempo suele ser alrededor de unas 12 h continuas y puede variar en función del uso que se está haciendo del entorno a la semana).

  Como ventaja adicional, los estudiantes de la UCM disponen de una cuota de almacenamiento de Google Drive muy amplia y además de licencia educativa para utilizar Google Drive Desktop, lo cual facilitara enormemente la carga y gestión de los datasets ya que con la interfaz web de Google Drive es mucho más lento.

- **Kaggle**: Como se ha comentado previamente, es una plataforma gratuita que pone a disposición de la comunidad, diferentes competiciones relacionadas con

la ciencia de datos, machine learning, etc. Tras estas competiciones, tanto código como datasets permanecen públicos (aunque algunos no tengan licencia CC) por lo que es una fuente excelente para encontrar datos y ejemplos para un problema como el planteado en este trabajo.

Además de la enorme cantidad de datasets proporcionados, esta plataforma permite el procesamiento en la nube, igual que Google Colab si somos usuarios.

Al ser usuario, disponemos directamente a acceso a un entorno de trabajo parecido a Google Colab. En este caso sí que están muy definidos los tiempos de ejecución de los que disponen los usuarios para CPU, GPU y TPU:

- CPU: indefinido.
- GPU: 30h semanales
- TPU: 30h semanales

Kaggle nos permite alojar nuestros propios datasets o utilizar de forma nativa cualquier dataset público, al igual que ocurre en Google Colab con Google Drive, solo que en Kaggle podemos agregar al entorno de trabajo de forma muy fácil cualquier dataset y utilizarlo directamente.

Todo el código, los datos y los resultados de las ejecuciones podemos decidir si se guardan de forma privada o de forma pública, en cuyo caso cualquier usuario de la plataforma podría utilizar tanto el dataset como el código para otros fines (podríamos por ejemplo publicar solo el dataset, pero no el código o viceversa).

Para llevar a cabo el entrenamiento del modelo para clasificar residuos se han realizado varias pruebas y entrenamientos, analizando los resultados en función del tipo de datos empleado y haciendo uso de ambas plataformas.

A continuación, se describen las pruebas realizadas para decidir el modelo a utilizar.

3.1.1.1 Conjuntos de datos de entrenamiento

Para poder entrenar el modelo, en primer lugar, se necesita disponer de datos. Los datos empleados en los entrenamientos provienen de dos fuentes diferentes.

- **Original Garbage Classification Data**: Dataset alojado en la plataforma Kaggle que proporciona 1796 imágenes divididas entre las clases: Cartón, Metal, Plástico y Vidrio. Originalmente este dataset estaba formado por 2532 imágenes divididas entre las clases: carboard, glass, metal, paper, plastic, trash como podemos ver en la figura 3-2 de diferentes tamaños, por lo que se ha renombrado a Original Garbage Classification y redimensionado todas sus imágenes a 224 x 224, además de dividirlo en dos subconjuntos, entrenamiento y test donde el conjunto de test contiene el 20% de las imágenes del dataset, como se muestra en la figura 3-3:

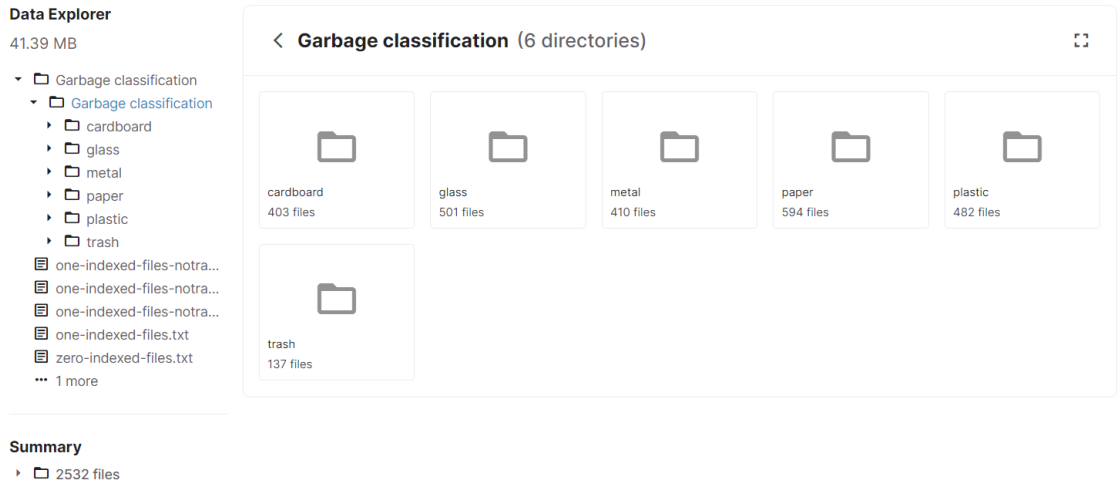


Figura 3-2. Garbage Classification Data (Comunidad Kaggle)

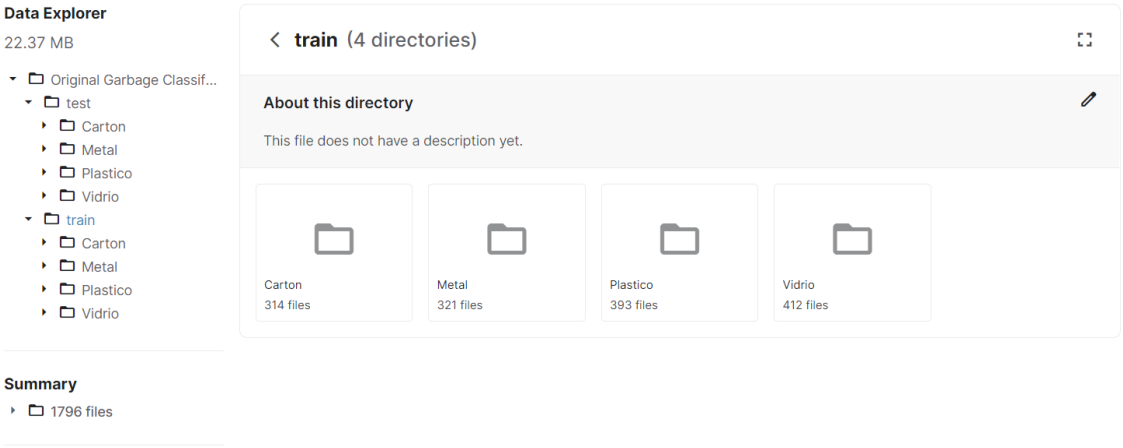


Figura 3-3. Original Garbage Classification (Modificado a 224 x 224)

Ejemplos ilustrativos de estos datos son los que se muestran en las figuras 3-4, 3-5, 3-6 y 3-7 agrupados según distintas categorías:

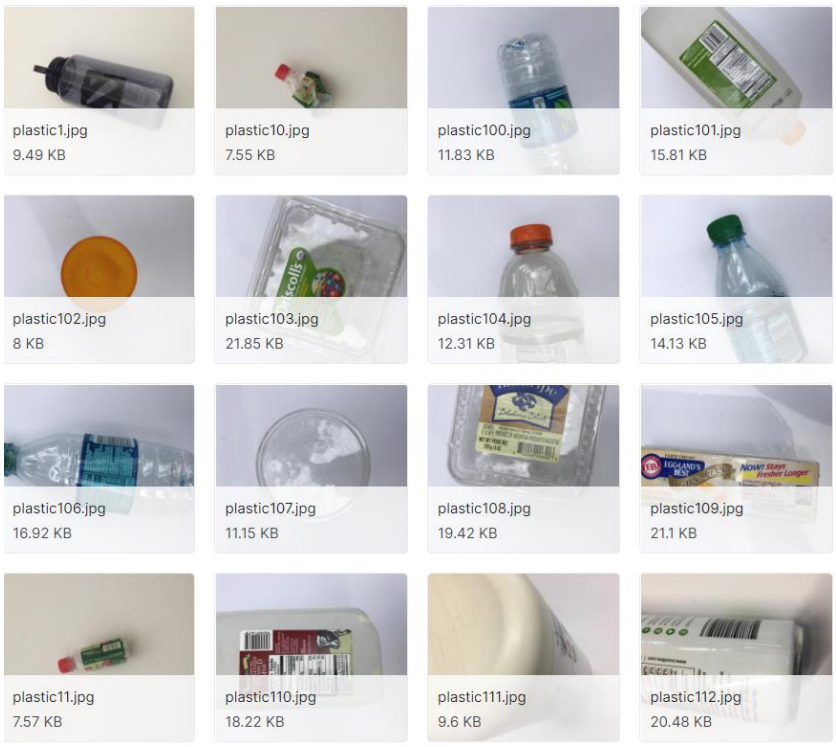


Figura 3-4. Ejemplos plástico Original Garbage Classification

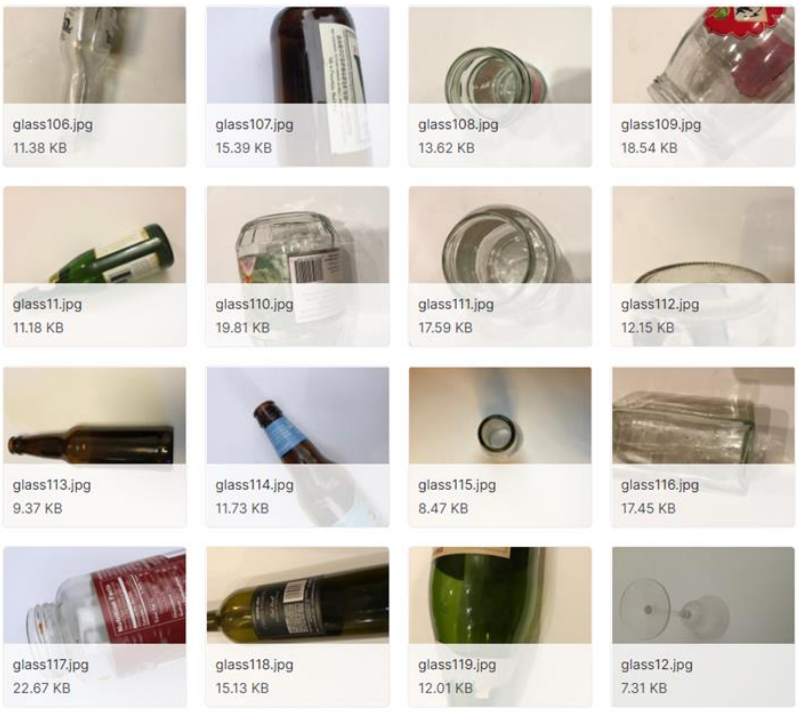


Figura 3-5. Ejemplo vidrio Original Garbage Classification

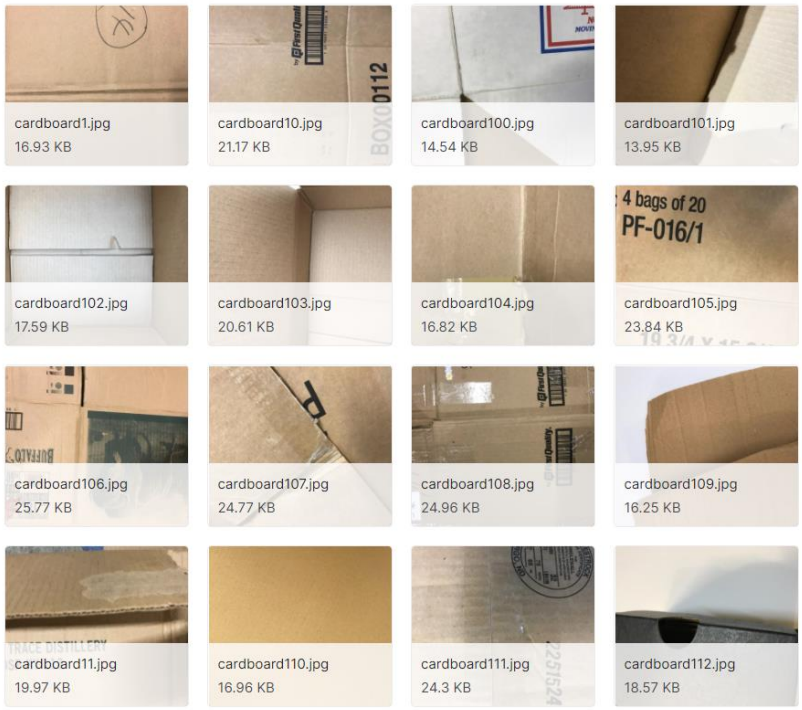


Figura 3-6. Ejemplos cartón Original Garbage Classification

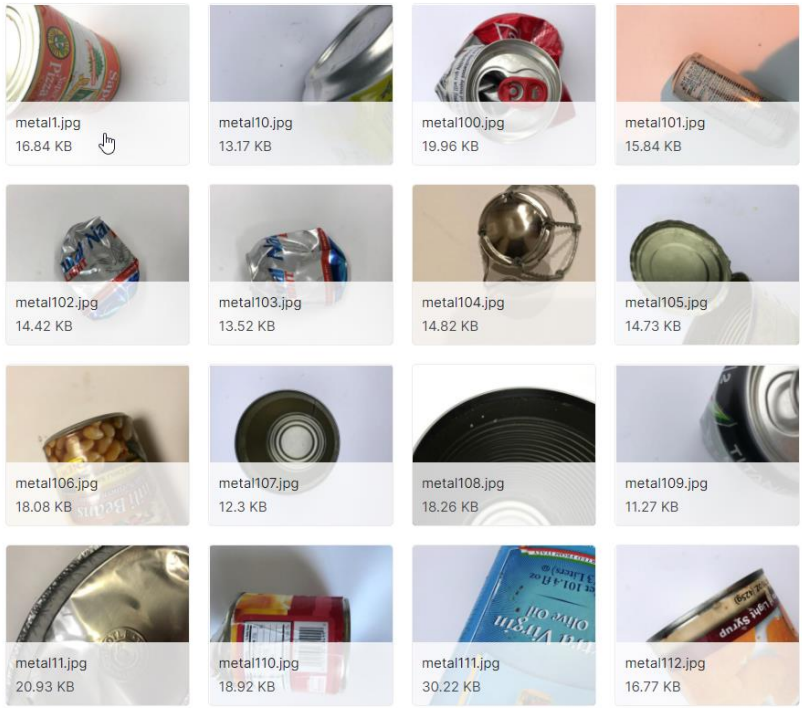


Figura 3-7. Ejemplos metal Original Garbage Classification

- **Garbage classification**: Dataset alojado en Kaggle creado para fines académicos (en concreto para este proyecto). Este dataset consta de 1921 imágenes divididas entre las clases: Cartón, Metal, Vidrio, Plástico, como se puede observar en la figura 3-8. Al igual que el anterior, todas las imágenes están en tamaño 224 x 224 y divididas en los conjuntos de entrenamiento y test con una proporción de 80 y 20%:

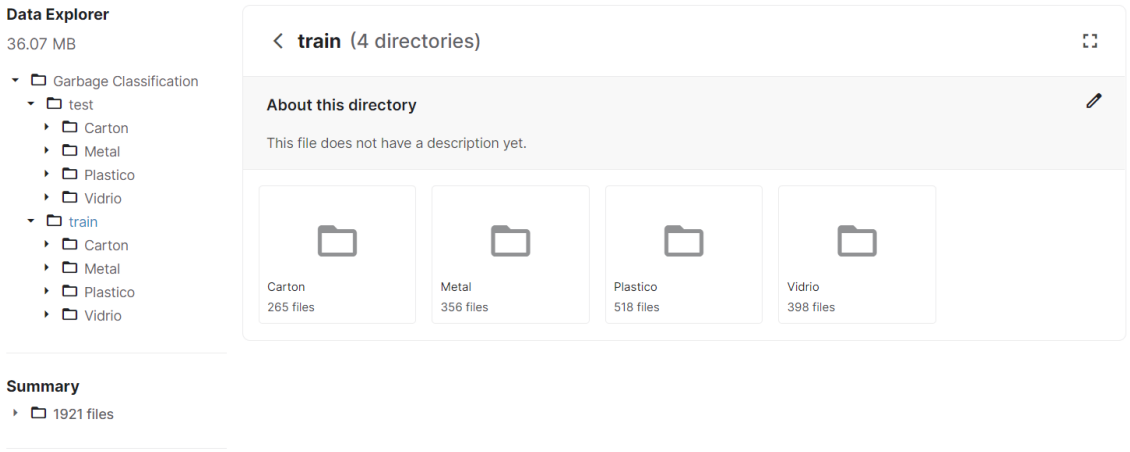


Figura 3-8. Garbage classification dataset

En cuanto al tipo de datos que contiene este dataset, en las figuras 3-9, 3-10, 3-11, 3-12 se pueden visualizar algunos ejemplos ilustrativos, según diferentes categorías.

Figura 3-9. Ejemplos plásticos Garbage classification

Figura 3-10. Ejemplo vidrio Garbage classification

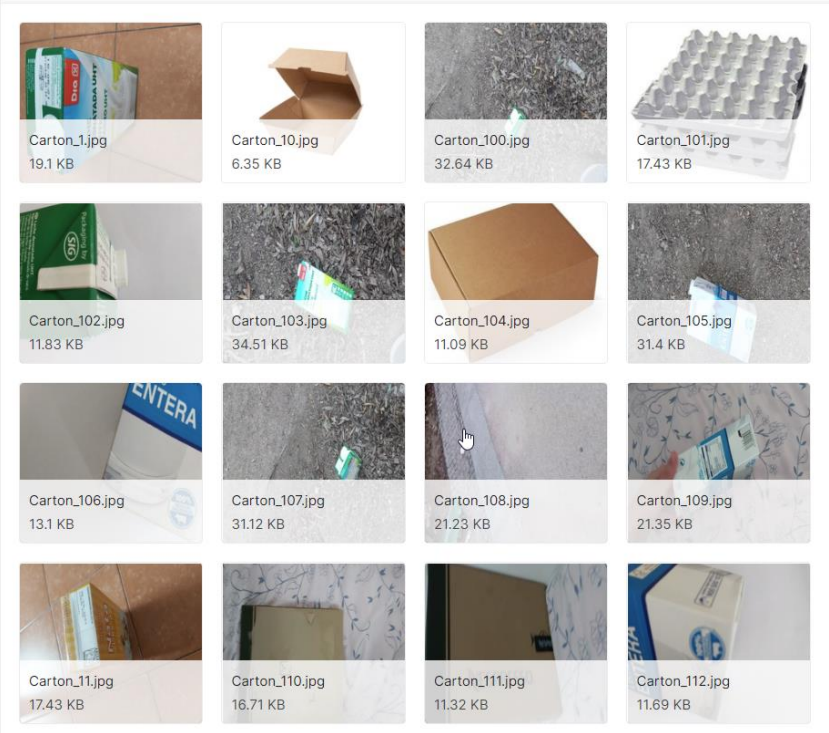


Figura 3-11. Ejemplos cartón Garbage classification

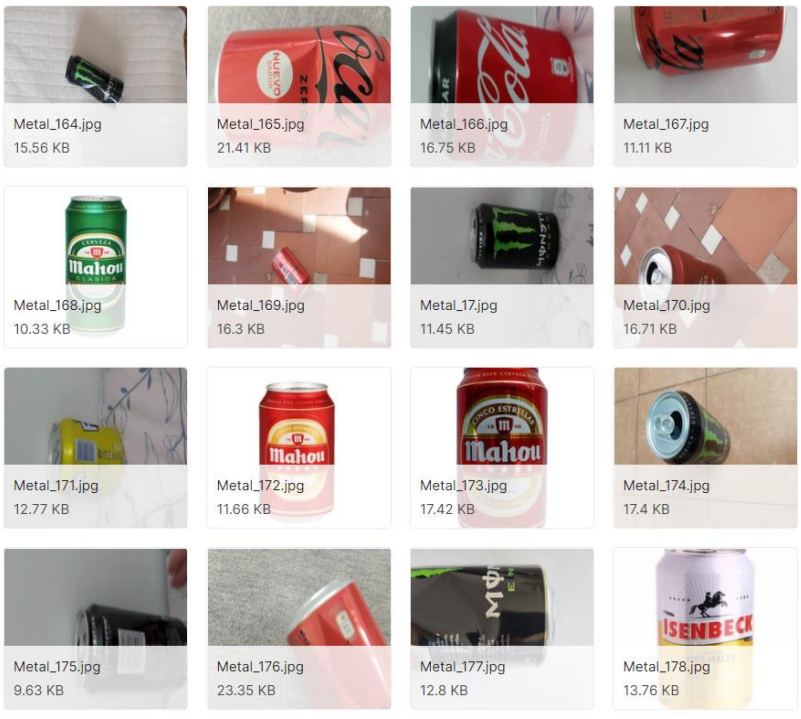


Figura 3-12. Ejemplo metal Garbage classification

3.1.1.2 Data agumentation

Antes de comenzar a realizar el entrenamiento y la configuración del modelo final, se ha investigado sobre esta técnica o método de pre-procesamiento.

Cuando se dispone de un conjunto de datos reducido, además de otras configuraciones se puede utilizar lo que se conoce como "*data agumentation*", una forma que permite "aumentar" el tamaño del dataset mediante ajustes en las imágenes proporcionadas (imágenes del dataset utilizado).

Estos ajustes se realizan de manera automática con la clase ImageDataGenerator proporcionada por TensorFlow en el módulo de Keras. Algunos de estos posibles ajustes (parámetros) son:

- **Shear_range**: Intensidad de corte en sentido antihorario. (Ver figura 3-13)
- **Rescale**: Factor de escalado que se aplicará a todas las imágenes.
- **Rotation_range**: Rango de grados para realizar rotaciones aleatorias.
- **Zoom_range**: Float o rango [mínimo, máximo] con el que se ampliará la imagen.
- **Width_shift_range**: Float, array o entero que indicaran el valor con el que se va a mover una imagen a lo horizontalmente. (Ver figura 3-14)
- **Height_shift_range**: Float, array o entero que indicaran el valor con el que se va a mover una imagen verticalmente. (Ver figura 15)
- **Horizontal_flip**: Tipo booleano. Indica si deseamos girar las imágenes horizontalmente de forma aleatoria.
- **Vertical_flip**: Tipo booleano. Indica si deseamos girar las imágenes verticalmente de forma aleatoria.
- **Brightness_range**: Tupla o lista de dos Float que nos permiten indicar el nivel de brillo.
- **Shuffle**: Booleano que indica si las imágenes se cargan de manera aleatoria. Si es falso se cargarán en orden alfabético.

Podemos encontrar la lista completa de parámetros en la documentación de Keras "*Image data preprocessing*".

A continuación, en las figuras 3-13 a 3-15 se representan de forma gráfica los ajustes que se realizan en las imágenes mediante algunos de los parámetros mencionados.

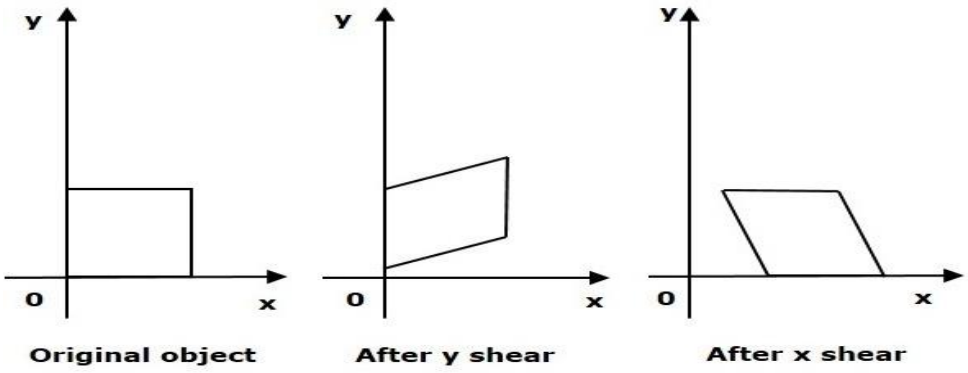


Figura 3-13. Shear range

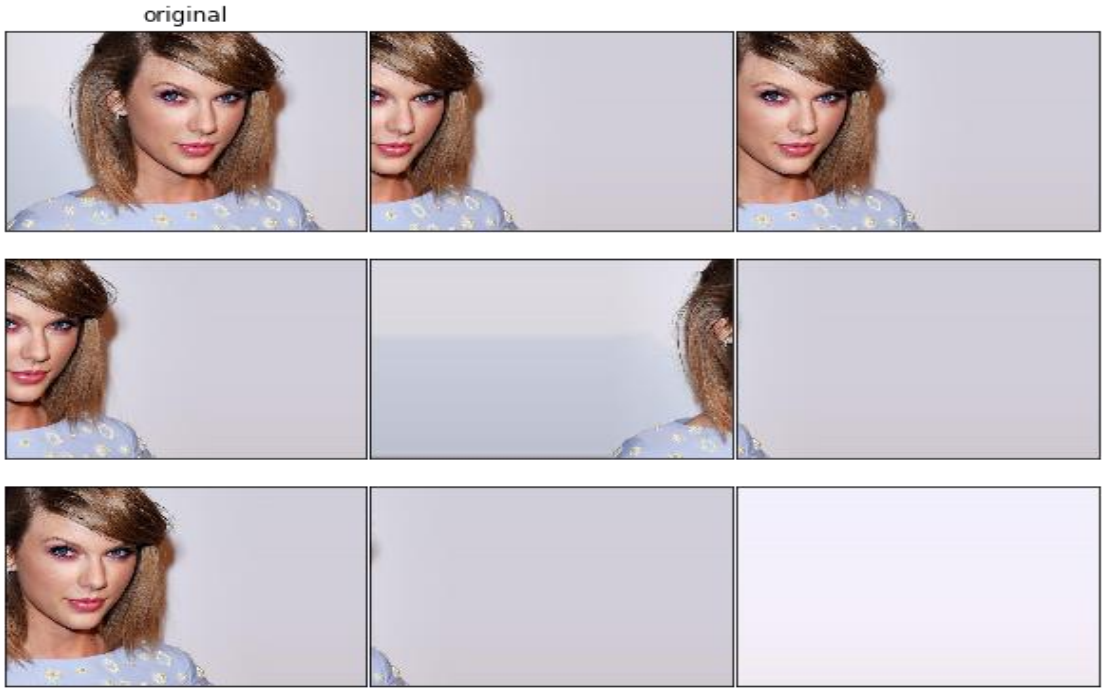


Figura 3-14. Transformaciones realizadas por width_shift_range

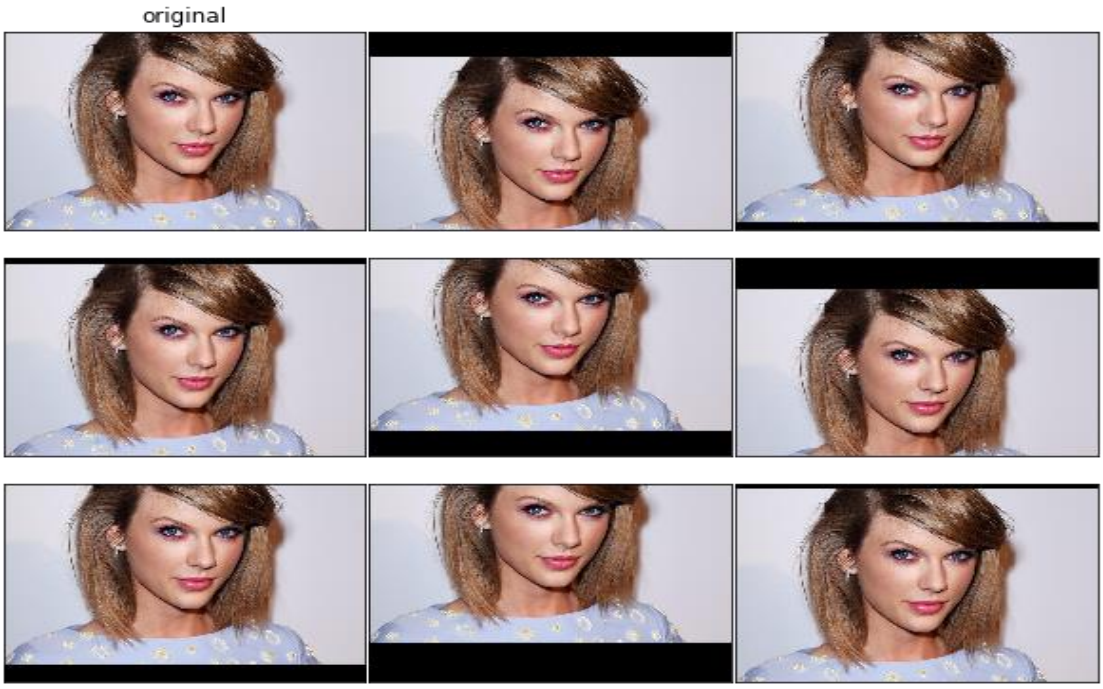


Figura 3-15. Transformaciones realizadas por height_shift_range

3.1.1.3 Ajuste fino de un modelo

El modelo que se va a generar y entrenar utilizará como base el modelo MobileNetV2, haciendo uso de todo el conocimiento adquirido previamente por este modelo.

Al utilizar modelos pre-entrenados disponemos de dos opciones:

- Crear un modelo nuevo, utilizando todas las capas del modelo pre-entrenado sin alterar y añadir nuevas capas al final para personalizar el modelo, que sí se van a entrenar (además de modificar el número de neuronas de la última capa).
- Crear un modelo nuevo, utilizando la mayoría de las capas del modelo pre-entrenado alterando los pesos de las últimas capas (ajuste fino) más las capas que se añaden al final para personalizar el modelo.

En este caso, se han utilizado ambas implementaciones de forma secuencial, de forma que en primer lugar se realiza la configuración con el modelo pre-entrenado y los pesos originales, se añaden las capas personalizadas y se realiza el entrenamiento. La configuración sería la que se indica en la figura 3-16, que muestra el esquema del modelo pre-entrenado, sin el ajuste fino.

```
[ ]
    # Create the base model from the pre-trained MobileNet V2
    base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                   include_top=False,
                                                   )

    base_model.trainable = False

    Downloading data from https://storage.googleapis.com/tensorflow/keras-ap
    9412608/9406464 [==============================] - 0s 0us/step
```

```
[ ]  loss_f = tf.keras.losses.CategoricalCrossentropy(
         reduction=tf.keras.losses.Reduction.AUTO,
        name='categorical_crossentropy'
    )
    model = tf.keras.Sequential([
        base_model,
        tf.keras.layers.Dropout(0.4),
        tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D(1),
        tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Dropout(0.4),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(units=OUTPUT_SIZE, activation='softmax')
    ])
```

Figura 3-16. Configuración del modelo

Una vez creado el modelo y entrenado, realizamos otro entrenamiento, pero indicando que queremos alterar algunas capas del modelo pre-entrenado, como se indica en la figura 3-17, donde se mantienen sin alterar las primeras 100 capas del modelo y permitiendo al resto de capas variar sus pesos:

```
base_model.trainable = True
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-6),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=4

history_fine = model.fit(train_generator,
                         epochs=EPOCHS_FINE,
                         validation_data=val_generator,
                         callbacks=[early_stop]
                         )
```

Figura 3-17. Ajuste fino del modelo

Es recomendable en este tipo de entrenamientos utilizar una razón de aprendizaje (*learning rate*) baja, ya que el modelo ya ha sido entrenado y lo único que se pretende es ganar algo más de precisión.

Otra opción sería realizar esta configuración desde el inicio, sin entrenar el modelo previamente, por lo que tendríamos la configuración de la figura 3-16 (sin realizar ningún entrenamiento), a continuación, realizamos el ajuste fino como se indica en la figura 3-17 y entrenamos.

En este segundo caso, estaríamos cambiando los pesos de todas las capas, excepto de las 100 primeras (pertenecientes al modelo de MobileNetV2), el resto (las capas finales del modelo de MobilenetV2 más las del ejemplo) cambiarían sus pesos.

3.1.1.4 Entrenamiento de un modelo combinando ambos datasets

Para realizar pruebas con un conjunto de datos más amplio, se ha decidido combinar los dos datasets mencionados en los apartados anteriores, dando lugar al dataset **Mixed Garbage Classification**, también alojado en Kaggle con un total de 3561 imágenes divididas en los conjuntos de entrenamiento y test con su respectivo 80 y 20% del total de los datos, como se puede comprobar en la figura 3-18:

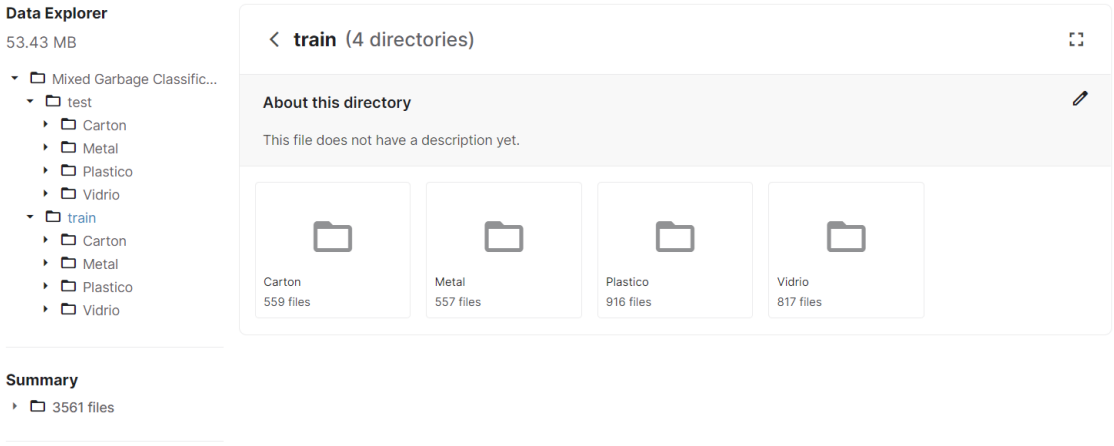


Figura 3-18. Mixed Garbage Classification

En cuanto a la red neuronal utilizada, como podemos observar en la figura 3-19, se utiliza el modelo pre-entrenado de MobilenetV2 al que se le añaden unas capas convolucionales, de *pooling* y *dense* para finalmente definir la última capa con las 4 posibles salidas (Cartón, Metal, Vidrio y Plástico).

```
loss_f = tf.keras.losses.CategoricalCrossentropy(
     reduction=tf.keras.losses.Reduction.AUTO,
    name='categorical_crossentropy'
)
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(units=OUTPUT_SIZE, activation='softmax')
])
model.compile(loss=loss_f,
              optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              metrics=['accuracy'])
model.summary();
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 7, 7, 64)          737344
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 3, 3, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 32)          18464
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_20 (Dense)             (None, 256)               8448
_________________________________________________________________
dense_21 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_22 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_23 (Dense)             (None, 4)                 260
=================================================================
Total params: 3,063,652
Trainable params: 2,667,108
Non-trainable params: 396,544
```

Figura 3-19. Modelo pre-entrenado con capas extra

Los parámetros utilizados han sido los que se muestran en la figura 3-20:

```
path_to_data = '../input/mixed-garbage-classification/Mixed Garbage Classification/train'
path_to_test = '../input/mixed-garbage-classification/Mixed Garbage Classification/test'

IMAGE_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 100
EPOCHS_FINE = 100
OUTPUT_SIZE = 4

base_learning_rate = 1e-4
fine_learning_rate = base_learning_rate/10
```

Figura 3-20. Parámetros óptimos

Además, se ha utilizado la clase ImageDataGenerator con la configuración que aparece en la figura 3-21:

```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    shear_range=0.1,
    zoom_range=0.1,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2,
    brightness_range=[0.6,1.1]
)

train_generator = train_datagen.flow_from_directory(
    path_to_data,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False,
    color_mode='rgb',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    path_to_data,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False,
    color_mode='rgb',
    subset='validation'
)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)

test_generator = test_datagen.flow_from_directory(
    path_to_test,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False,
    color_mode='rgb'
)
```

```
Found 1231 images belonging to 4 classes.
Found 306 images belonging to 4 classes.
Found 384 images belonging to 4 classes.
```

Figura 3-21. Configuración data agumentation

A la hora de realizar el entrenamiento, TensorFlow nos permite configurar lo que se conoce como parada temprana, como se muestra en la figura 3-22:

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=4,restore_best_weigh
ts=True, verbose=1)

history = model.fit(train_generator,
                    epochs=EPOCHS,
                    workers=4,
                    validation_data=val_generator,
                    callbacks=[early_stop]
                   )
```

Figura 3-22. Parada temprana

Como se puede observar en la figura 3-22, se utiliza la función "*EarlyStopping*" que ayuda a evitar el *overfitting* (sobreajuste) monitorizando como en este caso el valor de "*val_loss*" (error de validación o valor de pérdida) con una tolerancia de 4 (cuatro valores por encima del menor valor de validación. En la figura 3-24 se puede ver el efecto de la parada temprana a la hora de realizar el ajuste fino del modelo, donde se monitoriza la función de pérdida del entrenamiento, al contrario que en la figura 3-22 donde se monitoriza la función de pérdida de la validación del primer entrenamiento que no altera las capas del modelo pre-entrenado.

Al cumplirse esta condición, automáticamente se restauran los pesos obtenidos cuatro epochs más atrás.

En cuanto al modelo empleado para realizar las pruebas ha sido el que se muestra en la figura 3-23.

```
loss_f = tf.keras.losses.CategoricalCrossentropy(
     reduction=tf.keras.losses.Reduction.AUTO,
    name='categorical_crossentropy'
)
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
    tf.keras.layers.Conv2D(8, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(units=OUTPUT_SIZE, activation='softmax')
])

model.compile(loss=loss_f,
              optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              metrics=['accuracy'])
model.summary();
```

Figura 3-23. Modelo optimo

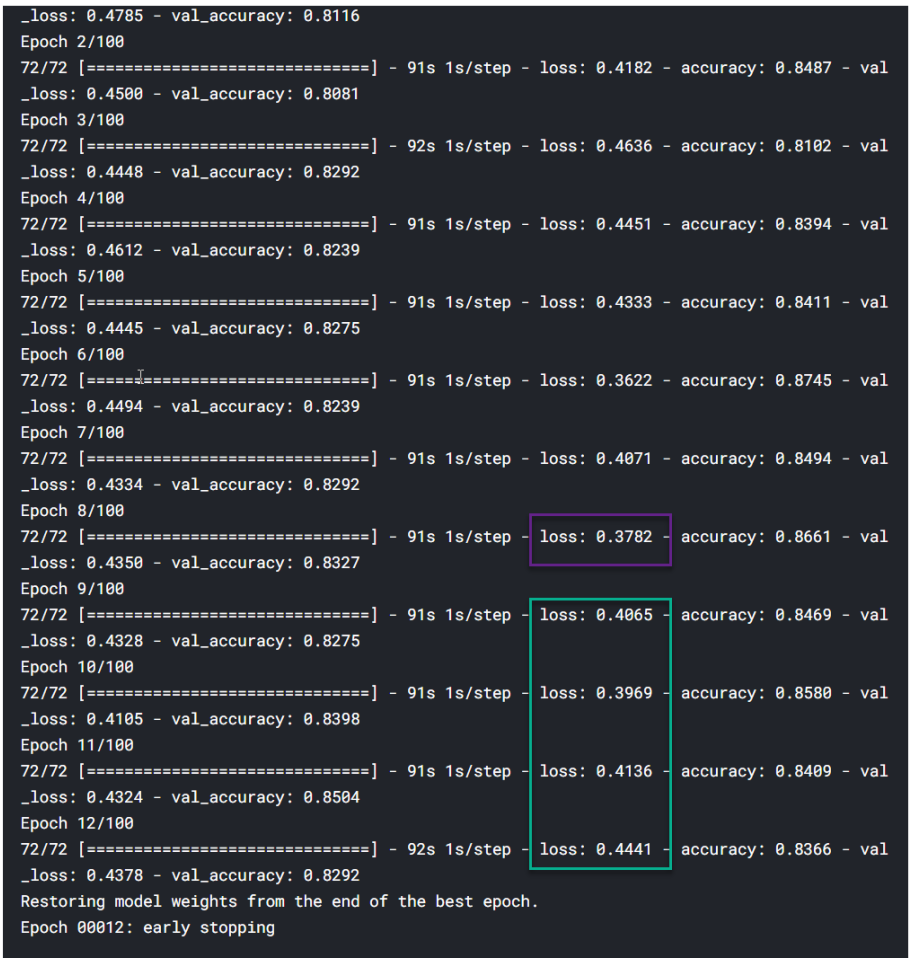


Figura 3-24. Parada temprana en el primer entrenamiento con datos combinados

A continuación, se detallan los entrenamientos y algunas de las pruebas realizadas con las configuraciones comentadas:

Al cargar los datos de este dataset mixto, obtenemos los valores mostrados en la figura 3-25 para cada subconjunto de entrenamiento, validación y test respectivamente:

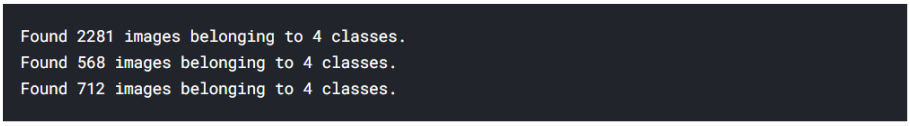


Figura 3-25. Conjunto de entrenamiento, validación y test respectivamente

Hay que recordar que el modelo empleado ha sido el que se muestra en la figura 3-23, donde las capas del modelo pre-entrenado permanecen intactas.

Como resultado de este entrenamiento se ha obtenido la gráfica de la figura 3-26, donde se indica la evolución del "Accuracy" y del coste:

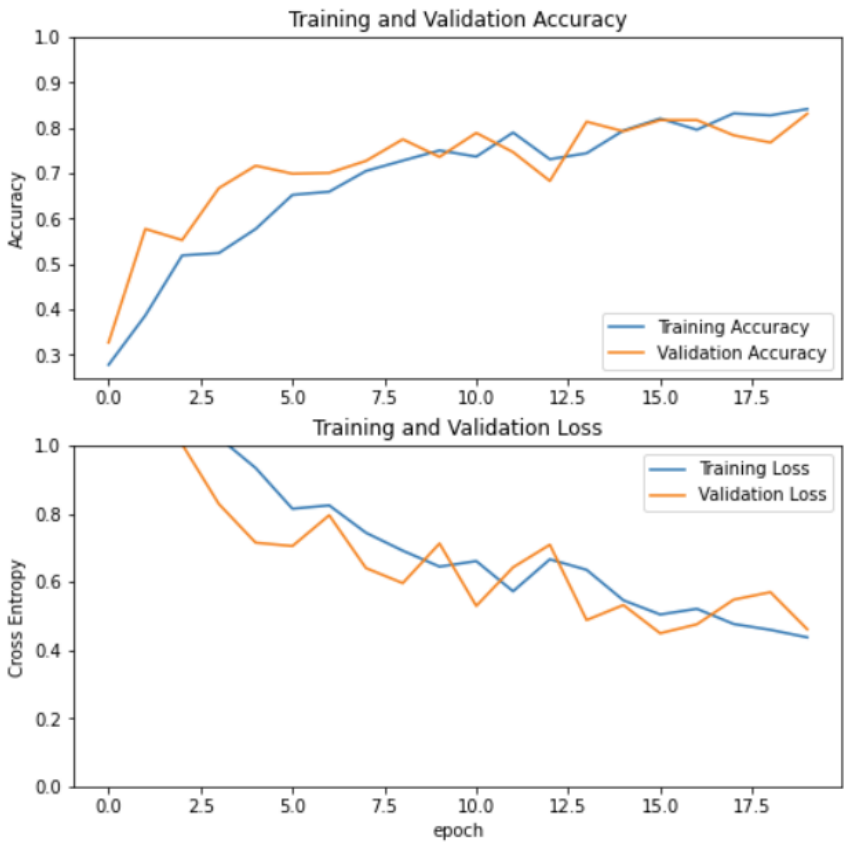


Figura 3-26. Curvas de aprendizaje primer entrenamiento con datos combinados

En la mencionada figura 3-26 se puede observar que tanto "validation loss" como "training loss" progresan más o menos a la par, indicando en ocasiones un leve *underfitting* y en otras un leve *overfitting*, lo que nos indica que está siendo un entrenamiento bastante bueno, pese a que el coste no llega a bajar del 0.48.

Si evaluamos el modelo tanto con el conjunto de validación como con el de test, obtenemos los resultados mostrados en la figura 3-27:

```
18/18 [==============================] - 18s 967ms/step - loss: 0.4881 - accuracy: 0.8046

Validation accuracy 0.8045774698257446

Validation loss 0.4881451427936554
23/23 [==============================] - 18s 797ms/step - loss: 0.4358 - accuracy: 0.8371
\Test accuracy 0.8370786309242249

Test loss 0.4357622563838959
```

Figura 3-27. Resultado al aplicar el conjunto de test al modelo

En la figura 3-27 lo que destaca en especial es el valor de "accuracy" del conjunto de test que es del 83% y de coste del 43%.

A continuación, en la figura 3-28 se representa la matriz de confusión después de realizar este entrenamiento y aplicar el conjunto de test:

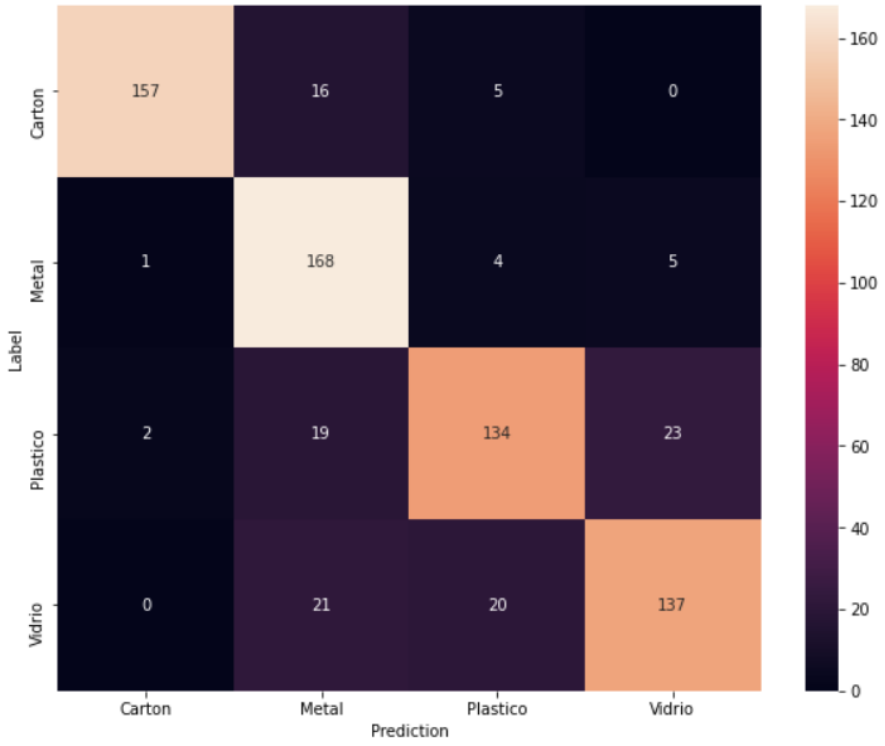


Figura 3-28. Matriz de confusión conjunto de test después del primer entrenamiento

Pese a que hay algunos fallos, parece que clasifica bien la mayoría. Para ello nos fijamos en los valores de la diagonal principal, ya que cuantos mayores sean estos valores en relación a los que aparecen en las celdas fuera de la diagonal, mejor es el ajuste. En este caso, la gama de colores expresa el grado de acierto en función de la escala de la derecha. Un ejemplo más visual es el que se muestra en la figura 3-29:

Figura 3-29. Clasificaciones con el primer modelo

Una vez analizados estos resultados, se realizará el ajuste fino obteniendo los resultados de las figuras 3-30, 3-31 y 3-32 donde se observa que el ajuste fino en este caso no aporta demasiado conocimiento al modelo, al contrario, empeoran un poco los resultados.

Figura 3-30. Ajuste fino Mixed Garbage Classification

```
18/18 [==============================] - 17s 945ms/step - loss: 0.4445 - accuracy: 0.8116

 Validation accuracy 0.8116196990013123

Validation loss 0.44454315304756165

23/23 [==============================] - 19s 804ms/step - loss: 0.4264 - accuracy: 0.8371

 Test accuracy 0.8370786309242249

Test loss 0.4263592064380646
```

Figura 3-31. Evaluación ajuste fino Mixed Garbage Classification

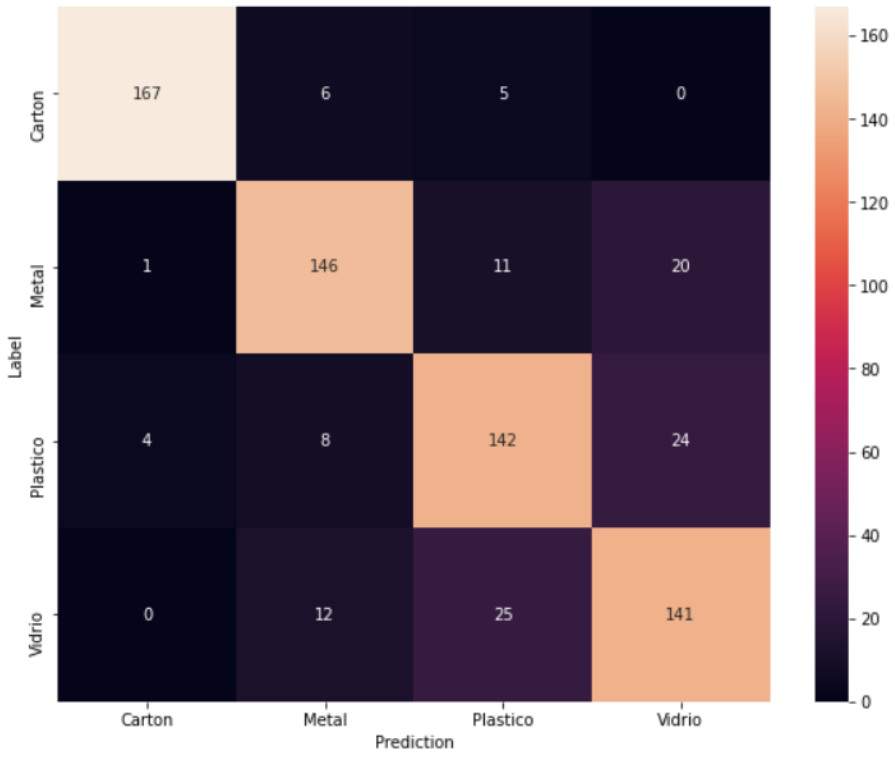


Figura 3-32. Matriz de confusión ajuste fino Mixed Garbage Classification

3.1.1.5 Entrenamiento únicamente con el dataset Garbage classification

Hay que recordar que se utilizará el mismo modelo y configuraciones de red especificado en el apartado anterior, pero aplicadas únicamente a este conjunto de datos, formado por los subconjuntos que se muestran en la figura 3-33:

```
Found 1231 images belonging to 4 classes.
Found 306 images belonging to 4 classes.
Found 384 images belonging to 4 classes.
```

Figura 3-33. Ejemplos de entrenamiento, validación y test de Garbage classification

En este caso, tras 39 *epochs* se detecta *overfitting*, por lo que se han considerado 35 *epochs* (debido a la parada temprana) para dar lugar a la gráfica de la figura 3-34:

Figura 3-34. Primer entrenamiento Garbage Classification

Al aplicar el conjunto de test sobre este modelo, se obtienen los resultados mostrados en la figura 3-35:

```
10/10 [==============================] - 10s 919ms/step - loss: 0.3244 - accuracy: 0.8725

Validation accuracy 0.8725489974021912

Validation loss 0.3244127333164215
12/12 [==============================] - 10s 838ms/step - loss: 0.2000 - accuracy: 0.9375
\Test accuracy 0.9375

Test loss 0.1999545693397522
```

Figura 3-35. Evaluación modelo Garbage Classification

Para poder interpretar mejor estos resultados utilizamos la matriz de confusión asociada:

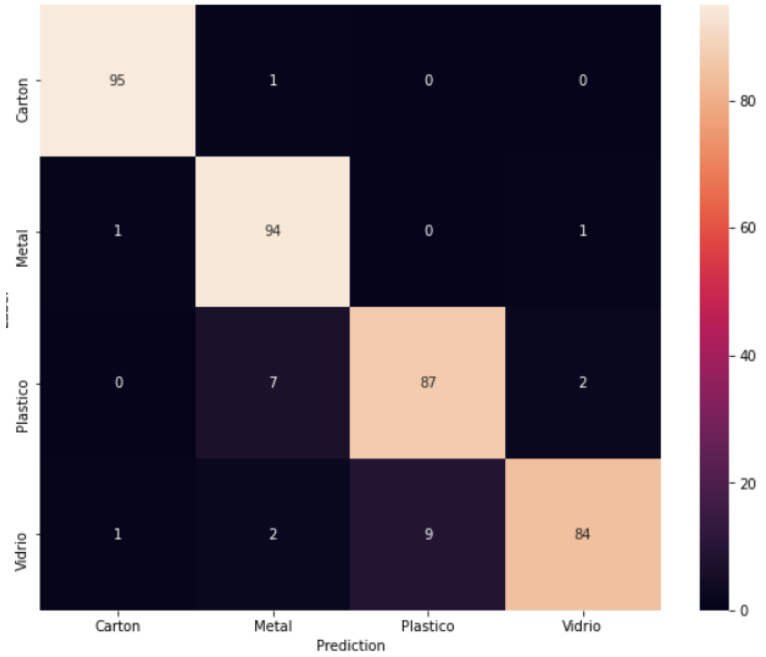


Figura 3-36. Matriz de confusión Garbage Classification conjunto de test

Al realizar el ajuste fino, lo que se consigue es sobre ajustar un poco más el modelo, reduciendo muy poco el error y mejorando poco la precisión (en realidad no aporta nada el ajuste fino en este caso) como se muestra en las figuras 3-37, 3-38 y 3-39:

Figura 3-37. Ajuste fino Garbage Classification

```
10/10 [==============================] - 10s 931ms/step - loss: 0.3394 - accuracy: 0.8889

 Validation accuracy 0.8888888955116272

Validation loss 0.33941030502319336

12/12 [==============================] - 10s 828ms/step - loss: 0.1921 - accuracy: 0.9349

 Test accuracy 0.9348958134651184

Test loss 0.1920517235994339
```

Figura 3-38. Evaluación después de ajuste fino Garbage Classification

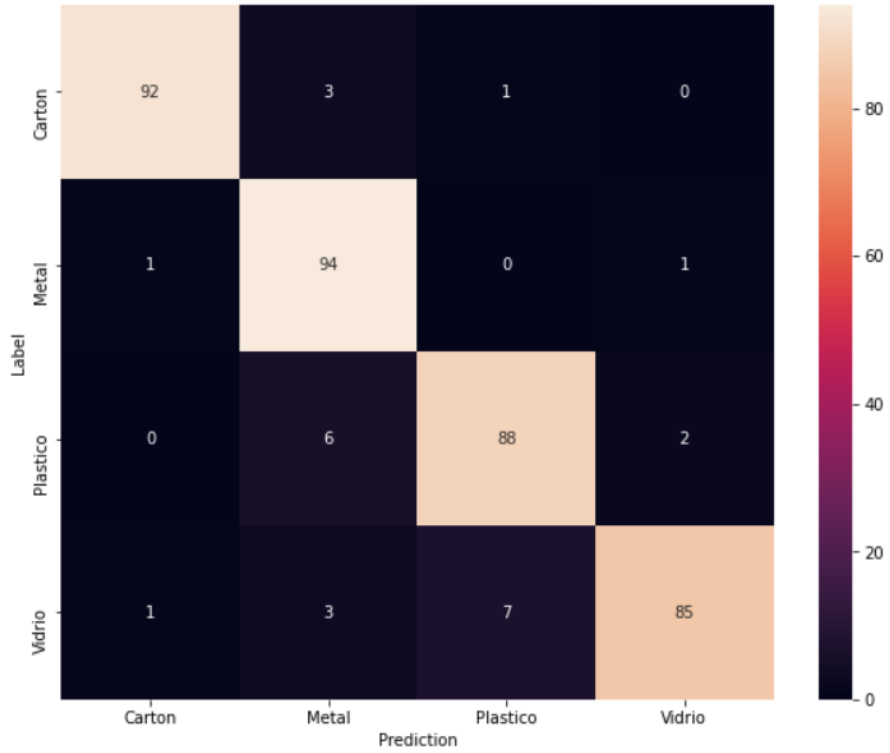


Figura 3-39. Matriz de confusión con el ajuste fino de Garbage classification

Para que estos resultados se puedan interpretar de forma gráfica, se han realizado algunas clasificaciones utilizando muestras el conjunto de test obteniendo los resultados de la figura 3-40:

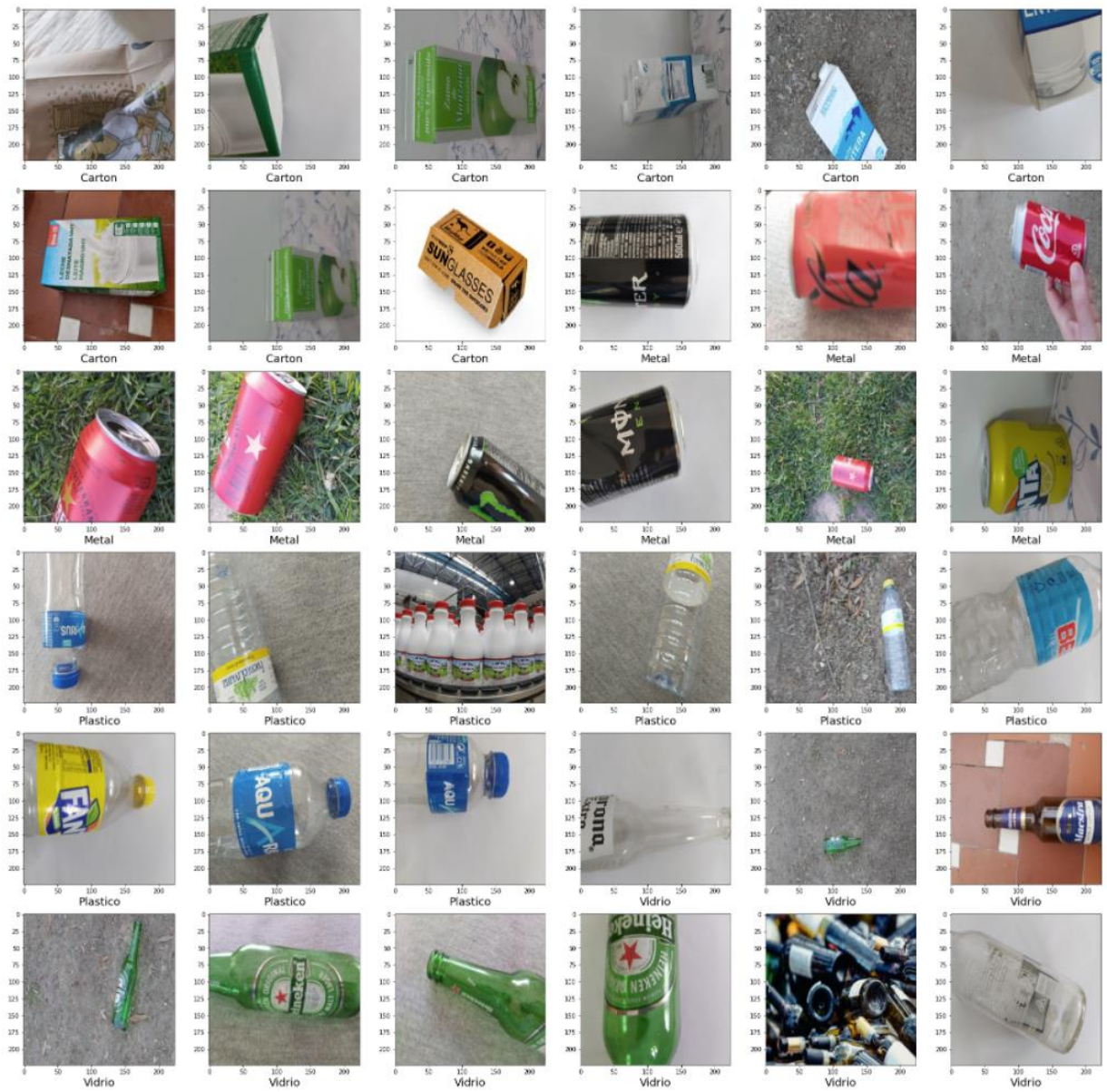


Figura 3-40. Clasificación de imágenes del conjunto de test Garbage classification

Como se puede observar en este caso el modelo es capaz de aprender y clasificar mucho mejor las muestras de este dataset obteniendo un accuracy del 93% y un error bastante bajo.

Entre otros aspectos, esto puede deberse a la propia naturaleza de los datos, ya que al combinar ambos datasets, aunque todos contengan residuos en las imágenes, estas son muy dispares entre sí, por lo que es comprensible que haya peores resultados en el mixto y mejores en este último.

Llegados a este punto también hay que tener en cuenta que interesa más, que el modelo sea capaz de clasificar bien de forma general cualquier tipo de residuo o que clasifique muy bien unos muy específicos.

El problema de los residuos es que su forma, colores, marcas, etc. cambian mucho en función de su origen, por lo que para este trabajo se decide utilizar el modelo entrenado únicamente con el dataset **Garbage classification** que funcionaría mejor en plantas de clasificación de residuos en España, ya que esa clase de residuos son más comunes.

3.2 Aplicación de clasificación de residuos

La aplicación de clasificación de residuos está formada por los componentes integrados en la figura 3-40.

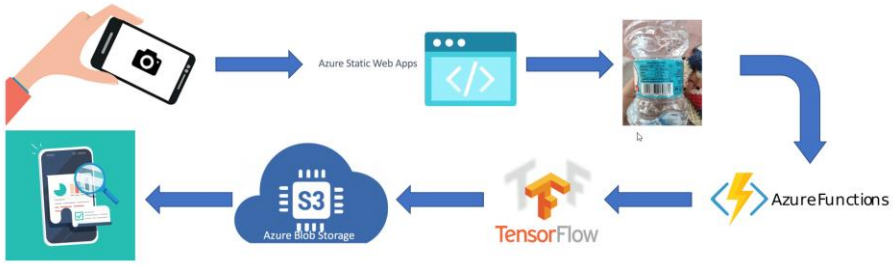


Figura 3-41. Comunicación entre componentes de la aplicación de clasificación de residuos

Como se puede observar en la figura 3-41 para llevar a cabo la aplicación se utilizan diferentes herramientas y tecnologías, algunas de ellas mencionadas en capítulos anteriores. A continuación, se va a detallar el uso de todas las componentes que se han utilizado a la hora de desarrollar la aplicación de clasificación de residuos.

3.2.1 Interfaz visual con Azure Static Web Apps

Para permitir a los usuarios tomar imágenes se ha desarrollado una página web utilizando el servicio de aplicaciones web estáticas de Azure que permite desarrollar aplicaciones front-end estáticas. Para desarrollar esta aplicación se han utilizado las siguientes tecnologías:

- **ReactJS**: Junto a Angular, React es uno de los *frameworks* para JavaScript más utilizados hoy en día. Desarrollado por Facebook, React facilita la creación de interfaces de usuario basadas en componentes encapsulados que reciben unas

propiedades y manejan su propio estado. Cabe mencionar que en las últimas versiones React se integra tanto la utilización de componentes que heredan de la clase "Component" (en generar se utilizarán para el desarrollo de lógica más compleja) y el uso de componentes funcionales (son una especie de funciones donde se desarrollan métodos y código HTML sencillo). Sean componentes funcionales o componentes que heredan de la clase "Component" ambos disponen del estado y de las propiedades mencionadas:

- **Propiedades**: Al crear un componente éste debe definir una interfaz con las propiedades necesarias para su funcionamiento. En esta interfaz se puede especificar que métodos, objetos o variables ha de pasar el componente principal. En la figura 3-42 se puede ver la interfaz que establece que el componente principal debe pasar al componente Camera un método que recibe un parámetro de tipo "string", como se muestra en la figura 3-42. En la figura 3-42 vemos como un componente principal utiliza el componente Camera donde se pasa la referencia a la función "*onTakePhoto*" definida también en este componente principal.

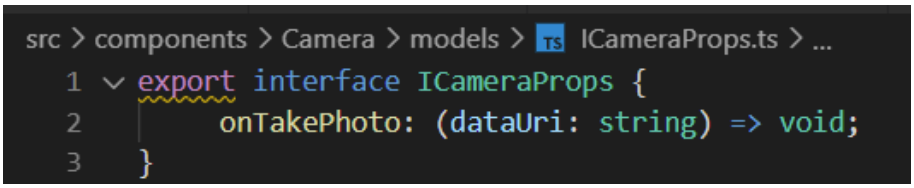


Figura 3-42. Interfaz que define las propiedades del componente Camera

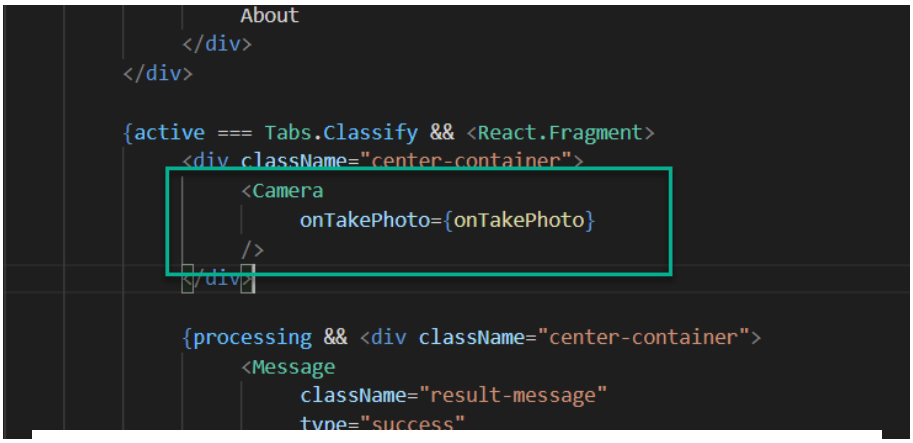


Figura 3-43. Componente que utiliza el componente Camera

```
const onTakePhoto = async (image: string) => {
    setProcessing(true);

    console.log("imagen", image);

    try {
        const responsePromise = await fetch(ENDPOINTS.PREDICTION_API, {
            headers: {
                "Content-Type": "application/json"
            },
            method: "POST",
            body: JSON.stringify({
                image: image
            })
        });
        const response: Array<IResultData> = await responsePromise.json();
        setProcessResult(response);
    }
    catch (e) {
        console.error(e);
    }
    setProcessing(false);
};
```

Figura 3-44. Función definida en un componente principal

Por tanto, cuando un usuario pulsa el botón para tomar una foto, dicho botón se encuentra definido en el componente Camera y ejecuta una función también definida en el componente Camera que al acabar el procesamiento local de ese componente llama a la función "*onTakePhoto*" que se le ha proporcionado como propiedad pasándole como argumento la imagen tomada en base64. La ejecución continua en la función "*onTakePhoto*" que la envía para su procesamiento y almacenamiento en la nube mediante una llamada HTTP de tipo POST.

- **Estado**: Como se ha comentado, cada componente dispone de un objeto llamado estado que va cambiando conforme a la interacción del usuario o debido a algún ciclo de vida de React que se puede desencadenar gracias a un evento (un evento puede ser la interacción del usuario, una llamada HTTP asíncrona que guarda el resultado en el estado, etc.). Esto permite que la aplicación web desarrollada no actualice la página entera, sino solo actualizará el elemento correspondiente mejorando así el rendimiento y la experiencia del usuario. En función de si estamos en un

componente funcional o un componente propiamente dicho, la forma en la que modificamos el estado puede cambiar.

Además, hay que tener en cuenta que al heredar de la clase "*Component*" el componente desarrollado define una serie de métodos públicos que forman el ciclo de vida de React y el método render, lo que permite al desarrollador ejecutar métodos en momentos determinados. Estas funciones que forman el ciclo de vida son "*ComponentDidMount*", "*ComponentDidUpdate*", "*ComponentDidUnmount*", etc.

Por otro lado, la función render es la que contiene el código HTML del componente y cambia de forma dinámica al cambiar una propiedad o el estado.

En la figura 3-45 podemos ver la definición de un componente que hereda de "*Component*" y utiliza "*ComponentDidMount*" para establecer que cada segundo se llame al método "*tick*", que actualiza el estado del componente.

Figura 3-45. Componente de React que actualiza el estado cada segundo

En los componentes funcionales la idea es la misma, pero utilizando métodos definidos en la librería de React ya que no heredamos de "*Component*" lo que supone que serán componentes más ligeros.

- **TypeScript**: Es un lenguaje de programación de código abierto basado en JavaScript que proporciona tipos, facilidades para documentación y la creación de un código más legible y mantenible. Al realizar el proceso de compilación todo el código TypeScript acaba siendo un fichero JavaScript.
- **NodeJs:** Entorno en tiempo de ejecución asíncrono, multiplataforma, de código abierto basado en JavaScript y con una arquitectura orientada a eventos que facilita la escalabilidad de las aplicaciones.
- **Git**: Es un control de versiones distribuido de código abierto y gratuito que permite gestionar el código de proyectos de cualquier tamaño de forma rápida y eficiente.
- **GitHub:** Es una plataforma que utiliza el control de versiones Git para alojar proyectos de desarrollo software, así como múltiples herramientas como Wikis, visor de ramas, sistemas de seguimiento de problemas, herramientas de revisión de código además de flujos de trabajo (workflows) que permiten establecer la integración y entrega continua con diferentes plataformas y servicios.

En la figura 3-46 podemos ver la interconexión de todas las tecnologías y componentes empleados:

Figura 3-46. Despliegue de aplicación web mediante GitHub Actions en Azure Static Web Apps

Por tanto, la aplicación web desarrollada con *TypeScript* y *React* sobre *NodeJs* se despliega en *Azure Static Web* haciendo uso del control de versiones *Git* para subir el código a *GitHub* que mediante *GitHub Actions*, al recibir un "push" o un "*pull request*" en una determinada rama, compila el proyecto y si la compilación ha tenido éxito despliega una nueva versión de la aplicación en la nube.

3.2.2 Procesamiento de la imagen en Azure Function

Además de la interfaz web, se ha utilizado el servicio de *Azure Function* para procesar la imagen tomada mediante la interfaz web, guardar la imagen en *Azure Blob Storage* y guardar los resultados de las predicciones en los metadatos del fichero, como se puede observar en la figura 3-47.

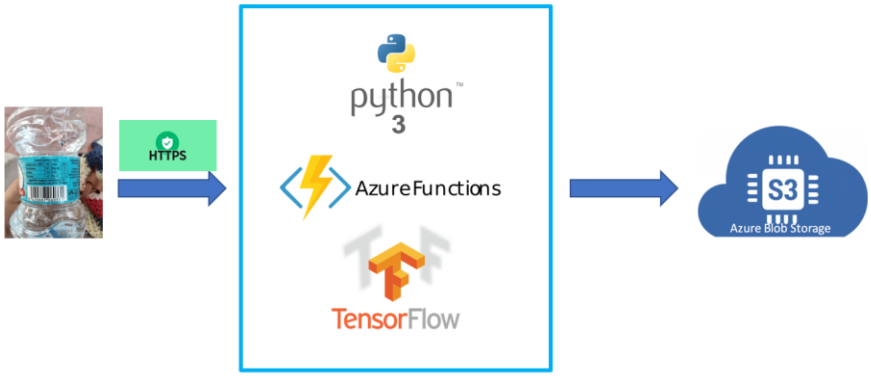


Figura 3-47. Procesamiento y almacenamiento de las imágenes tomadas con la aplicación móvil

La interfaz de usuario desarrollada al tomar una imagen la envía mediante una llamada HTTP de tipo POST al punto de acceso que se ha configurado en la *Azure Function*. Este servicio contiene un fichero Python que se encarga de preparar la imagen ajustando sus dimensiones a 224 x 224 y convertirla a un formato concreto para poder realizar la clasificación mediante un modelo de redes neuronales creado y entrenado previamente utilizando los entornos de *Google Colaboratory* y *Kaggle* que se describen en el apartado 3.1.1.

La función de *Azure* hará uso de la biblioteca de *TensorFlow* en *Python* por lo que cuando se recibe una imagen, se redimensiona y se convierte a un tensor mediante la función "*img_to_array*" disponible en "*tensorflow.keras.preprocessing.image*". Una vez se ha transformado la imagen de tamaño 224 x 224 a un tensor de dimensión (1, 224, 224, 3) donde 224 es la altura y anchura de la imagen respectivamente y 3 el número de canales de la imagen a color, en el modelo RGB se realiza la clasificación obteniendo un porcentaje de pertenencia a cada una de las clases disponibles, Cartón, Metal, Plástico o Vidrio. Este resultado como se ha comentado se adjunta a los metadatos de la imagen en forma de JSON y se almacena en *Azure Blob Storage* para que desde la

interfaz web se puedan recuperar todas las predicciones realizadas y guardar de esta forma el histórico de clasificaciones y de imágenes.

Una vez que la imagen se ha transformado al formato tensor se cargará el modelo de *TensorFlow* previamente entrenado como se comenta en el apartado 3 y se realizará una clasificación que se transforma al formato adecuado para asociarse como metadato a la imagen que se va a guardar en *Azure Blob Storage*.

3.2.3 Casos de uso y diagramas de secuencia

En cuanto a los casos de uso, como se representa en la figura 3-48 se tiene en cuenta la realización de la clasificación y el almacenamiento de los resultados. Ambos casos de uso no tienen ninguna limitación en cuanto a plataforma, pero sí en cuanto a dispositivos y conexión, ya que, en el caso de querer realizar una clasificación, el dispositivo empleado debe contar con una cámara, además de conexión a internet.

- **Clasificación de imagen**: Permite tomar una imagen, enviarla a la plataforma *Azure* para su procesamiento y almacenamiento además de retornar el resultado de la clasificación.

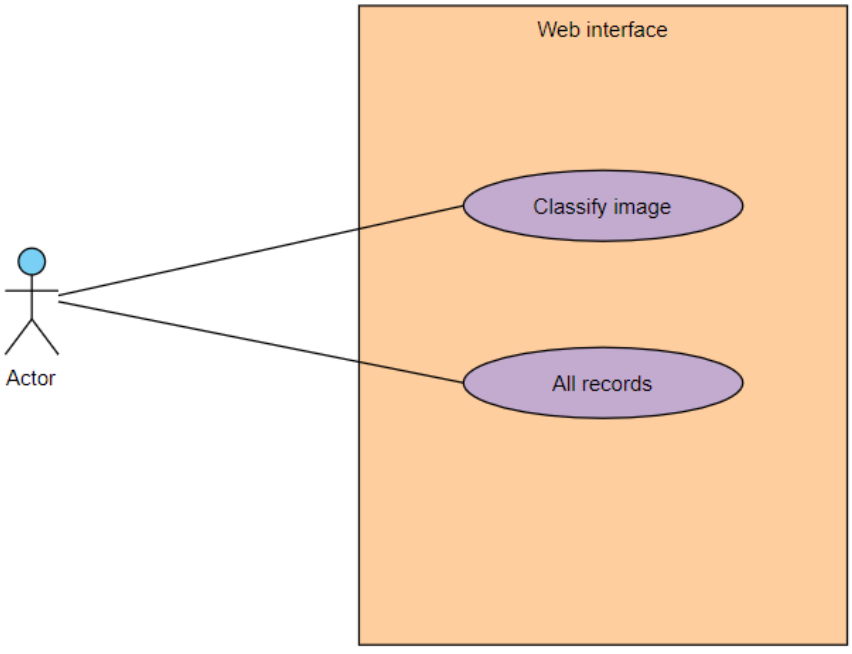


Figura 3-48 - Casos de uso aplicación móvil

- **Histórico de clasificaciones público**: Los usuarios pueden consultar el histórico de todas las clasificaciones realizadas desde el despliegue de la aplicación, así como observar los resultados obtenidos en las diferentes clasificaciones.

En cuanto al flujo de ejecución, en la figura 3-49 y 3-50 se detallan los pasos realizados tanto en el dispositivo como en la plataforma:

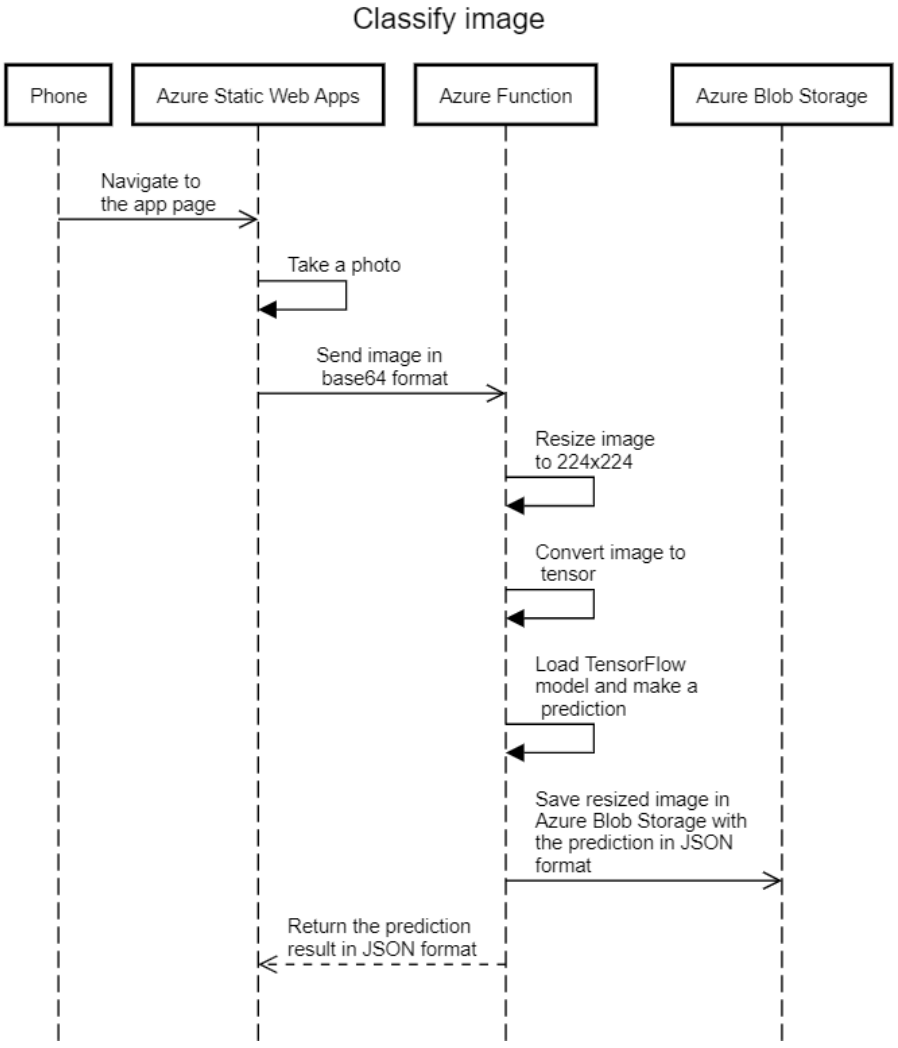


Figura 3-49 - Diagrama de secuencia de clasificación

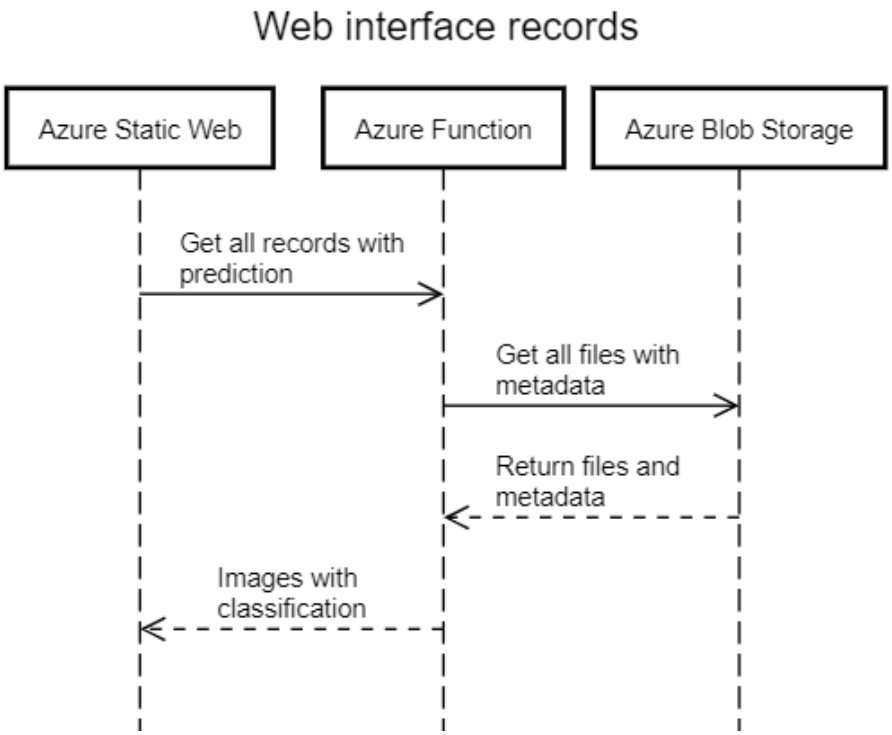


Figura 3-50 - Acceso al histórico de clasificaciones

3.3 Clasificación de residuos automatizada en la nube

En este apartado se describen los aspectos técnicos y funcionales relativos a la primera versión del ecosistema IoT desarrollado que hace uso de diferentes tecnologías.

Para llevar a cabo esta primera versión se ha utilizado el dispositivo ESP-EYE versión 2.1 cuyas características son las que aparecen en la figura 3-51:

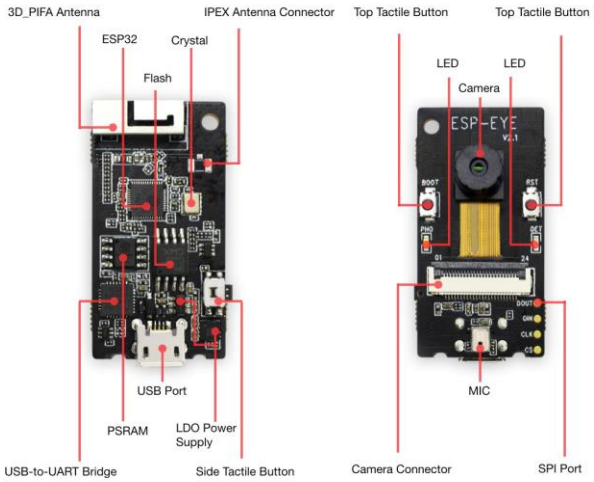


Figura 3-51. Características ESP-EYE

Al utilizar el chip de ESP32 este dispositivo nos permite enviar la imagen tomada a través de Wi-Fi o Bluetooth.

Para facilitar la comprensión de la descripción del flujo que sigue la ejecución completa de la clasificación de un residuo, se puede observar el diagrama de secuencia de la figura 3-52:

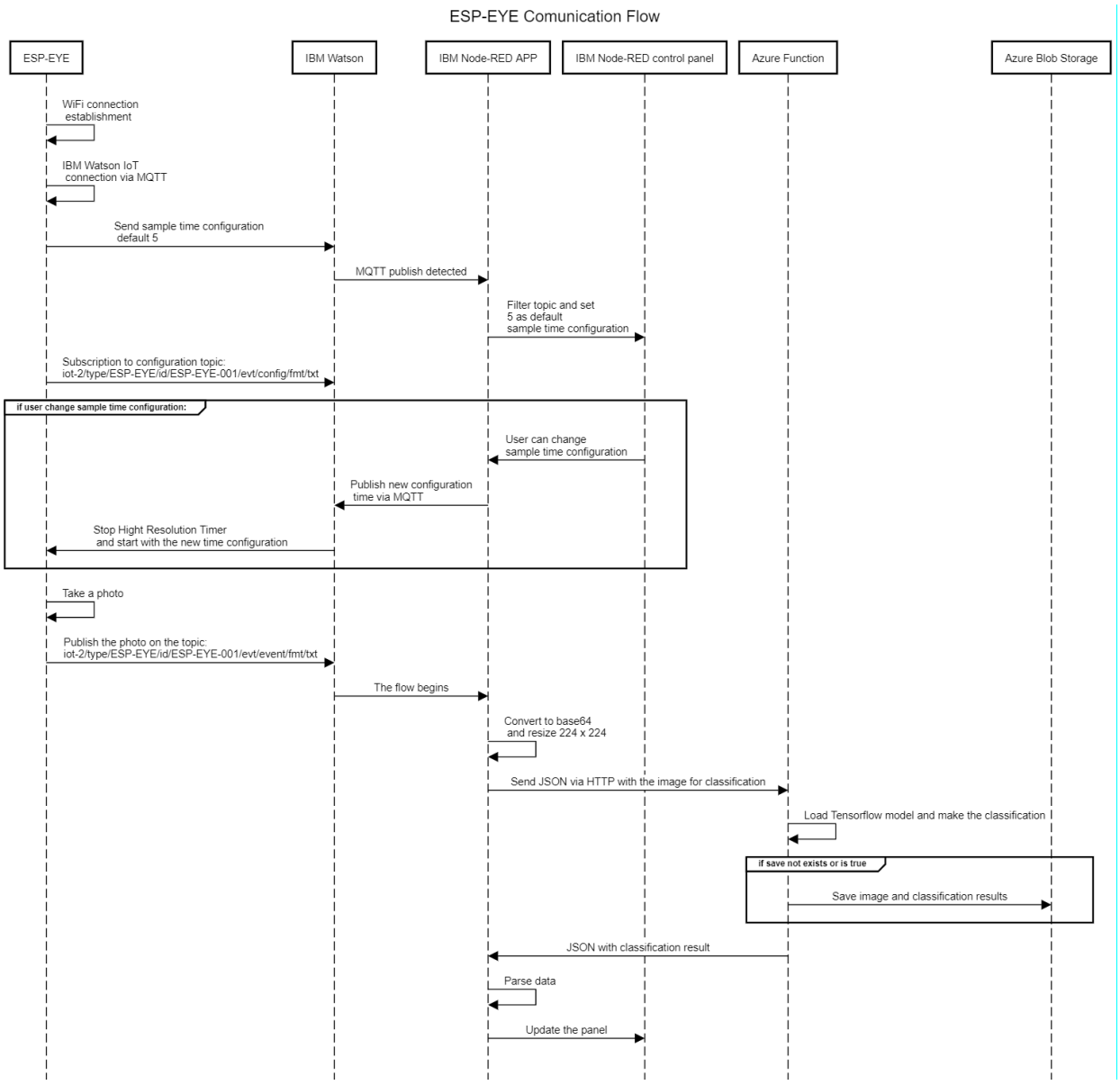


Figura 3-52. Flujo de configuraciones y clasificación de una imagen

En apartados anteriores se ha mencionado la facilidad que ofrecen algunas plataformas en la nube para registrar dispositivos y de esta forma disponer de acceso,

conectividad, actualizaciones de forma centralizada, eficaz y rápida entre otros aspectos. En este desarrollo se han creado los servicios que aparecen en la figura 3-53.

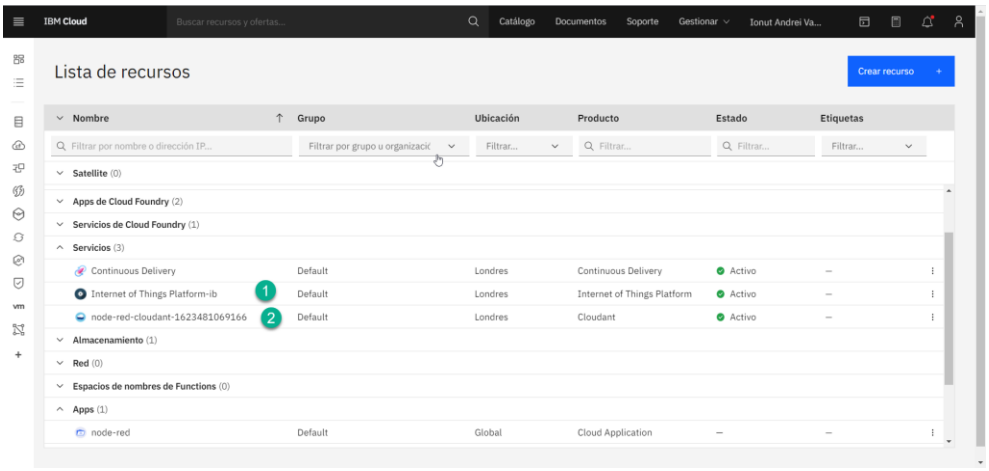


Figura 3-53. Servicios creados en IBM Cloud para interactuar con ESP-EYE

En esta versión se ha hecho uso de IBM Watson IoT que con el plan lite permite registrar el ESP-EYE en la plataforma y provee toda la infraestructura necesaria para establecer la comunicación entre la plataforma y el dispositivo.

Además, con este plan se ha registrado una aplicación de tipo node-red en IBM Cloud que permite la fácil interacción con el dispositivo mediante nodos desarrollados específicamente para servicios alojados en la nube de IBM tal como se muestra en la figura 3-53, donde se accede a la aplicación mediante la URL https://ivaduva-node-red.eu-gb.mybluemix.net/.

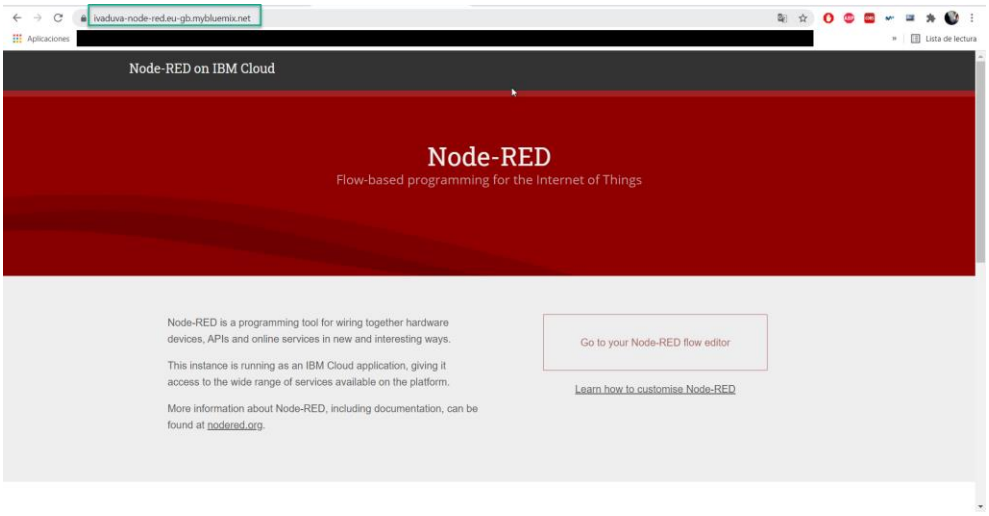


Figura 3-54. Node-red desplegado en IBM Cloud

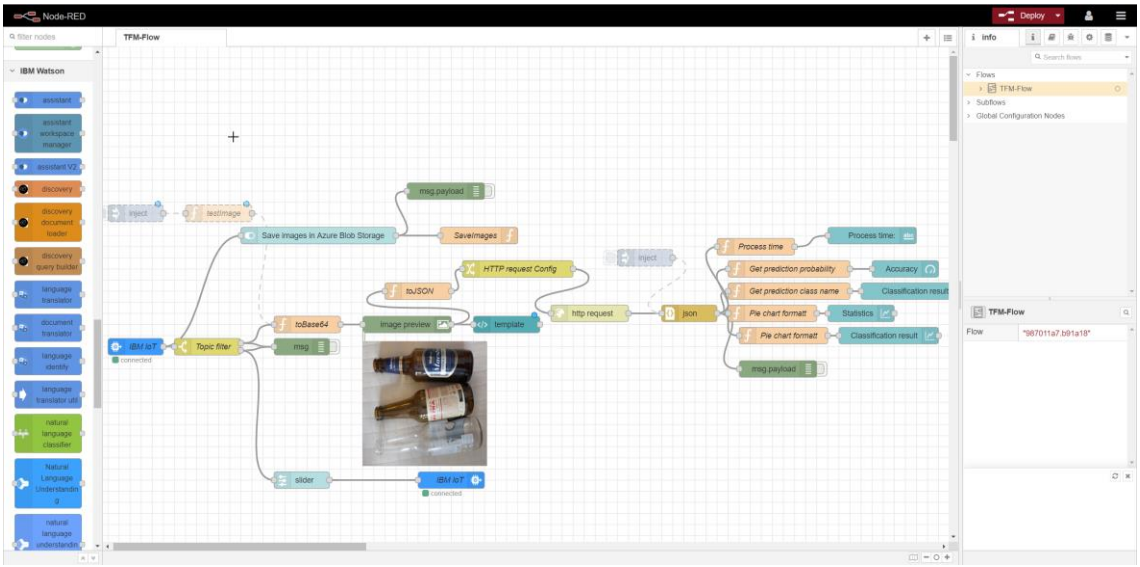


Figura 3-55. Flujo de node-red 1

Como se puede observar en la figura 3-55, el primer nodo es IBM IoT, un nodo que permite establecer la comunicación con el dispositivo registrado en IBM Watson cuyo servicio aparece creado en la figura 3-53 y cuyos detalles aparecen en la figura 3-56 considerando que el registro del dispositivo es intuitivo por lo que lo único que se requiere es rellenar el correspondiente formulario:

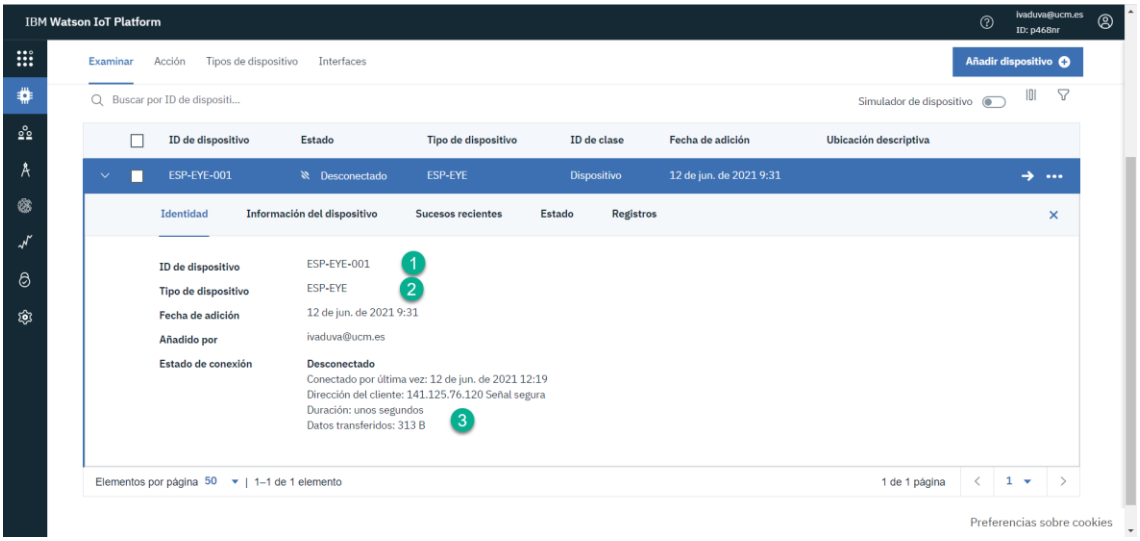


Figura 3-56. ESP-EYE virtual en IBM Watson IoT

Al disponer de este dispositivo registrado, IBM nos provee de toda la infraestructura necesaria para realizar la comunicación tanto desde la plataforma al dispositivo como viceversa, tal como se comenta previamente.

Para establecer la comunicación con el dispositivo desde *node-red*, tal como se ha comentado, se utilizará el nodo IBM IoT, cuya configuración realizada es la que aparece en la figura 3-57:

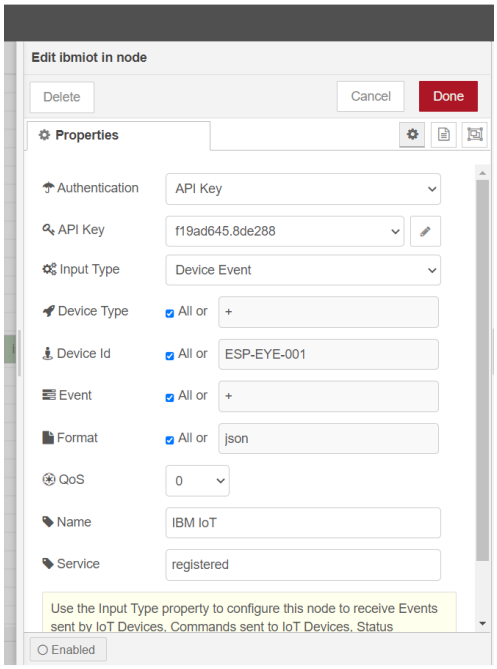


Figura 3-57. Conexión con ESP-EYE desde node-red

Como se puede observar, para establecer esta conexión antes debemos registrar una aplicación que va a permitir realizar la autenticación entre el dispositivo y la nube.

En la figura 3-57 se establece que cada vez que se produzca un evento en el dispositivo con ID ESP-EYE-001(que se ha registrado previamente según el panel de la figura 3-56) este se habilite, si bien en realidad cualquiera porque se ha marcado la casilla "All", pero como solo hay uno, no hay que aplicar ningún filtro más. Además, los datos del evento pueden estar en cualquier formato y la opción QoS del bróker de MQTT es 0, es decir que no se encolan los mensajes.

A continuación, desde el dispositivo se publicará la imagen en formato buffer al TOPIC "*iot-2/type/ESP-EYE/id/ESP-EYE-001/evt/event/fmt/txt*" cuya nomenclatura se establece en la documentación de IBM. Dicha estructura contiene información relativa al tipo de dispositivo (ESP-EYE), el id de ese dispositivo (ESP-EYE-001) y después información relativa al id de evento (evento) y al formato de los datos (txt). También se pueden aplicar todas las *wildcards* establecidas en la documentación de MQTT.

La imagen llegará desde el dispositivo con resolución 240 x 240 y en formato buffer, por lo que se va a convertir a base64 y se va a redimensionar al tamaño 224 x 224.

En la figura 3-55 se puede observar cómo dicha imagen se enviará vía HTTP a la función de Azure comentada en apartados anteriores, que se encargará de realizar la clasificación y devolver un JSON con dicha clasificación además de guardar tanto la imagen como el resultado en Azure Blob Storage (si así se establece en la llamada) para que se pueda consultar desde la aplicación web.

Finalmente, se ha desarrollado un panel de control que permite observar todos los detalles de las clasificaciones realizadas, así como permitir a los usuarios interactuar con el dispositivo cambiando la frecuencia con la que se toman las imágenes, decidir si las imágenes se deben guardar en Azure Blob Storage y observar el tiempo de procesamiento de la imagen, entre otras, tal como muestra la figura 3-58:

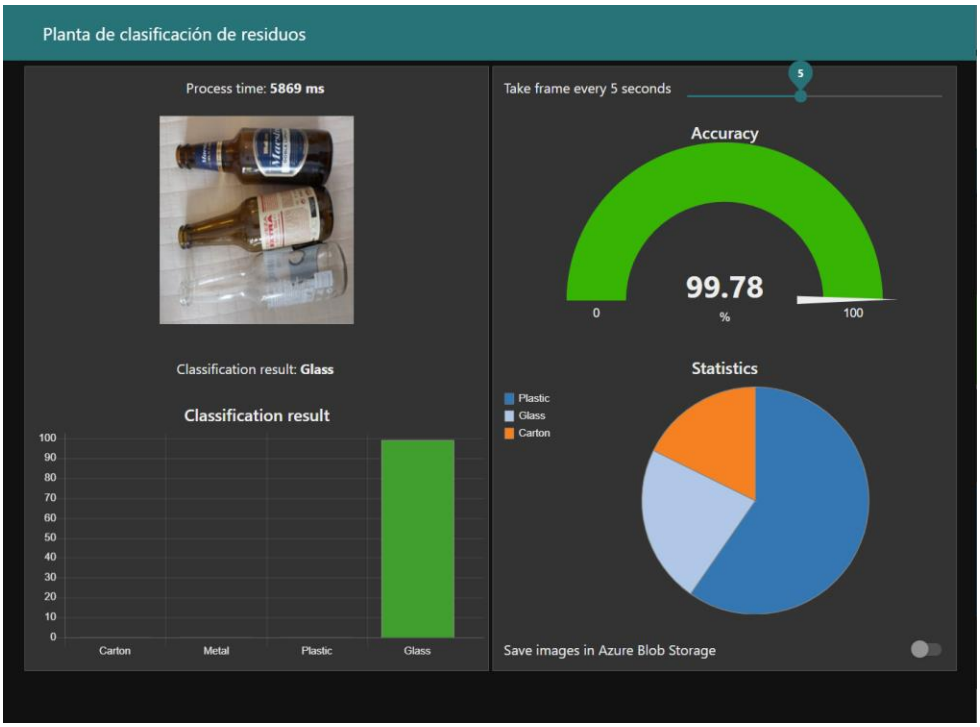


Figura 3-58. Panel de control

El panel de control consta de cuatro partes. En la esquina superior izquierda podemos observar la imagen enviada desde el ESP-EYE y la etiqueta de la clasificación realizada justo debajo. Encima de la imagen se puede identificar el tiempo que ha transcurrido desde que la imagen ha llegado al nodo "toBase64" (ver figura 3-55) hasta que se

obtiene la clasificación realizada en Azure Function, es decir hasta el nodo "Process time".

En la esquina inferior izquierda se puede observar la proporción asignada a cada clase en la clasificación. En otras palabras, el resultado completo de la clasificación realizada en Azure.

En la esquina superior derecha se puede observar el valor del "accuracy" que el modelo ha determinado para esa imagen. También el usuario puede cambiar la frecuencia de muestreo, que por defecto es de cinco segundos. Los valores permitidos están comprendidos en el rango [1,10].

En la esquina inferior derecha se ha generado un gráfico por cada clasificación realizada ofreciendo información valiosa sobre la cantidad de residuos clasificada y de que tipo. Además, el usuario puede decidir si almacenar las imágenes enviadas por el ESP-EYE en el contenedor de Azure Blob Storage.

Tras la descripción del procesamiento realizado en la nube y una vez que llega la imagen a continuación se describen los aspectos técnicos relativos al firmware desarrollado para que el ESP-EYE sea capaz de tomar la imagen y enviarla vía MQTT a IBM Cloud.

Como el ESP-EYE está basado en el SoC ESP32 de Espressif se ha decidido utilizar el lenguaje de programación C en el entorno de desarrollo de ESP-IDF desarrollado por Espressif.

Además, se ha utilizado como base el proyecto de Camera Web Server disponible en el repositorio oficial de Espressif, esp-who.

Esta decisión se ha tomado ya que, aunque no se haya utilizado nada de la funcionalidad del proyecto Camera Web Server, este proyecto proporcionaba toda la estructura (*scaffolding*) necesaria además de la principal ventaja como es la configuración ya establecida con la librería esp32-camera de Espressif, que es la que proporciona todos los drivers necesarios para tomar las capturas, así como utilidades para la gestión de las imágenes.

En cuanto a la funcionalidad desarrollada, en primer lugar, se ha establecido que el dispositivo debe conectarse a una red Wi-Fi para su correcto funcionamiento, por lo que, si no se proveen las credenciales para esta conexión, el ESP-EYE no va a realizar la captura de las imágenes.

Una vez establecida la conexión Wi-Fi, el dispositivo se va a conectar mediante MQTT a IBM Watson IoT haciendo uso de las credenciales obtenidas en apartados anteriores. Al igual que con la conexión Wi-Fi, si el dispositivo no se conecta a la nube correctamente, lo volverá a intentar hasta que lo consiga. Hasta que no esté correctamente conectado no enviará ninguna muestra.

Una vez establecida la conexión con IBM Watson, se enviará la configuración por defecto en lo relativo al tiempo de captura de imágenes, el dispositivo se va a suscribir al topic "*iot-2/type/ESP-EYE/id/ESP-EYE-001/evt/config/fmt/txt*" para poder recibir los posibles cambios al tomar las muestras y se desencadenará un temporizador de alta resolución (*High resolution timer*) cuyo valor por defecto es de cinco segundos y configurable vía *menuconfig*.

Cuando el timer vence, se desencadena la ejecución de una función encargada de tomar una imagen y enviarla mediante MQTT a IBM Watson, momento a partir del cual se realiza el procesamiento mencionado anteriormente.

Por último, cabe mencionar que todos los parámetros relativos a la conexión con IBM Watson son configurables vía *menuconfig*, así como las credenciales de la red Wi-Fi.

3.4 Clasificación de residuos automatizada en el edge

A continuación, se describe el proceso para realizar la clasificación de los residuos directamente en el propio dispositivo ESP-EYE, evitando de esta forma costes de licenciamiento y mejorando los costes computacionales.

Para llevar a cabo la clasificación en el ESP-EYE se ha generado un nuevo modelo mucho más sencillo que el que se describe en apartados anteriores debido a las limitaciones del dispositivo (principalmente los 4 Mb de memoria que posee).

Este nuevo modelo de red está formado por 8.894 parámetros y su estructura es la que se muestra en la figura 3-59.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 24)        672
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 24)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 24)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 12)        2604
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 6)           654
_________________________________________________________________
flatten (Flatten)            (None, 486)               0
_________________________________________________________________
dense (Dense)                (None, 10)                4870
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 66
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 28
=================================================================
Total params: 8,894
Trainable params: 8,894
Non-trainable params: 0
_________________________________________________________________
```

Figura 3-59. Modelo básico para clasificación en ESP-EYE

Dicho modelo establece que las imágenes han de ser a color con un tamaño de 28 x 28, lo que da lugar a la dimensión de entrada del modelo de (28, 28, 3).

Para realizar el entrenamiento y la búsqueda de los parámetros óptimos de este modelo se ha utilizado el mismo conjunto de datos que en ejemplos anteriores (Garbage Classification) redimensionado a tamaño 28 x 28. Para ello se ha generado un nuevo dataset alojado en Kaggle con estas características, denominado Garbage Classification Color 28x28.

Como parámetros se ha utilizado un learning rate de 1e-3, un batch size de 16 y 500 epoch. Los resultados obtenidos se pueden ver en las figuras 3-60 y 3-61.

Como se puede observar en la figura 3-60, a partir de la *epoch* 80 aproximadamente aparece un claro *overfitting*. Al evaluar el modelo con el conjunto de test se obtiene una precisión del 74% con un error del 78%, por lo que es de esperar que los resultados de las clasificaciones de este modelo no sean muy precisos, debido a la simplicidad del propio modelo.

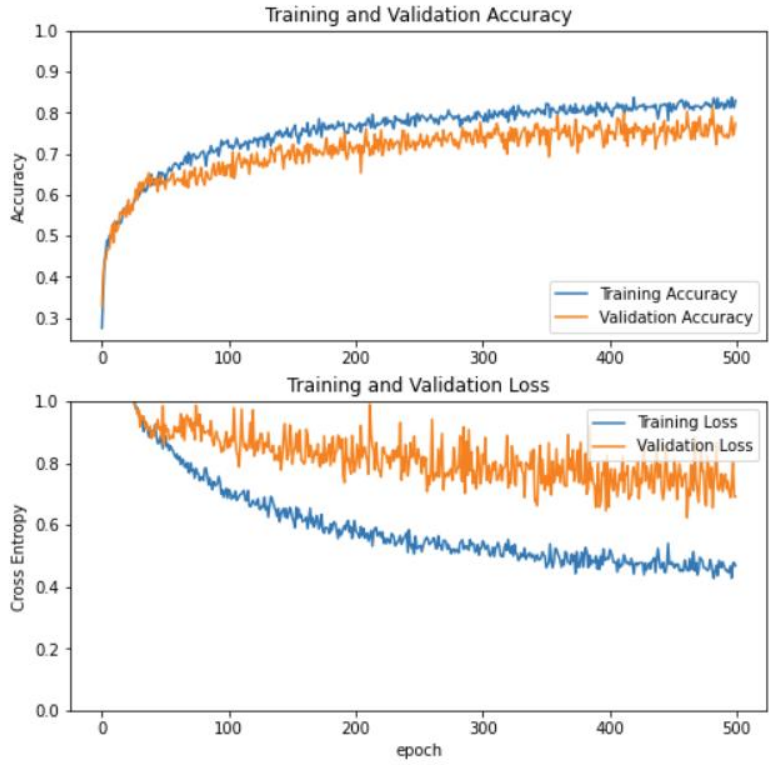


Figura 3-60. Resultado obtenido al entrenar el modelo básico del ESP-EYE

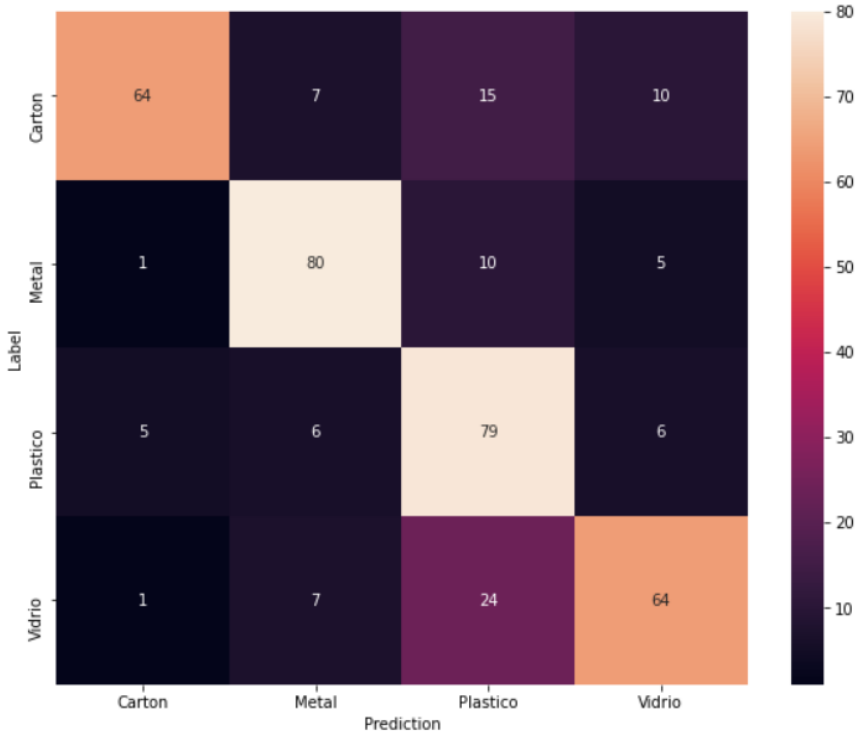


Figura 3-61. Matriz de confusión del conjunto de test sobre el modelo básico

Una vez entrenado el modelo, TensorFlow proporciona funciones nativas para convertir dicho modelo a su versión *lite* que a su vez se puede transformar a un array que el dispositivo ha de cargar en memoria para poder realizar la inferencia.

Hay que tener en cuenta que el tamaño mínimo del *frame* que permite la librería esp32-camera de Espressif es de 96x96, por lo que en este caso se tomarán imágenes de tamaño 96x96x3, es decir en formato RGB que se transforman a 28x28x3 en el propio dispositivo para poder realizar la inferencia.

Además, una vez realizada la inferencia, el dispositivo enviará los resultados de la clasificación y la propia imagen (en este caso para poder determinar el tipo de residuo de que se trata y con fines académicos, en un caso práctico no haría falta) vía MQTT al panel de control que se muestra en el apartado anterior procediendo a su interpretación.

Para ello, se han utilizado dos "*topic*" nuevos: "*iot-2/type/ESP-EYE/id/ESP-EYE-001/evt/classification/fmt/txt*" y "*iot-2/type/ESP-EYE/id/ESP-EYE-001/evt/image/fmt/txt*", donde el primero hace referencia al resultado de la clasificación en formato JSON y el segundo la imagen(ambos se envían como texto plano).

Al realizar la clasificación de los residuos en el propio dispositivo los tiempos mejoran significativamente, de forma que el proceso completo de tomar una imagen, procesarla, clasificarla y enviarla a la nube junto con la clasificación realizada tardaría alrededor de 290-350 ms, como se puede observar en la figura 3-62.

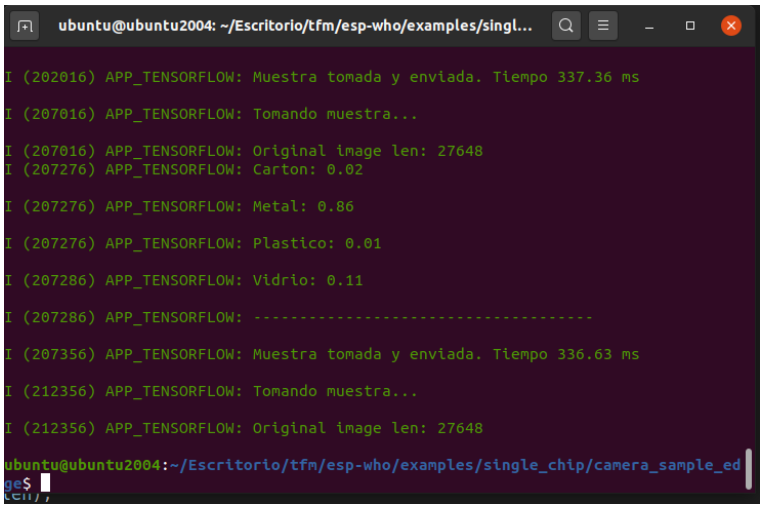


Figura 3-62. Tiempos de inferencia modelo básico en ESP-EYE

Capítulo 4 - Conclusiones y trabajo futuro

4.1 Conclusiones

En este trabajo se propone una solución conceptual para la clasificación de residuos en una planta de reciclaje. Tras la revisión de una serie de tecnologías existentes y posibilidades en uso se ha llegado a la mencionada solución IoT propuesta, que servirá para ayudar a las personas a clasificar correctamente los residuos antes de ser depositados en el cubo correspondiente, así como mejorar la eficiencia de las plantas de clasificación de residuos urbanos haciendo uso de dispositivos IoT.

Para ello se han analizado las diferentes plataformas en la nube que ofrecen algún tipo de servicio relacionado con IoT, así como sus ventajas e inconvenientes, se han utilizado dos de ellas, Azure e IBM Cloud gracias a los planes para estudiantes o de prueba, para llevar a cabo el desarrollo de una aplicación web multiplataforma y un ecosistema IoT capaz de clasificar residuos en diversas categorías, cartón, plástico, metal y vidrio en un rango de tiempos entre tres y diez segundos debido a los planes y las suscripciones utilizadas, donde es de suponer que al mejorar estas suscripciones (aumentando el coste evidentemente) mejoraría el rendimiento.

Además, se ha generado un firmware capaz de tomar una imagen y enviarla vía MQTT a la nube para su procesamiento y otro capaz de tomar una imagen y realizar el procesamiento en el propio dispositivo, para finalmente enviar tanto el resultado de la clasificación como la imagen vía MQTT a un panel de control, obteniendo tiempos de respuesta mucho menores que en la clasificación en la nube, en un rango de 290-350 ms, pero perdiendo bastante precisión debido a la simplicidad del modelo.

Finalmente se puede concluir que se ha creado y entrenado un modelo de redes neuronales capaz de clasificar con una eficiencia de más del 90% residuos urbanos como el metal, cartón, vidrio y plástico. Dicho modelo es independiente de las aplicaciones que se le puedan dar, por tanto, reutilizable como se ha podido comprobar al utilizar el mismo modelo para permitir la clasificación de residuos mediante la aplicación web (especialmente pensada para su ejecución en dispositivos móviles) y mediante conexiones entre diferentes plataformas en la nube como puede ser IBM

Watson IoT que se comunica con diferentes dispositivos ESP-EYE que toma imágenes y enviar para su procesamiento y análisis, facilitando de esta forma la gestión de los residuos tanto en los núcleos urbanos antes de su recogida como en las propias plantas de clasificación.

4.2 Trabajo futuro

Seguidamente se describen los aspectos que podrían mejorar o explorar la continuación de este proyecto.

Como ya se ha comentado en apartados anteriores, se podría aplicar parte de este trabajo (por ejemplo, el modelo empleado para la clasificación de residuos) en dispositivos instalados directamente en los cubos de basura, de esta forma se asegura siempre una clasificación parcial, antes de que los residuos lleguen a las plantas de clasificación.

También debemos tener en cuenta que los residuos llegan a las plantas de clasificación en camiones que suelen seguir siempre las mismas rutas y depositar los residuos en lugares específicos de la planta en función de su origen, por lo que se podría aplicar una solución IoT de forma que en función de la ruta que realiza el camión redirigirlo al vertedero correspondiente sin necesidad de la intervención de operarios, además de la posibilidad de enviar estadísticas directamente a una base de datos para su futuro análisis en lugar de recolectar esos datos manualmente.

En cuanto a la solución propuesta, es fundamental realizar el procesamiento de la imagen en el Edge, es decir, en el propio dispositivo para obtener un tiempo de respuesta mucho mejor a la hora de realizar la clasificación además de mejorar los costes tanto energéticos como de procesamiento en la nube. En este punto el modelo utilizado para realizar la inferencia en el dispositivo se podría mejorar para que identifique mejor los residuos.

Si la clasificación de la imagen no se puede realizar en el dispositivo por alguna limitación, sería interesante contemplar soluciones de caja ofrecidas por IBM Cloud, ya que dispone de una importante cantidad de servicios inteligentes.

Por otro lado, aunque acabamos de comprobar que un problema de clasificación puede satisfacer la necesidad de clasificar residuos, sería interesante comparar tanto resultados como tiempos de respuesta utilizando detectores de objetos tipo YOLO.

Desde el punto de vista de la clasificación o detección de imágenes, es necesario disponer de un conjunto de datos más amplio, ya que el utilizado en este proyecto se ha generado para tal fin y será público, pero es de un tamaño reducido, lo que limita las posibilidades de rendimiento.

Por último, se podrían cruzar varios tipos de datos antes de decidir el tipo de residuo, haciendo uso de otros sensores, como por ejemplo una cámara térmica, espectrómetro o cualquier tipo de dispositivo capaz de aportar información extra sobre el residuo a detectar, ya que de esta forma la clasificación de residuos sería más fiable.

Capítulo 5 - Introduction

Every day, new technological advances are taking place that are gradually changing people's daily lives. These advances are considered part of what is known as the Industrial Revolution Charter, or technically Industry 4.0.

In this new industry, digital technologies that seek to deepen the search for advanced production techniques, intelligent automated operations, people management and safety, among others, stand out. The fields of robotics, massive data analysis, artificial intelligence and above all the Internet of Things (IoT) stand out.

This paper will make use of artificial intelligence and IoT to improve processes, efficiency, and profits by applying smart solutions in municipal waste sorting in municipal waste sorting plants.

To achieve the process improvements discussed above, image recognition techniques and algorithms will be applied to specific parts of the recycling process using cloud-based platforms and devices.

5.1 Motivation

Technology is constantly evolving to better meet human needs. As technology advances, parallel progress is made in various areas of society, healthcare and industry in general. Since the beginning of the 20th century, the world's population has grown exponentially thanks to technological, health and social advances, among others. With the increase in population, needs have increased, so a revolution has begun in the industrial field to satisfy these needs. The aim of this industrial revolution is to improve processes to generate more products at lower cost and increase profits, as well as to satisfy human needs.

As the population, needs and, ultimately, consumption increase, a large amount of waste is generated, which will lead to the creation of landfills. In the middle of the 20th century, waste began to pose an environmental problem because, unlike in previous centuries when most of the waste generated was organic or its disintegration time was short, in the middle of the 20th century waste with much longer disintegration times

began to appear and, as previously mentioned, largely due to the increase in population, consumption and industrialization. This waste is largely made up of inorganic remains, such as paper, cardboard, aluminum, glass and above all plastic, which are the elements considered in this work.

This gives rise to the need to manage waste efficiently and sustainably in order to reduce pollution at all levels, such as atmospheric, water, soil, etc.

Due to this environmental problem generated, different solutions appear to mitigate it, including recycling plants, sorting plants, waste division and the establishment of specific points for waste collection (garbage cans, containers, etc.).

Over the years, waste has been classified in various ways according to its physical state, its origin or its chemical structure. This work focuses on the classification of inorganic solid waste such as cardboard, metal, glass and plastic generated in urban centers and coming from domestic activities, restaurants or public places.

As previously mentioned, the management of these wastes is complicated since large quantities of waste materials are generated every day, so this would be the starting point for their treatment. Once generated, they are deposited in containers and/or waste garbage cans within the urban centers where they are not always correctly classified. Therefore, one phase of the management of this waste is the collection, which can be general (no classification is taken into account) or selective (specific containers that identify the waste by the same characteristics).

It is at this point in the municipal solid waste management process that IoT ecosystem-based solutions could already be applied to facilitate sorting, even before sorting at a specific waste treatment plant. In this way, smart containers capable of sorting waste optimally and without errors could be made available, allowing a simpler and more efficient waste management, but increasing the cost of this management by requiring smart containers.

Continuing with the waste management process, once waste has been classified in one way or another, it must be transported to specific facilities (landfills, waste treatment plants, clean point, etc.).

To solve the problem that could involve the cost of containers today, as Industry 4.0 is in its infancy, this work proposes the development of a multiplatform web application that will help people to make a correct classification of waste in case of doubt, in addition to the development of a hypothetical IoT ecosystem, and therefore conceptual in nature, to be applied in waste sorting plants, which raises the increase in efficiency and therefore improving the treatment and management of urban waste.

5.2 Analysis of current solutions

Before starting with the development, a research on existing projects that could be interesting in the context of a waste sorting plant has been carried out.

Among the projects analyzed, the following stand out:

- DustBot: It is a project developed by a group of Italian researchers for a duration of 36 months that started in 01/12/2006 and ended in 30/11/2009. The objectives of this project are the creation of robots capable of collecting garbage in urban environments, streets, parks, residences through the use of machine learning and different sensors. The robot would also be able to collect information on air quality in saturated urban areas and then send it to a platform capable of storing and processing it in the form of graphs to facilitate decision-making. The project also proposes that the robots send information in order to improve the robot's own learning.
- Robot-based Autonomous Refuse (ROAR): is a project developed by thirty-five students from three universities in 2017. The objective of this project is the collection of garbage bins in urban areas. To achieve this goal they use a garbage truck improved thanks to IoT, making use of a camera mounted on the back of the truck to allow the driver to see the process of emptying the container, a robot capable of receiving coordinates to go around obstacles to collect the bin and take it to the truck and a drone that will be mounted on top of the truck so that when a bin is to be collected, the drone first takes altitude to find the location of the bin and then send it to the collecting robot so that it can go to the bin and take it to the truck to be emptied.

- Seabin: This project was born in 2014 with Australians Andrew Turton and Pete Ceglinski as a startup. The main idea comes from the garbage bins used on land extrapolating the same idea in the water. The project begins with the purchase of a disused furniture factory in Palma de Mallorca in 2014 and thanks to micro-financing campaigns (crowdfunding) and obtaining partners such as Poralu Marine (company that offers high-end solutions in marine environments) and La Grande Motte French marina, they manage to boost the project and create the first prototype in 2016 which is installed in Mallorca. In 2017 they continue to promote the project through online platforms such as booking.com, thus getting more partners to finally start its commercial use and large-scale distribution. Today they have around 860 bins deployed that capture 3612 kg of waste every day. On the technical side, no manuals or detailed technical specifications on the construction and operation of these garbage cans have been found, just some features such as low power consumption (500 watts) capable of capturing micro plastics with a size greater than 2 millimeters, weight of the garbage can with the support of 55 kg, 6 meters of cable, dimensions 500 x 500 mm x 1800 mm.

- River Trash Collector System (RTCS): This project is a prototype developed by several faculties of Universiti Teknikal Malaysia Melaka (UTeM). This project is based on the idea of the Seabin project and is conceived as a possible improvement/adaptation for rivers. Some of the improvements are conceptual such as the use of water to improve the buoyancy of the container instead of using an iron or stone ballast placed under the container, they propose to open four valves that fill the walls of the container with water placing it at river level. They also propose the use of materials in different proportions to the original project such as 10 kg of metal for the water pump part, 4 kg of PVC coating, 15 kg of wood underlay and other components to a lesser extent resulting in a container weight of 30 kg (20 kg less than the Seabin v5) and it is proposed that the container floats freely down the river while the Seabin project container remains anchored.

5.3 Objectives

The solutions described in the previous section focus on the problem of automated or semi-automated waste collection. The solution proposed in this work approaches the problem from another perspective: the classification of waste already collected and transported to a sorting plant, although as will be seen in the section on future improvements, part of the ecosystem could be transferred to different environments.

To automate the sorting process in such plants, it is proposed to use different cameras, Figure 1-4, placed in various locations. These cameras will take images that will be processed in two different ways, first, we will see the cloud processing analyzing the times and responses for comparison with the processing in the device itself, where two convolutional neural network models will be used, within the IoT paradigm, to classify the image and return the result to perform the corresponding actions.

In addition, a mobile application is available, Figure 1-5, which allows taking an image that is sent to the cloud to perform the detection of a specific waste to make it easier for people to deposit the waste in the correct container in case of doubt. The images taken are stored in the cloud in order to obtain a waste history that could be used to improve the training of the neural network used to classify a particular waste, obtain statistics on non-conventional waste that cannot be processed in an urban waste sorting plant, for example, waste containing hazardous substances or consisting of several types of materials that cannot be associated with a specific class such as household appliances, batteries, batteries, batteries, aerosols, paints, thermometers, coffee capsules, etc.

The following are the objectives identified in this work taking into account the above premises:

- Create and train a model based on convolutional neural networks (within the deep learning paradigm) capable of classifying waste into four categories: cardboard, metal, glass and plastic.
- Develop a firmware capable of taking an image and sending it for processing.
- Develop a firmware capable of taking an image and classifying it in the device itself, sending the classification result to a control panel via MQTT.

- Have a cloud infrastructure for bidirectional communication with the device (IBM Watson IoT).
- Have an application that, given an image, makes it possible to resize and classify it.
- To have a control panel for monitoring purposes to represent different statistics, as well as to be able to communicate with the device in a friendly way.

5.4 Work plan, contributions and organization of the report

This section will describe the work plan followed to carry out the design and implementation of this project.

First of all, we have analyzed different existing projects and proposals regarding efficient waste management using tools and technologies in the IoT field. Not having found an existing solution that applies IoT and artificial intelligence to improve the efficiency of sorting plants, the solutions whose objectives are mentioned in the previous section have been proposed.

Before starting with the development, in chapter 2 a deep analysis of some of the most important cloud platforms that offer different types of services, some of them directly related to IoT as it is the case of IBM Cloud, Bosh IoT Suite or Azure IoT that can be used in this context, in addition to search and analyze an already trained neural network model that provides input knowledge in waste sorting.

Once the existing solutions, platforms and objectives have been analyzed, the design and decision making process begins. These aspects are detailed in Chapter 3, where the proposed IoT ecosystem and the development of the web application are defined, in addition to answering the questions:

- Where is the waste classification going to be performed?
- Will the images taken by the cameras be stored?
- What will be the performance of the solution if it is decided to perform the sorting in the cloud? And on the device itself?

- Will any service be used to provide the communication infrastructure between the platform and the device?
- How has the model been trained to perform the classification?
- What technologies have been used for the development of the web application?

Finally, Chapter 4 describes the conclusions obtained and some possible improvements and evolutions of this work.

About the contributions made by this work, the following aspects can be highlighted:

- Creation, training and analysis of a neural network model with an "accuracy" of 93% that allows classifying four types of waste: cardboard, metal, glass and plastic.
- Storage of waste images in the cloud in order to have a history and also to be able to generate better data sets to improve the efficiency of the models used.
- Availability of a multiplatform web application that allows any user with a mobile device to classify waste.
- Availability of a public Azure cloud service that, upon receiving an image of any size, is able to perform a classification and return the result in JSON format, i.e. the generation of an intelligent waste classification service that anyone could integrate into their application.
- Development of a hypothetical IoT ecosystem that through the firmware developed for the ESP-EYE device takes images (with a configurable frequency) and sends them via MQTT to the IBM Cloud platform where it is processed by a Node-RED application that communicates with the service developed in Azure to perform the classification and display the image, the classification result, as well as other relevant statistics in a dashboard.
- Analysis of the classification response time in the cloud and the results obtained.
- Analysis of the classification results and response time on the device itself.
- Proposal of improvements and evolutions of this work.

Chapter - Conclusions and future work

The aspects that could improve or explore the continuation of this project are described below.

As discussed in previous sections, part of this work (e.g., the model used for waste sorting) could be applied to devices installed directly in the garbage bins, thus always ensuring a partial sorting, before the waste reaches the sorting plants.

We must also take into account that the waste arrives at the sorting plants in trucks that usually always follow the same routes and deposit the waste in specific locations of the plant depending on its origin, so an IoT solution could be applied so that depending on the route that the truck performs redirect it to the corresponding landfill without the need for the intervention of operators, in addition to the possibility of sending statistics directly to a database for future analysis instead of collecting that data manually.

Regarding the proposed solution, it is essential to perform the image processing on the Edge, i.e., on the device itself to obtain a much better response time when performing the classification in addition to improving both energy and processing costs in the cloud. At this point the model used to perform the inference on the device could be improved to better identify the debris.

If the image classification cannot be performed on the device due to some limitation, it would be interesting to contemplate box solutions offered by IBM Cloud, as it has a significant amount of intelligent services.

On the other hand, although we have just verified that a classification problem can satisfy the need to classify waste, it would be interesting to compare both results and response times using YOLO-type object detectors.

From the point of view of image classification or detection, a larger dataset is needed, as the one used in this project has been generated for that purpose and will be public, but it is of a small size, which limits the performance possibilities.

Finally, several types of data could be crossed before deciding the type of waste, making use of other sensors, such as a thermal camera, spectrometer or any type of device

capable of providing extra information about the waste to be detected, as this way the waste classification would be more reliable.

BIBLIOGRAFÍA

[1] IBM Cloud. [En línea]. Available: 10/04/2021 https://www.ibm.com/es-es/cloud

[2] EAM. [En línea]. Available: 10/04/2021 https://www.dynaway.com/blog/what-is-enterprise-asset-management-eam

[3] IBM Visual Recognition. [En línea]. Available: 10/04/2021 https://cloud.ibm.com/apidocs/visual-recognition/visual-recognition-v3

[4] IBM Watson IoT Platform. [En línea]. Available: 10/04/2021 https://www.ibm.com/es-es/cloud/watson-iot-platform

[5] Watson Assistant. [En línea]. Available: 10/04/2021 https://www.ibm.com/cloud/watson-assistant

[6] Watson Discovery. [En línea]. Available: 10/04/2021 https://www.ibm.com/cloud/watson-discovery

[7] IBM Maximo Application Suite: [En línea]. Available: 10/04/2021 https://www.ibm.com/products/maximo/predictive-maintenance

[8] IBM Watson Machine Learning. [En línea]. Available: 10/04/2021 https://www.ibm.com/cloud/machine-learning

[9] Eclipse Vorto. [En línea]. Available: 11/04/2021 https://www.eclipse.org/vorto/

[10] Bosch IoT Edge. [En línea]. Available: 11/04/2021 https://developer.bosch-iot-suite.com/service/edge/

[11] Bosch IoT Suite. [En línea]. Available: 11/04/2021 https://developer.bosch-iot-suite.com/

[12] Bosch IoT Hub. [En línea]. Available: 11/04/2021 https://developer.bosch-iot-suite.com/service/hub/

[13] Bosch IoT Gateway. [En línea]. Available: 11/04/2021 https://developer.bosch-iot-suite.com/service/gateway-software/

[14] Bosch IoT Things. [En línea]. Available: 11/04/2021 https://developer.bosch-iot-suite.com/service/things/

[15] Microsoft Azure. [En línea]. Available: 11/04/2021 https://azure.microsoft.com/es-es/

[16] Azure IoT Hub. [En línea]. Available: 11/04/2021 https://azure.microsoft.com/es-es/services/iot-hub/

[17] Azure IoT Edge. [En línea]. Available: 11/04/2021 https://azure.microsoft.com/es-es/services/iot-edge/ https://docs.microsoft.com/es-es/azure/iot-edge/about-iot-edge?view=iotedge-2020-11

[18] Azure IoT Central. [En línea]. Available: 11/04/2021 https://azure.microsoft.com/es-es/services/iot-central/

[19] Azure Function. [En línea]. Available: 11/04/2021 https://azure.microsoft.com/es-es/services/functions/ https://docs.microsoft.com/es-es/azure/azure-functions/create-first-function-vs-code-python

[20] Azure Static Web. [En línea]. Available: 11/04/2021 https://azure.microsoft.com/es-es/services/app-service/static/

[21] Lenguajes y entornos sorportados por Azure Function. [En línea]. Available: 11/04/2021 https://docs.microsoft.com/es-es/azure/azure-functions/supported-languages

[22] Desencadenadores de Azure Function. [En línea]. Available: 11/04/2021 https://docs.microsoft.com/es-es/azure/azure-functions/functions-triggers-bindings?tabs=csharp

[23] Azure Insights. [En línea]. Available: 11/04/2021 https://docs.microsoft.com/es-es/azure/azure-monitor/app/app-insights-overview

[24] Dataset Search. [En línea]. Available: 11/04/2021 https://datasetsearch.research.google.com/

[25] Google Research. [En línea]. Available: 11/04/2021 https://research.google/tools/datasets/

[26] Open Images Dataset v6. [En línea]. Available: 11/04/2021 https://storage.googleapis.com/openimages/web/index.html

[27] Kaggle Datasets. [En línea]. Available: 11/04/2021. https://www.kaggle.com/datasets

[28] Waste classification data. [En línea]. Available: 11/04/2021. https://www.kaggle.com/techsash/waste-classification-data

[29] Waste classification data v2. [En línea]. Available: 11/04/2021. https://www.kaggle.com/sapal6/waste-classification-data-v2

[30] Drinking waste classification. [En línea]. Available: 11/04/2021. https://www.kaggle.com/arkadiyhacks/drinking-waste-classification

[31] DustBot. [En línea]. Available: 11/04/2021. https://www.dustbot.org/

[32] ROAR. [En línea]. Available: 11/04/2021. https://www.chalmers.se/en/departments/e2/news/Pages/The-refuse-collecting-robot-has-been-successfully-tested.aspx

[33] Seabin: [En línea]. Available: 11/04/2021. https://seabinproject.com/

[34] River Trash Collector System. [En línea]. 11/04/2021. https://iopscience.iop.org/article/10.1088/1742-6596/1529/4/042029

[35] Poralu Marine. [En línea]. Available: 11/04/2021. https://www.poralu.com/es/lo-que-hacemos/

[36] Bishop, C.M. (2006). Pattern Recognition and Machine Learning, Springer, NY, USA.

[37] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In Proc. of the IEEE conference on computer vision and pattern recognition, pp. 770-778.

[38] Imagenet (2021). Disponible on-line: http://www.image-net.org (Accedido Abril 2021).

[39] Kingma, D.P., Ba, J.L. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980v9. In Proc. 3rd International Conference on Learning Representations (ICLR 2015), pp. 1-15.

[40] Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Proc. 25th Int. Conf. on Neural Information Processing Systems (NIPS'12), vol. 1, pp. 1097-1105.

[41] Murphy, K.P. (2012). Machine Learning: A Probabilistic Perspective. The MIT Press, Cambridge, Massachusetts, USA.

[42] Pajares, G., Herrera, P.J., Besada, E. (2021). Aprendizaje Profundo. RC-Libros, Madrid.

[43] Ruder, S. (2017). An overview of gradient descent optimization algorithms. arXiv:1609.04747v2 [cs.LG].

[44] Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986). Learning representations by back-propagating errors. Nature. 323 (6088), 533–536.

[45] Sandler, M., Howard, A., Zhu, W., Zhmoginov, A., Chen, L.C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18), pp. 4510-4520. arXiv:1801.04381.

[46] Sutskever, I., Martens, J., Dahl, G. Hinton, G.E. (2013). On the importance of initialization and momentum in deep learning. In Proc. 30th Int. Conf. on machine learning (ICML-13) (Dasgupta, S. and Mcallester, D. ed.), 28 Atlanta, GA. pp. 1139–1147.

[47] NodeJS. [En línea]. Available 18/04/2021. https://nodejs.org/es/about/

[50] TypeScript. [En línea]. Available 18/04/2021. https://www.typescriptlang.org/

[51] React. [En línea]. Available 18/04/2021. https://es.reactjs.org/

[52] Git. [En línea]. Available 18/04/2021. https://git-scm.com/

[53] GitHub. [En línea]. Available 18/04/2021. https://github.com/ https://conociendogithub.readthedocs.io/en/latest/#

[54] TensorFlow. [En línea]. Available 19/04/2021. https://www.tensorflow.org/?hl=es-419

[55] Azure Blob Storage. [En línea]. Available 19/04/2021. https://docs.microsoft.com/es-es/azure/storage/blobs/storage-blobs-introduction

APÉNDICE

Material y código desarrollado

En cuanto al material y código desarrollados para llevar a cabo el presente trabajo, se dispone de la siguiente lista de repositorios y recursos a los que se puede acceder de forma pública:

- **Aplicación web para clasificación de residuos**: Este proyecto está desplegado en Azure, de forma que no se necesitan instalaciones ni configuraciones a realizar (solo un dispositivo con cámara, móvil o pc). Para acceder a esta aplicación basta con utilizar este enlace. Hay que tener en cuenta que es una suscripción gratuita por lo que la primera vez que se acceda a este servicio, la respuesta de éste es lenta, pero después estándar.
- **Panel de control**: Este proyecto está desplegado en IBM Cloud, por lo que no necesita despliegue ni configuraciones extra. Para acceder tanto al panel como a la aplicación y flujo de node-red desarrollados, se debe emplear este enlace para ver el flujo y este enlace para ver el panel de control y las siguientes credenciales, nombre de usuario: "miot-tfm", contraseña: "HomerSimpson".
- **Camera-sample y Camera-sample-edge**: El código de estos proyectos se encuentra en este repositorio público desde el 22/06/2021. Para desplegar este código en el dispositivo se debe disponer de un entorno de desarrollo configurado para soportar la versión de ESP-IDF versión 4.1.1, cuyas indicaciones aparecen en la documentación oficial de Espressif, en este enlace. Una vez configurado el entorno, se deben utilizar los siguientes comandos (para el desarrollo se ha utilizado Ubuntu 18.04) para compilar y flashear el dispositivo ESP-EYE:
    - Idf.py build
    - Idf.py flash monitor
- **MIOT-TF-PREDICT-API**: El código de este proyecto se encuentra en este repositorio público. Dicha carpeta contiene el código de la aplicación de funciones

desplegada en Azure que realiza la clasificación. Es una API como su nombre indica, la interfaz del usuario se encuentra en el siguiente proyecto.

- **TFM-Interface**: Es un repositorio independiente que utiliza un pipeline de CI/CD con Azure Static Webs para desplegar el front-end de la aplicación web con React. Para su despliegue hay que subir dicho código a un repositorio vinculado a un tenant de Azure donde se establece la integración continúa siguiendo este enlace.
- **Demo camera-sample** (procesamiento en la nube): https://drive.google.com/file/d/1LU7FxoULoF7eMLu11pOxuL6evBaKc08k/view?usp=sharing
- **Demo camera-sample-edge** (procesamiento en el ESP-EYE): https://drive.google.com/file/d/1bzDifrpoSYNw4eN_9RSs7T1_UMzXonwM/view?usp=sharing

  Además, ambos videos se encuentran en el repositorio de los proyectos para prevenir problemas de acceso a Google Drive.