universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Comparación de arquitecturas CNN en la clasificación de objetos del Dataset RoboCup@Home-Objects

Autor: Xiao Yang

Tutora: Lidia Sánchez González

(Junio, 2023)

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías
GRADO EN INGENIERÍA INFORMÁTICA
Trabajo de Fin de Grado

ALUMNO: Xiao Yang
TUTORA: Lidia Sánchez González

TÍTULO: Comparación de arquitecturas CNN en la clasificación de objetos del Dataset RoboCup@Home-Objects

CONVOCATORIA: Junio, 2023

RESUMEN:

Este Trabajo Fin de Grado se centra en desarrollar un sistema de reconocimiento de objetos en robótica social usando visión por computadora y aprendizaje supervisado con los principales frameworks de deep learning basados en redes neuronales como Keras y Yolo. El trabajo consta de diferentes etapas: un estudio exhaustivo del problema, construcción del dataset con las imágenes que se utilizan en la competencia de RoboCup@Home-OBJECTS del año 2019, gestión de proyecto software, implementación de arquitecturas, evaluación de resultados y conclusiones con posibles mejoras. Resultados preliminares demuestran que el enfoque de aprendizaje supervisado con redes neuronales es prometedor en el reconocimiento de objetos en robótica social, logrando alto rendimiento en clasificación. Este trabajo avanza en visión por computadora y su aplicación en interacción humano-robot, sentando bases para futuras investigaciones en percepción visual en robótica social.

Palabras clave: deep learning, reconocimiento de objetos,
aprendizaje supervisado, robótica social, RoboCup@Home-OBJECTS,
redes neuronales, arquitecturas, YoloV8.

ABSTRACT:

This Final Degree Project focuses on developing an object recognition system in social robotics using computer vision and supervised learning with the main deep learning frameworks based on neural networks such as Keras and Yolo. The work consists of different stages: a comprehensive study of the problem, construction of the dataset with the images used in the 2019 RoboCup@Home-OBJECTS competition, software project management, implementation of architectures, evaluation of results and conclusions with possible improvements. Preliminary results show that the supervised learning approach with neural networks is promising in object recognition in social robotics, achieving high classification performance. This work advances computer vision and its application in human-robot interaction, laying the groundwork for future research in visual perception in social robotics.

Índice general

Índice de figuras

Índice de tablas

Glosario de términos

Catálogo de términos específicos del contexto del trabajo.

Callbacks : Funciones utilizadas durante el entrenamiento de una red neuronal para llevar a cabo acciones específicas en momentos determinados, como evitar sobreajuste, guardar el mejor modelo, etc.

Capas : Componentes individuales de una red neuronal que realizan operaciones específicas, como convoluciones, agrupaciones, activaciones, entre otras, para procesar los datos de entrada y generar una salida.

Dataset : Un conjunto estructurado y organizado de datos que se utiliza para entrenar y evaluar modelos de inteligencia artificial.

Entrenamiento : El proceso de ajustar los pesos y parámetros de un modelo de inteligencia artificial utilizando un conjunto de datos de entrenamiento, con el objetivo de mejorar su rendimiento en una tarea específica.

Fine Tuning : Un proceso en el cual se ajustan los pesos y parámetros de un modelo pre-entrenado para adaptarlo a un conjunto de datos o tarea específicos.

librería : Un conjunto de funciones y herramientas predefinidas que facilitan la implementación de algoritmos y modelos de inteligencia artificial.

Métrica : Una medida cuantitativa utilizada para evaluar el rendimiento de un modelo de inteligencia artificial.

Modelo/Arquitectura : La estructura o diseño de una red neuronal, que define cómo se conectan las neuronas y cómo se procesan los datos en una tarea específica.

Optimizador : Un algoritmo utilizado para ajustar los pesos y parámetros de un modelo de inteligencia artificial durante el proceso de entrenamiento, con el objetivo de minimizar la función de pérdida y mejorar el rendimiento.

Red neuronal : Un modelo computacional inspirado en el sistema nervioso biológico que se utiliza en inteligencia artificial para aprender y llevar a cabo tareas mediante el procesamiento de datos.

RoboCup@Home : Una competición internacional de robótica en la que los robots autónomos deben realizar tareas domésticas y de asistencia en un entorno realista.

Transfer Learning : Un enfoque de entrenamiento de redes neuronales en el que se aprovechan los conocimientos previos de un modelo pre-entrenado en un conjunto de datos grande y luego se adapta a un conjunto de datos más pequeño o a una tarea diferente.

YoloV8 : Una arquitectura de red neuronal convolucional (CNN) utilizada para la detección/clasificación de objetos en imágenes y videos, conocida por su enfoque rápido y preciso.

Introducción

En este trabajo fin de grado se explicará la importancia del reconocimiento de objetos en robótica social, qué aplicaciones prácticas pueden tener y se probará los modelos de clasificación de imágenes utilizando la tecnología de redes neuronales para conseguir el mejor modelo que adapte a este tipo de problema problemas.

Planteamiento del problema

La presencia de robots en hogares, lugares de trabajo y espacios públicos es cada vez más evidente, ya que los robots desempeñan un papel clave en la realización de tareas cotidianas. Aunque estas tareas pueden tener finalidades muy diversas, en la mayoría de los casos requieren que los robots tengan la capacidad de ser conscientes de su entorno, es decir, que sean capaces de identificar los objetos en su entorno. Esto ha sido posible gracias a los avances tecnológicos de los últimos años, sobre todo en tecnología de aprendizaje automático, que ha permitido crear sistemas de visión artificial que reconocen objetos, lugares, etc. de forma similar a los humanos.

El objetivo es permitir a los robots puedan interactuar plenamente con los humanos y prestarles ayuda en las tareas domésticas. Para ello, los robots deben ser capaces de determinar su ubicación en la casa de forma autónoma y en tiempo real. Desde una perspectiva sociocognitiva de los robots, el reconocimiento de objetos se convierte en un aspecto clave, que facilita la navegación del robot y prestar servicios adaptados a su entorno.

En el desarrollo de modelos para la clasificación de objetos domésticos han surgido en muchos enfoques, destacando el aprendizaje profundo y, en particular, las redes neuronales convolucionales. Si bien estas redes representan un gran avance, también suponen un reto debido al gran número de arquitecturas y técnicas disponibles. Por lo tanto, se necesitan amplios conocimientos para seleccionar el modelo que mejor se adapte al problema que hay que resolver de la mejor manera posible.

Esto pone de manifiesto la estrecha relación que existe entre la robótica y la inteligencia artificial, ya que cada vez están más entrelazadas. Aunque hay especialistas en cada campo, no siempre hay expertos en ambos, lo que dificulta la integración, sobre todo teniendo en cuenta que se puede trabajar con modelos diferentes. Esto plantea un reto a la hora de aplicarlos en un entorno realista con robots.

Objetivos

El objetivo principal de este trabajo fin de grado es desarrollar y evaluar sistemas de clasificación de imágenes utilizando tecnologías de redes neuronales de inteligencia artificial para enfocar el problema del reconocimiento de objetos en robótica social. Los objetivos específicos de este trabajo son:

1. Investigar y comprender la importancia del reconocimiento de objetos en robótica social y sus aplicaciones prácticas en diversas áreas, como asistencia en el hogar, cuidado de personas mayores, atención médica, entre otras. [45]

2. Explorar los avances en tecnología de aprendizaje automático, particularmente en redes neuronales convolucionales, para el reconocimiento de objetos en entornos de robótica social.

3. Utilizar el conjunto de datos que ha sido utilizado en la competencia robótica RoboCup@Home-Objects [26], que consta de 196.195 imágenes, divididas en 8 categorías principales y 180 subcategorías, se entrenar y evaluar los modelos de clasificación de imágenes para construir el mejor sistema que sea capaz de clasificar los objetos domésticos, y se puede exportar para mejorar la interacción de un robot doméstico.

4. Probar y evaluar diferentes modelos de clasificación de imágenes, incluyendo MobileNetV2, EfficientnetB0, Yolov8, Xception, InceptionV3, Resnet150V2, etc, utilizando la técnica de transfer learning y aplicando fine-tuning con una tasa de aprendizaje relativamente baja para mejorar su rendimiento.

5. Identificar el modelo que mejor se adapte al problema del reconocimiento de objetos en entornos robóticos sociales.

Con estos objetivos se pretende contribuir al avance de la visión por computador y la robótica social mediante el desarrollo de modelos de clasificación de imágenes que puedan implementarse en robots para mejorar su capacidad de interacción y prestación de servicios en entornos domésticos y sociales.

Metodología

En este proyecto fin de grado, el desarrollo del proyecto se sigue la metodología ágil de Scrum [42], Scrum proporciona un enfoque sistemático e iterativo para desarrollar y evaluar el trabajo, en cada iteración Sprint dura dos semanas, excepto la última que tiene 3 semanas de duración, en cada periodo de iteración se trabajaba de manera individual en las tareas asignadas por Scrum Master, en este caso mi tutora Lidia, y en cada reunión de planificación del sprint se empieza con unas tareas nuevas y también se resuelve las dudas que han surgido durante el desarrollo.

En la metodología ágil Scrum existe tres actores principalmente:

- **Equipo desarrollador:** En este caso el alumno que realiza este trabajo fin de grado, es responsable de realizar todas las tareas necesarias para llevar el cabo el proyecto.
- **Scrum Master y Product Owner:** Mi tutora, Lidia Sánchez González, como Scrum Master, identifica y resuelve los obstáculos y problemas que puedan afectar al progreso del proyecto y, como Product Owner, asegura que el trabajo desarrollado cumple con los requisitos y expectativas establecidos.

Tecnologías utilizadas

Las tecnologías utilizadas para realizar este trabajo fin de grado son las siguientes:

- Python
- Visual Studio Code
- Google Colab
- Anaconda
- TensorFlow
- Keras
- matplotlib
- Keras
- numpy
- seaborn
- scikit-learn

Estructura del trabajo

En esta sección se presenta la estructura del trabajo, que aporta una explicación concisa de los contenidos de cada apartado.

- Capítulo 1: Estudio del problema 1, En este capítulo se proporciona un análisis detallado del problema del reconocimiento de objetos en robótica social. Demuestra la importancia de esta tarea en el ámbito de la interacción humano-robot.
- Capítulo 2: Gestión de proyecto software 2, En este capítulo se describe la gestión del proyecto de desarrollo de modelos de clasificación de objetos. Se define el alcance del proyecto, incluyendo la estimación de las tareas y recursos necesarios.
- Capítulo 3: Solución 3, En este capítulo se presenta la solución propuesta, que se basa en la implementación de modelos de redes neuronales convolucionales. Se detallan las métricas seleccionadas, las arquitecturas y técnicas utilizadas. Se detalla el proceso de adaptación de estos modelos mediante transfer learning y fine-tuning.
- Capítulo 4: Evaluación 4, En este capítulo se lleva a cabo la evaluación de los modelos implementados. Se realiza la visualización de los resultados, se compara de los resultados obtenidos y apartar ejemplos de clasificaciones correctas e incorrectas.
- Capítulo 5: Conclusión 5, En el último capítulo se presentan las conclusiones obtenidas de este trabajo. Además, se plantean posibles líneas de trabajo futuro, con el objetivo de mejorar y ampliar la solución propuesta.

Capítulo 1

Estudio del problema

En este apartado se realiza una investigación de la situación actual sobre el reconocimiento de los objetos en robótica social, presentará el problema que ha motivado el trabajo, las herramientas que nos ayuda a resolver el problema y la solución que se ha propuesto.

1.1. El contexto del problema

La robótica social es un campo en rápido crecimiento cuyo objetivo es mejorar la interacción entre humanos y robots en situaciones cotidianas. Para lograr una interacción eficaz, es fundamental que los robots comprendan su entorno y se adapten a las necesidades y preferencias individuales de las personas.

Conocer el entorno es importante por varias razones, como por ejemplo garantizar la seguridad tanto del robot como de las personas que lo rodean. Al comprender la disposición del espacio, reconocer los obstáculos e identificar los peligros potenciales, los robots pueden navegar y operar sin causar accidentes ni daños.

Además, la comprensión eficaz del entorno permite a los robots ejecutar sus tareas con eficiencia. Ya se trate de un robot de limpieza que maniobra entre los muebles, un asistente personal que localiza y trae objetos, la comprensión del entorno permite a los robots planifica, ejecuta sus acciones de forma adecuada y adaptarse a distintas situaciones y preferencias de los usuarios.Al comprender el contexto, los robots pueden modificar su comportamiento, respuestas y funcionalidades en consecuencia. Esta adaptabilidad es importante en la robótica social, donde la personalización y las interacciones a medida son fundamentales para establecer la confianza y satisfacer las necesidades del usuario.

Al comprender los entornos que rodea el robot, nos puede ofrecer una experiencia más intuitiva y natural. Ya sea anticipándose a las necesidades del usuario, adaptándose a sus preferencias o ayudándole proactivamente en sus tareas cotidianas, un robot que comprende su entorno puede crear una interacción más envolvente y satisfactoria para los usuarios.

En resumen, el reconocimiento de los objetos permite el robot comprende el entorno que se rodea, y eso es fundamental para que los robots sociales garantizan la seguridad, ejecutan tareas con eficacia, se adaptan a distintas situaciones, colaboran sin problemas con los humanos y mejoran la experiencia general del usuario. De este modo, los robots se convierten en agentes sociales más capaces e inteligentes, lo que facilita su integración en diversos aspectos de la vida cotidiana.

1.2. El estado de la cuestión

La investigación en el campo de la robótica social y el reconocimiento de objetos se ha beneficiado enormemente de la intersección entre diferentes disciplinas, como la inteligencia artificial, la visión por computadora y el aprendizaje automático. Estas áreas de estudio están estrechamente relacionadas y se complementan entre sí, lo que ha llevado a importantes avances en la capacidad de los robots para interactuar y comprender su entorno. [33]

La inteligencia artificial (IA) se refiere a la capacidad de las máquinas que simula a los seres humanos que realiza tareas que normalmente requieren de la inteligencia humana, como el razonamiento, la percepción, el aprendizaje y la toma de decisiones. Fue en el año 1956 cuando se acuñó el término "inteligencia artificial" durante una conferencia en Dartmouth College de New Hampshire, donde se reunieron los pioneros del campo. El término fue propuesto por John McCarthy, considerado uno de los padres fundadores de la IA.

La aparición de la inteligencia artificial se debe al deseo de los científicos y expertos en computación de replicar la inteligencia humana en máquinas y desarrollar sistemas capaces de realizar tareas complejas de manera autónoma. La base de la Inteligencia artificial es en realidad la acumulación de funciones matemáticas. Pero es capaz de obtener conseguir la capacidad de aprender, razonar y tomar decisiones basadas en datos, similar a cómo lo hacen los seres humanos.

Hoy en día, la inteligencia artificial se ha vuelto extremadamente importante debido a su amplia aplicación en una variedad de industrias y campos. Con el avance

de la tecnología, los algoritmos de IA y el poder de cómputo han mejorado significativamente, lo que ha permitido el desarrollo de sistemas más sofisticados y eficientes. La IA se utiliza en áreas como la medicina, la automoción, la atención al cliente, la seguridad, la logística, la finanzas, la robótica social, y cada vez beneficia a más sectores. [37]

La importancia de la inteligencia artificial radica en su capacidad para procesar y analizar grandes cantidades de datos de manera rápida y precisa, lo que brinda oportunidades para descubrir patrones, realizar predicciones y tomar decisiones informadas. Además, la IA está impulsando la automatización de tareas y procesos, lo que puede aumentar la eficiencia, reducir costos y liberar a los seres humanos de tareas repetitivas.

Existe diferentes tipos de aprendizaje, como el aprendizaje supervisado y no supervisado. En el aprendizaje supervisado, se utilizan conjuntos de datos etiquetados, donde se proporciona información previa sobre las entradas y las salidas deseadas. El algoritmo aprende a asignar las etiquetas correctas a las nuevas entradas en función de los ejemplos proporcionados. Por otro lado, en el aprendizaje no supervisado, el algoritmo debe descubrir patrones o estructuras ocultas en los datos sin la guía de etiquetas previas. [37]

La aparición de la inteligencia artificial ha sido un factor clave en el desarrollo de la robótica social y el reconocimiento de objetos. La capacidad de las máquinas para realizar tareas cognitivas y tomar decisiones se ha vuelto cada vez más sofisticada gracias a los avances en algoritmos y hardware. Esto ha permitido que los robots sean más autónomos y capaces de comprender y responder a su entorno de manera más efectiva.

La visión por computadora es una subrama de la inteligencia artificial que fue desarrollada en los dos últimas décadas, que se centra en enseñar a las máquinas a "ver" e interpretar imágenes o vídeos. Ha surgido como una herramienta fundamental en el reconocimiento de objetos, ya que permite a los robots adquirir información visual del entorno y procesarla para identificar y comprender objetos específicos. Al aplicar técnicas de procesamiento de imágenes y reconocimiento de patrones, la visión por computadora capacita a los robots para percibir y comprender su entorno visual. [30]

En la figura 1.1 se demuestra una esquema de un proceso de análisis de imágenes.

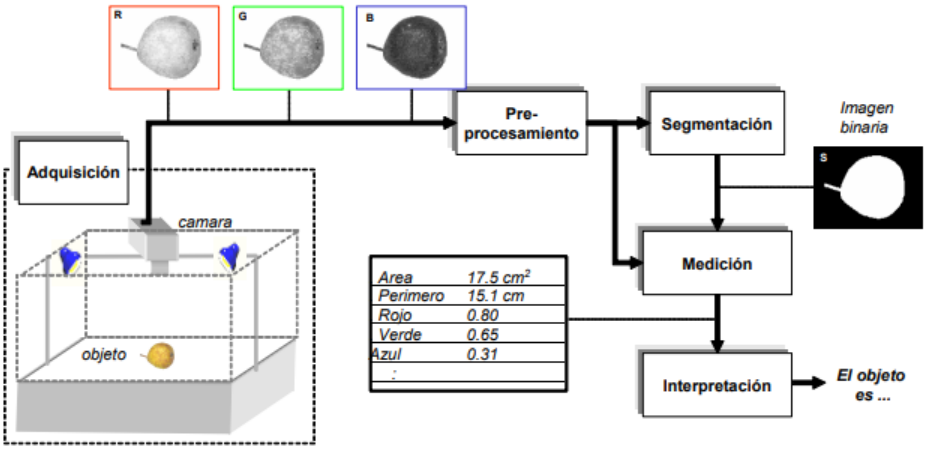

Figura 1.1: esquema de un proceso de análisis de imágenes
Fuente: `http://dmery.sitios.ing.uc.cl/Prints/Books/2004-ApuntesVision.pdf`

El aprendizaje automático una de las ramas más estudiadas actualmente dentro de la inteligencia artificial que se basa en algoritmos y modelos estadísticos para permitir que las máquinas aprendan de los datos y mejoren su rendimiento en tareas específicas sin ser programadas explícitamente. En el contexto del reconocimiento de objetos, el aprendizaje automático permite a los robots adquirir conocimiento y habilidades para identificar y clasificar objetos en función de ejemplos etiquetados previamente [31].

El aprendizaje profundo, a su vez, es una subcategoría del aprendizaje automático que se ha convertido en un pilar fundamental en el reconocimiento de objetos. Basado en redes neuronales artificiales de múltiples capas, estas son modelos computacionales inspirados en el funcionamiento del cerebro humano. Están compuestas por capas de nodos interconectados llamados neuronas, que procesan y transmiten información. Las redes neuronales pueden aprender a través de algoritmos de aprendizaje y ajustar sus pesos y conexiones para realizar tareas específicas, como el reconocimiento de objetos en imágenes.

El entrenamiento de las redes neuronales puede requerir una gran cantidad de recursos computacionales, especialmente si la red es profunda y tiene una gran cantidad de parámetros. Las redes neuronales profundas, como las redes neuronales convolucionales (CNN) utilizadas en el reconocimiento de objetos, pueden requerir entrenamientos prolongados y computacionalmente intensivos. Esto se debe a que el aprendizaje implica la propagación hacia adelante y hacia atrás de grandes volúmenes de datos a través de múltiples capas de la red [24]

En la figura 1.2 demuestra la estructura de una red neuronal.

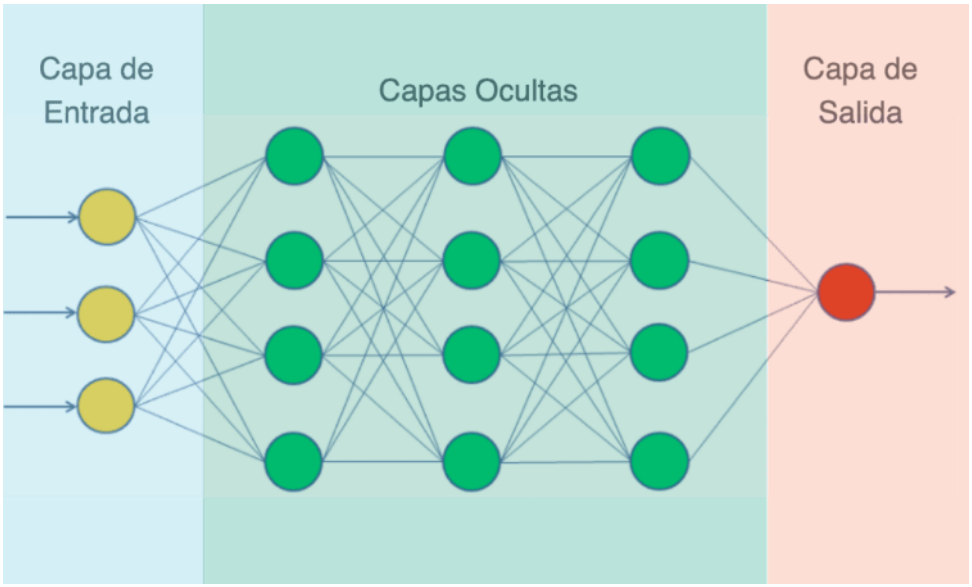

Figura 1.2: Características de una red neuronal
Fuente: *https://aprendeia.com/que-son-las-redes-neuronales-artificiales/*

Por el motivo de gran gasto en los recursos, se promueve la técnica del transfer learning o aprendizaje por transferencia. Esta técnica permite aprovechar el conocimiento y las representaciones aprendidas por una red entrenada previamente en una tarea relacionada y aplicarlas a una nueva tarea similar. Al utilizar transfer learning, se pueden reutilizar los pesos y las características aprendidas de una red pre-entrenada, lo que puede acelerar el entrenamiento y mejorar el rendimiento en tareas específicas. Esto es especialmente útil cuando los conjuntos de datos disponibles para la nueva tarea son limitados. [44]

En la figura 1.3 se muestra la diferencia entre el aprendizaje automático tradicional y usar el transfer learning.

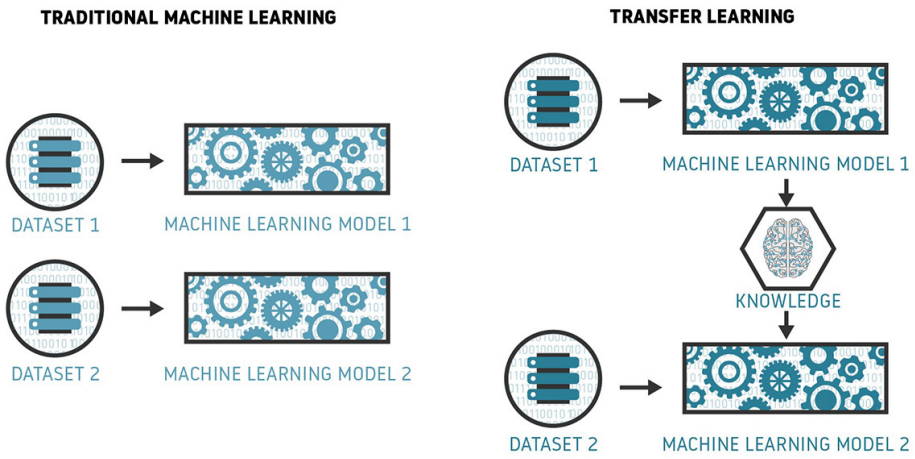

Figura 1.3: Aprendizaje Automático Tradicional vs. Transfer Learning
Fuente: `https://www.pinecone.io/learn/transfer-learning/`

Además del transfer learning, otra técnica que se utiliza en el contexto de redes neuronales y reconocimiento de objetos es el fine-tuning o ajuste fino, el fine-tuning es una técnica de ajuste fino que se utiliza junto con el transfer learning para adaptar una red pre-entrenada a una tarea específica. Permite aprovechar el conocimiento previo de la red y adaptarlo a nuevos conjuntos de datos, mejorando así el rendimiento y la precisión en el reconocimiento de objetos. Esta técnica es de gran relevancia en el campo de la visión por computadora y la robótica social, donde se busca obtener los mejores resultados en el reconocimiento de objetos y la interacción con el entorno. [40]

Esto ha llevado a una mejora significativa en la capacidad de los robots para reconocer y comprender una amplia gama de objetos en entornos cambiantes.

La aplicación de estas tecnologías en la robótica social y el reconocimiento de objetos ha permitido resolver diversos problemas. Por ejemplo, en entornos de cuidado asistido, los robots pueden utilizar la visión por computadora y el aprendizaje automático para reconocer y localizar objetos cotidianos, como medicamentos o utensilios de cocina, para ayudar a las personas mayores o con discapacidad en sus actividades diarias. Además, en el ámbito de la interacción humano-robot, el reconocimiento preciso de objetos permite a los robots comprender y responder a las necesidades de las personas de manera más intuitiva, mejorando la comunicación y la colaboración.

En resumen, la combinación de la inteligencia artificial, la visión por computadora y el aprendizaje automático ha impulsado el desarrollo de la robótica social y el reconocimiento de objetos. Estas disciplinas se complementan entre sí, permitien-

do a los robots adquirir habilidades de percepción visual, comprensión y toma de decisiones. Esto ha llevado a una amplia gama de aplicaciones en diversos campos, como el cuidado asistido, la interacción humano-robot y la automatización de tareas, resolviendo problemas y mejorando la calidad de vida de las personas [13].

1.3. La definición del problema

La intención de este trabajo fin de grado es descubrir sistemáticamente los mejores modelos que existe hoy en día y encontrar el mejor modelo para resolver problemas del reconocimiento de objetos en robótica social. En este contexto, se enfrenta a ellos diversas limitaciones y desafíos.

El reconocimiento preciso de objetos en entornos complejos y dinámicos presenta dificultades debido a la variabilidad de apariencias, iluminación cambiante y oclusiones. Además, la interacción social requiere una detección y comprensión precisa de objetos relevantes para el contexto, como personas, objetos cotidianos y señales de comunicación no verbal.

En nuestra búsqueda de soluciones, se considera diferentes alternativas para abordar este problema. Una opción podría haber sido utilizar métodos tradicionales de visión por computadora, como la extracción manual de características y el diseño de clasificadores basados en reglas heurísticas [36]. Sin embargo, estos enfoques presentan limitaciones en términos de escalabilidad y adaptabilidad a nuevos escenarios.

Para superar estas limitaciones se va a construir un sistema robusto y adaptable, recurrimos al uso de redes neuronales y aprendizaje automático. Estas técnicas aprovechan los avances en el campo de la inteligencia artificial y nos permiten capturar patrones complejos de manera automática a partir de conjuntos de datos etiquetados. Al entrenar las redes neuronales en datos específicos de reconocimiento de objetos en robótica social, se puede lograr una mayor precisión y capacidad de generalización en la detección de objetos relevantes en entornos sociales.

Además, el uso de transfer learning se revela como una estrategia efectiva. Al transferir el conocimiento adquirido por modelos pre-entrenados en tareas relacionadas, se puede acelerar el proceso de entrenamiento y mejorar el rendimiento en nuestro problema específico. Esta capacidad de aprovechar el conocimiento previo nos permite obtener mejores resultados incluso cuando los conjuntos de datos etiquetados son limitados.

Ya que la solución se basa en redes neuronales, aprendizaje automático y transfer learning para abordar los desafíos del reconocimiento de objetos en robótica social, se prueban los modelos de alto rendimiento hoy en día como Resnet, YoloV8, Xception, etc, y se compararan para obtener el mejor modelo que nos ayuda a superar los desafíos del reconocimiento de objetos en robótica social.

Capítulo 2

Gestión de proyecto software

En este capítulo, se especifica todas las restricciones y requisitos del proyecto [25], así como la estimación de las tareas y recursos, el presupuesto, el coste personal, los riesgos, etc, es decir, un gestión del desarrollo del proyecto.

2.1. Alcance del proyecto

2.1.1. Definición del proyecto

En este trabajo fin de grado tiene como objetivo principal el desarrollo de un sistema de visión por computadora para el reconocimiento de objetos en robótica social. Se busca proporcionar a un robot la capacidad de percibir y comprender su entorno a través de una cámara, permitiéndole detectar y clasificar los objetos presentes en su campo visual. Para lograr este propósito, se empleará la técnica de aprendizaje profundo utilizando el enfoque de transfer learning, con el fin de aprovechar los modelos preentrenados existentes y obtener los mejores resultados posibles.

El alcance del proyecto [10] comprende las siguientes actividades y entregables:

1. Investigación y estudio del problema: Los conceptos fundamentales de la visión por computadora, el reconocimiento de objetos y la robótica social se revisarán exhaustivamente. En esta etapa, será posible comprender completamente el contexto y los problemas del proyecto.

2. Recopilación y preparación de datos: se recopilará un gran conjunto de datos representativos, que incluirá imágenes de varios objetos que pueden encontrarse en el entorno de un robot social. Además, se llevará a cabo un proceso de

preparación y etiquetado de datos para su posterior uso en el entrenamiento del modelo clasificador.

3. Diseño e implementación del modelo clasificador: Se desarrollará un modelo de aprendizaje profundo utilizando la técnica de transfer learning. Se explorarán y evaluarán diferentes arquitecturas y configuraciones de modelos pre-entrenados para encontrar el mejor modelo posible en términos de precisión y rendimiento.

4. Entrenamiento y evaluación del modelo: El entrenamiento del modelo se llevará a cabo utilizando el conjunto de datos previamente preparado. Posteriormente, se realizará una evaluación completa del modelo utilizando las métricas de desempeño apropiadas, como la precisión y la tasa de aciertos. Este paso permitirá evaluar la eficacia del modelo en la detección y clasificación de objetos.

5. Conclusiones y posibles mejoras: Se realizará un análisis crítico de los resultados obtenidos. Además, se propondrán mejoras potenciales y futuras líneas de investigación que podrían abordarse con el fin de ampliar y mejorar el modelo en cuestión.

Finalmente genera la documentación que desempeña un papel fundamental en este proyecto [20], ya que permite garantizar la reproducibilidad y comprensión de los resultados obtenidos. A continuación, se describen los elementos clave de la documentación que se generarán durante el desarrollo del proyecto:

- Informe técnico: Se elaborará un informe técnico detallado que documente todas las etapas del proyecto, desde la investigación inicial hasta la evaluación final del modelo. El informe incluirá una descripción exhaustiva de los conceptos teóricos y las técnicas utilizadas, así como los resultados obtenidos y su análisis correspondiente. Además, se incluirán las referencias bibliográficas pertinentes que respalden las decisiones tomadas y los enfoques adoptados.

- Código fuente: Se proporcionará un repositorio de control de versiones que contendrá todo el código fuente desarrollado en el proyecto. Este repositorio estará estructurado de manera clara y organizada, con comentarios apropiados en el código para facilitar su comprensión. Además, se incluirán instrucciones detalladas sobre cómo configurar el entorno de desarrollo y ejecutar el código para replicar los resultados.

- Descripción del conjunto de datos: Se incluirá en la documentación el conjunto de datos utilizado en el proyecto. Incluirá información sobre la fuente de los

datos, la forma en que se recopilaron y prepararon, así como las etiquetas y categorías correspondientes a los objetos presentes en el conjunto de datos. Esta información permitirá a otros investigadores y desarrolladores comprender y utilizar de manera adecuada el conjunto de datos en futuros trabajos.

2.1.2. Presupuesto

A continuación se presenta detalladamente el presupuesto para personal, hardware y el total estimado.

Coste de personal

En el desarrollo del proyecto se basa en la metodología ágil Scrum y existen 4 tipos de perfiles que son: Desarrollador, Product Owner, Scrum Master y Project Manager.

El desarrollador se encarga de recolectar, organizar y preparar el dataset, aplicar las técnicas de Transfer Learning y Fine Tuning de los modelos y al final se crea diagramas para evaluar el rendimiento de los modelos y evitar el sobreajuste.
El Scrum Master identifica y resuelve obstáculos y problemas durante el desarrollo, garantizando un flujo de trabajo fluido y ayudando al equipo a superar dificultades.
El Product Owner colabora con el desarrollador en la evaluación de los modelos finales, asegurando que cumplan con los requisitos y expectativas establecidos.
El Jefe de Proyecto supervisa el desarrollo general del proyecto, asegurándose de que el equipo cumpla con los plazos, objetivos y requisitos, y también se encarga de la asignación de recursos y la gestión del presupuesto.

En la tabla 2.1 se indican los costes por el personal del proyecto. v18*80

Tabla 2.1: Costes de personal

Perfil	Horas	Euros/Hora	Total
Desarrollador	180	18	3240 €
Product Owner	7	30	210 €
Scrum Master	10	30	300 €
Jefe de Proyecto	10	40	400 €
Total			4150 €

Coste del hardware

Para la realización de este proyecto se necesitan los siguientes dispositivos. Se incluye en el proyecto un importe de 996,66 €en concepto de amortización de los mismos a 4 años.

Esto implica que el costo de los equipos se distribuirá a lo largo de un período de 4 años para reflejar su desgaste con el paso del tiempo. La amortización se realiza generalmente de manera lineal, dividiendo el costo de los equipos entre el número de años de vida útil estimada. [23]

1. Ordenador con AMD Ryzen 5:

   Placa base: MSI B550M PRO-VDH WIFI (MS-7C95).

   Procesador: AMD Ryzen 5 PRO 4650G

   RAM: 16GB DDR4 3200Mhz 2x8GB CL16

   Disco duro: 1TB SSD M.2

   Tarjeta gráfica: Nvidia GeForce RTX 3070

   Monitor: 15.6"

   - Precio (sin IVA): 996,66 €

Como el proyecto dura aproximadamente 4 meses según la planificación del proyecto, como se puede ver en la figura 2.1, la amortización aproximada del hardware es de un **8,33 %** que equivale a **83,05€** sin incluir los impuestos.

Coste total

El coste total del proyecto es de 6453,69 €, incluyendo los costes indirectos (15 %), el beneficio (11 %) y los impuestos (ver Tabla 2.2).

Tabla 2.2: Costes totales

Concepto	Coste (Euros)
Costes de personal	4150,00 €
Amortización de equipos informáticos	83,05 €
Costes indirectos (15 %)	634,95 €
Beneficio (11 %)	465,63 €
Total sin IVA	5333,63 €
IVA (21 %)	1120,06 €
Total	**6453,69 €**

2.2. Plan de trabajo

2.2.1. Identificación de tareas

En esta sección se identifica todas las tareas que debe realizar para llevar al cabo el proyecto, que son las siguientes:

- **Análisis de las principales tecnologías en aprendizaje automático.**
  Esta tarea consiste en investigar y los métodos y tecnologías del campo del aprendizaje automático. Se discutirán conceptos básicos, algoritmos y frameworks como tensorflow y keras, descubriendo las últimas tendencias y los avances en el campo. El objetivo es obtener un panorama completo de las tecnologías relevantes para el proyecto y comprender sus fortalezas y debilidades. [16]

- **Construcción del dataset.**
  En esta tarea se recopilan y preparan un gran conjunto de imágenes para el entrenamiento y la evaluación del modelo. Se seleccionan las imágenes de diferentes objetos bien etiquetados. Además, se realizarán tareas de preprocesamiento de los datos para ajustar el modelo que se va a implementar.

- **Elección de los modelos de clasificación de objetos.**
  En esta tarea se investigarán y evaluarán varios modelos de clasificación de objetos en el campo de la visión por computadora. Se analizan arquitecturas más popular, sobre todo los modelos ya preentrenados para la clasificación de los objetos como Xception, Yolo, etc... en función de factores como la exatitud, precisión, y el rendimiento, se seleccionarán los modelos que se consideren más apropiados para el proyecto.

- **Creación de los modelos mediante Transfer Learning.**
  En esta tarea se crea los modelos utilizando la técnica de transfer learning. se adapta y se utiliza los modelos preentrenados en el ámbito del reconocimiento de objetos en robótica social. Se ajusta los pesos del modelo para acelerar el proceso de entrenamiento y obtener resultados más rápidos y precisos.

- **Elección de las métricas.**
  En esta etapa, se selecciona las métricas adecuadas para evaluar el rendimiento de los modelos clasificadores. Se considera las métricas comunes como exatitud (accuracy), precisión, exatitud en el top 5 y matriz de confusión.

- **Mejorar el modelo mediante Fine Tuning.**
  En esta tarea se centra en mejorar aún más nuestro modelo a través del fine

tuning. Se realiza Fine Tuning de los modelos ya preentrenados. El objetivo es ajustar los pesos del modelo a nuestro conjunto de datos para conseguir un resultado más preciso y robusto para la clasificación de objetos en el contexto de la robótica social.

- **Visualización de los resultados.**
  En esta etapa, se visualiza los resultados para comprender cómo evoluciona la exatitud y presición durante el proceso de entrenamiento y evaluación de nuestros modelos. Para llevar a cabo esta tarea, se utiliza la librería matplotlib, que nos proporciona herramientas gráficas para crear visualizaciones efectivas.

- **Comparación de los modelos.**
  En esta etapa, se compara y analiza los diferentes modelos que se han desarrollado y evaluado. Se considera el valor numérico de las métricas seleccionados, y otros aspectos relevantes para determinar cuál de ellos es más adecuado para clasificar los objetos en el contexto de la robótica social.

- **Conclusión del proyecto.**
  En esta fase final, se realiza una conclusión general del proyecto, destacando los logros alcanzados, las lecciones aprendidas y las posibles áreas de mejora, resumir los resultados principales, discutir las implicaciones y ofrecer recomendaciones para futuros trabajos en el campo del reconocimiento de objetos en robótica social. Esta conclusión cierra nuestro trabajo fin de grado de manera concisa y significativa.

2.2.2. Estimación de tareas y recursos

La estimación de las tareas y recursos se establece en función de los Sprint:

Sprint 1

- **Tareas:**
  1. Establecer requisitos del proyecto
  2. Introducción de visión por computadora
  3. Introducción de redes neuronales
- **Fecha de inicio:** 01/03/2023
- **Fecha de fin:** 14/03/2023

Sprint 2

- **Tareas:**
  1. Introducción a la librería Keras
  2. Análisis de las arquitecturas en Keras
  3. Definir las métricas para evaluar los modelos
  4. Definir las configuraciones que se utiliza
- **Fecha de inicio:** 15/03/2023
- **Fecha de fin:** 28/03/2023

. . . . . .

Sprint 3

- **Tareas:**
  1. Implementación del modelo EfficientNetB0
  2. Fine-tuning del modelo EfficientNetB0
  3. Creación de las diagramas de los resultados
- **Fecha de inicio:** 29/03/2023
- **Fecha de fin:** 11/04/2023

. . . . . .

Sprint 4

- **Tareas:**
  1. Implementación del modelo Xception
  2. Fine-tuning del modelo Xception
  3. Implementación del modelo MobileNetV2
  4. Fine-tuning del modelo MobileNetV2
- **Fecha de inicio:** 12/04/2023
- **Fecha de fin:** 25/04/2023

. . . . . .

Sprint 5

- **Tareas:**
  1. Implementación del modelo Resnet50

2. Fine-tuning del modelo Resnet50
3. Preprocesamiento del conjunto de datos para ajustar el modelo YOLOv8

- **Fecha de inicio:** 26/04/2023
- **Fecha de fin:** 09/05/2023

. . . . . .

Sprint 6

- **Tareas:**

1. Implementación y Fine Tuning del modelo YOLOv8
2. Implementación del modelo InceptionV3
3. Fine-Tuning del modelo InceptionV3

- **Fecha de inicio:** 10/05/2023
- **Fecha de fin:** 23/05/2023

. . . . . .

Sprint 7

- **Tareas:**

1. Implementación del modelo Resnet152V2
2. Fine-Tuning del modelo Resnet152V2
3. Evaluación de los modelos implementados mediante accuracy top-1, accuracy top-5, precisión y la pérdida
4. Exportación de los modelos

- **Fecha de inicio:** 24/05/2023
- **Fecha de fin:** 06/06/2023

. . . . . .

Sprint 8

- **Tareas:**

1. Creación de la matriz de confusión de todos los modelos
2. Análisis de los modelos implementados
3. Conclusión del mejor modelo obtenido
4. Documentación final

- **Fecha de inicio:** 07/06/2023
- **Fecha de fin:** 28/06/2023

2.2.3. Planificación de tareas

Respecto a la planificación, como se puede ver en el apartado anterior, las tareas se realizan por las iteraciones de los Sprints, los detalles se pueden ver en el diagrama de Gantt en la figura 2.1, el diagrama de Gantt se proporciona una vista global simplificada del flujo del trabajo.

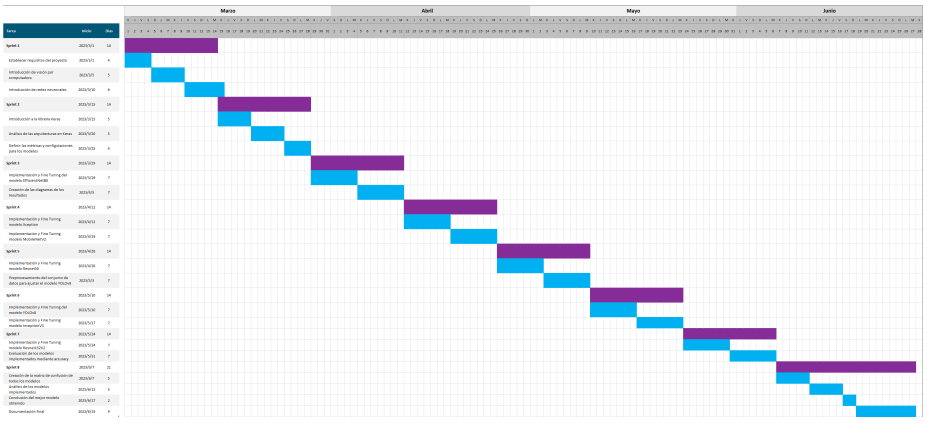

Figura 2.1: Diagrama de Gantt de la planificación de las tareas

2.3. Gestión de recursos

La gestión de los recursos es fundamental para garantizar el desarrollo exitoso del proyecto. A continuación se detallan la especificación de recursos y la asignación de recursos.

2.3.1. Especificación de recursos

- **Recursos Humanos:**
  En este proyecto, el recurso humano principal está representado por mi tutora Lidia, quien desempeña los roles de Scrum Máster y Product Owner, y yo como el desarrollador del sistema, quien es responsable de la implementación. Ambos se colaboran para garantizar una comunicación fluida, una gestión eficiente y una toma de decisiones efectiva en todas las fases del proyecto.

- **Recursos Tecnológicos:**
  Los recursos tecnológicos incluye una computadora con capacidad de procesamiento y memoria suficientes para ejecutar los algoritmos de aprendizaje profundo y manipular grandes conjuntos de datos.
- **Recursos Financieros:**
  Los recursos financieros se refieren a los medios económicos disponibles llevar al cabo el proyecto.
- **Recursos de tiempo:**
  El tiempo es un recurso fundamental en cualquier proyecto, depende del tiempo disponible puede desarrollar el sistema de mayor complejidad o no.

2.3.2. Asignación de recursos

La asignación de los recursos mencionados está descrita en las tablas anteriores: Recursos humanos y recursos de tiempos se presenta en las tablas del sprint2.2.2 Recursos financieros y tecnológicos está definida en la sección 2.1.2

2.4. Gestión de riesgos

2.4.1. Identificación de riesgos

Los riesgos que pueden ocurrir en el desarrollo del proyecto [15] y se pueden clasificar en:

- R1. Riesgos en la planificación
- R2. Riesgos en la implementación de los modelos
- R3. Riesgos en la recolección de datos
- R4. Riesgos de calidad de datos
- R5. Riesgos de tiempos
- R6. Riesgos en hardware
- R7. Riesgos de pérdida de datos
- R8. Riesgos de pérdida de códigos

2.4.2. Análisis de riesgos

El análisis de riesgos supone evaluar y comprender los posibles riesgos asociados a una determinada situación, actividad o proyecto [17]. En este proyecto, se han identificado 8 posibles riesgos relacionados con la aplicación desarrollada, los cuales se presentan en la tabla 2.3.

2.5. Legislación y normativa

- ISO/IEC 29155-4:2016(en) Systems and software engineering — Information technology project performance benchmarking framework — Part 4: Guidance for data collection and maintenance

  El dataset que se utiliza en el proyecto cumple con los requisitos establecidos por la normativa ISO/IEC 29155-4:2016(en) sobre benchmarking del rendimiento de proyectos de tecnología de la información y software. Ha seguido las directrices recomendadas para la recopilación y mantenimiento de datos, lo cual garantiza la calidad y fiabilidad de la información en nuestro proyecto de visión por computadora para el reconocimiento de objetos en robótica social. La normativa nos ha brindado una estructura clara para la gestión de datos, asegurando su adecuada documentación y mantenimiento.

ID	Riesgos	Probabilidad	Impacto	Causas	Gestión del riesgo
R1	Riesgos en la planificación	Media	Alto	Mala planificación del proyecto	Corregir los fallos
R2	Riesgos en la implementación de los modelos	Media	Alto	Falta del conocimiento	ver tutoriales o pedir ayudas
R3	Riesgos en la recolección de datos	Baja	Alto	No existe el dataset en la internet	Construir uno más pequeño manualmente
R4	Riesgos de calidad de datos	Alto	Media	Imagenes de mala calidad o mal clasificada las imágenes	Limpiar el dataset o buscar otro dataset alternativa
R5	Riesgos de tiempos	Alto	Alto	Mal asignación de tiempo	Adaptarse a los nuevos plazos
R6	Riesgos en hardware	Bajo	Alto	Cualquier impacto externos que destruye el ordenador	Conseguir otro equipo para trabajar
R7	Riesgos de pérdida de datos	Medio	Bajo	impactos externos que destruye el disco duro, o la eliminación accidentalmente	vuelve a descargar en la página y reconstruir el dataset
R8	Riesgos de pérdida de códigos	Medio	Medio	eliminación accidentalmente de los códigos	recuperar la última versión en el Git Hub

Tabla 2.3: Tabla de análisis de riesgos

Capítulo 3

Solución

En los siguientes apartados se detallan las etapas de desarrollo de la solución realizada.

3.1. Descripción de la solución

El propósito principal de este Trabajo de Fin de Grado es desarrollar un sistema para el reconocimiento de objetos en robótica social utilizando la técnica de transfer learning y fine tuning de diversos modelos, como MobileNetV2, EfficientnetB0, Yolov8, Xception, InceptionV3, Resnet150V2, entre otros. Con este sistema, un robot podrá capturar una imagen de un objeto y reconocer y clasificar los objetos en su entorno.

Para lograr este desarrollo, se utiliza un conjunto de datos jerárquico compuesto por 8 categorías principales y 180 categorías secundarias.

Este conjunto de datos contiene imágenes relevantes para el campo de la robótica social y se utiliza como conjunto de entrenamiento y evaluación para el sistema de reconocimiento de objetos.

El proceso de desarrollo del sistema se basa en la técnica de transfer learning, que consiste en aprovechar modelos preentrenados en conjuntos de datos más grandes y generalizados. Estos modelos tienen la capacidad de extraer características aprendidas que se pueden utilizar como punto de partida para el reconocimiento de objetos en un conjunto de datos más específico.

En este caso, se seleccionan diversos modelos preentrenados que han demostrado buenos resultados en tareas de reconocimiento de objetos. Estos modelos se adaptan

y ajustan utilizando el conjunto de datos jerárquico mencionado anteriormente, con el objetivo de que puedan reconocer y clasificar objetos específicos en el contexto de la robótica social.

A continuación, se detallará el proceso de desarrollo, que incluye el análisis del sistema, el diseño de la adaptación y ajuste de los modelos, y la implementación de los mismos.

3.2. El proceso de desarrollo

El proceso de desarrollo del sistema de reconocimiento de objetos en robótica social consta de varias etapas, desde el análisis inicial hasta la implementación y las pruebas del sistema.

El objetivo del proceso de desarrollo es crear un sistema que es capaz de clasificar las imágenes de objetos domésticos, el dicho sistema es capaz de integrar en los robots domésticos como aspirador, limpiador o incluso robots de asistencia doméstico [45] para mejorar la calidad de vida de las personas.

Para conseguir mejor resultados en el sistema, se utiliza el mismo dataset que se ha utilizado en la competencia RoboCup@home en el año 2019, RoboCup es un proyecto internacional que fue fundado en 1997 con el objetivo de promover la investigación y educación en inteligencia artificial a través de competiciones que involucran robots autónomos, por lo tanto contiene un gran conjunto de imagenes muy especializado en el campo robótica social.

Al construir un sistema que aporta valor, es necesario analizar el estado de la cuestión actual, las mejores y nuevas tecnologías, las arquitecturas y las librerías útiles para conseguir un resultado optimizado, en este caso los modelos basados en redes neuronales, se implementa con el lenguaje de programación Python Utilizando las librerías como Keras, tensorflow, Ultralytics(YOLO), scikit-learn, ...

El desarrollo se seguirá los principios de la metodología ágil Scrum para permitir una respuesta rápida a los cambios y un desarrollo iterativo e incremental.

El enfoque se centró en cumplir con los siguientes criterios clave para el software:

- Ser exportable para futuros integraciones en robots.
- Ser suficientemente preciso y consistente.
- Estar suficientemente detallado.

- Utilizar las arquitecturas modernas.
- Ser lo más simple posible.

3.2.1. Análisis

Para poder identificar y documentar las necesidades que deben ser satisfechas por el sistema, se ha utilizado la especificación de requisitos.

Esta especificación comprende la definición de los requisitos funcionales, los cuales detallan las funciones que el sistema debe llevar a cabo, las respuestas que debe proporcionar ante diferentes entradas y las salidas que debe generar. Asimismo, incluye los requisitos no funcionales, que engloban las restricciones del producto y las propiedades o características que afectan al nivel de satisfacción del sistema. Al emplear esta metodología, se garantiza una adecuada comprensión y cumplimiento de los requerimientos del sistema.

Definición de requisitos

- Requisito funcionales
  - **Requisito funcional 1:** Resolver un problema típico de las competiciones de robótica de servicios
  - **Requisito funcional 2:** Clasificar imagen de un objeto
  - **Requisito funcional 3:** Exportar el modelo para integrar en los robots domésticos
  - **Requisito funcional 4:** Evaluar el rendimiento de distintos modelos
  - **Requisito funcional 5:** Generar la matriz de confusión para saber las clases fáciles y difíciles de clasificar
- Requisito no funcionales
  - **RNF1:** Rendimiento en tiempo real, el sistema debe ser capaz de procesar y clasificar las imágenes en tiempo real, con una latencia mínima.
  - **RNF2:** Escalabilidad, el sistema debe ser capaz de manejar un creciente volumen de datos y adaptarse a futuras ampliaciones de la base de datos de objetos.
  - **RNF3:** El sistema debe ser fácil de mantener, los modelos entrenados utilizan los controles de versiones.

- **RNF4:** El sistema debe ser fácil de configurar.
- **RNF5:** El sistema debe ser fácil de exportar.

Especificación de requisitos

Después de enumerar los requisitos necesarios para el desarrollo del sistema, a continuación se describen de manera individual y en lenguaje claro cada uno de ellos, con el fin de facilitar la comprensión de las funcionalidades que se deben implementar en el proyecto.

RF1: Resolver un problema típico de las competiciones de robótica de servicios.

- **Versión:** 1.0
- **Descripción:** Resolver el problema típico de las competiciones de robótica de servicios, permitiendo que la aplicación desarrollada se pueda usar en estos contextos robotics24.net/es/servicios/competition.
- **Actores:** El sistema.
- **Precondición:** El modelo debe estar bien implementado y el dataset bien estructurado.
- **Flujo básico:**
  1. Se inicializa los modelos.
  2. Se carga el conjunto de datos en la memoria.
  3. El sistema se evalúa especificando pasando el dataset.
  4. Demuestra el resultado obtenido.
- **Flujo alternativo:**
- **Postcondición:** Imprime las salidas esperadas.

. . . . . .

RF2: Clasificar imagen de un objeto.

- **Versión:** 1.0
- **Descripción:** Clasificar la clase que pertenece el objeto pasando la imagen al modelo.
- **Actores:** El sistema.

- **Precondición:** El modelo debe estar cargado correctamente.
- **Flujo básico:**
  1. Se inicializa el modelos.
  2. pasar la imagen al sistema.
  3. predecir la clase que pertenece la imagen.
- **Flujo alternativo:**
- **Postcondición:** Imprime por la pantalla la clase que pertenece la imagen.

. . . . . .

RF3: exportar el modelo para integrar en los robots domésticos.

- **Versión:** 1.0
- **Descripción:** Exportar el modelo para integrar en los robots domésticos para que el robot obtiene la capacidad de clasificar los objetos, con el fin de mejorar su interacción con el entorno y aumentar su rendimiento.
- **Actores:** El robot, el sistema.
- **Precondición:** Un robot con capacidad de capturar imágenes y procesar datos, y el modelo preparado/exportado.
- **Flujo básico:**
  1. Conectar el robot con un ordenador.
  2. Comprobar la compatibilidad.
  3. Instalar el sistema en el robot.
  4. Fin de la integración.
- **Flujo alternativo:**
- **Postcondición:** Robot con la capacidad de clasificar las imágenes domésticos.

. . . . . .

RF4: Evaluar el rendimiento de distintos modelos.

- **Versión:** 1.0
- **Descripción:** Comparar los resultados obtenidos por distintos modelos, se examinará la accuracy top 1, accuracy top 5, precisión, valor de pérdida, ...

- **Actores:** El usuario y el sistema.
- **Precondición:** Conjunto de datos para evaluar.
- **Flujo básico:**
  1. Se inicializa los modelos.
  2. Se carga el conjunto de datos en la memoria.
  3. El sistema se evalúa especificando pasando el dataset.
  4. Demuestra los resultados obtenidos de los modelos.
  5. Comparar los resultados.
- **Flujo alternativo:**
- **Postcondición:** Saber qué modelo es mejor para el conjunto de datos que se necesita.

.....

RF5: Generar la matriz de confusión para saber las clases fáciles y difíciles de clasificar.

- **Versión:** 1.0
- **Descripción:** Generar la matriz de confusión que permite el usuario analizar el rendimiento del sistema para conocer cuáles son las clases difíciles/fáciles de clasificar y con qué clases tienen mayor características en común.
- **Actores:** El usuario y el sistema.
- **Precondición:** Tener el conjunto de datos etiquetados
- **Flujo básico:**
  1. Se inicializa los modelos.
  2. Se carga el conjunto de datos en la memoria.
  3. El sistema se clasifica las clases que pertenece cada imagen.
  4. Genera la matriz de confusión.
- **Flujo alternativo:**
- **Postcondición:** Saber qué modelo es mejor para el conjunto de datos que se necesita.

. . . . . .

3.2.2. Dataset

En este trabajo fin de grado, se utiliza el mismo dataset que se ha utilizado en una competencia del proyecto RoboCup@home 2019.

El proyecto RoboCup@Home 2019 utiliza un dataset llamado "RoboCup@Home-Objects" que desempeña un papel fundamental en el desarrollo de robots domésticos capaces de realizar tareas en un entorno hogareño. A continuación, se proporciona informaciones cuantitativas del dataset, destacando su tamaño y características.

Tamaño del dataset

El dataset RoboCup@Home-Objects es un conjunto de datos considerablemente grande que contiene de 196K imágenes. Las resoluciones de las imágenes no son únicos, pero la mayoría está entre 100 píxeles hasta 300 píxeles. Esta amplia cantidad de imágenes proporciona una diversidad significativa, permitiendo a los investigadores y desarrolladores entrenar y evaluar algoritmos de reconocimiento de objetos en una variedad de escenarios.

Categorías y estructuras

El dataset abarca una amplia gama de categorías de objetos comunes que se encuentran en un hogar típico, sus categorías están organizada jerárquicamente, que contiene 8 clases padres y 180 clases hijos, como se puede ver en la tabla 3.1.

Tabla 3.1: Categorías del dataset

Parent	**Number of children**
Artículos de la limpieza	37
Contenedores	17
Cubiertos	15
Bebidas	17
Alimentos	22
Frutas	23
Snacks	26
Vajilla	23
Total	**180**

Esta estructura es válida para dividir el conjunto de datos en un conjunto entrenamiento y un conjunto validación utilizando la clase ImageDataGenerator de la librería Keras para nuestro proyecto en la clasificación de las clases hijos con

los modelos utilizables en Keras, pero para clasificar las clases padres, y para el modelo Yolov8 se necesita otra estructura diferente, se detalla en el apartado de implementación de los modelos.

El dataset RoboCup@Home-Objects utiliza una estructura jerárquica en la que las imágenes se agrupan en clases padres y clases hijas. Esto permite una clasificación y acceso eficiente a las imágenes según sus categorías. La organización en niveles facilita la búsqueda y el entrenamiento de algoritmos de reconocimiento de objetos. Además, esta estructura es escalable y adaptable a diferentes escenarios.

```
Directorio principal del dataset
|
|-- Clase Padre 1
| |-- Clase Hijo 1
| | |-- imagen1.jpg
| | |-- imagen2.jpg
| | |-- imagen3.jpg
| | |-- ...
| |
| |-- Clase Hijo 2
| | |-- imagen4.jpg
| | |-- imagen5.jpg
| | |-- ...
| |
| |-- ...
|
|-- Clase Padre 2
| |-- Clase Hijo 1
| | |-- imagen6.jpg
| | |-- imagen7.jpg
| | |-- ...
| |
| |-- Clase Hijo 2
| | |-- imagen8.jpg
| | |-- imagen9.jpg
| | |-- ...
| |
| |-- ...
```

|
|...

Diversidad del dataset

El dataset RoboCup@Home-Objects busca representar una variedad de escenarios y condiciones de iluminación que se encuentran en un hogar real. Las imágenes se capturaron en diferentes entornos, lo que proporciona variaciones en la apariencia de los objetos debido a la iluminación, el ángulo de visión y otros factores, como se puede ver en las figuras 3.1, 3.2 y 3.3. Esta diversidad de escenarios ayuda a los robots a generalizar su capacidad de reconocimiento de objetos en diferentes condiciones del mundo real.

Figura 3.1: Diversidad de las imágenes 1.

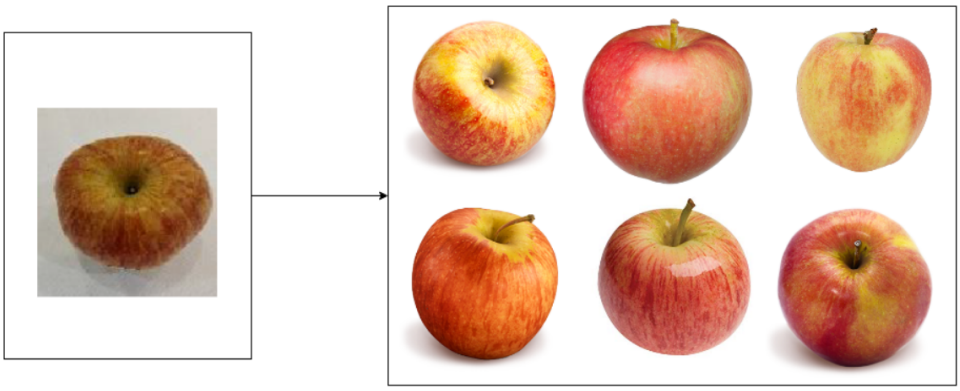

Figura 3.2: Diversidad de las imágenes 2.

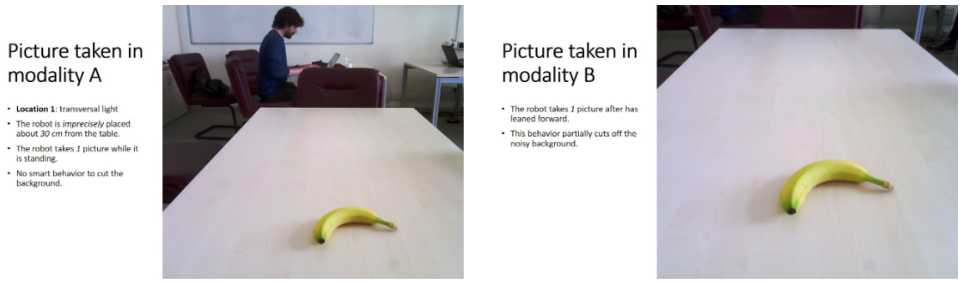

Figura 3.3: Diversidad de las imágenes 3.

Análisis del dataset

En esta sección, visualizamos algunos imágenes que contiene el dataset.

Como las imágenes que se utiliza en RoboCup@Home-Objects son las imágenes capturados en un ámbito real domésticos, se puede ver que algunas imágenes no distinguen bien las clases, incluso desde nuestra perspectiva humana, dependiendo de muchas razones, como los siguientes:

1. Imágenes muy raros o engañosos, como se muestra en la figura 3.4.

Figura 3.4: Imágenes raros Fruta/Melones.

2. Los objetos que tienen alta similitud como por ejemplo snacks, frutas y alimentos, la clase snack y fruta puede ser una subclase de la clase alimento, o un contenedor y una bebida, como aparece en la figura 3.5, por este motivo la predicción se ha complicado mucho, como los resultados que se muestra en la figura 3.6 del modelo autoML, que realmente la imagen puede pertenecer a ambas categorías.

Figura 3.5: Imágenes Similares Fruta/Melons y Snack/FruitCup.

Figura 3.6: Imágenes Similares, predicción autoML.

3. En una imagen se representa varios elementos, como se muestra en la figura 3.7

Figura 3.7: Imagen que contiene varios objetos. plato, tarta y fresa.

Por la ambigüedad que tiene las imágenes, ya que una imagen puede pertenecer a varias categorías a la vez, se da mucha importancia a la métrica accuracy en el top 5 en la fase de evaluación de los modelos.

Las implementaciones existentes y los resultados

En el siguiente artículo/paper [28] se puede encontrar algunas implementaciones existentes del dataset de la competencia roboCup@home-Objects, a continuación se detalla las implementaciones y los resultados obtenidos tanto para clasificar las clases padres como para las clases hijos.

Se procedió a dividir el conjunto de entrenamiento de RoboCup@Home-Objects en un 80 % de imágenes para el entrenamiento y un 20 % para la validación. Se llevó a cabo un Fine Tuning de un AlexNet y un GoogleNet preentrenados en ILSVRC12 de Imagenet utilizando el marco Caffe en el Sistema de Entrenamiento de GPU de Aprendizaje Profundo de NVIDIA (DIGITS).

Se congelaron todas las capas de las redes y se aprendió la última capa completamente conectada para Alexnet y la última capa de agrupación para Googlenet. Se estableció una tasa de aprendizaje inicial de 0.001, la cual se redujo un 10 % cada 7.5 épocas. Ambas redes se entrenaron durante 30 épocas utilizando el optimizador de descenso de gradiente estocástico (SGD).

Como resultado del entrenamiento, se obtuvieron 4 modelos: entrenados en todas las categorías hijas, Alexnet@home180 y Googlenet@home180; entrenados solamente en las categorías principales, Alexnet@home8 y Googlenet@home8. Posteriormente, se mapea de categorías en los modelos entrenados con las 180 categorías hijas para asignar los resultados a las 8 categorías principales. La figura 3.8 muestra el porcentaje de accuracy obtenido por los 4 modelos en el conjunto de validación, y se puede observar que al tener menos categorías para aprender facilitó la tarea para los modelos.

Validation accuracy

Model	Accuracy
Alexnet@home8	77.89%
Alexnet@home180	47.85%
Googlenet@home8	81.91%
Googlenet@home180	53.55%

Figura 3.8: Porcentaje de accuracy de validación de los 4 modelos.
Fuente: `https://www.researchgate.net/publication/337689438_RoboCupHome-Objects_Benchmarking_Object_Recognition_for_Home_Robots`

3.2.3. Arquitecturas

Para la clasificación de los objetos se implementa las arquitecturas basadas en las redes neuronales, una red neuronal consta de tres componentes fundamentales: la capa de entrada, las capas ocultas y la capa de salida [22], como se muestra en la figura 1.2

- **Capa de entrada:** Esta es la primera capa de la red neuronal y se encarga de recibir los datos de entrada, para clasificar las imágenes, admite por entrada un tensor de (ancho, largo, canal).
- **Capas ocultas:** Estas capas se encuentran entre la capa de entrada y la capa de salida. Se les llama "ocultas" porque sus salidas no son directamente visibles en el resultado final. Las capas ocultas están formadas por neuronas que aplican transformaciones a las entradas recibidas. Estas transformaciones están determinadas por los pesos y sesgos asociados a cada conexión entre las neuronas. Las capas ocultas pueden tener diferentes arquitecturas y funciones de activación, y su número y tamaño varían la estructura de la red neuronal.
- **Capa de salida:** Esta es la última capa de la red neuronal y proporciona los resultados o predicciones deseadas. puede utilizar una capa de salida con múltiples neuronas, donde cada neurona representa una clase y se utiliza la activación softmax para obtener las probabilidades de pertenencia a cada clase.

Las arquitecturas o modelos que se puede aplicar en las capas ocultas están compuestas por una serie de bloques, un bloque es una estructura que agrupa un conjunto de capas con una función específica dentro del modelo de red neuronal, para mejorar la comprensión de la funcionalidad de la arquitectura se muestra ahora las principales capas utilizadas en la construcción de modelos de redes neuronales.

1. **Capa Convolucional:** La capa convolucional [9] es una de las capas más importantes en las arquitecturas de redes neuronales convolucionales. Esta capa se encarga de realizar operaciones de convolución en los datos de entrada. La convolución es un proceso matemático que permite extraer características relevantes de las imágenes o datos de entrada, como bordes, texturas o formas.
2. **Capa Dense:** La capa Dense, también conocida como capa totalmente conectada, es una capa en la que cada neurona está conectada a todas las neuronas de la capa anterior. Esta capa es comúnmente utilizada en las últimas etapas de una arquitectura de red neuronal para realizar la clasificación final o la regresión.
3. **Capa AvgPooling:** La capa AvgPooling [46] es una capa de submuestreo utilizada para reducir la dimensionalidad de los datos en una red neuronal convolucional. Esta capa calcula el promedio de los valores en cada región de una imagen y reduce su tamaño. El promedio obtenido representa una característica general de la región analizada.

4. **Capa MaxPooling:** La capa MaxPooling [39] es similar a la capa AvgPooling, pero en lugar de calcular el promedio de los valores en cada región, esta capa selecciona el valor máximo. Al igual que la capa AvgPooling, el MaxPooling se utiliza para reducir la dimensionalidad y extraer características importantes de los datos.

5. **Capa Concatenación:** La capa de concatenación se utiliza para fusionar dos o más conjuntos de datos en una única representación. Esta capa es especialmente útil cuando se trabaja con modelos que tienen múltiples fuentes de entrada o cuando se desea combinar características extraídas de diferentes bloques de la arquitectura

6. **Capa Normalización:** La capa de normalización [11], como Batch Normalization, se utiliza para normalizar las activaciones de una red neuronal, lo que ayuda a mejorar la estabilidad y el rendimiento del modelo. Esta capa ajusta la media y la desviación estándar de las activaciones para mantenerlas dentro de un rango óptimo.

7. **Capa Expansión:** La capa de expansión, también conocida como capa de upsample, se utiliza para aumentar la resolución de los datos en una red neuronal. Esta capa aplica una operación inversa a la submuestreo, aumentando la dimensionalidad de los datos y permitiendo una representación más detallada.

8. **Capa Adición:** La capa de adición, también llamada capa de suma, combina dos o más conjuntos de datos sumando sus valores correspondientes. Esta capa es útil cuando se desean fusionar características de diferentes bloques en una arquitectura y conservar la información valiosa de ambos.

Además, en cada capa de un modelo neuronal, se encuentra una función de activación. Estas funciones permiten que el modelo aprenda y represente relaciones y patrones complejos en los datos. Sin las funciones de activación, las redes neuronales se asemejarían a modelos lineales, lo cual limitaría su capacidad para aproximar funciones no lineales. [38]

Existen varios tipos de funciones de activación utilizadas en las redes neuronales, cada una con características propias. Algunas de las funciones de activación más comunes que se usan en nuestros modelos son las siguientes:

1. **Función Sigmoide (Logística):** Esta función tiene forma de "S" y mapea los valores de entrada a un rango entre 0 y 1. Es útil en problemas de clasificación binaria.

2. **Función ReLU (Rectified Linear Unit):** Esta función devuelve 0 para valores de entrada negativos y el mismo valor de entrada para valores positivos. Se utiliza ampliamente en redes neuronales debido a su eficiencia computacional y su capacidad para mitigar el problema de la desaparición del gradiente.
3. **Función Tangente Hiperbólica (Tanh):** Es similar a la función sigmoide, pero mapea los valores de entrada a un rango entre -1 y 1. También se utiliza principalmente en problemas de clasificación.
4. **Función Softmax:** Esta función se utiliza comúnmente en la capa de salida de una red neuronal para problemas de clasificación multiclase. Convierte las salidas en probabilidades que suman 1, lo que permite seleccionar la clase más probable.

Resnet50V2

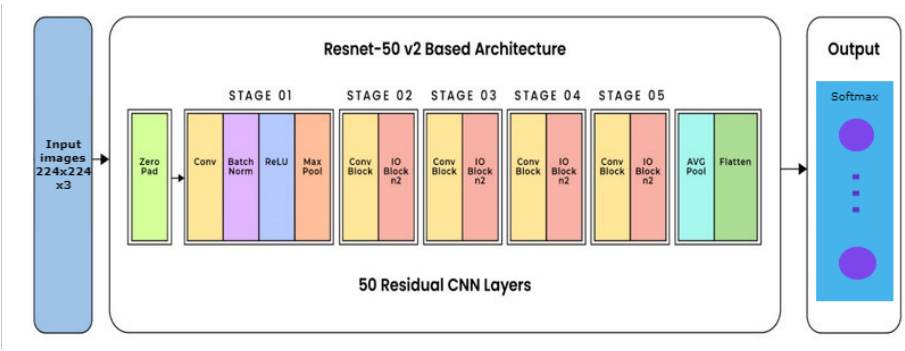

Figura 3.9: Arquitectura del modelo Resnet50V2.
Fuente: [35]

La arquitectura básica del modelo Resnet50V2 está compuesta por 50 capas de CNN(Convolutional Neural Network) residuales en 5 bloques como se puede ver en la figura 3.9.

Resnet152V2

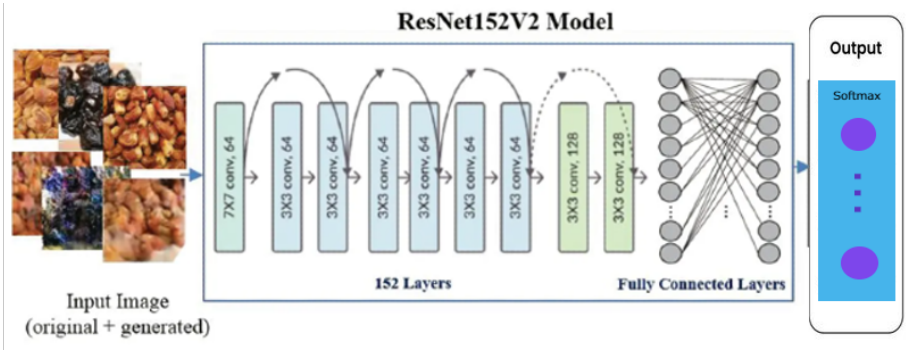

Figura 3.10: Arquitectura del modelo Resnet152V2.
Fuente: [18]

La arquitectura del modelo Resnet152V2 está basada en la arquitectura básica de Resnet50V2, pero más compleja, tiene 152 capas convolucionales y termina con varias capas densas totalmente conectadas como se muestra en la figura 3.10.

MobileNetV2

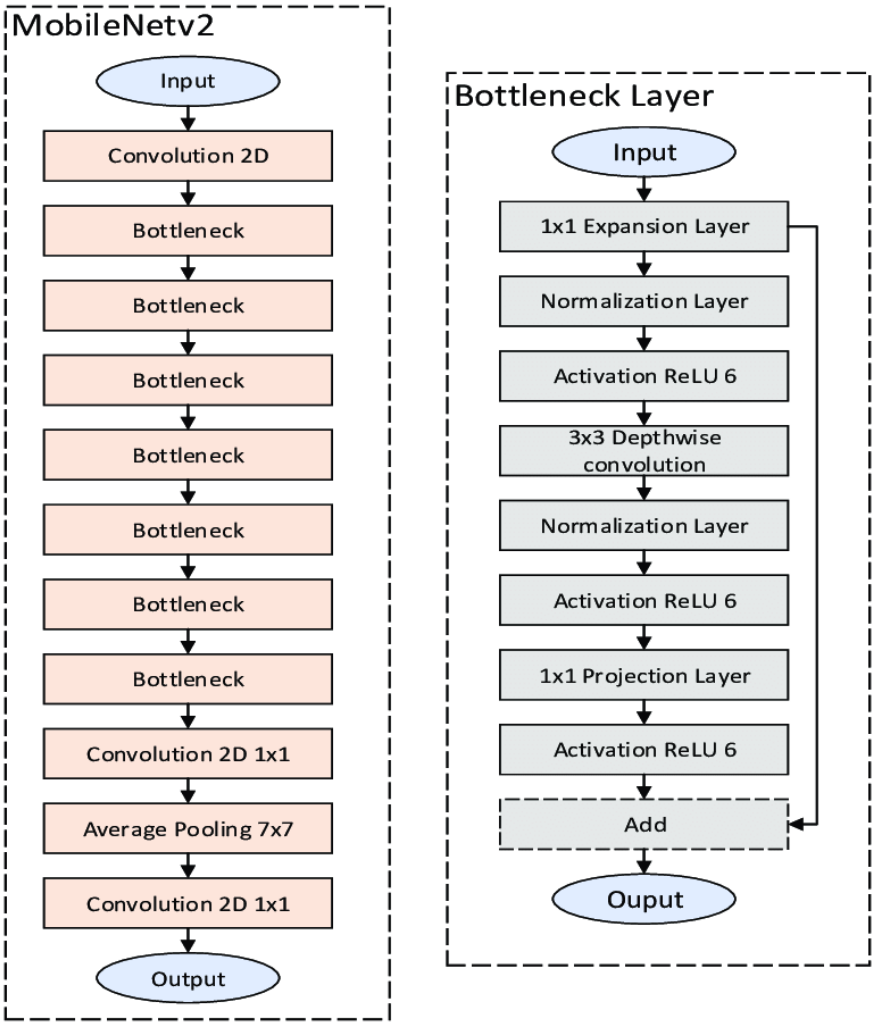

Figura 3.11: Arquitectura del modelo mobileNetV2.
Fuente: [41]

MobileNetV2 se compone de capas de convolución 2D, capas de "bottleneck" (cuello de botella) y capas de pooling. La capa de "bottleneck" a su vez incluye subcapas de expansión, normalización, activación, suma y convolución de profundidad 3x3 como se puede ver reflejado en la figura 3.11. La operación de expansión tiene como objetivo ampliar el espacio de datos al aumentar el número de canales de los datos de entrada mediante un factor definido por los hiperparámetros del modelo. Por otro lado, las operaciones de proyección disminuyen la cantidad de canales de los datos de entrada, reduciendo así el espacio de datos. Las capas de normalización, pooling, convolución y activación emplean las operaciones correspondientes necesarias para el entrenamiento y la inferencia de la red neuronal. [41]

EfficientNetB0

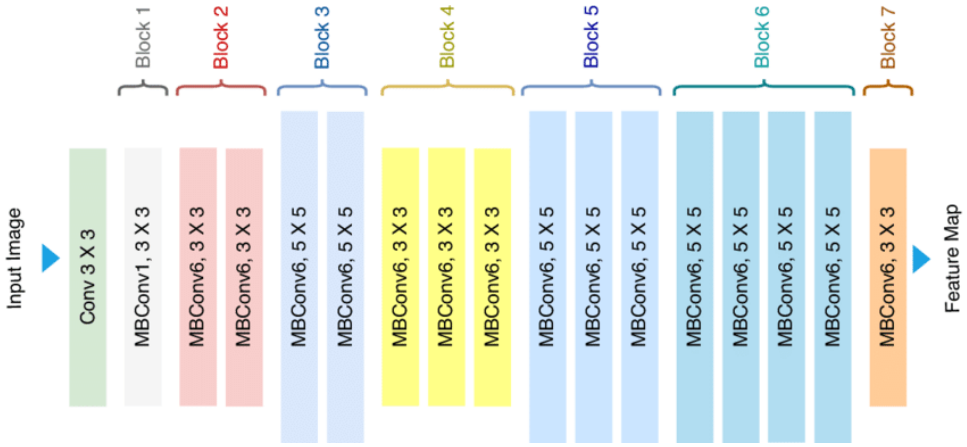

Figura 3.12: Arquitectura del modelo EfficientNetB0.
Fuente: [8]

EfficientNetB0 tiene 7 bloques como se puede ver en la figura 3.12, en cada bloque contiene bloques MBConv, que son módulos de bloques convolucionales móviles. Cada bloque MBConv se compone de varias capas, incluyendo convoluciones, normalización, funciones de activación y conexiones residuales. Estas conexiones residuales permiten el flujo directo de información y ayudan a evitar problemas de desvanecimiento del gradiente durante el entrenamiento.

InceptionV3

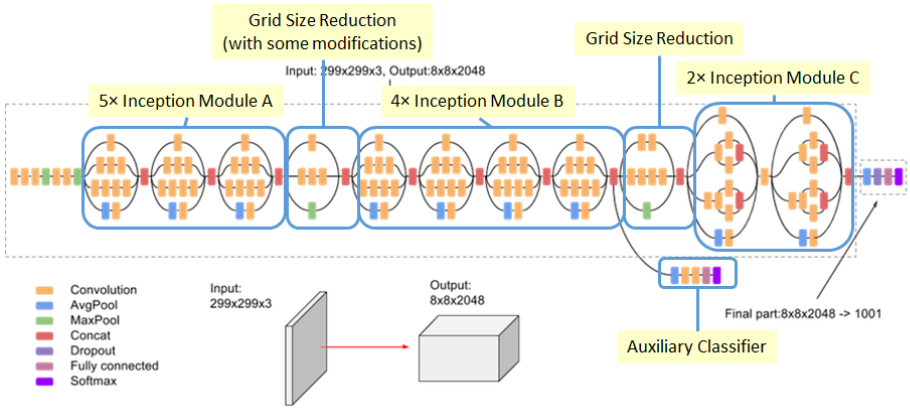

Figura 3.13: Arquitectura del modelo InceptionV3.
Fuente: [14]

InceptionV3 tiene 42 capas como se representa en la figura 3.13, las cuales incluyen convoluciones, MaxPooling, AveragePooling, concatenaciones, capas totalmente

conectadas y dropout. También cuenta con capas de softmax que se utilizan para calcular el valor de pérdida.

Xception

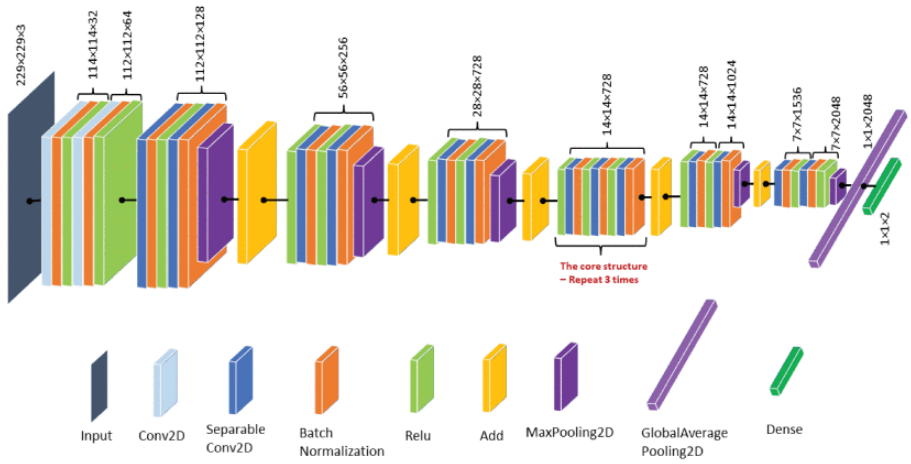

Figura 3.14: Arquitectura del modelo Xception.
Fuente: [29]

Xception es una red neuronal convolucional con 71 capas de profundidad en 7 bloques, empieza por el bloque que contiene la dimensión de las entrada y salidas más grandes y disminuye progresivamente, un bloque está formado por un conjunto de capas que son los siguientes: capas convolucionales, capas de normalización, capas de adición, capas MaxPolling2D, capas dense,... como se muestra en la figura 3.14.

YoloV8

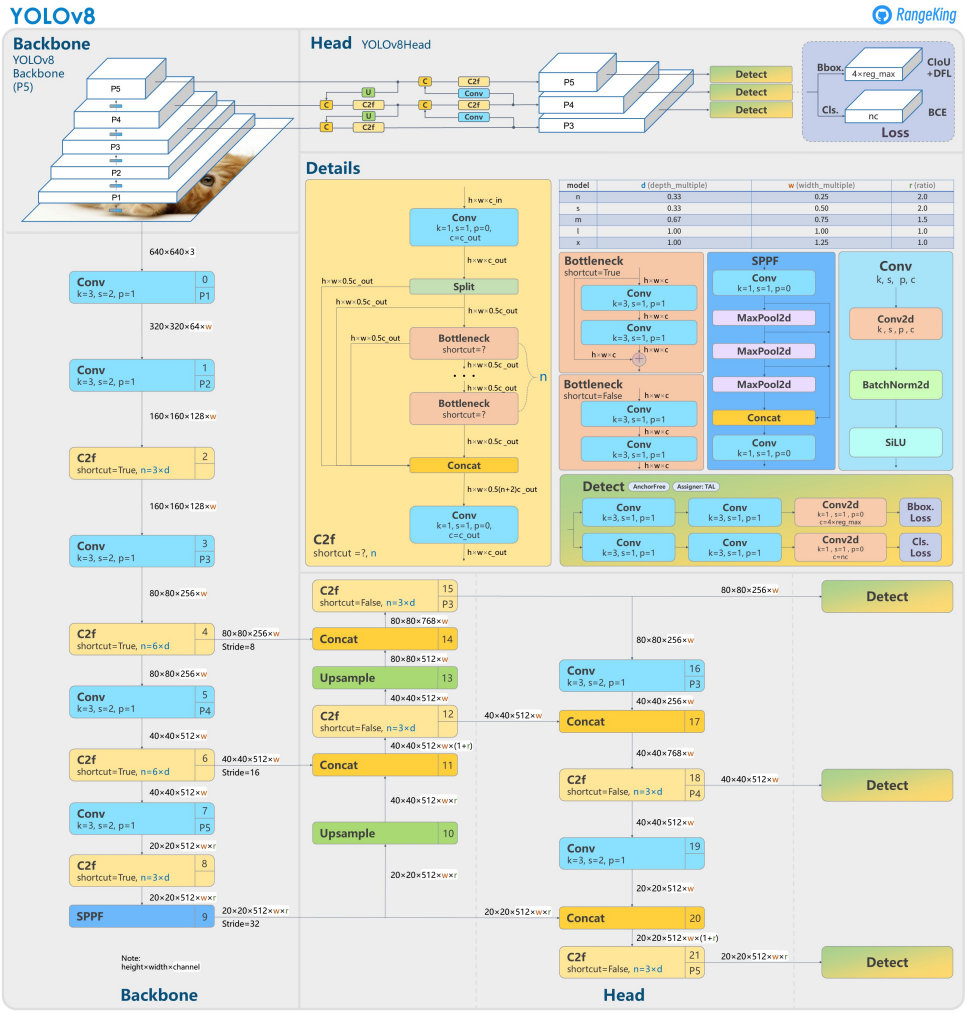

Figura 3.15: Arquitectura del modelo YoloV8.
Fuente: [43]

YOLOv8 ha sido desarrollado con el objetivo de ser rápido, preciso y fácil de utilizar. Esto lo convierte en una opción excepcional para una amplia variedad de tareas, como detección y seguimiento de objetos, segmentación de instancias, estimación de poses y clasificación de imágenes. [43]

La arquitectura de YoloV8 se puede ver en la figura 3.15, se parece a un pirámide de 5 bloques, la entrada se percibe por el bloque más abajo, con la dimensión de las entrada y salidas más grandes y se disminuye la dimensión en cada bloque.

Cada bloque del yolo, contiene múltiples capas de distintos tipos, capas convolucionales(Conv), capas de cuello de botella(Bottneck), las capas de MaxPooling y las capas de Concatenación (Concat).

3.2.4. Herramientas utilizadas

En este sección se va a detallar el lenguaje, las herramientas y las librerías que se utilizan en el proyecto.

Lenguaje

1. **Python:** [7] Python es uno de los lenguajes de programación más populares y versátiles en el campo del aprendizaje automático. Su sintaxis sencilla y legible, junto con una amplia colección de librerías especializadas, lo convierten en una opción ideal para implementar modelos de redes neuronales. Python proporciona una gran flexibilidad y facilidad de uso, lo que facilita el desarrollo, la depuración y la implementación de algoritmos de aprendizaje automático.

Herramientas

1. **Visual Studio Code:** [6] Visual Studio Code es un entorno de desarrollo integrado (IDE) poderoso y fácil, ampliamente utilizado y altamente configurable. Ofrece una amplia gama de extensiones y complementos que hacen que el desarrollo sea más eficiente. Con características como resaltado de sintaxis, depuración interactiva, autocompletado y control de versiones integrado.

2. **Google Colab:** [12] Google Colab es una plataforma en la nube que permite escribir y ejecutar código de Python en un entorno de Jupyter Notebook. Es especialmente útil cuando se trabaja con grandes conjuntos de datos o se requiere una gran cantidad de capacidad de procesamiento. Google Colab ofrece acceso gratuito a GPU y TPU, lo que acelera significativamente el tiempo de entrenamiento de los modelos de redes neuronales. Además, proporciona una integración perfecta con otras herramientas y servicios de Google como Google Drive, lo que facilita el desarrollo y la colaboración en proyectos de aprendizaje automático.

3. **Anaconda:** [1] Anaconda es una plataforma de distribución de Python y R que simplifica la gestión de paquetes y entornos. Proporciona un gestor de paquetes llamado conda, que permite instalar y actualizar fácilmente las librerías requeridas para implementar modelos de redes neuronales. Además, Anaconda ofrece la capacidad de crear entornos virtuales independientes, lo que facilita la gestión de diferentes configuraciones de librerías y versiones para diferentes modelos.

Librerías

1. **TensorFlow:** [19] TensorFlow es una de las librerías más populares y ampliamente utilizadas para la implementación de modelos de redes neuronales. Desarrollada por Google, TensorFlow proporciona una amplia gama de herramientas y recursos para construir y entrenar redes neuronales de manera eficiente. Su arquitectura flexible y escalable permite la implementación de modelos de redes neuronales complejos, incluyendo redes neuronales convolucionales, recurrentes y generativas. TensorFlow también ofrece un entorno de ejecución eficiente que aprovecha al máximo el poder de las unidades de procesamiento gráfico (GPU) y las unidades de procesamiento tensorial (TPU).

2. **Keras:** [21] Keras es una librería de alto nivel escrita en Python que se ejecuta sobre TensorFlow. Proporciona una forma sencilla y elegante para la creación y entrenamiento de modelos de redes neuronales. Keras permite desarrollar modelos de manera rápida y sencilla, ya que abstrae gran parte de la complejidad del desarrollo de redes neuronales. Además, Keras ofrece una amplia variedad de capas, funciones de activación y algoritmos de optimización, lo que facilita la personalización y experimentación con diferentes configuraciones de modelos.

3. **Matplotlib:** [2] Matplotlib es una librería de visualización de datos en 2D que se utiliza ampliamente en el campo del aprendizaje automático. Permite crear gráficos, diagramas y visualizaciones interactivas para analizar y representar los resultados de los modelos de redes neuronales. se permite generar diagramas de precisión, de accuracy, de pérdida,... lo que ayuda a comprender mejor el rendimiento y el comportamiento de los modelos.

4. **Numpy:** [3] Numpy es una librería fundamental para el procesamiento numérico en Python. Proporciona una estructura de datos multidimensional llamada "array" que permite realizar operaciones matemáticas eficientes en grandes conjuntos de datos. Numpy es ampliamente utilizado en la implementación de modelos de redes neuronales para realizar cálculos numéricos, operaciones de álgebra lineal y manipulación de matrices.

5. **Seaborn:** [5] seaborn es una librería de visualización de datos en Python que se basa en matplotlib. Ofrece una interfaz de alto nivel para crear visualizaciones estadísticas atractivas e informativas.

6. **Scikit-learn:** [4] Scikit-learn es una biblioteca de aprendizaje automático en Python que ofrece una variedad de algoritmos y herramientas. Entre sus fun-

ciones, Scikit-learn incluye la capacidad de generar y trabajar con matrices de confusión.

3.2.5. Métricas

En esta sección, se detalla cuáles son las métricas, la matriz de confusión y la función de pérdida que se utilizan para evaluar el rendimiento de los modelos. [32]

1. **Accuracy Top 1:** El accuracy es la métrica más utilizada para evaluar el rendimiento del modelo, esta métrica que mide la proporción de predicciones correctas en su clase objetivo. Se calcula dividiendo el número de predicciones correctas entre el número total de muestras.

2. **Accuracy Top 5:** El accuracy top 5 es similar al accuracy Top 1, pero en lugar de evaluar solo la mejor predicción, se consideran las cinco mejores predicciones. Esto significa que una predicción se considera correcta si la etiqueta correcta se encuentra entre las cinco primeras predicciones más probables. El accuracy top 5 es útil en nuestro proyecto por la ambigüedad que tiene en el dataset 3.2.2.

3. **Precisión:** La precisión es una métrica que evalúa la proporción de predicciones positivas correctas en comparación con el número total de predicciones positivas. Una alta precisión indica que la red neuronal tiene una baja tasa de errores al clasificar correctamente las muestras positivas.

Matriz de confusión

Una matriz de confusión es una tabla que muestra la calidad de las predicciones realizadas por un modelo de clasificación. Es especialmente útil cuando se trabaja con un problema de clasificación multiclase. [27]

La matriz de confusión tiene una estructura cuadrada, donde las filas representan las clases reales y las columnas representan las clases predichas por el modelo. Cada celda de la matriz muestra la cantidad de instancias que fueron clasificadas correctamente o incorrectamente para cada combinación de clases.

La estructura de una matriz de confusión se divide en cuatro elementos principales:

1. **Verdaderos positivos (TP):** Son los casos en los que el modelo clasifica correctamente una instancia como positiva.

2. **Verdaderos negativos (TN):** Son los casos en los que el modelo clasifica correctamente una instancia como negativa.
3. **Falsos positivos (FP):** Son los casos en los que el modelo clasifica incorrectamente una instancia como positiva, cuando en realidad es negativa.
4. **Falsos negativos (FN):** Son los casos en los que el modelo clasifica incorrectamente una instancia como negativa, cuando en realidad es positiva.

Podemos ver una muestra en la figura 3.16

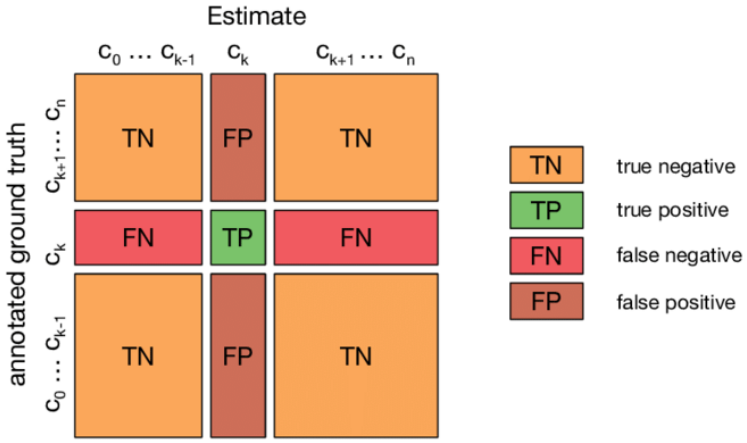

Figura 3.16: Estructura de la matriz de confusión

Función de pérdida

La función de pérdida utilizada en la red neuronal es un componente crítico para guiar su entrenamiento. En este proyecto se utiliza la función de pérdida **categorical crossentropy (entropía cruzada categórica)** , es la función comúnmente utilizada en problemas de clasificación multiclase.

La entropía cruzada categórica mide la discrepancia entre la distribución de probabilidad predicha por la red neuronal y la distribución de probabilidad real de las etiquetas de entrenamiento. El objetivo es minimizar esta discrepancia para mejorar la precisión de la red. [47]

3.2.6. Configuraciones

En esta sección se explica las configuraciones utilizadas para encontrar el mejor resultado de los modelos.

Optimizador

El optimizador es el algoritmo que se emplea para ajustar los pesos de la red neuronal durante el proceso de entrenamiento. Uno de los optimizadores más populares es Adam (*Adaptive Moment Estimation*).

Adam combina las ventajas de otros optimizadores, como el descenso de gradiente estocástico (SGD) y el método de momentum. Ajusta adaptativamente las tasas de aprendizaje para cada parámetro, lo cual lo hace eficiente para lograr la convergencia del modelo. [48]

Callbacks

Los callbacks son funciones que se ejecutan en determinados puntos durante el entrenamiento de una red neuronal para mejorar su rendimiento o detenerlo si se cumplen ciertas condiciones. Un callback útil es el *EarlyStopping* (detención temprana).

El *EarlyStopping* se utiliza para prevenir el sobreajuste del modelo deteniendo el entrenamiento cuando no se observa una mejora significativa en la métrica deseada. Esto se logra mediante la monitorización de la métrica objetivo y la aplicación de un límite de paciencia. Si la métrica no mejora después de un número determinado de épocas, el entrenamiento se detiene, evitando así el sobreajuste y ahorrando tiempo y recursos. [34]

3.2.7. Implementación

En este apartado se detalla los pasos que se siguen en la implementación de los modelos, en el proyecto se utiliza principalmente dos librerías de Python para facilitar el proceso del desarrollo, que son Keras y Ultralytics.

Keras: MobileNetV2, EfficientNetB0, InceptionV3, Resnet50V2, Resnet152V2 y Xception

Para implementar los modelos preentrenados en Keras, que son los siguientes: MobileNetV2, EfficientNetB0, InceptionV3, Resnet50V2, Resnet152V2 y Xception, se sigue los siguientes pasos:

1. **Cargar el dataset:** Antes de comenzar a construir nuestro modelo de aprendizaje automático, debemos cargar el conjunto de datos en el que vamos a entrenarlo. Utilizaremos la biblioteca keras.preprocessing.image.ImageDataGenerator para cargar las imágenes y dividir el conjunto de datos en un 80 %(157028) para entrenamiento y un 20 %(39167) para validación.

```
from keras.preprocessing.image import ImageDataGenerator

data_path = './dataset_superclass'
train_datagen = ImageDataGenerator(rescale=1.0 / 255,
    preprocessing_function=tf.keras.applications.efficientnet.
    preprocess_input, validation_split=0.2, horizontal_flip=True)
train_batches = train_datagen.flow_from_directory(
    data_path, target_size=(IMG_SIZE,IMG_SIZE), batch_size=BATCH_SIZE,
        shuffle = False, seed=42, subset='training')
valid_batches = train_datagen.flow_from_directory(
    data_path, target_size=(IMG_SIZE,IMG_SIZE), batch_size=BATCH_SIZE,
        shuffle = False,seed=42, subset='validation')
```

2. **Construcción del modelo (Transfer Learning):** El transfer learning es una técnica que aprovecha los conocimientos aprendidos por un modelo previamente entrenado en un conjunto de datos grande y lo aplica a un problema similar. En este proyecto se utiliza la biblioteca keras.applications para importar un modelo preentrenado y permite también agregar unas capas personalizadas. En este caso, crea una capa de GlobalAveragePooling2D y luego una capa de salida Dense con la función de activación softmax.

```
from tensorflow import keras

base_model = keras.applications.ResNet152V2(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False)  # Do not include the ImageNet classifier at the
        top.

#Then, freeze the base model.
base_model.trainable = False

#Create a new model on top.
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you
    will
```

```
    # learn in a few paragraphs.
    x = base_model(inputs, training=False)
    # Convert features of shape `base_model.output_shape[1:]` to vectors
    x = keras.layers.GlobalAveragePooling2D()(x)
    outputs = keras.layers.Dense(180, activation='softmax')(x)

    model = keras.Model(inputs, outputs)
    model.summary()
```

El modelo construido tiene una estructura como se muestra en la figura 3.17

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 resnet152v2 (Functional)    (None, 7, 7, 2048)        58331648

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 180)               368820

=================================================================
Total params: 58,700,468
Trainable params: 368,820
Non-trainable params: 58,331,648
```

Figura 3.17: Estructura del modelo construido

3. **Compilación del modelo:** Una vez que se ha construido el modelo, se debe compilarlo para que esté listo para el entrenamiento. Para ello se especifica el optimizador como "Adam", la función de pérdida como "Categorical crossentropy" y las métricas seleccionadas, que son: el accuracy en el top 1, el accuracy en el top 5 y la precisión.

```
    model.compile(optimizer=keras.optimizers.Adam(),
                  loss="categorical_crossentropy",
                  metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=1,
                      name='accuracy'), tf.keras.metrics.
                      TopKCategoricalAccuracy(k=5, name='top_5_accuracy'),
                  tf.keras.metrics.Precision(name='precision')])
```

4. **Entrenamiento del modelo:** Con el modelo compilado, se empieza a entrenarlo en nuestro conjunto de datos. Es importante controlar el proceso de entrenamiento para evitar el sobreajuste. Para ello, se utiliza la función EarlyStopping de callbacks para detener el entrenamiento si no se produce ninguna mejora significativa en 5 etapas.

```
1    import tensorflow as tf
2
3    callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
         patience=5)
4    history = model.fit(train_batches, epochs=20, validation_data=
         valid_batches, callbacks=[callback])
```

5. **Construcción de las gráficas:** Después de que el modelo haya terminado de entrenarse, es útil visualizar las métricas de rendimiento a lo largo del tiempo. Utilizaremos la biblioteca matplotlib para construir gráficas que muestren la precisión y la pérdida del modelo durante el entrenamiento.

```
1    # Construyamos horarios de entrenamiento
2    import matplotlib.pyplot as plt
3
4    plt.figure(figsize=(15,5))
5    plt.plot(range(history.epoch[-1]+1),history.history['val_accuracy'],label
         ='val_accuracy')
6    plt.plot(range(history.epoch[-1]+1),history.history['accuracy'],label='
         trn_accuracy')
7    plt.title('Accuracy'); plt.xlabel('Epoch'); plt.ylabel('Percent');plt.
         legend();
8    plt.show()
```

En la figura 3.18 se representa una gráfica de accuracy del modelo Resnet152V2 para clasificar las clases padres.

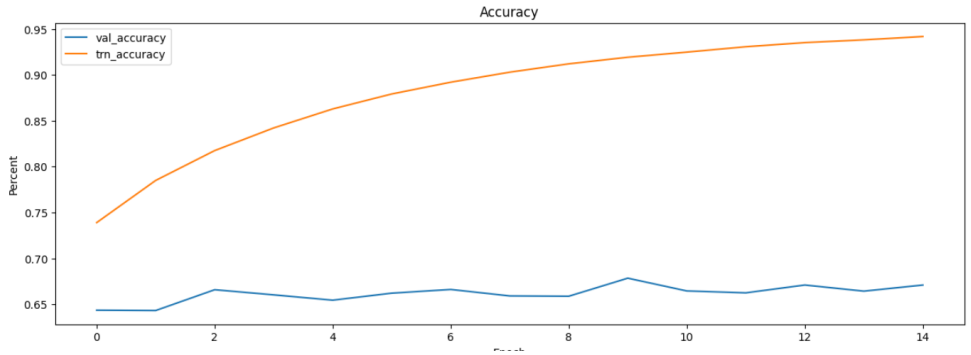

Figura 3.18: Gráfica de accuracy de Resnet152V2 para clases padres

6. **Ajuste fino del modelo (Fine Tuning):** Una vez que el modelo ha sido entrenado, se puede aplicar Fine Tuning para adaptarlo aún más al dataset y mejorar su rendimiento. Esto implica descongelar capas del modelo y volver a entrenar el modelo completo con una tasa de aprendizaje relativamente baja.

```
1    model.trainable=True
2    model.summary()
```

Ahora el modelo tiene más parámetros que pueden ser entrenados, como se puede ver en la figura 3.19 y comparar con la figura 3.17 que tiene mucho menos parámetros que pueden ser entrenados.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_2 (InputLayer)         [(None, 224, 224, 3)]     0

 resnet152v2 (Functional)     (None, 7, 7, 2048)        58331648

 global_average_pooling2d (G  (None, 2048)              0
 lobalAveragePooling2D)

 dense (Dense)                (None, 180)               368820

=================================================================
Total params: 58,700,468
Trainable params: 58,556,724
Non-trainable params: 143,744
```

Figura 3.19: Estructura del modelo al aplicar Fine Tuning

7. **Guardar el modelo en Drive** Después de todo el proceso de Transfer Learning y Fine Tuning, es importante guardar nuestro modelo para poder utilizarlo en el futuro sin tener que volver a entrenarlo desde cero. Podemos guardar el modelo en Google Drive utilizando bibliotecas como google.colab.drive para facilitar su exportación.

```
1  # En Colab notebook
2  from google.colab import drive
3
4  drive.mount('/content/drive')
5  model.save('/content/drive/MyDrive/Colab Notebooks/save_models/
       ResNet152V2Model_superClass_fineTuning')
```

8. **Generar matriz de confusión:** Para la evaluación el rendimiento de los modelos, conociendo cuáles son las clases difíciles/fáciles de clasificar se genera una matriz de confusión. Se utiliza las bibliotecas numpy, seaborn y scikit-learn para crear esta matriz, que nos dará una visión general del accuracy del modelo en la clasificación de diferentes clases.

```
1  from sklearn.metrics import confusion_matrix
2  import numpy as np
3  import seaborn as sns
4
5  class_names = ['Cleaning_stuff', 'Containers', 'Cutlery', 'Drinks', 'Food
       ', 'Fruits', 'Snacks', 'Tableware']
```

```
6
7       # Obtener las predicciones del modelo en el conjunto de validacion
8       predictions = model.predict_generator(valid_batches)
9       predicted_classes = np.argmax(predictions, axis=1)
10
11      # Obtener las etiquetas reales del conjunto de validacion
12      true_classes = valid_batches.classes
13
14      # Calcular la matriz de confusion
15      confusion_mtx = confusion_matrix(true_classes, predicted_classes)
16
17      # Plot the confusion matrix
18      plt.figure(figsize=(10, 8))
19      sns.heatmap(
20          confusion_mtx, xticklabels=class_names, yticklabels=class_names,
                annot=True, fmt="g"
21      )
22      plt.xlabel("Prediction")
23      plt.ylabel("Label")
24      plt.title("Validation Confusion Matrix")
25      plt.show()
26
27      #----------Matriz de confusion normalizada----------
28
29      # Calcular el porcentaje de acierto para cada clase
30      class_totals = np.sum(confusion_mtx, axis=1)  # Total de instancias por
            clase
31      confusion_mtx_norm = confusion_mtx.astype('float') / class_totals[:, np.
            newaxis]  # Dividir cada elemento por el total de instancias en la
            clase correspondiente
32
33      # Plotear la matriz de confusion normalizada
34      plt.figure(figsize=(10, 8))
35      sns.heatmap(
36          confusion_mtx_norm, xticklabels=class_names, yticklabels=class_names,
                annot=True, fmt=".2f"
37      )
38      plt.xlabel("Prediction")
39      plt.ylabel("Label")
40      plt.title("Normalized Validation Confusion Matrix")
41      plt.show()
```

La matriz de confusión generada del modelo Resnet152V2 podemos ver en la figura 3.20, y la normalizada en la figura 3.21

Figura 3.20: Matriz de confusión Resnet152V2 de las clases padres

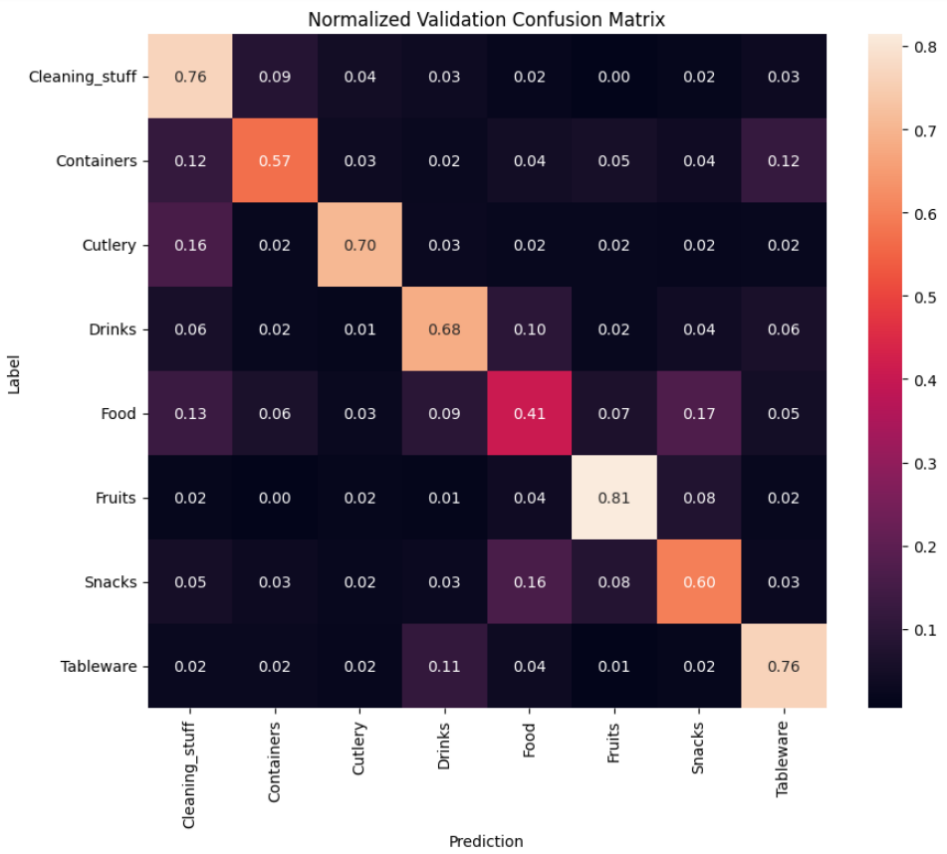

Figura 3.21: Matriz de confusión normalizada Resnet152V2 de las clases padres

Ultralytics: YoloV8

El modelo preentrenado de Yolo(You Only Look Once) no pertenece a la biblioteca Keras, se debería utilizar la biblioteca Ultralytics, por los tanto la implementación de YoloV8(You Only Look Once) es diferente que los demás, los pasos son los siguientes:

1. **Ajustar el dataset:** la estructura que se utiliza el Yolo para la clasificación a diferencia de la estructura que se utiliza en Keras 3.2.2 tiene siguiente aspecto:

```
Directory-/
|
|-- train/
|   |-- clase1/
|   |   |-- 10008_airplane.png
|   |   |-- 10009_airplane.png
|   |   |-- ...
|   |
```

```
|   |-- clase2/
|   |   |-- 1000_automobile.png
|   |   |-- 1001_automobile.png
|   |   |-- ...
|   |
|   |-- ...
|
|-- test/
|   |-- clase1/
|   |   |-- 10_airplane.png
|   |   |-- 11_airplane.png
|   |   |-- ...
|   |
|   |-- clase2/
|   |   |-- 100_automobile.png
|   |   |-- 101_automobile.png
|   |   |-- ...
|   |
|   |-- ...
```

Para ello se ha creado una función que permite hacer este transformación y además dividir aleatoriamente las imágenes para el conjunto de entrenamiento y el conjunto de validación, el porcentaje de las imágenes de entrenamiento y de validación sigue siendo 80 % y 20 %.

```
1   import shutil
2   import random
3
4   def transform_dataset(root_dir, output_dir, train_ratio=0.8):
5       # Create the output directory
6       os.makedirs(output_dir, exist_ok=True)
7       train_dir = os.path.join(output_dir, 'train')
8       val_dir = os.path.join(output_dir, 'val')
9       os.makedirs(train_dir, exist_ok=True)
10      os.makedirs(val_dir, exist_ok=True)
11
12      # Get the list of classes in the root directory
13      classes = [d for d in os.listdir(root_dir) if os.path.isdir(os.path.
            join(root_dir, d))]
14
15      for class_name in classes:
16          class_dir = os.path.join(root_dir, class_name)
17          if not os.path.isdir(class_dir):
18              continue
```

```
19
20              # Get the list of images in the current class
21              images = [f for f in os.listdir(class_dir) if f.lower().endswith
                    (('.png', '.jpg', '.jpeg'))]
22
23              # Shuffle the images randomly
24              random.shuffle(images)
25
26              # Calculate the number of training images based on the specified
                    ratio
27              train_size = int(len(images) * train_ratio)
28
29              # Create the class directories in the training and validation
                    sets
30              train_class_dir = os.path.join(train_dir, class_name)
31              val_class_dir = os.path.join(val_dir, class_name)
32              os.makedirs(train_class_dir, exist_ok=True)
33              os.makedirs(val_class_dir, exist_ok=True)
34
35              # Copy the training images
36              for img in images[:train_size]:
37                  src_path = os.path.join(class_dir, img)
38                  dst_path = os.path.join(train_class_dir, f"{os.path.splitext(
                        img)[0]}_{class_name}.png")
39                  shutil.copy(src_path, dst_path)
40
41              # Copy the validation images
42              for img in images[train_size:]:
43                  src_path = os.path.join(class_dir, img)
44                  dst_path = os.path.join(val_class_dir, f"{os.path.splitext(
                        img)[0]}_{class_name}.png")
45                  shutil.copy(src_path, dst_path)
46      root_dir   = '/content/dataset_superclass'   # Path to the original
            dataset
47      output_dir  = '/content/dataset_super'      # Path for the resized
            dataset
48
49      transform_dataset(root_dir, output_dir, train_ratio=0.8)
```

2. **Compilación y entrenamiento y validación del modelo:** La compilación, el entrenamiento y la validación de Yolo es muy sencillo por la facilidad que ha proporcionado de la librería Ultralytics.

```
1       from ultralytics           import YOLO
2       # Load a model
3       model = YOLO('yolov8x-cls.yaml').load('yolov8x-cls.pt')  # build from
            YAML and transfer weights
4
5       # Train the model
6       model.train(data='/content/dataset_super', epochs=70, patience=70, imgsz
            =224,
7         verbose=True, batch=64,
8         project='/content/drive/MyDrive/Colab Notebooks/save_models', name='
              yolo_70epoch_superClass/classify')
9       model.val()  # no arguments needed, dataset and settings remembered
```

Como se puede ver en el código, en la línea 6 cuando se entrena el modelo, se permite añadir y cambiar cualquier parámetro que aparece en la página `https://docs.ultralytics.com/modes/train/#arguments`. Y por defecto:

- Se calcula el accuracy en top 1 y accuracy en top 5.
- Se genera la matriz de confusión.
- Se le asigna automáticamente el mejor optimizador entre SGD, Adam, Adamax, AdamW, NAdam, RAdam y RMSProp.
- Se le asigna la tasa de aprendizaje 0.01.

El modelo entrenado se guarda en la ruta project/name que se pasa en los parámetros para los siguientes usos.

3.3. El producto del desarrollo

Los componentes generados en este proyecto son los modelos que tienen el mejor peso ajustado para clasificar los objetos en el ámbito doméstico, como se ve en la figura 3.22, se guardar en cada carpeta un archivo "saved_model.pb", que contiene los pesos de la arquitectura en cada momento, antes de aplicar Fine Tuning, y después de aplicar Fine Tuning, los que sirven para clasificar los 180 clases hijos y los que sirven para clasificar 8 clases padres.

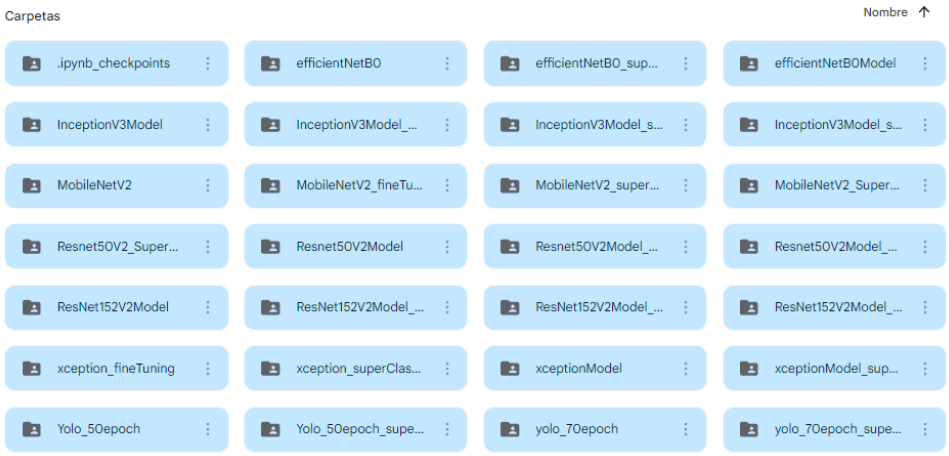

Figura 3.22: Los modelos resultantes

La funcionalidad de los modelos resultantes es que se pueden integrar en cualquier software que es capaz de procesar las imágenes y que tiene memoria suficiente para descargar el modelo, son los modelos especialmente diseñado para los robots

domésticos, ayuda a los robots consiguen la capacidad de identificar los objetos que hay en su entorno, y como consecuencia se pueden mejorar la interacción con las personas y realizar tareas más adecuadamente que permite aumenta la calidad de la vida de las personas.

Capítulo 4

Evaluación

En este apartado se evalúa los modelos implementados en el capítulo anterior, y realizar una comparación de los resultados tanto para las clases padres y como para las clases hijos, sabiendo que el dataset tiene una estructura jerárquica.

4.1. Proceso de evaluación

Utilizando las métricas definidas en la sección 3.2.5 y las configuraciones indicadas en la sección 3.2.6, se ha realizado un conjunto de experimentos con el dataset explicado en la sección 3.2.2 junto con los modelos considerados. A continuación se muestran los resultados obtenidos de todos los modelos.

4.1.1. Resultados de los experimentos para el modelo Resnet152V2

Los valores de validación de la arquitectura Resnet152V2 para la clasificación de las clases padres y hijos se presenta en la tabla 4.1, así como su correspondiente matriz de confusión en las figuras 4.1, 4.2 y 4.3.

Tabla 4.1: Resultados de Resnet152V2

Resnet152V2	**clases padres (8)**	**Clases hijos (180)**
Accuracy	67.09 %	53.62 %
Accuracy top 5	94.42 %	79.63 %
Precisión	71.56 %	57.62 %
Pérdida	1.1476	3.3750

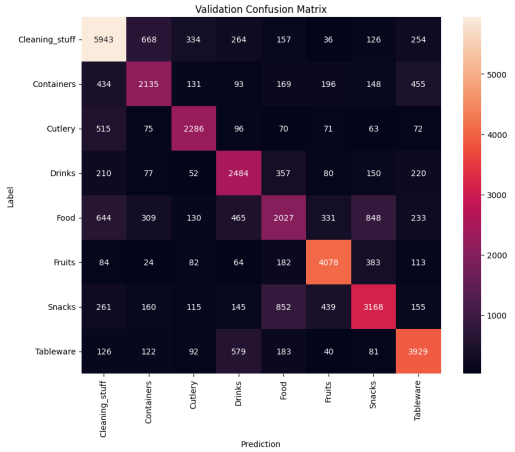

Figura 4.1: Matriz de confusión de Resnet152V2

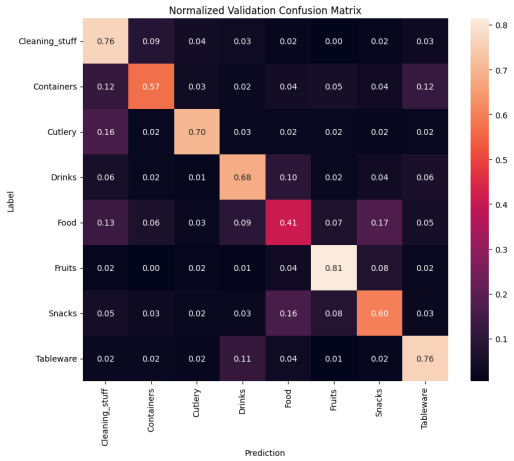

Figura 4.2: Matriz de confusión normalizada de Resnet152V2

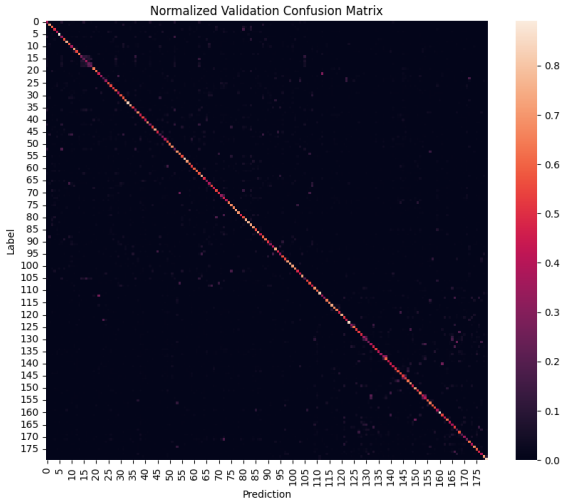

Figura 4.3: Matriz de confusión de las clases hijos de Resnet152V2

4.1.2. Resultados de los experimentos para el modelo Resnet50V2

Los valores de validación de la arquitectura Resnet50V2 para la clasificación de las clases padres y hijos se presenta en la tabla 4.2, así como su correspondiente matriz de confusión en las figuras 4.4, 4.5 y 4.6.

Tabla 4.2: Resultados de Resnet50V2

Resnet50V2	**clases padres (8)**	**Clases hijos (180)**
Accuracy	66.32 %	47.74 %
Accuracy top 5	94.24 %	75.68 %
Precisión	51.46 %	56.86 %
Pérdida	1.2174	3.8702

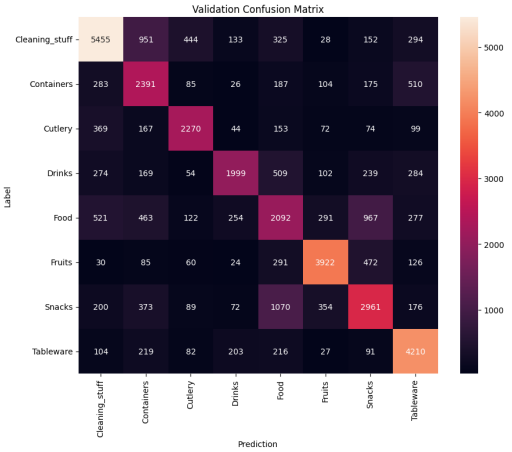

Figura 4.4: Matriz de confusión de Resnet50V2

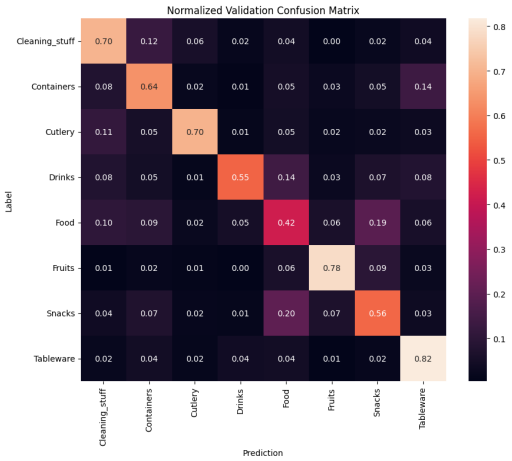

Figura 4.5: Matriz de confusión normalizada de Resnet50V2

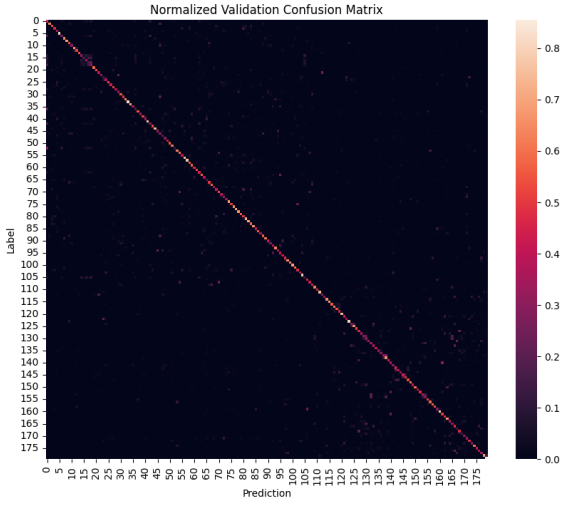

Figura 4.6: Matriz de confusión de las clases hijos de Resnet50V2

4.1.3. Resultados de los experimentos para el modelo InceptionV3

Los valores de validación de la arquitectura InceptionV3 para la clasificación de las clases padres y hijos se presenta en la tabla 4.3, así como su correspondiente matriz de confusión en las figuras 4.7, 4.8 y 4.9.

Tabla 4.3: Resultados de InceptionV3

InceptionV3	**clases padres (8)**	**Clases hijos (180)**
Accuracy	67.58 %	55.41 %
Accuracy top 5	93.88 %	81.68 %
Precisión	71.64 %	58.87 %
Pérdida	1.1440	3.2291

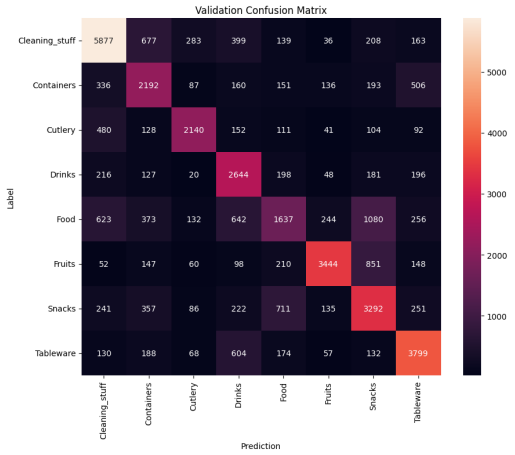

Figura 4.7: Matriz de confusión de InceptionV3

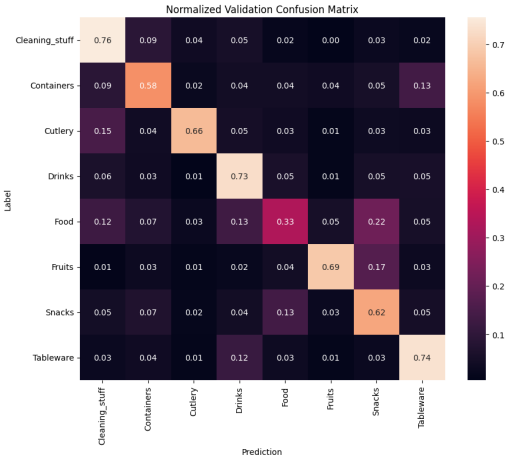

Figura 4.8: Matriz de confusión normalizada de InceptionV3

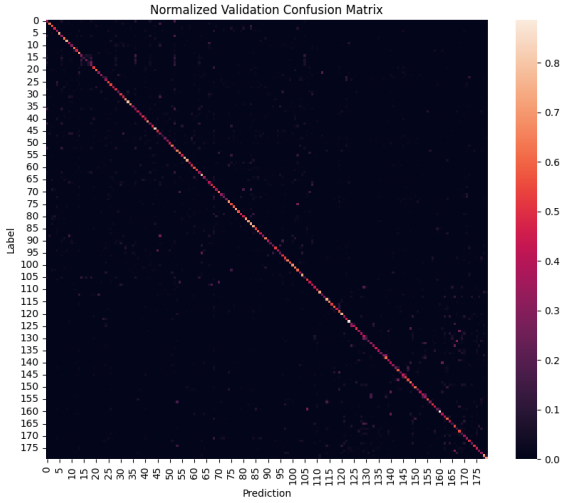

Figura 4.9: Matriz de confusión de las clases hijos de InceptionV3

4.1.4. Resultados de los experimentos para el modelo EfficientNetB0

Los valores de validación de la arquitectura EfficientNetB0 se presenta en la tabla 4.4, así como su correspondiente matriz de confusión en las figuras 4.10, 4.11 y 4.12.

Tabla 4.4: Resultados de EfficientNetB0

EfficientNetB0	**clases padres (8)**	**Clases hijos (180)**
Accuracy	20.02 %	0.69 %
Accuracy top 5	72.63 %	3.33 %
Pérdida	2.0441	5.1834

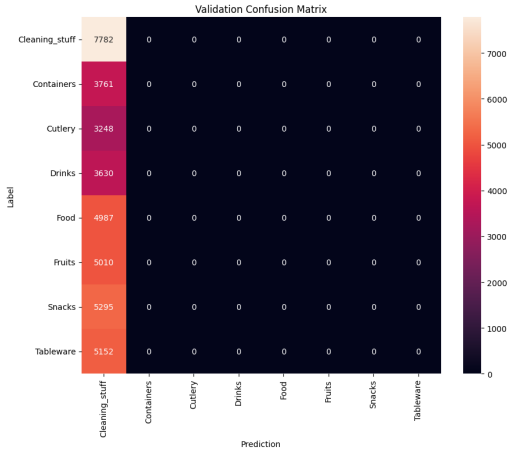

Figura 4.10: Matriz de confusión de EfficientNetB0

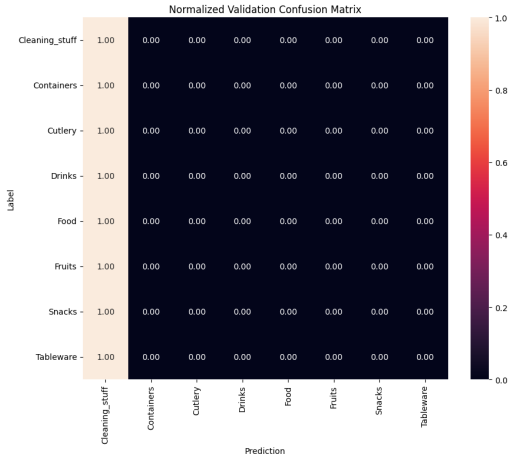

Figura 4.11: Matriz de confusión normalizada de EfficientNetB0

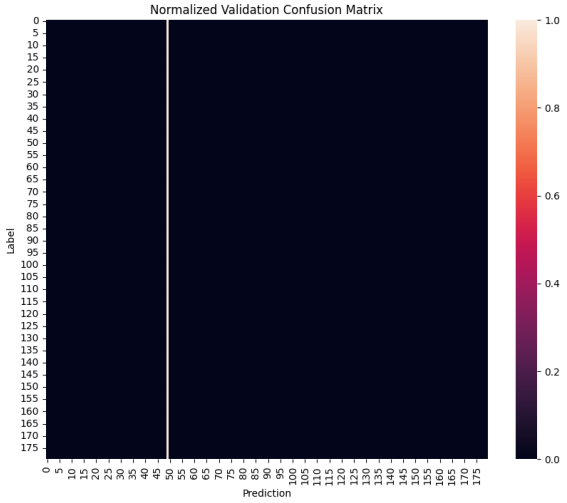

Figura 4.12: Matriz de confusión de las clases hijos de EfficientNetB0

4.1.5. Resultados de los experimentos para el modelo MobileNetV2

Los valores de validación de la arquitectura MobileNetV2 se presenta en la tabla 4.5, así como su correspondiente matriz de confusión en las figuras 4.13, 4.14 y 4.15.

Tabla 4.5: Resultados de MobileNetV2

MobileNetV2	clases padres (8)	Clases hijos (180)
Accuracy	66.38 %	50.53 %
Accuracy top 5	94.46 %	78.69 %
Precisión	72.33 %	62.32 %
Pérdida	1.1139	2.284

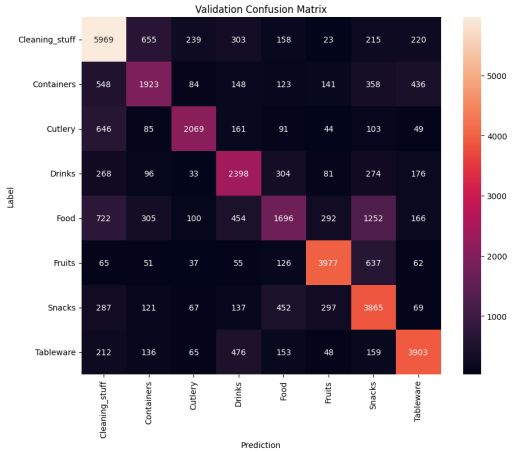

Figura 4.13: Matriz de confusión de MobileNetV2

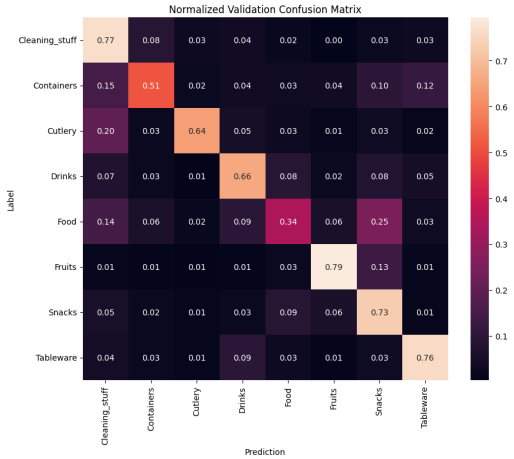

Figura 4.14: Matriz de confusión normalizada de MobileNetV2

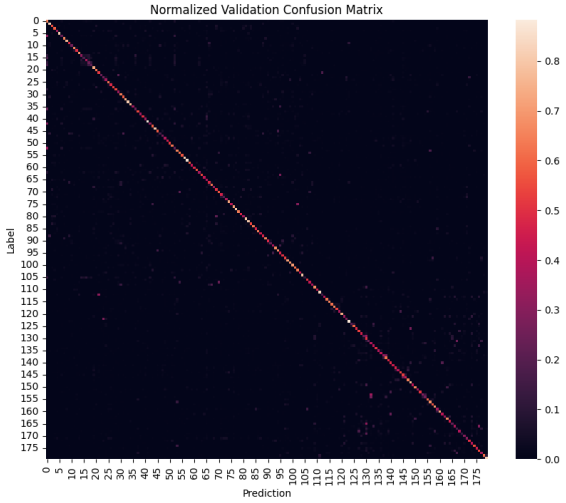

Figura 4.15: Matriz de confusión de las clases hijos de MobileNetV2

4.1.6. Resultados de los experimentos para el modelo Xception

Los valores de validación de la arquitectura Xception se presenta en la tabla 4.6, así como su correspondiente matriz de confusión en las figuras 4.16, 4.17 y 4.18.

Tabla 4.6: Resultados de Xception

Xception	**clases padres (8)**	**Clases hijos (180)**
Accuracy	63.84 %	42.27 %
Accuracy top 5	93.41 %	70.86 %
Precisión	68.32 %	49.82 %
Pérdida	1.3948	4.9396

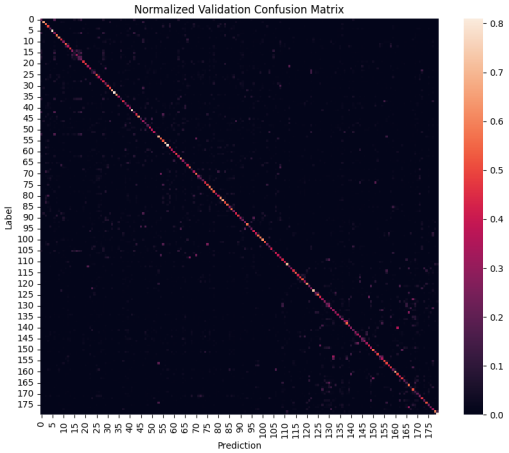

Figura 4.16: Matriz de confusión de Xception

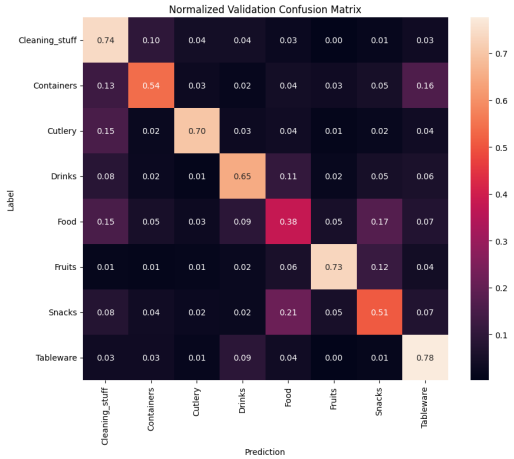

Figura 4.17: Matriz de confusión normalizada de Xception

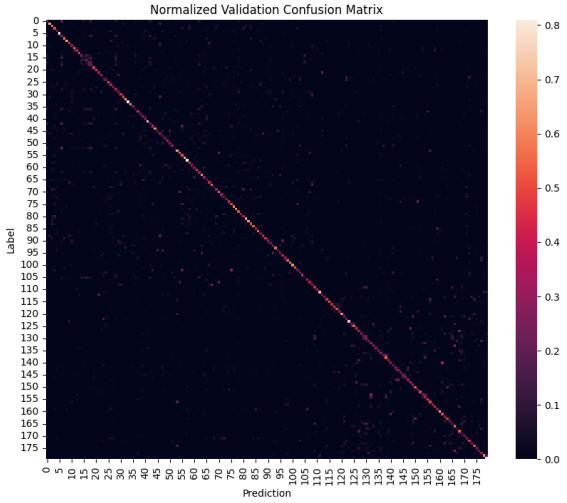

Figura 4.18: Matriz de confusión de las clases hijos de Xception

4.1.7. Resultados de los experimentos para el modelo YoloV8

Los valores de validación de la arquitectura YoloV8 se presenta en la tabla 4.7, así como su correspondiente matriz de confusión en las figuras 4.19, 4.20 y 4.21.

Tabla 4.7: Resultados de YoloV8

YoloV8	**clases padres (8)**	**Clases hijos (180)**
Accuracy	83.10 %	58.60 %
Accuracy top 5	98.70 %	83.90 %
Pérdida	0.538	1.094

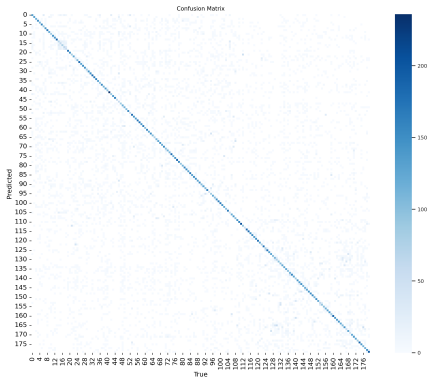

Figura 4.19: Matriz de confusión de YoloV8

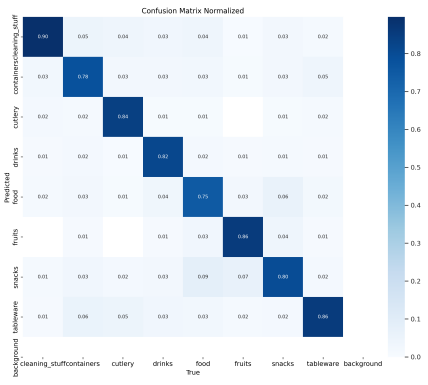

Figura 4.20: Matriz de confusión normalizada de YoloV8

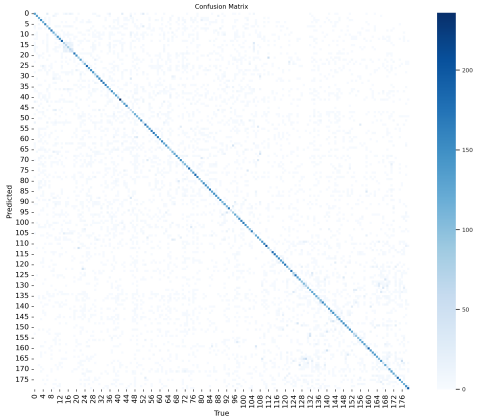

Figura 4.21: Matriz de confusión de las clases hijos de YoloV8

4.2. Análisis de resultados

En esta sección se realiza la comparación de los resultados de los modelos.

Como se puede observar en la tabla 4.8, el mejor modelo obtenido para la clasificación de los objetos domésticos divididos en 8 clases padres es el modelo YoloV8, por su alto rendimiento comparando con los demás, el accuracy se alcanza a un 83.10 %, los modelos Resnet152V2, Resnet50V2, InceptionV3, MobileNetV2 y Xception tienen un rendimiento muy similar, el accuracy está entorno al 65 %, y el modelo EfficientNetB0 no ha sido capaz de clasificar bien, como se muestra en la figura 4.10, predice todos los imágenes como productos de la limpieza.

Tabla 4.8: Resultados de los modelos para las clases padres

Cla. Clases padres	Accuracy	Accuracy top 5	Precisión	Pérdida
Resnet152V2	67.09 %	94.42 %	71.56 %	1.1476
Resnet50V2	66.32 %	94.24 %	51.46 %	1.2174
InceptionV3	67.58 %	93.88 %	71.64 %	1.1440
EfficientNetB0	0.69 %	3.33 %	-	2.0441
MobileNetV2	66.38 %	94.46 %	72.33 %	1.1139
Xception	63.84 %	93.41 %	68.32 %	1.3948
YoloV8	83.10 %	98.70 %	-	0.538

Como se puede ver en la tabla 4.9, el modelo YoloV8 también muestra un rendimiento relativamente alta en la clasificación de las clases hijos. Con un accuracy del 58.60 % y un accuracy en el top 5 del 83.90 %, supera a los demás modelos evaluados. Además, presenta la menor pérdida con un valor de 1.094, el rendimiento de los modelos Resnet152V2 y InceptionV3 no es mucho peor, pueden ser dos modelos buenos

también para la clasificación, y el modelo EfficientNetB0 no es capaz clasificar bien este tipo de problemas.

Tabla 4.9: Resultados de los modelos para las clases hijos

Cla. Clases hijos	Accuracy	Accuracy top 5	Precisión	Pérdida
Resnet152V2	53.62 %	79.63 %	57.62 %	3.3750
Resnet50V2	47.74 %	75.68 %	56.86 %	3.8702
InceptionV3	55.41 %	81.68 %	58.87 %	3.2291
EfficientNetB0	0.69 %	3.33 %	-	5.1834
MobileNetV2	50.53 %	78.69 %	62.32 %	2.284
Xception	42.27 %	70.86 %	49.82 %	4.9396
YoloV8	58.60 %	83.90 %	-	1.094

A continuación se realiza algunos ejemplos de clasificación con el modelo que tiene mejor rendimiento, el modelo YoloV8.

4.2.1. Ejemplos de clasificaciones correctas

En la figura 4.22 se muestran algunas imágenes que se han clasificado correctamente.

4.2.2. Ejemplos de clasificaciones incorrectas

En la figura 4.23 se muestran imágenes que no se han clasificado correctamente.

A veces es difícil ser precisos (acertar) por los motivos mencionados en la sección 3.2.2.

El primer imagen contiene a la vez alimento(food), frutas y un plato de plástico, la segunda imagen se confunde entre una bebida y producto de la limpieza, por la mayor similitud que tiene, la tercera imagen tiene un animal que ocupa la mayor parte de la imagen, pero la etiqueta del imagen es fruta, ya que el modelo no clasifica los animales, la cuarta imagen es una bebida, pero contenido en los vasos.

Como muchas imágenes pueden pertenecer a varias clases, mostrar varios productos en una sola imagen o que tienen características de ambas categorías como se puede ver en las matrices de confusión de cualquier modelo, la clasificación entre las categorías food y snack ha sido difícil de precisar(acertar).Pero el resultado obtenido del YoloV8, según viendo las imágenes del ejemplo se puede considerar un modelo excelente para el objetivo de este trabajo, la clasificación de los objetos en robótica social.

Figura 4.22: Ejemplos de clasificaciones correctas

Figura 4.23: Ejemplos de clasificaciones incorrectas

Capítulo 5

Conclusión

En este apartado, se explicará la conclusión alcanzada después de llevar a cabo el proyecto, donde se detallará los aportaciones realizadas y se proporcionará un breve resumen de los trabajos futuros que se pueden realizar, los problemas encontrados durante el desarrollo y las opiniones personales.

Aportaciones realizadas

En la realización de este trabajo fin de grado, ha realizado las tareas mencionadas en la sección 2.2.1 para conseguir el cumplimiento de los requisitos definidos en la sección 3.2.1, se ha probado los modelos de alto rendimiento que se encuentra hoy en día adaptando el dataset RoboCup@Home-Objects mediante la aplicación de las técnicas Transfer-Learning y Fine-Tuning.

Con los resultados obtenidos de cada uno de los modelos se ha realizado una comparación entre ellos y finalmente se obtiene el mejor modelo para la clasificación de los objetos domésticos, que es el modelo YoloV8. El modelo YoloV8 ofrece la capacidad de clasificación en tiempo real, y su diseño optimizado lo hace adecuado para una variedad de aplicaciones, siendo fácilmente adaptable a diferentes plataformas de hardware, desde dispositivos periféricos hasta API en la nube.

Con todo esto se cierra este trabajo fin de grado cumpliendo el objetivo del proyecto.

Trabajos futuros

Los posibles trabajos futuros que pueden seguir para mejorar la clasificación de los objetos domésticos son los siguientes:

1. Integración del modelo al robot doméstico para mejorar la experiencia del usuario.
2. Adaptación del dataset dependiendo el uso de la clasificación.
3. Más Experimentaciones de los modelos para aumentar el rendimiento.
4. Ampliación de las funcionalidades:clasificación detección y clasificación por el vídeo.

Problemas encontrados

Durante el desarrollo del proyecto ha surgido diversos problemas como los siguientes:

1. **Tiempo de ejecución excesivamente larga:** al utilizar un dataset muy compleja que contiene 196.195 imágenes, el tiempo de ejecución es más larga que el tiempo que permite mantener la sesión de Google Colab, y por este motivo la implementación se divide por bloques pequeños, primero crea el modelo básico con Transfer-Learning y guardar el modelo entrenado en Google Drive para Fine-Tuning posteriormente, así sucesivamente para todos los modelos tanto para las clases padres y hijos sucesivamente para no perder los trabajos en medio.
2. **Herramientas diversas:** La existencia de numerosas herramientas disponibles para el desarrollo, es una ventaja pero también se complica el trabajo al principio, como no tenía mucho conocimiento previamente se me complica encontrar las herramientas útiles para conseguir un desarrollo sistemáticamente de los modelos.
3. **Recursos limitados:** Los recursos ofrecidos por Google Colab son limitados, como el tiempo de uso de la tarjeta gráfica y la capacidad de la memoria RAM, lo cual ralentiza el proceso del desarrollo. Sin embargo al ser un entorno diseñado específicamente para trabajar con arquitecturas de redes neuronales, sigue siendo una opción preferible al desarrollar en local, además no tiene problemas con las versiones de las librerías.

Opiniones personales

El uso de la inteligencia artificial está aplicando en muchos ámbitos, en este trabajo en el ámbito de robótica social y pienso que en próximos años podrá haber

más tipos de proyectos vinculados a la inteligencia artificial. Este trabajo aunque no es muy complicado, pero me parece un trabajo muy interesante y he aprendido mucho con ellos, que puede ser un trabajo significativo para mis futuros estudios y trabajos.

Lista de referencias

[1] Anaconda | the world's most popular data science platform. https://www.anaconda.com/, (Accessed on 06/20/2023)

[2] Matplotlib — visualization with python. https://matplotlib.org/, (Accessed on 06/20/2023)

[3] Numpy. https://numpy.org/, (Accessed on 06/20/2023)

[4] scikit-learn: machine learning in python — scikit-learn 1.2.2 documentation. https://scikit-learn.org/stable/, (Accessed on 06/21/2023)

[5] seaborn: statistical data visualization — seaborn 0.12.2 documentation. https://seaborn.pydata.org/, (Accessed on 06/20/2023)

[6] Visual studio code - code editing. redefined. https://code.visualstudio.com/, (Accessed on 06/20/2023)

[7] Welcome to python.org. https://www.python.org/, (Accessed on 06/20/2023)

[8] Ahmed, T., Sabab, N.: Classification and understanding of cloud structures via satellite images with efficientunet. SN Computer Science **3** (01 2022). https://doi.org/10.1007/s42979-021-00981-2

[9] Albawi, S., Mohammed, T.A., Al-Zawi, S.: Understanding of a convolutional neural network. In: 2017 international conference on engineering and technology (ICET). pp. 1–6. Ieee (2017)

[10] Ariza, D.A.: Efectividad de la gestión de los proyectos: una perspectiva constructivista. Obras y proyectos (22), 75–85 (2017)

[11] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)

[12] Bisong, E., Bisong, E.: Google colaboratory. Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners pp. 59–64 (2019)

[13] Breazeal, C., Dautenhahn, K., Kanda, T.: Social robotics. Springer handbook of robotics pp. 1935–1972 (2016)

[14] Cloud: Advanced guide to inception v3. `https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=es_419` (01 2023)

[15] Cordero Morales, D., Ruiz Constanten, Y., Torres Rubio, Y.: Sistema de razonamiento basado en casos para la identificación de riesgos de software. Revista Cubana de Ciencias Informáticas **7**(2), 222–239 (2013)

[16] De-La-Hoz, E.J., De-La-Hoz, E.J., Fontalvo, T.J.: Metodología de aprendizaje automático para la clasificación y predicción de usuarios en ambientes virtuales de educación. Información tecnológica **30**(1), 247–254 (2019)

[17] De Leon, J.G.M.P.: Introducción al análisis de riesgos. Editorial Limusa (2007)

[18] Dina M. Ibrahim, N.M.E.: Improving date fruit classification using cyclegan-generated dataset. Computer Modeling in Engineering & Sciences **131**(1), 331–348 (2022). https://doi.org/10.32604/cmes.2022.016419, `http://www.techscience.com/CMES/v131n1/46639`

[19] Ertam, F., Aydın, G.: Data classification with deep learning using tensorflow. In: 2017 international conference on computer science and engineering (UBMK). pp. 755–758. IEEE (2017)

[20] García-Peñalvo, F.J., Cruz-Benito, J., et al.: Informe de buena práctica-proyecto europeo vals y semester of code: Prácticas virtuales en empresas y fundaciones relacionadas con el software libre a nivel europeo (2015)

[21] Gulli, A., Pal, S.: Deep learning with Keras. Packt Publishing Ltd (2017)

[22] Gupta, N., et al.: Artificial neural network. Network and Complex Systems **3**(1), 24–28 (2013)

[23] Infoautónomos: ¿qué son las amortizaciones? tablas de amortización y gastos deducibles. `https://www.infoautonomos.com/contabilidad/tablas-de-amortizacion-para-los-bienes-de-una-empresa/`, (Accessed on 07/05/2023)

[24] Izaurieta, F., Saavedra, C.: Redes neuronales artificiales. Departamento de Física, Universidad de Concepción Chile (2000)

[25] de Jiménez, R.L.M.: Metodologías ágiles de desarrollo de software aplicadas a la gestión de proyectos empresariales. Revista tecnológica **8** (2015)

[26] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: Proceedings of the first international conference on Autonomous agents. pp. 340–347 (1997)

[27] Krüger, F.: Activity, Context, and Plan Recognition with Computational Causal Behaviour Models. Ph.D. thesis (12 2016)

[28] Massouh, N., Brigato, L., Iocchi, L.: Robocup@home-objects: Benchmarking object recognition for home robots. pp. 397–407 (12 2019). https://doi.org/10.1007/978-3-030-35699-6$_3$1

[29] Mehmood, A.: Efficient anomaly detection in crowd videos using pre-trained 2d convolutional neural networks. IEEE Access **PP**, 1–1 (10 2021). https://doi.org/10.1109/ACCESS.2021.3118009

[30] Mery, D.: Visión por computador. Santiago de Chile. Universidad Católica de Chile (2004)

[31] Moreno, A., Armengol, E., Béjar Alonso, J., Belanche Muñoz, L.A., Cortés García, C.U., Gavaldà Mestre, R., Gimeno, J.M., Martín Muñoz, M., Sànchez-Marrè, M.: Aprendizaje automático (1994)

[32] Ollivier, Y.: Riemannian metrics for neural networks i: feedforward networks. Information and Inference: A Journal of the IMA **4**(2), 108–153 (2015)

[33] Pérez Vidal, A.J., Castro-González, Á., Alonso Martín, F., Castillo, J.C., Salichs, M.Á., et al.: Evolución de la robótica social y nuevas tendencias. Actas de las XXXVIII Jornadas de Automática (2017)

[34] Prechelt, L.: Early stopping-but when? In: Neural Networks: Tricks of the trade, pp. 55–69. Springer (2002)

[35] Qazi, E., Zia, T., Almorjan, A.: Deep learning-based digital image forgery detection system. Applied Sciences **12**, 2851 (03 2022). https://doi.org/10.3390/app12062851

[36] Rivera, S.H., Lema, L.Z., Freire, A.M., Rojas, L.V., Villa, A.E.: Métodos de clasificación en minería de datos meteorológicos. Perfiles **2**(20), 107–113 (2018)

[37] Rouhiainen, L.: Inteligencia artificial. Madrid: Alienta Editorial (2018)

[38] Sharma, S., Sharma, S., Athaiya, A.: Activation functions in neural networks. Towards Data Sci **6**(12), 310–316 (2017)

[39] Sigaki, H.Y., Lenzi, E.K., Zola, R.S., Perc, M., Ribeiro, H.V.: Learning physical properties of liquid crystals with deep convolutional neural networks. Scientific Reports **10**(1), 1–10 (2020)

[40] Tajbakhsh, N., Shin, J.Y., Gurudu, S.R., Hurst, R.T., Kendall, C.B., Gotway, M.B., Liang, J.: Convolutional neural networks for medical image analysis: Full training or fine tuning? IEEE transactions on medical imaging **35**(5), 1299–1312 (2016)

[41] Tragoudaras, A., Stoikos, P., Fanaras, K., Tziouvaras, A., Floros, G., Dimitriou, G., Kolomvatsos, K., Stamoulis, G.: Design space exploration of a sparse mobilenetv2 using high-level synthesis and sparse matrix techniques on fpgas. Sensors **22**, 4318 (06 2022). https://doi.org/10.3390/s22124318

[42] Trigás Gallego, M.: Metodologia scrum (2012)

[43] ultralytics: Ultralytics yolov8. `https://github.com/ultralytics/ultralytics` (05 2023)

[44] Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. Journal of Big data **3**(1), 1–40 (2016)

[45] Yamazaki, K., Ueda, R., Nozawa, S., Kojima, M., Okada, K., Matsumoto, K., Ishikawa, M., Shimoyama, I., Inaba, M.: Home-assistant robot for an aging society. Proceedings of the IEEE **100**(8), 2429–2441 (2012)

[46] Zhang, S., Yu, H.: Person re-identification by multi-camera networks for internet of things in smart cities. IEEE Access **6**, 76111–76117 (2018)

[47] Zhang, Z., Sabuncu, M.: Generalized cross entropy loss for training deep neural networks with noisy labels. Advances in neural information processing systems **31** (2018)

[48] Zhang, Z.: Improved adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS). pp. 1–2. Ieee (2018)

Anexo A

Control de versiones

Se ha utilizado como sistema de control de versiones Git y su versión web Github. Todas las implementaciones de los modelos están alojados en el repositorio `https://github.com/TheSmileXiao/TFG` y para reproducir lo que se ha implementado en este proyecto se pueden descargar el dataset que se utilizó en la competencia RoboCup@Home-OBJECTS en la siguiente página: `https://sites.google.com/diag.uniroma1.it/robocupathome-objects`.

También se ha usado Google Drive, que permite subir los archivos grandes como el dataset y los modelos entrenados que no ha permitido almacenar en git, el enlace del drive es el siguiente: `https://drive.google.com/drive/folders/12Vw7ecwYbihCHBCfY9KRjdgpw_B1DJNX?usp=sharing`.

Anexo B

Manual de usuario

Se han utilizado jupyter notebooks para desarrollar los modelos así que su uso es muy intuitivo, simplemente ejecutando cada celda que los forman.

Los códigos están organizando linealmente de arriba hacia abajo por celdas, y las funciones de cada celda están explicado por palabras como se puede ver en la figura B.1.

Ejemplo para el modelo Yolo

- Importar las librerías B.1

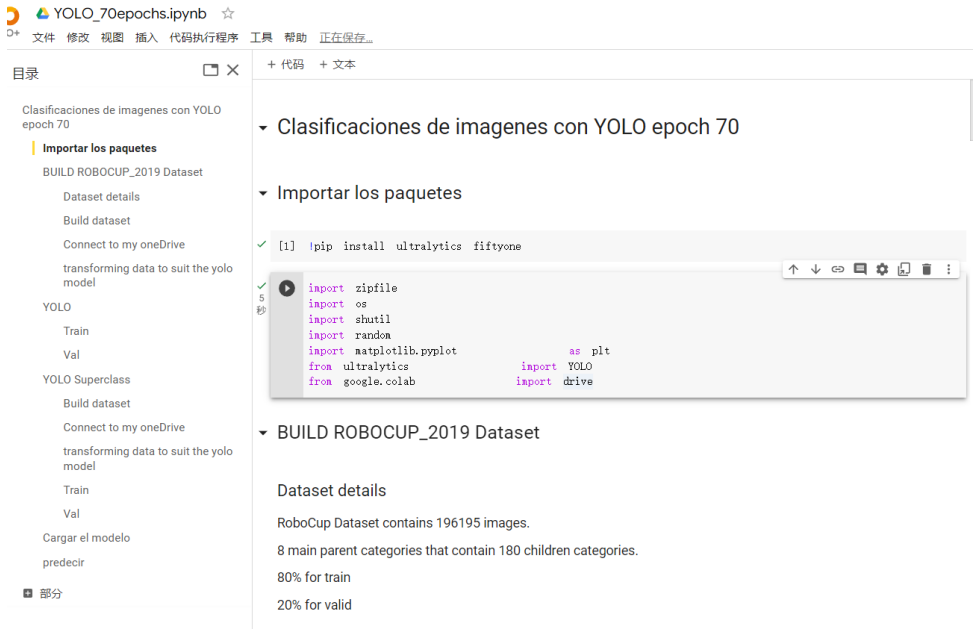

Figura B.1: Manual de uso 1

- Conectar al Drive y descomprimir el dataset B.2

Figura B.2: Manual de uso 2

- Entrenar el modelo B.3

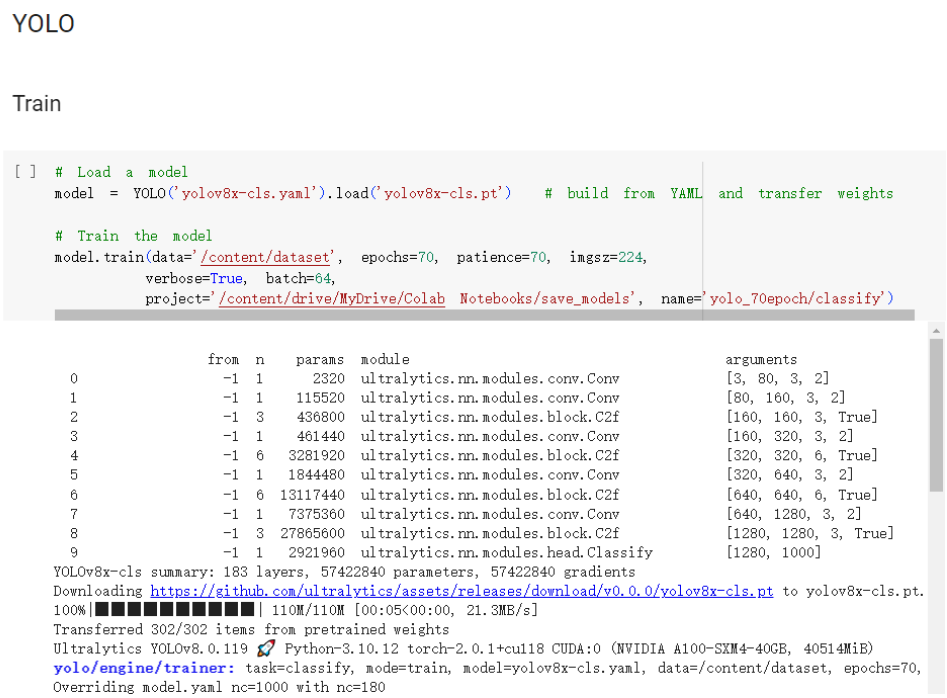

Figura B.3: Manual de uso 3

- Evaluar el modelo B.4

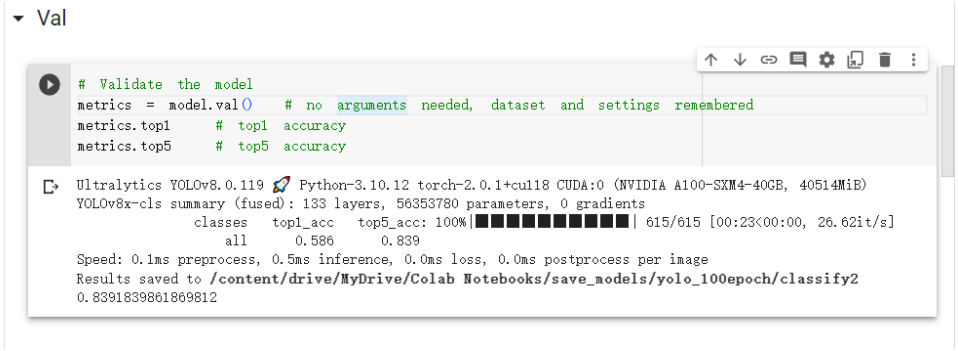

Figura B.4: Manual de uso 4

- Cargar el modelo entrenado y predecir las imágenes B.5

Figura B.5: Manual de uso 5