UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Espantapájaros selectivo basado en visión por computador

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Acosta Triana, José Miguel

Tutor/a: Martínez Hinarejos, Carlos David

CURSO ACADÉMICO: 2021/2022

Resumen

Los espantapájaros han sido una herramienta utilizada desde antaño para protegerse de pájaros dañinos. A día de hoy los pájaros pueden causar estragos en ciudades, contaminando con sus excrementos, deteriorando con ello los edificios, consumiendo plantas o cultivos de jardines urbanos e incluso transmitiendo enfermedades peligrosas.

En este proyecto se moderniza la idea del espantapájaros y se convierte en un sistema completo sencillo de utilizar y accesible. Es capaz de darle al usuario la libertad de mantener a pájaros no deseados fuera de su propiedad; para ello no hace falta adquirir ningún hardware específico para su funcionamiento, pues se puede ejecutar en cualquier dispositivo Android, poniendo al alcance de cualquier persona una poderosa herramienta donde mantener a pájaros deseados y espantar a los dañinos es posible con unos cuantos toques de la pantalla.

Para el desarrollo de esta solución se utilizan técnicas de visión por computador, *machine learning*, servicios *cloud*, arquitectura cliente/servidor y, por supuesto, desarrollo de aplicaciones Android.

Palabras clave: aplicación, Android, Python, visión por computador, pájaros, clasificador

Abstract

Scarecrows have been a tool used since ancient times to protect against harmful species of birds. Nowadays, birds can cause havoc in cities, contaminating with their excrements, deteriorating the exteriors of buildings, consuming plants and crops in urban gardens and even transmitting dangerous diseases.

In this project, the idea of the scarecrow is revamped and is transformed into a complete system that is easy to use and accessible. It is capable of giving the user the freedom to keep unwanted birds out of their property without the need of acquiring specialized hardware to operate, as it is able to run in any Android device. This fact gives access to any user to a powerful tool that can scare away harmful bird species while keeping wanted birds with just a few touches of the screen.

To develop this solution, computer vision techniques are used; the system also includes other technologies such as machine learning, cloud services, client/server architecture and Android app development.

Keywords: app, Android, Python, computer vision, birds, classifier

Tabla de contenidos

1. Introducción

1.1 Motivación

Los pájaros son muy comunes en zonas urbanas, e independientemente de su valor positivo, presentan peligro para los humanos. Su presencia puede causar problemas tales como contaminar fuentes de agua, pueden comer plantas que se tengan accesibles al exterior, dañar estructuras con la corrosión de sus excrementos, ensuciar tejados, balcones y vehículos. También son capaces de transmitir enfermedades a los humanos, lo que se vuelve más común a medida que aumenta su nivel poblacional. Algunas de las enfermedades pueden ser por ejemplo salmonelosis o psittacosis. Incluso que el animal haya estado en cierto espacio y haya defecado ya lo hace un peligro, ya que inhalar accidentalmente el polvo dejado por sus heces secas puede transmitir problemas de salud [1], sobre todo a individuos con inmunodepresión. Además, son portadores de ácaros, garrapatas y pulgas.

Varias aves tales como gorriones, estorninos o palomas, que son habitualmente halladas en entornos urbanos, pueden presentar peligros para edificios y vehículos debido al daño que causan sus excrementos en los materiales [2].

También hay personas que disfrutan del acompañamiento de las aves urbanas, y desean que se aproximen algunas especies y no otras, ya sea por gusto personal, deseo de estudiarlas, sacarles fotografías, o por beneficios que pueden traer, por ejemplo, eliminar plagas tales como insectos.

Por estos motivos la presencia de muchos pájaros urbanos puede ser nociva, pero se puede también desear que otras especies que no se consideren dañinas se aproximen a comederos, balcones o incluso jardines en zonas urbanas. Ya que no todas las especies pueden ser consideras dañinas debemos tener alguna forma de distinguir y excluir las aves no deseadas.

Existen ya algunos dispositivos que son capaces de espantar pájaros, pero no tienen ningún tipo de discriminación ni personalización asociada. Incluso muchos de los dispositivos que hay funcionan simplemente por sensores de movimiento, pudiendo confundir pájaros con muchos otros sucesos que activen el sensor. Además, no son capaces de detectar la especie en caso de que deseemos

La principal motivación de este trabajo es resolver este impedimento, a través de la creación de un software que sea capaz de distinguir los pájaros, seleccionar aquellos que puedan ser considerados no deseados por el usuario y acto seguido espantarlos para evitar los problemas mencionados anteriormente.

Personalmente, es un problema que me ha afectado, ya que en varios sitios donde he residido no es raro que aves, sobre todo palomas, excreten en la fachada, balcón e incluso ventanas, sitios en los que luego es difícil limpiar debido al limitado acceso a estas zonas exteriores. Sin embargo, hay otras especies que no suelen aproximarse a

mi balcón y sin embargo me resultan agradables. Por tanto, deseo tener una forma de evitar los tipos de pájaro que yo considere que dan problemas y mantener a los que no cerca, gozando así de un espacio más natural dentro de la urbe.

Además, desde que entré en la carrera siempre he deseado trabajar con visión por computador, ya que me parece de gran interés observar cómo un algoritmo puede detectar objetos y tomar respuesta según lo que ve, como si se tratase de un ser vivo capaz de reconocer sus alrededores y reaccionar a estos estímulos visuales.

1.2 Objetivos

Con este proyecto, el primer objetivo es recoger datos de varias de las especies de pájaros más comunes en entornos urbanos, concretamente en Valencia, que es donde se lleva a cabo este trabajo, recolectando imágenes de estas especies con las que podamos entrenar un detector y clasificador.

Tras esto es importante elegir un modelo de detección de objetos que sea capaz de detectar un pájaro en una imagen de forma fiable, independientemente del entorno, y después escoger un modelo de clasificación de imágenes adecuado que permita discernir las diferentes aves las unas de las otras tomando como entrada una fotografía.

Todo esto iría integrado en una aplicación móvil compatible con sistemas Android, que se tendrá como objetivo que sea compatible con la mayor cantidad de dispositivos posibles. Esta aplicación se comunicaría con un servidor remoto, que recibiría sus peticiones para detectar aves en la imagen, determinaría su especie y luego devolvería una respuesta a la app móvil. Este servidor estará siendo ejecutado en un entorno *cloud* que habría que escoger. Esto facilitaría su uso, escalabilidad y daría una alta disponibilidad.

Esta aplicación se espera que esté disponible en la Google Play Store para su fácil acceso desde cualquier dispositivo que cumpla unos mínimos requisitos para el correcto funcionamiento de este software. De esta forma, cualquier usuario con unos mínimos conocimientos del funcionamiento de un dispositivo Android podría hacer uso de esta aplicación.

La aplicación final debería ser capaz de ser personalizable al gusto del usuario para poder espantar los pájaros considerados no deseados, que pueden ser distintos según cada usuario.

Debe contar también con una precisión razonable (se espera que sea de al menos 75%, que es aceptable en un sistema móvil de rápida ejecución) y que su tiempo de respuesta sea rápido para no mantener a los pájaros no deseados mucho tiempo sin espantarlos (se considerará apto que tarde menos de 8 segundos en detectar un pájaro y tomar las acciones correspondientes para espantarlo).

1.3 Metodología

Al tener que llevar a cabo un proyecto con diversas partes acopladas entre sí, surge la necesidad de tener un control sobre el orden y la implementación de cada uno de los módulos software que formarán este trabajo. Para ello habrá que realizar un análisis de los requisitos de la aplicación para escoger un modelo de trabajo adecuado.

Por tanto, hay que elegir un enfoque metodológico para organizar el proceso de creación. Nosotros en este caso hemos optado por la metodología *Waterfall [3]*, también conocida como “cascada” en castellano. Es una de las más populares en cuanto a desarrollo de software se refiere y la más tradicional, además de secuencial. Se basa en subdividir el trabajo en fases o pasos, donde se irán realizando de forma secuencial y no se comenzará el siguiente hasta no haber terminado todos los anteriores.

Esto requiere una gran cantidad de planificación antes de comenzar, ya que es una metodología que carece de flexibilidad para adaptarse a grandes modificaciones de los requisitos establecidos. Pero a cambio proporciona unos objetivos claros y con un progreso fácilmente medible, y ya que deseamos crear una aplicación con unos requisitos muy marcados desde el primer momento, ayudará a entender cuál es el progreso y organizar mejor el tiempo del que se dispone para dedicar a cada módulo. Además, es una metodología idónea para el desarrollo de software con múltiples módulos, ya que permite ir realizándolos uno a uno y luego acoplarlos entre sí, lo que permite probar cada uno por separado y realizar un proceso de testeo y depuración más simple, pudiendo descubrir errores de diseño de forma temprana y mitigarlos antes de que causen problemas mayores. Esto permite realizar un desarrollo con más robustez, ya que no es importante que la aplicación sea utilizable desde un primer instante, proporcionando más tiempo para ajustar cada parte de la aplicación.

Usualmente, esta metodología divide el proceso en cinco partes distintas:

- Análisis: Se organizan los requisitos que tendrá que cumplir la aplicación, los tiempos aproximados a dedicar a cada parte y los riesgos que pueden surgir en cada fase; deben ser claros, ya que con esta metodología tener que cambiar los requisitos posteriormente puede resultar desastroso.
- Diseño: En esta fase se seleccionan las soluciones a los requisitos apuntados anteriormente, se diseñan los componentes y la estructura general que tendrá el proyecto.
- Implementación: Se programa cada parte del software; en nuestro caso también tomaremos una aproximación secuencial en esta fase, no empezando el desarrollo de una parte de la aplicación hasta no tener las anteriores terminadas; se realizan también pruebas unitarias.
- Verificación: Se unen todos los componentes de la aplicación y se realizan las pruebas de sistema de integración, corrigiendo errores que surjan y retocando pequeños detalles.
- Mantenimiento: Con la aplicación ya terminada, se pone en funcionamiento y se despliega, teniendo ya uso real; posteriormente a la entrega de este

proyecto, se podrán arreglar errores o añadir mejoras en base a retroalimentación proporcionada por usuarios.

Para la organización de las tareas y medición del progreso hemos hecho uso del software de administración de proyectos Trello, que permite crear tarjetas, donde cada una representa una tarea y se le puede asignar una aproximación en horas, una fecha límite o subobjetivos a ir cumpliendo. Esto facilita la resolución de problemas y ayuda a gestionar los requisitos de forma visual e interactiva, además de tener en todo momento una idea clara de qué se está desarrollando y qué es lo que queda.

1.4 Estructura

Este documento se subdivide en diferentes capítulos, cada uno abordando una temática dentro del proyecto.

En este primer capítulo se ha introducido el proyecto, estableciendo los motivos que han llevado a su realización. También se presentan los objetivos que nos hemos propuesto para la creación de la aplicación, lo que se espera conseguir con este software y, finalmente, se explica la metodología a usar y las razones por las que se ha escogido, incluyendo una breve descripción de su funcionamiento.

En el segundo capítulo se estudia el estado del arte, es decir, la tecnología y los métodos usados actualmente para hacer frente a los diferentes problemas que deseamos resolver con la aplicación y explicar su funcionamiento básico. También se explica qué es lo que aporta nuestra aplicación que no hayan realizado otros anteriormente.

En el tercer capítulo se expone un análisis detallado de los problemas a resolver. También se incluye una descripción del plan de trabajo seguido y de los costes económicos y temporales que tiene el proyecto.

En el cuarto capítulo se detallan las bases teóricas sobre las que está basado el software a desarrollar y se explica su funcionamiento.

En el quinto capítulo se realiza un análisis de la solución, las partes por las que está compuesto, un diseño detallado y las tecnologías que han sido utilizadas, comparando con algunas alternativas y argumentando el uso de las seleccionadas.

En el sexto capítulo quedan descritas las pruebas realizadas en cada fase del proyecto y los resultados obtenidos al respecto. Se hace además un análisis de problemas encontrados a lo largo de la realización del proyecto y las soluciones que se han tomado para mitigarlos.

En el séptimo capítulo se realizan las conclusiones del proyecto, mencionando nuevamente los objetivos planteados y si se han logrado cumplir satisfactoriamente. También se comentan ampliaciones o partes que han podido quedar sin aplicar debido a limitaciones de distinto carácter, así como mejoras a realizar en un futuro para la

ampliación de este proyecto. Se detalla además la relación que posee este trabajo con lo aprendido en los estudios cursados.

Por último se encuentran las referencias utilizadas para la realización de esta memoria y de donde se ha obtenido principalmente la información.

Adicionalmente, se encuentra un anexo al final de la memoria donde se detallan los objetivos de desarrollo sostenible (ODS). Aquí se explica cómo puede usarse esta aplicación para cumplir objetivos en lo referente al entorno y sociedad.

2. Estado del arte

Desgraciadamente, no existe una aplicación similar a la nuestra con la que poder comparar resultados ni ampliar sobre ella. Estamos tratando un problema considerablemente específico donde no se han realizado desarrollos de software anteriormente para satisfacer la solución a nuestro problema. Pero, aun así, podemos dividir los conceptos de la aplicación en varias partes y explorar cada uno de ellos por separado, observando cuál es el estado de las tecnologías actuales.

2.1 Detección de objetos

Actualmente hay muchos algoritmos distintos enfocados en la detección de objetos. No es un campo novedoso, de hecho lleva más de 20 años evolucionando, por lo que tenemos técnicas más refinadas, precisas y rápidas, que además han mejorado drásticamente desde la introducción del *Deep Learning*. No es un problema sencillo de solucionar, ya que no se puede resolver sin una gran cantidad de trabajo y hay que enfrentar múltiples problemas. Por ejemplo, un objeto puede ser visualizado desde muchos ángulos, posiciones y puede ocupar tamaños distintos según la imagen, lo que requiere una gran cantidad de fotografías para entrenar el modelo.

No hay un “mejor algoritmo” como tal, sino que la elección del algoritmo a usar depende mucho del caso de uso, donde en general realizan dos tareas. Una primera tarea en la que se intenta encontrar un número arbitrario de objetos (posiblemente incluso cero) y otra tarea seguidamente que consiste en clasificar cada objeto y rodearlo con una *Bounding Box* para determinar su posición en la imagen y tamaño.

Algunos detectores realizan el proceso en los dos pasos descritos anteriormente y otros lo unifican en un único paso, sacrificando precisión a cambio de más rendimiento. Los detectores de dos pasos son comúnmente utilizados en aplicaciones que requieren una precisión muy alta, y los de un paso se usan por lo general en aplicaciones de detección de objetos en fotos o vídeos a tiempo real, o en casos donde el rendimiento es un factor determinante.

En primer lugar, tenemos a *R-CNN* [4], que luego se fue desarrollando en una familia de algoritmos. Esta técnica surge en 2014 y funciona dividiendo la imagen en zonas de interés, a través de un algoritmo especializado (normalmente búsqueda selectiva) que se encarga de agrupar píxeles en regiones con otros píxeles similares. Después, R-CNN toma estas regiones, las escala a un tamaño adecuado y las pasa por una red clasificadora, que devuelve la clase a la que pertenece el objeto, por ejemplo: coche, persona, pájaro, bicicleta, etcétera. Por último, se le puede aplicar opcionalmente a cada una de estas regiones un algoritmo de regresión de *Bounding Boxes* [5] para mejorar la estimación de la posición y tamaño del objeto. El problema con este modelo es que, debido al algoritmo de búsqueda selectiva, se proponen muchas regiones por imagen. Además, este algoritmo se ejecuta en CPU, haciendo que el tiempo de inferencia aumente, y como todas las regiones de interés tienen que ser escaladas y suministradas

al clasificador se pierde una cantidad considerable de tiempo. La figura 1 muestra una representación gráfica de este proceso.

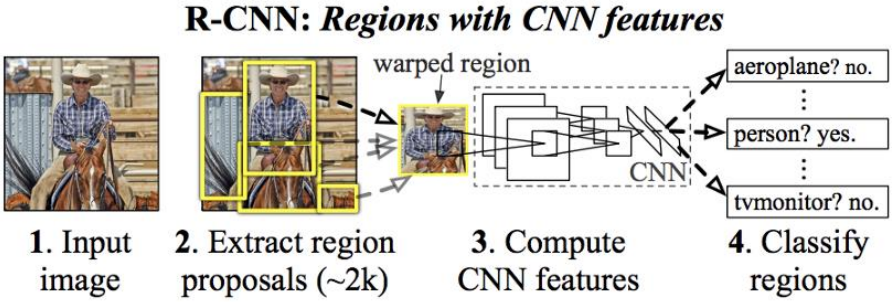

Figura 1: Arquitectura de R-CNN. Fuente: [4]

Una aproximación del mismo autor es Fast R-CNN [6]. Surge un año más tarde y es 9 veces más rápido entrenándose y 213 veces más rápido analizando una imagen; y además de esto, logra una precisión muy similar al modelo anterior. Este nuevo modelo es un detector de objetos de un paso, a pesar de estar basado en uno de dos pasos, pues en Fast R-CNN se unifican las dos fases. En vez de coger todas las regiones de interés y clasificarlas individualmente, se alimenta toda la imagen a un extractor de características que devolverá una matriz de características, que será compartida por todas las regiones de interés, lo que tiene sentido ya que muchas de ellas se solaparán. Cada región se pasa por una capa llamada “RoI pooling layer”, que ajusta al tamaño adecuado las regiones; tras esto se mandan al clasificador y se clasifican, obteniendo la etiqueta del objeto y su *Bounding Box*. Esto es una mejora substancial a R-CNN, ya que no hace falta clasificar cada imagen individualmente (lo que ralentiza mucho el proceso), pero no está libre de inconvenientes, ya que también usa búsqueda selectiva para las regiones de interés, por lo que se consume mucho tiempo en esta parte.

La última aproximación a esta familia de algoritmos es Faster R-CNN [7], un detector de objetos de dos pasos nuevamente, donde se decide eliminar la búsqueda selectiva, que era el principal problema de Fast R-CNN ya que penalizaba el rendimiento. En este nuevo algoritmo no se usa búsqueda selectiva para encontrar regiones de interés, sino que, en la primera fase, usando una red convolucional extra conocida como “regional proposal network”, se le alimenta la matriz de características y devuelve las regiones de interés, las que son alimentadas a la capa “RoI pooling layer” para ajustar su tamaño, y de nuevo se pasan por un clasificador, devolviendo así la clase y la *Bounding Box* asociada. A pesar de ser unas diez veces más rápido que su predecesor Fast R-CNN en test, sigue teniendo inconvenientes. En primer lugar, el algoritmo no mira la imagen completa de una vez, sino que se enfoca secuencialmente en distintas partes de la imagen, lo que hace que requiera varias pasadas para extraer todos los objetos. Además de esto, como hay varios sistemas dependientes entre sí, los sistemas finales dependerán del rendimiento de los anteriores.

Cabe también destacar una modificación reciente de este algoritmo, conocida como G-RCNN [8], que funciona de forma similar a Faster R-CNN, pero, a diferencia de los

anteriores algoritmos, toma como entrada vídeo en vez de una imagen, y trabaja con datos espacio-temporales de los diferentes fotogramas para poder granular y extraer mejor las regiones de interés. Este algoritmo es el más rápido de esta familia de algoritmos y demuestra mayor precisión al disponer de datos espacio-temporales, pero aún no está ampliamente implementado en aplicaciones modernas.

Otro algoritmo muy popular de detección de objetos es SSD (*Single Shot Detector*) [9]. Es muy utilizado para aplicaciones de detección de objetos a tiempo real debido a su alto rendimiento. Es un detector de objetos de un único paso que primero divide la imagen en celdas y utiliza una red neuronal llamada VGG16 [10], con la que extrae las características de la foto haciendo predicciones por cada celda de la imagen. Tras esto, detecta los objetos utilizando una capa llamada "Conv4_3", donde cada predicción consta de una *Bounding Box* y N+1 puntuaciones distintas, donde N es el número de clases distintas que es capaz de detectar y se le suma una puntuación más que simboliza que no haya ningún objeto. SSD no utiliza una red especializada para obtener las regiones de interés, sino que utiliza un método muy simple. Este método se basa en aplicar filtros convolucionales a cada celda, que son unos filtros utilizados en capas convolucionales para extraer características específicas de los datos de entrada, ya que los datos son fundamentalmente matrices. Esto da como salida la clase y *Bounding Box* de cada objeto. Debido a su aproximación usando celdas, tiene un rendimiento peor que Faster R-CNN, pero funciona muy bien para objetos grandes.

Por último, pero no menos importante, se encuentra YOLO (*You Only Look Once*) [11], que es otro de los modelos más utilizados actualmente debido a su gran rendimiento, pues es fácilmente capaz de procesar vídeos a tiempo real cuando es ejecutado en una GPU lo suficientemente potente. Este algoritmo de un único paso funciona dividiendo la imagen en N cuadrículas, todas del mismo tamaño; cada casilla es responsable de la detección y localización del objeto que contiene. Correspondientemente, estas casillas predicen B coordenadas de *Bounding Box* relativas a sus coordenadas de casilla, además de la etiqueta del objeto y la probabilidad de que esté presente en esa casilla. Después de obtener las predicciones, se eliminan las *Bounding Boxes* por debajo de cierta probabilidad; a las restantes se les aplica un proceso de *non-max supression*, que se encarga de eliminar los objetos de una misma clase que estén solapados, pero no tengan el valor máximo entre las *Bounding Boxes*. Actualmente, YOLO ha tenido varias iteraciones que han mejorado su rendimiento y su precisión. YOLOv5 es la última versión disponible, pero YOLOv4 tiene mucho más soporte y facilidad de integración.

2.2 Clasificación de imágenes

La clasificación de imagen consiste en analizar una imagen e identificar la clase a la que pertenece, o al menos la probabilidad que tiene de ello. Una clase es simplemente una categoría de objeto (coche, avión, hamburguesa, gato, etcétera). Para los humanos es un proceso sencillo, de segunda naturaleza, pero las máquinas requieren procesos más complejos para lograr una tasa de acierto similar a los humanos.

Actualmente la norma es utilizar aprendizaje profundo para lograr buenos resultados. Aprendizaje profundo es un tipo de *machine learning* que utiliza sistemas llamados redes neuronales profundas, con las que se pueden construir modelos que dada una imagen detectarán su clase.

Uno de los modelos más actuales y también más efectivos es Florence [12], desarrollado por Microsoft, que permite obtener una precisión muy alta, siendo de los más precisos que existen. Concretamente es capaz de tener 99,02% de precisión top-5 en clasificación de imagen con el conjunto de datos ImageNet [13], es decir, que uno de los 5 resultados con más probabilidad es la clase correcta para la imagen dada. Esto coloca a este modelo como el clasificador con más precisión top-5 de todo ImageNet. Ha sido entrenado precisamente con este conjunto de datos, pero tiene una ayuda extra, y es que debido a los grandes recursos de Microsoft también se ha usado para su entrenamiento 900 millones de pares de imagen-texto, lo que mejora mucho su rendimiento. Este modelo no solo es útil para clasificación de imágenes, ya que puede ser adaptado a muchos otros escenarios, como detección de objetos, pruebas de validación visual, descripción de imágenes e incluso reconocimiento de acciones en la imagen. Es un modelo titánico que no sería posible entrenar por un usuario promedio, ni siquiera por una empresa media o pequeña debido a la cantidad masiva de datos que se han usado para ello.

Un modelo también muy preciso pero que sólo hace uso de ImageNet para su entrenamiento es EfficientNet [14], cuya mejor versión actual en términos de precisión es EfficientNet-L2 [15], que cuenta con varias modificaciones, siendo la más precisa en ImageNet “Meta Pseudo Labels” [16]. Esta versión presenta un método de aprendizaje semi-supervisado, donde hay una red “profesor” que genera pseudo-etiquetas en datos no etiquetados para enseñar a una red “estudiante”, donde el profesor va adaptando las etiquetas que produce según el rendimiento en clasificación del estudiante. Esta versión del modelo es la segunda mejor en precisión top-5, detrás del modelo mencionado anteriormente de Microsoft. “Meta Pseudo Labels” consigue una precisión top-5 de 98,8%, e incluso supera al modelo Florence en precisión top-1, con un 90,2%.

Un último modelo a destacar, que puede no ser tan preciso como los otros, pero es de interés debido al gran rendimiento que tiene tanto en facilidad y rapidez de entrenamiento como a la hora del tiempo utilizado en clasificar una imagen, es MobileNet [17]. Desarrollado por Google específicamente para dispositivos móviles, posee muy baja latencia y reduce el número de parámetros de la red considerablemente comparándolo con otros modelos basados en redes neuronales convolucionales. En su versión más reciente, MobileNetV3 logra unos resultados muy decentes, teniendo un 94,5% de precisión top-5 en el conjunto de datos ImageNet, y todo eso con una gran facilidad de entrenamiento y rapidez de clasificación. Posee casi una centésima parte de los parámetros que tiene “Meta Pseudo Labels” y es, a día de hoy, uno de los modelos más utilizados para clasificar directamente en dispositivos menos potentes.

2.3 Google Lens

Probablemente la aplicación que más se puede asemejar a nuestro proyecto es una que se popularizó hace pocos años, y es Google Lens. Presentada en 2017, esta aplicación desarrollada por Google está disponible para sistemas Android, donde viene incluida en la gran mayoría de móviles actuales dentro de la propia aplicación de la cámara. También se ha lanzado una versión para dispositivos móviles iOS. Incluso se ha integrado parcialmente en las últimas versiones del navegador Google Chrome para dispositivos de escritorio [17].

Esta aplicación funciona con detección de objetos y clasificación de objetos, entre otras tecnologías, usando principalmente redes neuronales convolucionales (CNN). Posee una miríada de funciones. La más utilizada es poder detectar objetos en la cámara del dispositivo y proporcionar información sobre ellos, siendo capaz de clasificar el tipo de objeto y ofrecer productos similares, encontrar otros objetos parecidos, recomendaciones o simplemente información adicional. Esto es aplicable también a la detección de texto escrito, donde es capaz de localizar el texto, procesarlo y convertirlo a un formato más legible. Puede además traducirlo desde casi cualquier idioma en tiempo real solo con enfocarlo con la cámara.

No hay información pública de los algoritmos que utiliza ni exactamente cómo está construida la aplicación, pero es capaz de transformar lo que la cámara ve en una barra de búsqueda visual. Posee una gran precisión y a fecha de 2018 era capaz de distinguir más de mil millones de artículos diferentes sumando los reconocidos con códigos de barras y visión por computador [18], cuatro veces más que en su lanzamiento un año antes. En la figura 2 se puede observar la interfaz de esta aplicación, que es capaz de identificar en tiempo real una lámpara y sugerir otras similares.

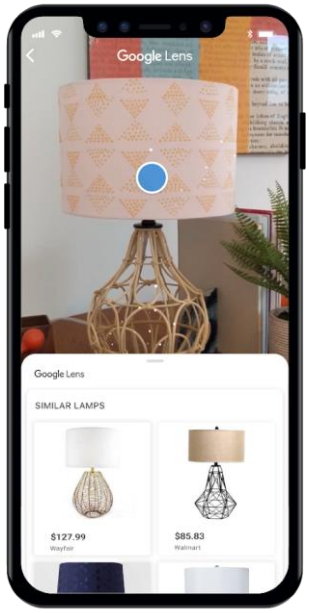

Figura 2: Interfaz de Google Lens. Fuente: [19]

2.4 Propuesta

Después de haber podido observar cuáles son las principales aplicaciones y técnicas utilizadas actualmente, hemos concluido que no existe ninguna aproximación similar al problema que estamos intentando solucionar, ya que las tecnologías empleadas son demasiado genéricas. Por tanto, necesitaremos mezclar varias técnicas y tecnologías para poder alcanzar el objetivo planteado.

La aplicación a desarrollar en principio es similar en cierto modo a Google Lens, pero reducido a un ámbito más definido; concretamente, un número reducido de especies de aves. De manera que aplicaremos técnicas de visión por computador, donde en primer lugar necesitaremos un algoritmo detector de objetos. Este algoritmo será de un único paso, ya que el rendimiento de la aplicación es importante, pues no se debe permitir que un pájaro excluido por el usuario esté por un periodo extendido de tiempo delante de la cámara. Luego también se integrará un modelo de clasificación de imágenes, donde de nuevo el rendimiento es importante, pero al trabajar con un conjunto de datos reducido no debería suponer un gran problema la precisión.

Se diferencia de las aplicaciones que circulan actualmente en que tiene un propósito más específico y combinará varias técnicas y modelos del estado del arte actual para formar una aplicación capaz de reconocer, clasificar y actuar sobre el mundo exterior.

3. Análisis del problema

3.1 Identificación y análisis de soluciones posibles

Para la creación de esta aplicación se pueden tomar varias aproximaciones. Por un lado, existe la opción de crear una aplicación Android donde se tome una foto y, mediante un modelo de detección de objetos y otro de clasificación de imágenes, se analice si existe un pájaro o no, y de existir, identificarlo para luego espantarlo o no según la selección del usuario. Esta opción por un lado presenta ciertas ventajas, y es que cada dispositivo trabajaría localmente; por tanto, no habría ninguna necesidad de tener que utilizar datos móviles o disponer de una conexión inalámbrica a la que conectarse. También permitiría por este mismo motivo utilizar la aplicación en entornos más remotos o menos urbanos, como parques o incluso fuera de la urbe, como montañas u otros lugares alejados, todo sin tener que agotar un plan de datos móviles del que disponga el usuario. Pero, por otro lado, esta solución puede tener complicaciones, y es que no todo el mundo dispone de un dispositivo capaz de procesar imágenes, detectar objetos, clasificar el pájaro y enviar una señal para espantarlo, ya que muchos móviles son antiguos, están ralentizados por una gran cantidad de aplicaciones funcionando simultáneamente o no disponen de la batería suficiente como para durar un tiempo útil realizando las tareas de la aplicación. Y aunque la idea sea usarlo en entornos urbanos, no todos los usuarios serán capaces de tener el dispositivo móvil conectado a una toma de corriente o a una batería portátil para que la batería tenga una duración suficiente.

La otra alternativa es crear una arquitectura cliente servidor, donde la aplicación tome una fotografía con la cámara trasera del dispositivo, la procese y envíe a un servidor capaz de ejecutar el modelo de detección de objetos y el clasificador, el cual devolvería una señal o código indicando el tipo de pájaro encontrado o si no se ha encontrado ninguno, siendo el dispositivo móvil el encargado de recibir y determinar si esta señal corresponde con una especie permitida o no. Esta opción permite que dispositivos más antiguos o modestos sean capaces de ejecutar la aplicación sin restricciones, solo necesitando como requisito una cámara trasera y una conexión a internet relativamente estable, ya que se puede hacer uso de un protocolo de red que tenga en cuenta las pérdidas de información en envío.

Los problemas que pueden surgir con esta aproximación son la necesidad de una conexión a internet, que no siempre puede ser garantizada en entornos más rurales (esta aplicación está diseñada para entornos urbanos, pero siempre se agradece ampliar su funcionamiento), o también sería problemático para personas que no tengan internet o datos móviles, lo que es una rara ocurrencia hoy en día. El segundo problema que puede darse es la saturación por parte del servidor, ya que si hay muchos dispositivos cliente ejecutando la aplicación y mandando continuamente peticiones al servidor para que analice las imágenes, éste se puede saturar y no ser capaz de responder en un tiempo adecuado. También puede haber problemas de disponibilidad según dónde se ejecute el servidor, ya que para que fuera efectiva esta opción debería

ejecutarse en un ordenador que esté siempre encendido, para conseguir una disponibilidad lo más alta posible. Habría también que tener cuidado con mensajes provenientes de otras fuentes al servidor, ya que habría que identificar que lo recibido es una imagen; en caso contrario podría haber problemas y lanzar errores, acabando la ejecución del servidor.

Otra duda que surge a la hora de desarrollar la aplicación es si usar un algoritmo detector de objetos que tome como entrada la imagen original y luego entrenar un clasificador de imágenes que reciba el pájaro recortado detectado en la imagen original para clasificarlo, o si en cambio es mejor entrenar directamente un clasificador de objetos con muestras de pájaros directamente y que los distinga desde el primer momento.

Por un lado, la aproximación de solo entrenar un detector de objetos, incluso tomar uno ya entrenado y mediante *transfer learning* [20] usar nuestro conjunto de datos, es muy buena, ya que ahorramos tiempo y no tenemos que interconectar dos modelos distintos y compatibilizar sus entradas y salidas, al solo tener que preocuparnos por un modelo. Además, puede resultar más rápido en ejecución que un sistema más complejo. El problema que puede presentar es que, al reentrenar un sistema de detección de objetos sin demasiados datos, pueden presentarse complicaciones a la hora de detectar al pájaro en ambientes no controlados, es decir, imágenes donde el pájaro esté entre muchos objetos u ocluido por el entorno.

La arquitectura de utilizar un detector de objetos y un clasificador puede resultar mucho más complicada de implementar, pero los clasificadores de imágenes funcionan muy bien con *transfer learning* y para un conjunto de datos pequeño como es nuestro caso no necesitan una cantidad desmesurada de datos para poder reentrenarse. Habría que tener especial cuidado y dedicar tiempo a comunicar los dos modelos, y el rendimiento podría sufrir al tener que ejecutar dos algoritmos distintos para una única imagen. En cambio, no habría que preocuparse por no detectar los pájaros, ya que los modelos pre-entrenados han aprendido de imágenes donde no se ven claramente, o al menos no en un primer plano, a los pájaros, y por tanto puede realizar una mejor detección que un detector de objetos entrenado con nuestros datos que son mucho más limitados en comparación.

3.2 Solución propuesta

Entre las opciones presentadas anteriormente, se ha decidido tomar la aproximación de tener una arquitectura cliente-servidor. Esto es así ya que, como se ha mencionado anteriormente, muchos dispositivos móviles no son capaces de hacer la detección y clasificación de una fotografía en pocos segundos, además de que necesitan una cantidad de memoria RAM más elevada si estos modelos se están ejecutando en el propio dispositivo en vez de mandar simplemente la imagen. Además, al ser una aplicación que estará centrada en entornos urbanos y las especies de ave a reconocer son también urbanas es lógico que se disponga de una conexión a internet relativamente estable con la que poder mandar la imagen al servidor. La disponibilidad del servidor

puede maximizarse haciendo uso de servicios *cloud* y haciendo que máquinas remotas ejecuten los algoritmos continuamente. El servidor puede replicarse o utilizar uno con soporte para GPU, con lo que se ejecutaría mucho más rápido y no habría problema al recibir una gran cantidad de peticiones de cliente. Con respecto a poder recibir alguna comunicación errónea, se puede usar TCP o algún protocolo similar para evitar estos problemas, y además se tratará la entrada por si acaso hay algún mensaje que no corresponde. Con esto obtendríamos el beneficio de que cualquier usuario con un dispositivo incluso básico sea capaz de hacer uso de nuestra aplicación.

Por otra parte, hemos tomado la decisión de utilizar tanto un detector de objetos como un clasificador de imágenes. Esto requiere más trabajo y posiblemente mayor tiempo de ejecución, pero no debería ser una cantidad muy elevada como para marcar una diferencia. Además, al usar un detector de objetos ya entrenado con una gran cantidad de imágenes no se corre el riesgo de no detectar un pájaro por falta de datos de entrenamiento.

Para implantar esta solución habrá que empezar recolectando todas las imágenes posibles de los distintos tipos de pájaros urbanos que deseamos distinguir, y posteriormente recortar estas imágenes a un primer plano del ave para poder utilizar en el entrenamiento del clasificador. Cuando esto esté listo, habrá que decidir qué modelos de detección de objetos y clasificación usar, y entrenar el clasificador y encontrar los parámetros más adecuados para nuestro uso. Habrá que comprobar el correcto funcionamiento y unir ambas partes, realizando además pruebas con otras imágenes distintas. Posteriormente, se procederá a unificar ambos modelos en un servidor que tome imágenes y devuelva un código asociado al pájaro, pudiendo realizar pruebas con un programa cliente. Luego se comenzará el desarrollo de la aplicación Android que tomará una foto y se comunicará con el servidor creado anteriormente, pudiendo ya realizar todas las pruebas pertinentes para el correcto funcionamiento del sistema. Por último, deberá encontrarse un servicio *cloud* adecuado al que subir el servidor y ejecutarlo remotamente, y realizar las pruebas finales.

3.3 Plan de trabajo

Para calcular las horas dedicadas a cada parte se usará horas-persona como medida de tiempo usado en cada fase del proyecto.

A la primera parte de recolectar imágenes correspondientes a las especies de aves se le asignan 15 horas-persona, ya que es una tarea que requiere muchas imágenes distintas, y luego recortar una por una cada imagen entre los cientos de los que se habrán obtenido.

La elección de modelos adecuados y entrenamiento del clasificador es otra tarea que podría tener en torno a 35 horas-persona de dedicación. Esto incluye investigar sobre los modelos que pueden ser más útiles para el proyecto, implementar su uso y entrenar el clasificador con el ajuste de parámetros correspondiente, incluyendo también las pruebas unitarias.

La tercera parte sería unir la salida del detector de objetos y la entrada del clasificador, y realizar las pruebas de integración entre estas partes. A esta fase se le dedican 10 horas-persona.

Se continuaría con la creación del servidor, elección del protocolo de comunicación y pruebas de integración. Se le dedican 5 horas-persona.

El desarrollo de la aplicación Android, su conexión con el servidor y pruebas unitarias y de integración son un proceso complejo y que llevará aproximadamente 60 horas-persona. En parte es debido a que no disponemos de experiencia y habría que realizar además un proceso de aprendizaje y arreglar más errores de lo normal, causados por la inexperiencia.

En último lugar habría que investigar qué opciones existen para ejecutar servidores *online* en máquinas virtuales y configurar y poner en funcionamiento una de estas máquinas virtuales. A esta tarea se le dedican 10 horas-persona.

3.4 Presupuesto

Para el cálculo del presupuesto hay que preguntarse cuál es el valor por hora de alguien que realiza estas tareas. Para ello podemos comparar con un salario de programador o desarrollador software junior, es decir, alguien que está empezando a trabajar como programador y aún no tiene un dominio considerable sobre las tecnologías que utiliza. Según páginas como *talent.com* [21] o *jobted.es* [22], el salario promedio de un programador junior es de unos 20.000€ brutos al año, lo que traducido a euros por hora es aproximadamente 11,5 euros por hora. Usaremos esta cantidad para los cálculos de cuánto vale el trabajo desarrollado.

Podemos ignorar costes de electricidad o de tener una GPU para poder entrenar correctamente el clasificador, ya que se puede realizar en Google Colab, una plataforma de ejecución en línea de libretas de código que permite ahorrarnos estos gastos, ya que permite ejecución gratuita de *notebook*s de código Python haciendo uso de GPU o incluso de unidades de procesamiento tensorial, lo que puede acelerar el entrenamiento de algunos modelos. Desgraciadamente, Google Colab tiene una limitación temporal en la ejecución del *notebook*, por lo que si deseamos hacer uso de esta herramienta no nos será posible entrenar un modelo con muchos parámetros, ya que no se podría completar este proceso. Este límite no está detallado explícitamente, debido a que depende de la carga de las máquinas que Google proporciona. De querer tener más tiempo de ejecución se puede pagar una suscripción mensual de 9,25€ [23], existiendo además otra mensualidad superior de 42,25€ mensuales que proporciona aún más tiempo de ejecución y la posibilidad de ejecutar código con el navegador cerrado de forma autónoma. Pero debido a que buscamos un modelo que sea ligero y con pocos parámetros, no harán falta las versiones de pago de este servicio.

En total, en el apartado 3.3 se estima una duración aproximada para el proyecto de 135 horas-persona, con lo que, aplicando la tasa de programador junior, se obtiene que solo en horas dedicadas se corresponde a 1552,5 euros.

Lo descrito anteriormente es solo la parte de esfuerzo horas-persona. Al ser un proyecto que posiblemente se desarrolle a largo plazo también habría que considerar cual sería el coste por mantener esta aplicación. Y es que si se va a ejecutar continuamente en un servidor hay que poder conocer el precio que se debe pagar por este servicio. Los cálculos se harán de forma mensual. Para una máquina virtual que pueda satisfacer nuestras necesidades en Amazon EC2, que es el principal proveedor de estos servicios, nos puede costar alrededor de 0,02 euros la hora [24] para una máquina virtual *t3.small*, que no dispondría de tarjeta gráfica para acelerar el cálculo de los modelos de detección y clasificación. Esto calculado mensualmente, suponiendo que dejamos en ejecución continuamente el servidor tal y como está planeado, llevaría a 14,4 euros mensuales.

Sin embargo, si de cara al futuro se desea utilizar una GPU para evitar problemas como saturación del servidor por muchas peticiones de distintos clientes, una máquina virtual también en Amazon EC2, costaría 50 céntimos de euro cada hora [25] por un modelo con una GPU AMD Radeon Pro V520, lo que supondría 360 euros al mes. Aunque si se llegase al punto de expandir hasta este nivel, se podría reservar durante más tiempo para obtener un descuento; si se reservase la máquina virtual durante 3 años se reduciría el gasto a 20 céntimos de euro la hora, lo que resultaría en 144 euros al mes.

4. Fundamentos teóricos

4.1 Redes neuronales

Las redes neuronales son comúnmente utilizadas en muchas técnicas de inteligencia artificial y aprendizaje automático, incluyendo detección de objetos y clasificación de imágenes [26]. Simulan el sistema nervioso humano y cómo procesa la información.

La unidad básica de una red neuronal es la neurona, también conocida como nodo, que normalmente se organizan en capas. Diferentes capas pueden realizar diferentes transformaciones en sus entradas. Existe una capa de entrada, cero o más capas ocultas y una capa de salida. Los datos que se desean procesar se insertarán en la capa de entrada, irán siendo procesados a lo largo de las siguientes capas ocultas y se obtendrá un resultado en la capa de salida. Es de notar que cuantas más capas ocultas posea el modelo más sofisticado será, pero esto puede conllevar un coste temporal y computacional mayor para entrenar y procesar datos con esta red. Actualmente se trabaja principalmente con redes neuronales profundas; aunque la definición aún genera discusión, se suele referir a estas redes como aquellas que tienen una capa de entrada, una o más capas ocultas y una de salida [26]. A día de hoy se suele trabajar con docenas o incluso cientos de capas ocultas.

Cada neurona o nodo se conecta con una o varias otras neuronas tanto a la entrada como a la salida, usualmente siendo estas conexiones entre distintas capas de la red. Cada una de estas conexiones tiene un peso asociado. El peso incrementa o decrementa la importancia de los datos enviados por una neurona a otra. Para decidir el valor de salida de una neurona se usan funciones de activación, que son funciones matemáticas encargadas de devolver un valor de salida generado por la neurona dada una entrada o conjunto de ellas. Cada capa de la red neuronal tiene una función de activación. Hay muchos tipos distintos de funciones de activación, utilizando cada una según qué tan rápida se desea que sea la convergencia, el rendimiento, o si se quiere acotar o no el valor de salida, entre otros factores.

Las redes neuronales dependen de datos de entrenamiento para poder mejorar su precisión; por tanto, van modificando sus pesos según los datos de entrenamiento con los que sean alimentadas, por lo que ejecuciones subsecuentes obtendrán mejores resultados. Para entrenar la red neuronal se van modificando los pesos de forma que si un nodo contribuye a la respuesta correcta su peso será más alto y si falla no lo consideraremos tan importante y se disminuirá su peso.

Algunos tipos de redes neuronales son:

- **Redes neuronales *feed-forward*:** Una de las variantes más simples de las redes; pasan información en un único sentido, es decir, los datos entran por la capa de entrada y salen por la de salida, pasando en orden por las capas ocultas intermedias si hay alguna. En la figura 3 se observa un esquema típico

de las conexiones entre las neuronas de las distintas capas de una red *feed-forward*.

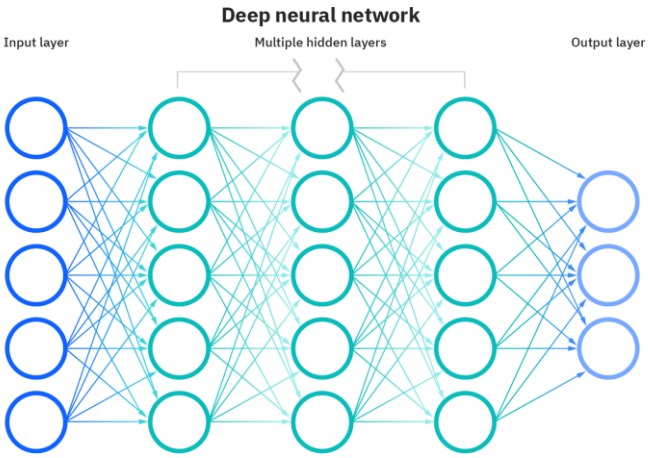

Figura 3: Estructura de una red neuronal feed-forward

- **Redes neuronales modulares:** Es una estructura formada por varias redes neuronales que trabajan sin comunicarse entre ellas; se divide la tarea a realizar entre las distintas redes y al final se combinan sus salidas para obtener un único resultado.

- **Redes neuronales convolucionales:** Este tipo de redes neuronales es similar a las redes *feed-forward*, pero tienen al menos una capa o bloque de convolución. Son muy utilizadas en el campo de visión por computador, reconocimiento de patrones y reconocimiento de imágenes. Se define por tener capas convolucionales. Estas redes son afines al sistema de córtex visual humano. Debido a su importancia en las tareas de visión por computador y reconocimiento de objetos, las cuales forman una gran parte del presente proyecto, resultan cruciales para el desarrollo del mismo.

4.1.1 Redes neuronales convolucionales

Este tipo de redes neuronales tiene un gran funcionamiento a la hora de capturar información local, como los píxeles vecinos en una imagen o palabras adyacentes en un texto. Sumado a su capacidad de reducir la complejidad del modelo al permitir un entrenamiento más rápido y con menos muestras, convierte a este tipo de redes en la herramienta predilecta para la mayoría de tareas de visión por computador.

Para poder utilizar imágenes con estas redes hay primero que tenerlas en un formato adecuado, con el que se pueda trabajar de forma sencilla. Para ello dividimos la imagen en canales, donde cada canal representa la imagen en escala de grises de su color asociado, que será uno de los colores primarios que componen la imagen. Es común utilizar un canal para rojo, verde y azul, ya que estos tres colores juntos pueden crear el resto de combinaciones de colores. Cada uno de estos canales se representa de

manera matricial, de forma que hay un valor en la matriz por cada píxel en la imagen, y este valor indica la intensidad del color representado por el canal en el píxel dado. En algunas ocasiones también se hace uso de un canal extra, llamado canal alfa, que indica el grado de transparencia u opacidad de la fotografía. Es utilizado para determinar cómo un píxel se modifica al mezclarlo con otro.

El funcionamiento de estas redes para tratar con imágenes, que es donde concierne a este proyecto, consta de dos tipos de bloques principales: el bloque convolucional, donde se obtienen las características relevantes de la imagen, y el bloque de clasificación, donde se discrimina entre las distintas salidas posibles. Puede haber más de un bloque de convolución.

La convolución es una operación que consiste en ir deslizando unas matrices, también conocidas como filtros o *kernels*, sobre la imagen de entrada. Para aplicar estos filtros, se van deslizando en orden por la matriz de entrada de la imagen y multiplicando cada valor del filtro con el valor correspondiente de la matriz y sumándolos; tras esto se desliza el filtro una cantidad definida, llamada *stride*, repitiendo así el proceso. El número de *kernels* aplicados al mismo tiempo es conocido como características (*features*); por ejemplo, si se fueran a aplicar cuatro *kernels* a la imagen, se tendrían cuatro características. Los valores de estos *kernel* se inicializan aleatoriamente por lo general, y se van actualizando durante el entrenamiento de la red mediante descenso por gradiente.

Una característica que hace que estas redes resulten más eficientes es que son capaces de compartir los pesos de los filtros, ya que éstos recorren toda la imagen, haciendo que las características de la imagen sean invariantes a la posición. Se suelen utilizar *kernels* de un tamaño pequeño, ya que, cuanto más grandes, más aumenta el coste computacional de las operaciones a realizar, ya que hay que hacer multiplicaciones matriciales más complejas.

Para reducir el coste computacional aún más se utiliza un proceso llamado submuestreo o *pooling*. Se basa en reducir las dimensiones de una matriz mientras se intenta mantener su valor. Es útil para añadir robustez ante distorsiones como cambio de tamaño, rotaciones o desplazamientos. Existen varios tipos de *pooling*, pero los más usados son *Max-pooling*, donde se mantiene el valor máximo de cada sección, y *Average-pooling*, en el que se hace una media de los valores de cada sección para la nueva matriz reducida.

En cada capa convolucional, la salida es la entrada de la siguiente capa, y a la salida de la última capa convolucional se debe pasar el resultado al bloque de clasificación, que tiene una entrada unidimensional. Esto implica realizar una operación conocida como aplanamiento, en la que se recorre cada salida y se ponen sus valores en el mismo orden en un vector, lo que hace que se pierda el sentido espacial.

El bloque de clasificación es una red completamente conectada, lo que significa que consta de una serie de capas que conectan cada neurona de una capa con todas las de la siguiente. La capa de entrada de este bloque tiene un tamaño igual al vector obtenido del aplanamiento, y la capa de salida es de tamaño igual al número de clases de las que

se disponen, siendo las capas intermedias de un tamaño normalmente decreciente. La última capa usa la función de activación *softmax*, lo que crea una distribución probabilística, donde cada neurona de la última capa representa una clase. Con esto podemos tomar la neurona que haya producido como salida el mayor valor y asignarle a la imagen la clase correspondiente a esa neurona.

4.2 Detección de objetos

La detección de objetos es una técnica de visión por computador que consiste en localizar e identificar objetos en una imagen o vídeo, dando como resultado una etiqueta que indica qué tipo de objeto es, además de su localización y dimensiones. Esta localización y dimensiones vienen dada por la *Bounding Box*, nombre por el que es conocido el conjunto de coordenadas que delimitan al objeto dentro de la imagen. A diferencia de la clasificación de imágenes no tiene por qué haber un único objeto en la imagen, ya que puede haber una gran cantidad de ellos.

Esta técnica es de gran interés, ya que puede ser utilizada en muchos campos distintos como vehículos autónomos, donde necesitan ser capaces de identificar los objetos de su alrededor para poder circular de forma eficiente y segura. También en detección de caras, o incluso en videovigilancia.

En general se pueden utilizar muchas características de la imagen para detectar objetos, como agrupar píxeles de colores similares y comparar el contraste para delinear un objeto. También se puede identificar por *template matching*, una técnica que permite identificar objetos sin necesidad de entrenar un modelo previamente, que funciona teniendo diferentes plantillas de objetos que van deslizándose a lo largo de la imagen hasta coincidir con una parte de la misma, y entonces se sabe que se ha detectado ese objeto; esto es especialmente útil cuando se quiere trabajar con un conjunto reducido de objetos o no se quiere entrenar un modelo para la tarea a realizar. Otra forma es a través de las formas y contornos de elementos de la imagen, donde se pueden comparar con otras formas ya conocidas.

Las aproximaciones actuales a este problema están basadas en redes neuronales convolucionales para realizar la detección de objetos, y hay muchas formas de implementarlo tal y como se ha visto brevemente en el capítulo 2 sobre el estado del arte. Suelen ser más complejas que simplemente usar una única característica de la imagen, y varían mucho sus aproximaciones e implementaciones, pero por lo general se entrena previamente un modelo con una gran cantidad de imágenes; cada una de estas imágenes tiene como dato adicional de entrada los distintos objetos que hay y sus *Bounding Boxes*, con lo que el modelo va aprendiendo a reconocer mejor estos objetos. En la figura 4 se representa una imagen con los diferentes objetos que han sido detectados, cada uno con su *Bounding Box* asociada y su etiqueta de clase.

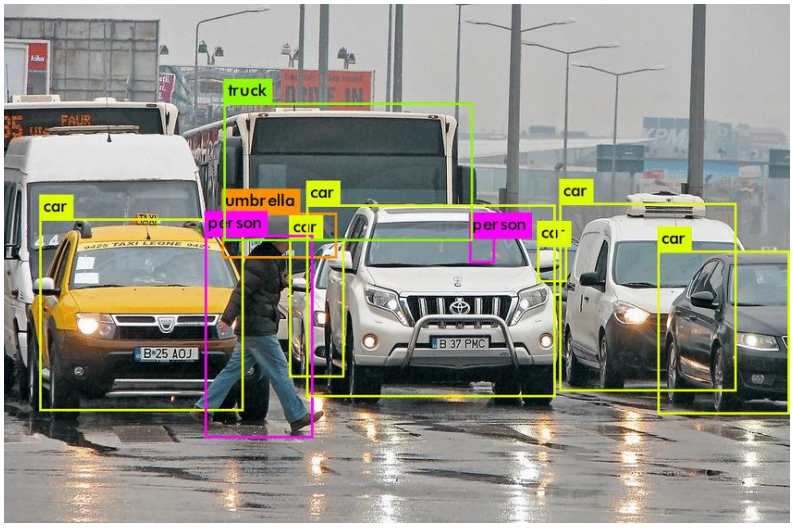

Figura 4: Ejemplo de Bounding Boxes con sus etiquetas asociadas. Fuente: [28]

Como ya fue mencionado en el análisis del estado del arte, los dos tipos de detectores de objetos son los de dos pasos y los de un paso. En los clasificadores de dos pasos se identifican una serie de regiones de interés dentro de la imagen; estas regiones son las candidatas a poder contener un objeto. En el segundo paso se trata de mirar todas y cada una de las regiones anteriores y clasificarlas, con lo que se sabrá si existe o no un objeto en ese espacio y, en caso de haberlo, conocer a qué tipo pertenece. Estos clasificadores son muy precisos, pero carecen de velocidad en comparación a los de un paso.

En los detectores de un paso todo este proceso es capaz de ser unificado en una única fase, donde se utilizan en muchas ocasiones *anchor boxes*, que son un conjunto de *Bounding Boxes* de un tamaño y dimensiones predeterminado. Cada *anchor box* es superpuesta en varias partes de la imagen o casillas y representan distintas clases de objeto, lo que permite hacer *k* predicciones por casilla, donde *k* es el número de *anchor boxes* a usar. Esto permite calcular la probabilidad para cada casilla y *anchor box* que intersecten con un objeto, y se utilizan predicciones para refinar cada una de ellas. Normalmente su tamaño y dimensiones se ajusta a los objetos en las imágenes empleadas para entrenar el modelo. Los detectores de este tipo son capaces de obtener resultados mucho más rápidos a costa de perder precisión en objetos más pequeños o alejados en una imagen.

4.3 Clasificación de imágenes

La clasificación de imágenes es una tarea donde se enfrenta el problema de reconocer a qué categoría o clase pertenece una imagen, en la que tenemos un conjunto definido de clases entre las que elegir. También existe la posibilidad de clasificación multi-clase, donde a una imagen le asignamos varias clases. Pertenece a la categoría de aprendizaje supervisado, es decir, que hace uso de datos etiquetados para poder aprender a clasificar correctamente. Cuando se hace una clasificación incorrecta el modelo puede saberlo gracias a la etiqueta proporcionada junto al dato de entrada.

Para poder utilizar un clasificador de imágenes se requieren varios pasos. En primer lugar, hay que recolectar una gran cantidad de imágenes de las clases que se desean reconocer en el clasificador. Esto se puede hacer de forma manual, recolectando imágenes y dándoles una etiqueta a cada una que representa la clase a la que pertenece, o con una base de imágenes ya creada. Uno de los ejemplos más populares de este último caso es ImageNet, que es un conjunto de datos que consta de más de un millón de imágenes con etiquetas que están divididas en mil clases distintas, con lo que se facilita la tarea de entrenar un clasificador más complejo.

Las imágenes se deben posteriormente dividir en tres conjuntos. El primero es un conjunto de entrenamiento, que serán las imágenes de las que se extraigan sus características y el clasificador aprenda de ellas. Existirá otro conjunto de validación, que será utilizado mientras se entrena el modelo para poder comprobar la precisión del clasificador obtenido. Por último se tendrá un conjunto de test, que será empleado para comprobar la precisión del clasificador final una vez se hayan ajustado los parámetros y se haya completado el entrenamiento, lo que nos permite conocer qué tan bien funciona el clasificador.

Muchas veces se desea entrenar con unas imágenes propias para resolver un problema específico, pero no se dispone de un número elevado de imágenes, o simplemente se quieren tener más. En estos casos, se puede utilizar una técnica conocida como *image augmentation [29]*, donde se generan más imágenes en base a las que ya se disponen. Esto se logra modificando las imágenes originales y aplicándole transformaciones tales como rotación, recortado o traslación. Existen además aproximaciones basadas en redes neuronales, llamadas redes generativas antagónicas [30], donde dos redes neuronales, una red generadora y otra discriminadora, compiten entre ellas. La red generadora crea imágenes con características similares a las originales mientras que la discriminadora se encarga de evaluarlas para comprobar su autenticidad.

Otra opción para resolver un problema específico es utilizar un clasificador pre-entrenado en vez de intentar hacerlo desde cero, ya que tomando un modelo que haya sido entrenado con millones de imágenes, resultará más fácil que pueda discernir entre las diferentes características de una imagen. La idea base es transferir tanto conocimiento como sea posible de un modelo existente a uno nuevo, aprovechando los aspectos generales de este clasificador y simplemente adaptándolo a una nueva tarea con las nuevas imágenes. Este proceso se suele conocer como *transfer learning* [20]. Uno de estos aspectos puede ser, por ejemplo, la capacidad de detectar los bordes de los objetos. Esto elimina el requisito de tener una cantidad enorme de imágenes de entrenamiento etiquetadas, disminuye el tiempo en entrenar el modelo y puede incluso proporcionar mejores resultados. Los parámetros del modelo necesitarán ser ajustados y refinados, pero la funcionalidad base ya estará implementada a través del *transfer learning*.

Si los resultados no son satisfactorios tras el entrenamiento con *transfer learning* o se desea obtener más precisión, se puede hacer un proceso llamado *fine tuning*, donde se añaden capas a la red neuronal o se modifican las ya existentes para “afinar” las

características del modelo base, convirtiendo el modelo en uno más apto para la tarea específica.

Para entrenar el clasificador, inicialmente se le da como entrada el conjunto de datos de entrenamiento con sus respectivas etiquetas. También se especifica el número de *epochs* deseados, esto es, el número de veces que se pasará por completo todo el conjunto de entrenamiento por la red. Además, añadimos como entrada el conjunto de validación, con el que al final de cada *epoch* se comprobará el rendimiento de la red utilizando estas imágenes, de forma que intentando clasificarlas se observa el número de errores y así se obtiene un porcentaje de aciertos. La meta final de un clasificador es reducir lo máximo posible la función de pérdida, que es una función que evalúa la desviación entre las predicciones realizadas y las etiquetas reales de las imágenes.

4.4 YOLO

You Only Look Once, también conocido como YOLO [11] es una arquitectura de detección de objetos de un paso. Detallada por primera vez en 2015 por Joseph Redmond, uno de los autores y contribuidores de este proyecto fue Ross Girshick, el desarrollador de la arquitectura R-CNN. La aproximación de YOLO hace uso de una única red neuronal entrenada para tomar una fotografía como entrada y predecir las *Bounding Boxes* y sus etiquetas directamente. Consta de 24 capas convolucionales, seguido de dos capas completamente conectadas. Esta arquitectura ofrece una menor precisión predictiva, pero a cambio es capaz de operar a gran velocidad. Cuando salió el modelo, detallaron que era capaz de ejecutarse a 45 fotogramas por segundo y con una versión optimizada para su ejecución hasta 155 fotogramas por segundo, convirtiéndola en una arquitectura capaz de trabajar con vídeos a tiempo real y realizar detecciones de objetos de forma muy veloz.

Este algoritmo está basado en regresión. En vez de seleccionar regiones de interés como algoritmos descritos anteriormente, lo que hace es predecir las clases y *Bounding Boxes* para toda la imagen en una única pasada del algoritmo. Cada *Bounding Box* puede ser descrita con las coordenadas de su centro, la altura y la anchura, además de un valor que representa la clase a la que pertenece el objeto. Junto a esto se predice un número real, que indica la probabilidad o confianza de que haya un objeto en la *Bounding Box*. En vez de buscar regiones de interés, divide la imagen en casillas, donde cada casilla es responsable de predecir un número *k* de *Bounding Boxes*. Se dice que un objeto está en una casilla específica si las coordenadas centrales de una *anchor box* pertenecen a esa casilla. Por este motivo, las coordenadas centrales se calculan siempre de forma relativa a la casilla mientras que la altura y anchura son relativas al tamaño de la imagen completa.

Con el conjunto de casillas, al predecir la clase asociada al objeto que alberga cada una, si es que existe, se crea un mapa de probabilidad de clases. En la figura 5 se puede observar en la parte de la izquierda la imagen inicial, que está dividida en las casillas que usará el algoritmo. Tras esto, se intenta detectar por cada casilla las *Bounding Boxes* que pueden existir, tal y como se muestra en la parte central superior de la figura,

donde se observa una cantidad elevada de *anchor boxes* innecesaria. También se obtiene de esta parte el mapa de probabilidad de clases, visible en la parte central inferior de la figura, donde está representado por colores distintos cada clase de la imagen.

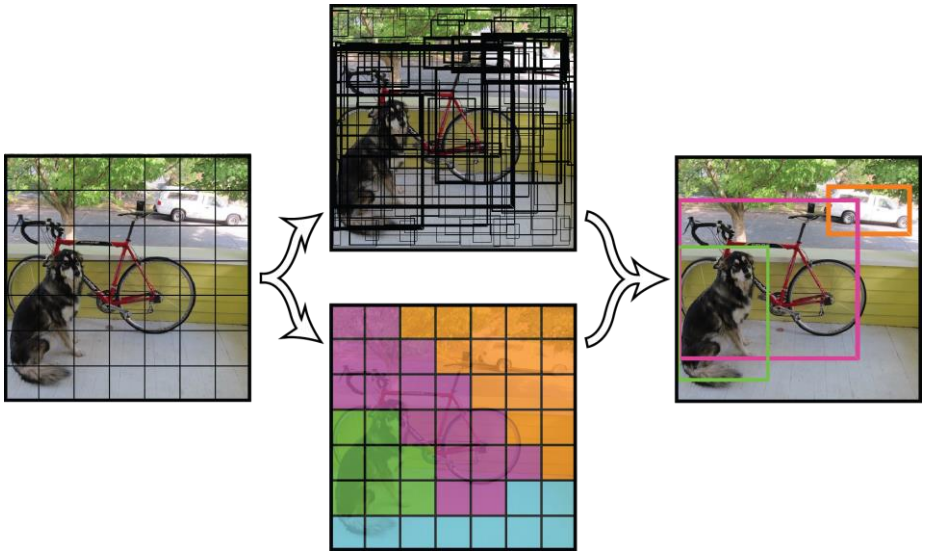

Figura 5: Ilustración del funcionamiento del algoritmo YOLO. Fuente: [27]

Una vez hemos hecho este proceso, solo queda eliminar todas las *anchor boxes* que no son deseadas. Para ello se trata la imagen con una técnica llamada *Non-Maximum Supression*, que consiste en eliminar las *Bounding Boxes* muy cercanas entre sí a través de una operación de intersección sobre unión (IoU). De esta manera, se calcula el valor de IoU para todas las *Bounding Boxes* con respecto a la que tenga mayor probabilidad de clase; por tanto, si hay dos *Bounding Boxes* cubriendo el mismo objeto pero una tiene una probabilidad menor, esta última es eliminada. Una vez hecho esto, el algoritmo encuentra la siguiente *Bounding Box* con mayor probabilidad de clase y vuelve a compararse con el resto, y de esta forma se repite hasta quedarse con todas las *Bounding Boxes* distintas. Por último, YOLO da como salida las *Bounding Boxes* encontradas en la imagen, su probabilidad y la clase asociada.

YOLO es un algoritmo sobre el que se ha iterado repetidas veces, existiendo múltiples mejoras realizadas. La primera de ellas es YOLOv2 [31], también conocido como YOLO9000; el cambio principal que trae es que usa otra red de entrenamiento distinta, llamada Darknet-19, que necesita muchas menos operaciones y solo hace uso de 19 capas convolucionales, 5 menos que su predecesor. Además, se eliminan las capas completamente conectadas, que estaban dedicadas a la clasificación de las *Bounding Boxes,* y se sustituyen por una capa de *Average-pooling*. En la siguiente iteración, YOLOv3 [32], se hace uso de una red más profunda de extracción de características, concretamente con 53 capas de convolución, por lo que fue nombrada como Darknet-53. YOLOv4 [33] es un cambio significativo, ya que trae mejoras de precisión y tiempo de inferencia, pues se optimiza para ser entrenada más rápidamente con una única GPU. Adicionalmente, se modifica el uso de la función de pérdida de intersección sobre unión para pasar a usar una modificación llamada CIOU (*Complete Intersection Over*

Union), que es capaz de converger más rápido. Existe una última versión que está en activo desarrollo llamada YOLOv5 [34]; está a la par de YOLOv4 en precisión, pero sin una gran mejora de rendimiento. Su ventaja es que posee una red más ligera y rápida de entrenar, además de un código más sencillo de entender con el objetivo de recibir mejoras por parte de otros desarrolladores. Esta versión no es muy popular debido a que no es tan innovadora como los anteriores saltos que ha dado la arquitectura ni proporciona un aumento de rendimiento considerable, y como aún está en constante desarrollo la versión más utilizada es YOLOv4.

4.5 Modelo cliente-servidor

Un modelo cliente-servidor es una estructura de aplicación donde se divide la carga de la tarea a realizar entre los proveedores de un recurso o servicio, llamado servidor, y los que solicitan ese servicio, llamados clientes [40]. En esta arquitectura, el cliente manda una petición, el servidor la acepta y procesa su solicitud y procesa los datos, enviando seguidamente una respuesta con los datos deseados.

Para conectarse con el servidor, el cliente debe conocer la IP del servidor, o al menos su URL si es una web, donde en ese caso se consultaría a un servidor DNS cuál es la IP correspondiente y se le devolvería esta información al cliente para que pueda realizar su petición.

Esta arquitectura presenta ventajas como poder centralizar el sistema teniendo todos los datos en un mismo lugar, es eficiente y, al tener el código separado, se puede modificar la capacidad tanto del cliente como del servidor sin afectar al otro. Pero también presenta desventajas; por ejemplo, los clientes son susceptibles a virus que puedan existir en el servidor. También puede existir la posibilidad de que sufran ataques DOS o DDOS, donde se deniega el servicio del servidor saturándolo. Otro inconveniente es que los paquetes pueden ser modificados o visualizados por terceras partes si no se cuenta con el cifrado y estructura correcta.

4.6 Computación en la nube

La computación en la nube, comúnmente conocida como *cloud computing*, es una tecnología que permite la ejecución remota y acceso en remoto a procesamiento de datos y almacenamiento [41], dando una alternativa a utilizar hardware local, con lo que no hay que realizar instalaciones en máquinas locales. Esto ofrece a los usuarios recursos adaptables, seguros, con alta disponibilidad y sencillos de montar y utilizar.

El funcionamiento de la computación en la nube es sencillo: se hace uso de una capa de red para conectar el dispositivo local del usuario a una máquina o recursos virtuales de un centro de datos remoto, que son capaces de procesar, proteger y guardar los datos personales del usuario. Estos centros de datos son instalaciones compuestas de ordenadores, sistemas de almacenamiento e infraestructuras varias de computación. Consta de muchas ventajas, ya que un modelo en local (*On Premise*) necesita gestionar

una gran cantidad de factores distintos, complicando más el proyecto que usando computación en la nube.

Existen tres tipos de computación en la nube:

- **Infraestructura como servicio (IaaS):** El proveedor de los servicios en la nube se encarga de llevar la infraestructura (es decir, los servidores, red, virtualización, seguridad y conexión), por lo que el cliente no se tiene que preocupar por actualizaciones hardware ni software o configuraciones adicionales. El usuario tiene acceso al servicio mediante una API o interfaz gráfica. Esencialmente, se alquila la infraestructura y el usuario puede seleccionar el sistema operativo y las aplicaciones a utilizar. Un ejemplo es Amazon Web Services, que permite alquilar máquinas virtuales para ejecutar programas elegidos por el cliente.

- **Plataforma como servicio (PaaS):** Tanto el hardware como la plataforma software son gestionados por el proveedor, mientras que el usuario gestiona las aplicaciones que funcionan en esta plataforma. Es útil principalmente para desarrolladores y programadores, ya que proporciona una plataforma compartida para desarrollar y gestionar sin tener que preocuparse por la infraestructura asociada. Un ejemplo es el motor de aplicaciones de Google, un sistema que permite ejecutar aplicaciones sobre la infraestructura de Google. Los *hosts* de páginas web son otro ejemplo, pues permiten desplegar páginas sin tener que gestionar el servidor ni detalles adicionales.

- **Software como servicio (SaaS):** Es un servicio que proporciona una aplicación software, la que es llevada por el proveedor. Son típicamente aplicaciones web de fácil acceso, que pueden ser normalmente usadas desde navegadores web o aplicaciones móviles. El usuario no tiene que hacer gestión de actualizaciones software y se elimina la necesidad de tener la aplicación instalada localmente, facilitando el uso grupal. Un ejemplo claro puede ser Gmail, una aplicación de correo gestionada por Google de fácil acceso remoto a través de navegador o aplicación móvil donde el usuario solo debe encargarse de su interacción con la aplicación.

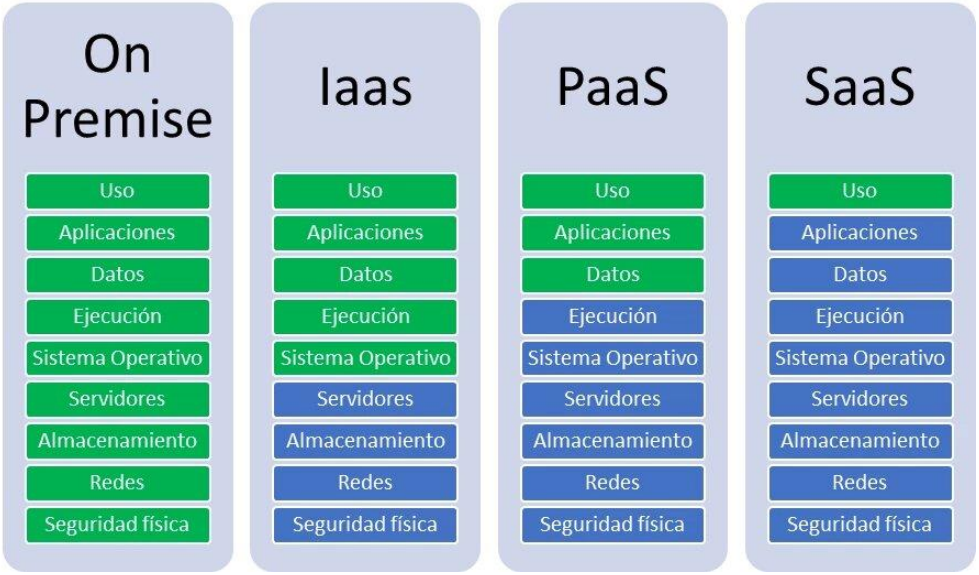

Figura 6: Diferencias en lo gestionado por usuario (en verde) y proveedor (en azul). Fuente: [42]

En la figura 6 se presentan cuatro columnas; cada una representa un tipo distinto de computación en la nube, donde se indican en verde las características gestionadas por el usuario y en azul las gestionadas por el proveedor de servicios en la nube.

5. Diseño de la solución

5.1 Arquitectura del sistema

La arquitectura seleccionada de cliente servidor nos dispone dos grandes bloques unidos entre sí, cada uno con sus propios componentes software, siendo estos dos bloques la aplicación Android y el servidor asociado. Estos están disponibles en un repositorio en línea donde cualquier persona interesada puede contribuir a su desarrollo[1].

Para la aplicación Android se ha optado por una interfaz sencilla, donde simplemente hay dos pantallas, y se puede navegar entre ellas mediante un menú en la parte inferior de la aplicación. Ambas tienen disposición tanto vertical como horizontal para facilitar su uso sin importar el ángulo en el que se tenga el dispositivo. También cuenta con traducciones de la interfaz tanto en castellano como en inglés; esto depende del idioma del sistema operativo y se cambia automáticamente si el usuario modifica su idioma por defecto.

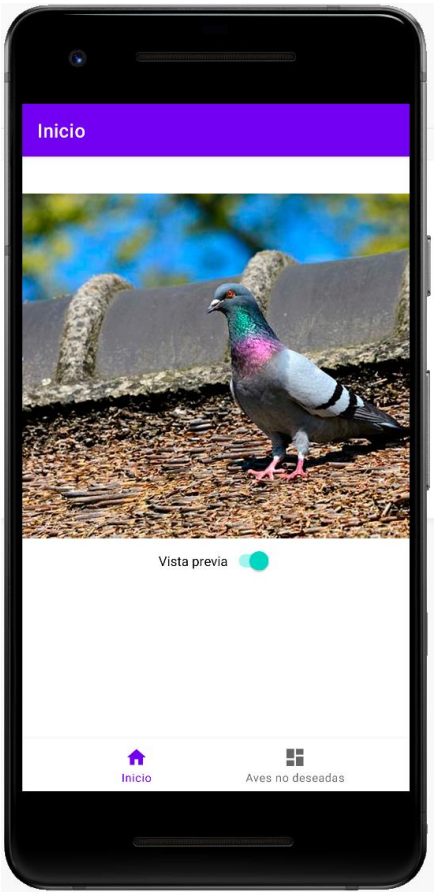

Figura 7: Pantalla de inicio de la aplicación

[1] https://github.com/joactr/eScarecrow

La primera pantalla es la de inicio, mostrada en la figura 7. Cuando el usuario esté posicionado en esta pantalla de la aplicación, estará en funcionamiento la comunicación con el servidor, capturando imágenes y mandándolas. Tras esto espera la respuesta y en caso de que un pájaro no sea aceptado emitirá una señal sonora para espantarlo. En esta pantalla de la aplicación hay un botón de tipo *switch* que permite al usuario poder visualizar u ocultar la vista previa de la cámara en la aplicación, lo que puede ayudar a reducir el consumo de batería, aunque sea ligeramente, ya que es una aplicación diseñada para su uso prolongado.

La pantalla secundaria es la de selección de aves no deseadas, es decir, las que activarán la señal sonora cuando sean detectadas en la imagen capturada. En esta pantalla se pueden marcar las distintas especies de pájaro pulsando encima del recuadro de selección correspondiente a cada especie. También hay un botón de guardado donde el usuario puede guardar las preferencias; cuando se aprieta este botón la aplicación guarda los ajustes, los que serán preservados incluso con el cierre de la aplicación. Estos cambios toman efecto inmediatamente. Las aves a espantar tendrán al lado de su nombre el recuadro activado en verde para poder diferenciarlas fácilmente. Esta pantalla secundaria se muestra en la figura 8.

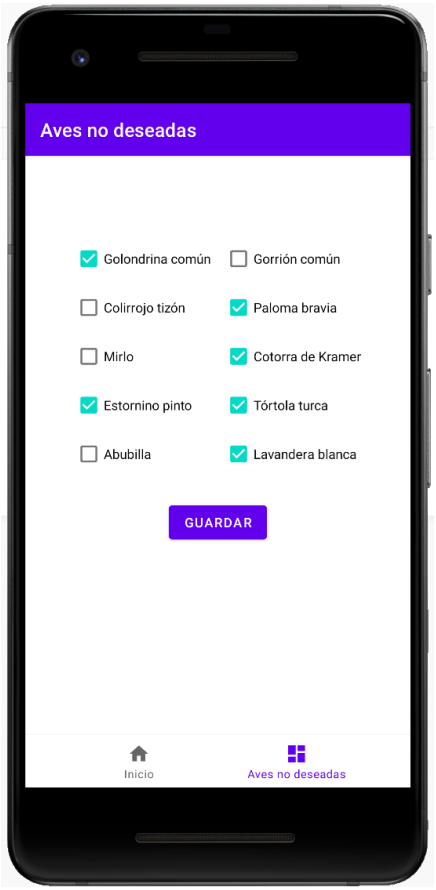

Figura 8: Pantalla de selección de aves no deseadas

La aplicación también posee un icono propio para poder ser localizada en la pantalla de inicio del dispositivo del usuario, donde se ven dos siluetas de pájaros de color rojo

sobre unos edificios minimalistas. Esto se ve representado en la figura 9. Además, tiene un nombre asociado; hemos decidido nombrarla “eScarecrow”, que proviene de la palabra “espantapájaros” en inglés junto a una “e” como prefijo indicando la naturaleza tecnológica de la aplicación.

Figura 9: Icono de la aplicación

Para el funcionamiento de la aplicación se necesita pedir permisos de uso de cámara, los que se pedirán la primera vez que se inicia la aplicación; en caso de no concederse, se solicitará también las veces posteriores que se inicie hasta que el usuario acepte conceder los permisos. También es necesario el desbloqueo de la orientación del dispositivo; para ello aparece un mensaje recordatorio en la parte inferior de la aplicación al iniciarla. En cuanto a la comunicación con el servidor, se captura una fotografía cada 15 segundos mediante la cámara frontal del dispositivo cuando la aplicación se encuentra en la pantalla principal. Junto a esta información se obtiene el ángulo a girar de la imagen; esto es necesario ya que muchos dispositivos Android por defecto giran la imagen al tomarla; este ángulo se envía al servidor adicionalmente para conocer en qué orientación se debe interpretar la imagen. Tras esto se abre una conexión al servidor y se envían los datos mencionados anteriormente. Posteriormente se espera a recibir una respuesta con un código numérico que indica la especie de pájaro encontrada o si no hay ninguno en la imagen. Con este dato la aplicación es capaz de comparar con las preferencias establecidas por el usuario y reconocer si el pájaro pertenece a la lista de aceptados o de rechazados. En caso de pertenecer a la lista de rechazados emite una señal sonora, actualmente es un maullido de gato como sonido provisional. A pesar de su carácter provisional no es ineficaz, ya que cualquier señal sonora repentina puede asustar a las aves, y más en este caso pues los gatos son depredadores naturales de la mayoría de pájaros urbanos.

Con respecto al servidor, éste se inicia cargando todos los módulos necesarios para su funcionamiento, como el detector de objetos y clasificador de imagen. Tras esto entra en un estado de escucha constante a posibles peticiones que pueda recibir de cualquier cliente. Cuando recibe una petición separa del *stream* de datos TCP los dos mensajes enviados (la imagen y el ángulo a rotar). Tras esto, se rota la imagen según el ángulo recibido, se convierte a forma matricial y se proporciona como entrada al detector de objetos, que indicará si hay o no un pájaro en la imagen. En caso de haber uno, se recorta y se le pasa este fragmento de imagen al clasificador, que indicará a qué especie

pertenece. Con este dato le damos respuesta a la aplicación devolviendo el código numérico correspondiente.

El servidor se ejecuta de forma constante en el servicio *cloud* de AWS, donde se dispone de una máquina virtual con características suficientes como para poder ejecutar el modelo y devolver una respuesta en un tiempo razonable. Se puede apagar y encender esta máquina virtual en caso de querer reiniciar o mantener apagada para ahorrar costes. También se puede conectar a ella para actualizar de forma sencilla el programa mediante el protocolo FTP para transferencia de archivos o SSH para trabajar con órdenes por consola.

5.2 Diseño detallado

Profundizando en el diseño de la aplicación Android, lo primero que hace al iniciarse es comprobar los permisos de acceso a cámara actuales; en caso de no poseerlos, se utilizarán las funciones nativas de Android para solicitarlo; si el usuario no concede estos permisos, la aplicación no funcionará, pero como este proceso se realiza con cada inicio de la aplicación se asegura que no haya que establecer estos permisos manualmente desde los ajustes del sistema. Se obtiene también el texto correspondiente al recordatorio de desbloquear la orientación y se muestra por pantalla con un módulo nativo de Android Studio llamado “Toast”, encargado de mostrar mensajes y alertas en la aplicación. Se inicializa además el reproductor de sonidos, usado en este caso para espantar pájaros.

En la pantalla secundaria de la aplicación se tiene un identificador asociado a cada caja seleccionable de especie de pájaro. Cuando el usuario activa el botón de guardado se obtienen los identificadores de todos los pájaros y si han sido o no seleccionados. Se procede entonces a guardar este dato adjunto al identificador de cada especie; esto se logra mediante una interfaz incluida nativamente llamada *SharedPreferences*; esta interfaz está diseñada para guardar, como su nombre indica, preferencias del usuario en el almacenamiento del dispositivo, pudiendo acceder a estos datos incluso tras un reinicio de la aplicación.

Debido al funcionamiento del ciclo de vida de las aplicaciones Android [35], cuando se cambia de pantalla o se cambia la disposición de la aplicación debido a la orientación hay que inicializar nuevamente algunas partes para su correcto funcionamiento, debido a que se pierden las variables locales. Para ello se inicializa la cámara con los parámetros adecuados y la previsualización asociada en la pantalla principal si procede. Además, se inicia el proceso de tomar una imagen, enviarla al servidor y recibir su respuesta cada 15 segundos; se realiza en un hilo de ejecución nuevo para no interrumpir o ralentizar la ejecución normal de la aplicación; así podemos hacer este proceso en segundo plano mientras el usuario puede seguir interactuando con la aplicación. El susodicho proceso consiste en tomar una imagen; para ello llamamos al método que se encarga de capturar una fotografía con la cámara inicializada; mediante un *callback*, podemos conocer si este proceso ha tenido éxito. En caso de ser así, obtenemos el ángulo que se debe girar la imagen, que es proporcionado por un método

de la librería de la cámara, y se procede a convertir esta imagen a un *buffer* de bytes, ya que es un formato adecuado para ser enviado al servidor en el *stream* de datos TCP. Una vez se obtienen los datos de la imagen, se abre una conexión con el servidor remoto a través de un *socket*, donde conocemos la dirección IP del servidor, ya que ésta es estática y no cambia; también debemos indicar el puerto, en este caso el 8050. Creamos un flujo de datos de salida y enviamos tanto el valor de rotación como la imagen. Este flujo de datos deberá ser posteriormente decodificado por el servidor en su recepción. Tras haber enviado estos datos se permanece a la espera de una respuesta por parte del servidor; cuando se recibe, obtenemos un código numérico, donde el 99 representa la ausencia de ave en la imagen y un número del 0 al 9 se asocia a cada una de las especies. Haciendo uso nuevamente de *SharedPreferences*, obtenemos las preferencias marcadas por el usuario y comparamos con la respuesta recibida. En caso de pertenecer la especie a un grupo no deseado, se activa la señal sonora mediante el inicio del reproductor de sonido y se cierra la conexión.

El servidor escucha en el puerto 8050 y va recibiendo los flujos de datos de los clientes tras aceptar la conexión. Para poder reconocer los distintos mensajes del flujo de datos de entrada se separa manualmente en dos partes: primero se recibe el ángulo de rotación de la imagen, que ocupará 3 bytes en total, y posteriormente la imagen, de la que no tenemos que preocuparnos de su tamaño ya que no hay ningún mensaje posterior que provenga del mismo cliente. En caso de que varios clientes manden peticiones al mismo tiempo no existe problema, ya que el servidor va guardando sus peticiones y atendiéndolas en orden de llegada, con lo que no se pierden en caso de estar ocupado procesando otra solicitud. De haber algún mensaje con una estructura distinta simplemente se ignorará, ya que se atienden únicamente peticiones de los clientes Android. Para poder detectar si hay un pájaro en la imagen se alimenta la imagen en forma de matriz al detector y éste proporcionará el código numérico de la clase a la que pertenece. En el caso de YOLO sería el número 14, que proviene del conjunto de datos Microsoft COCO [36] donde la clase de los pájaros es la decimocuarta. En caso de no detectar nada se responde con el código 99 al cliente, que significará ausencia de pájaro. Si se detecta uno, tomamos la *Bounding Box* calculada por el clasificador donde se puede localizar al ave y recortamos la imagen para solo dejar lo cubierto por la *Bounding Box*. Tras esto se escala la imagen al tamaño de entrada aceptado por el clasificador, en este caso 224x224 píxeles, y se dividen los valores de cada píxel de la imagen entre 255 con el fin de normalizar los valores y que se represente cada color entre el 0 y el 1, de forma que 0 es la ausencia de ese color en el píxel actual y un 1 indica la intensidad máxima posible de ese color. Una vez redimensionada y normalizada la imagen recortada, se predice la clase a la que pertenece seleccionando entre el vector de salida la que más probabilidad tiene y se envía de vuelta al cliente un número correspondiente a la especie detectada.

La instancia del servidor está ejecutándose en AWS EC2, una plataforma de cómputo elástica fácilmente ampliable, con lo que se podría replicar el servicio o cambiar la máquina en la que se ejecuta en cualquier momento. Actualmente, funciona en una instancia *t3.small*, que es un tipo de instancia de bajo costo que no cuenta con soporte para GPU; funciona con procesadores de primera o segunda generación Intel Xeon

Platinum 8000, de los que nuestra instancia tiene acceso a dos núcleos. Cuenta además con 2 gigabytes de memoria RAM, que es el mínimo requisito para poder ejecutar el algoritmo de procesamiento de imágenes, gestionar las peticiones y mantener en funcionamiento el sistema operativo, que en este caso es una versión de Linux propia de Amazon, llamada “Amazon Linux”. La instancia posee también una “dirección IP elástica”, que es una dirección IPv4 pública a la que se puede acceder desde internet. Ésta se asocia a la instancia para que posea una IP estática con la que poder comunicarse desde la aplicación. No es posible de otra forma ya que cuando se apaga o pausa una máquina virtual pierde su dirección actual, y no se garantiza que al reanudarla sea la misma máquina y no otra la que ejecutará el servidor. El servidor se ejecuta de forma constante a través de la orden “nohup” de Linux, que permite la ejecución en segundo plano de procesos. Es necesario el uso de esta orden, ya que todo lo que se esté ejecutando en la instancia tras cerrar la comunicación por SSH con la misma dejará de funcionar; de esta forma, creamos un proceso que funciona en segundo plano y evita la obligación de tener una conexión constante SSH a la instancia. Además de esto, se ha creado un *script* que contiene las órdenes de consola Linux necesarias para permitir la puesta en marcha del servidor de forma automática en caso de que la instancia se apagase o fallase; la pérdida de disponibilidad es un suceso poco usual en estos servicios *cloud*, pero es una posibilidad; por tanto, este *script* será ejecutado cada vez que se inicie de forma automática la instancia tras un fallo. La instancia no cuenta con una base de datos ni un almacenamiento extendido, ya que es innecesario para nuestro propósito, pues no se almacenan datos de ningún tipo exceptuando el modelo y los parámetros necesarios para el detector de objetos YOLO.

5.3 Tecnología utilizada

Para realizar este proyecto ha habido que tomar muchas decisiones respecto a las tecnologías utilizadas, para hacer uso de las que mejor se adapten a nuestros requisitos de la aplicación. En primer lugar, se ha hecho uso de Python para la realización de todo el entrenamiento e implementación del modelo clasificador de objetos, y para su conexión con el detector, ya que es el lenguaje de programación por defecto para todo lo relacionado con visión por computador. Esto es debido a que es un lenguaje muy sencillo y consistente, que posee soporte para una gran variedad de librerías de inteligencia artificial, aprendizaje automático y cálculo matemático. Es además independiente de la plataforma, flexible y ostenta una gran comunidad y soporte, haciendo más sencillo resolver cualquier duda o problema.

5.3.1 Detector de objetos

La principal prioridad del detector de objetos es el rendimiento, ya que la gran mayoría de ellos poseen una precisión suficiente para utilizar en nuestra aplicación, y vienen pre-entrenados con una gran cantidad de imágenes, por lo que no es necesario entrenar uno propio ni hacer un proceso de *transfer learning* por el que re-entrenamos uno ya existente. Simplemente hay que utilizar uno rápido y que ya cuente con la capacidad de detectar pájaros en las imágenes y etiquetarlos como tal.

El modelo a usar será YOLOv4 [33], ya que posee una precisión digna del estado del arte mientras que tiene un rendimiento más que adecuado, pues es capaz de procesar 65 imágenes por segundo con una resolución de 608x608 en una tarjeta gráfica Tesla V100. En el conjunto de datos Microsoft COCO [36] consigue una precisión media de 43,5%; esto puede parecer poco, pero en esta métrica, muy utilizada para detectores de objetos, YOLOv4 se sitúa entre los mejores detectores en este conjunto de datos, que es uno de los más populares a la hora de comprobar la precisión de estos modelos detectores de objetos. Es capaz de ejecutarse rápidamente en CPU también. Y ya que el conjunto de datos Microsoft COCO posee una etiqueta de clase de pájaros, no haría falta modificar nada para ponerlo en funcionamiento.

Además, a fecha del 16 de noviembre de 2020, una versión modificada de YOLOv4, llamada Scaled-YOLOv4 [37], era el detector de objetos que más destacaba en esta métrica mencionada anteriormente, logrando un desempeño de 55,5% de precisión media. Desgraciadamente, esta versión funciona considerablemente más lenta, siendo capaz de procesar un cuarto de imágenes por segundo con respecto a YOLOv4. Por tanto, no es tan interesante, ya que no poseeremos tarjeta gráfica en el servicio *cloud* donde se esté ejecutando el servidor con el detector de objetos. Hay que tener en cuenta que el rendimiento en CPU de estos algoritmos detectores de objetos es muy inferior al que puede tener en una GPU, incluso si esta tarjeta gráfica pertenece a una gama de entrada o más humilde.

También existen versiones incluso más rápidas, pero no logran tan buenos resultados como YOLOv4, y a pesar del interés que tenemos en una ejecución más veloz, se debe tener en cuenta que la precisión sea apta para el problema. Además, dado que usan resoluciones menores, es mucho más complicado que detecten correctamente el pájaro si no está en primer plano u ocupa un gran porcentaje de la imagen.

5.3.2 Clasificador

Tal y como se describió anteriormente, necesitamos un clasificador que sea capaz de proporcionar una precisión adecuada a nuestro problema y con un buen rendimiento, ya que se desea que el servidor sea capaz de proporcionar una respuesta sin retardo. Otro punto que debemos tener en cuenta es que debe ser re-entrenado rápidamente, ya que haremos uso de Google Colab para poder entrenar este modelo con nuestros propios datos, donde tenemos un límite de tiempo determinado en el que, si no somos capaces de finalizar el entrenamiento de esta red, se cortará la conexión con la libreta, perdiendo así el progreso. De esta forma se realiza un proceso de *transfer learning* donde aprovechemos las capacidades del modelo ya entrenado para beneficiar a nuestro problema, haciendo uso para ello de imágenes de pájaros recolectadas y etiquetadas.

Claramente, el modelo descrito en el estado del arte que mejor se ajusta a nuestras necesidades es MobileNet, que cuenta con varias versiones siendo la más reciente MobileNetv3 [38], creada también por Google. Realmente, para un número de clases reducido como el que tenemos no se necesita un clasificador con una gran cantidad de parámetros, pues solo son unas cuantas especies de pájaros y no cientos o miles de ellas, donde habría que considerar utilizar una arquitectura más compleja. En este caso MobileNetv3 cuenta con 2,9 millones de parámetros, que es una cantidad muy reducida

en comparación con la mayoría de los otros clasificadores del estado del arte; de hecho, esta tercera versión consigue reducir en medio millón el número de parámetros de la red con respecto a su predecesor MobileNetv2, con lo que resulta más rápido de entrenar y logra un tiempo de inferencia menor.

Aparte de las ventajas que posee MobileNet por sí mismo como clasificador, se han realizado anteriormente implementaciones [39] junto a YOLO como detector de objetos, que han dado buenos resultados y con una gran velocidad de inferencia. Recalcando lo mencionado, es importante la velocidad ya que al ejecutar con CPU tardará un tiempo considerablemente mayor. Pero la gran ventaja que posee este clasificador es que está diseñado en torno a los procesadores de teléfonos móviles, con lo que no hace falta uso de GPU para que los cálculos se completen en un tiempo razonable.

5.3.3 Protocolo de comunicación

Para la comunicación entre la aplicación móvil y el servidor hay dos protocolos principales, TCP (*Transmission control protocol*) y UDP (*User datagram protocol*) [40]. Estos protocolos se encargan de gestionar la comunicación y los mensajes enviados entre cliente y servidor, aparte de proporcionar su estructura.

TCP es un protocolo orientado a la conexión, es decir, que los dispositivos deben establecer primero una conexión entre ellos antes de poder transmitir datos, y después se debe cerrar esta conexión tras haber finalizado la transmisión. Es un protocolo muy fiable, ya que garantiza el envío de datos del emisor al receptor, de manera que si se pierde un mensaje se volverá a transmitir. Además, los paquetes siempre llegarán en el orden enviado al receptor. También cuenta con comprobación de errores en los mensajes.

UDP es el otro principal protocolo de transmisión de mensajes. Está orientado a datagramas, es decir, que no existe la necesidad de abrir y cerrar conexiones cada vez que se quiere mandar algo, lo cual resulta más rápido. A cambio de esto, pierde fiabilidad, ya que si se pierde un mensaje no se va a poder recuperar. Además, su mecanismo de detección de errores es muy básico, pues no posee ninguna forma por defecto de entregar los mensajes en un orden adecuado (hay que programarlo manualmente) y no se retransmiten los paquetes de datos perdidos. Por tanto, la principal ventaja que presenta es la velocidad y el bajo coste temporal de mandar mensajes.

Debido a que se va a trabajar con conexiones de datos móviles en gran parte de los casos de la aplicación Android, hay que tener en cuenta que cualquier pérdida de cobertura puede resultar problemática a la hora de retransmitir los datos al servidor y viceversa, y las mejoras de velocidad de UDP no son suficientemente grandes como para poder justificar su uso. Aparte, habría que implementar manualmente mecanismos de comprobación de errores y de gestión del orden de los mensajes. Por tanto, el protocolo de comunicación elegido ha sido TCP.

5.3.4 Servidor

Para el lenguaje de programación del servidor existen múltiples alternativas. En primer lugar, se puede usar Python de nuevo; es una buena opción ya que se usa para desarrollar la primera parte del proyecto y posee una gran compatibilidad con las librerías ya utilizadas. Es muy fácil de usar, se desarrolla muy rápidamente el código y se puede modificar de forma sencilla, ya que es un código de fácil lectura. Por otro lado, no es muy eficiente si tiene que gestionar una gran cantidad de peticiones al mismo tiempo, por lo que no es adecuado como servidor para aplicaciones de gran escala.

Otro lenguaje que se podría usar es JavaScript [43], con cualquiera de sus distintos *frameworks*. Es un lenguaje no compilado y es de los más utilizados para crear cualquier tipo de servidor; es muy similar a Python, pero está orientado a eventos, lo que permite mucha flexibilidad a la hora de programar un servidor, ya que atenderá automáticamente las peticiones y gestionará todo de manera muy sencilla. Tampoco es muy rápido en su ejecución. Permite escalar el servidor fácilmente para poder atender muchas peticiones simultáneamente.

Java [44] también es una alternativa viable; es un lenguaje compilado, que es capaz de realizar cálculos rápidamente y es fácilmente escalable. Un punto a favor es que es capaz de utilizar varios hilos de ejecución para acelerar el programa. Es muy apto para servidores de gran escala.

Hemos tomado la decisión de hacer uso de Python, ya que es donde se desarrolla la primera parte de este proyecto y tenemos familiaridad con su uso. El servidor en principio no estará pensado para recibir muchas peticiones simultáneamente; por tanto, no habría problema en hacer uso de este lenguaje. En caso de escalar la aplicación y tener muchos más usuarios el servidor es una parte que se puede rehacer sin demasiadas complicaciones. Las librerías compatibles son un punto a favor que permiten un desarrollo rápido y eficaz.

5.3.5 Aplicación móvil

Para la realización de la aplicación Android se pueden utilizar varios entornos de desarrollo distintos, cada uno con sus ventajas y desventajas.

Visual Studio [45] es un entorno de desarrollo que permite desarrollar código de casi cualquier ámbito, y las aplicaciones móviles no son una excepción. Es muy conocido y por tanto recibe mucho apoyo por parte de la comunidad, y también de los creadores en caso de cualquier problema, ya que es una herramienta creada por Microsoft. Utiliza Xamarin, una capa de abstracción encargada de comunicar el código compartido con el código plataforma subyacente, lo que permite desarrollar para varios sistemas al mismo tiempo y se utiliza C# para programar. Este entorno es mucho más fácil de ejecutar que otros competidores; por tanto, es muy buena opción para programar en máquinas más humildes o con menos prestaciones. Goza también de una amplia variedad de *plugins* aplicables de forma rápida.

Otro entorno de desarrollo es Android Studio [46], creado por Google y basado en el popular entorno de IntelliJ. Ofrece soporte nativo para Android, ya que es el sistema

operativo creado por Google, con lo que las librerías y *plugins* que vienen por defecto son las idóneas para el desarrollo nativo en Android. Utiliza principalmente Java como lenguaje de programación, pero existe la opción de utilizar Kotlin, que es más legible. Tiene una gran comunidad y es muy fácil encontrar respuestas a los problemas en foros on-line. La única pega que presenta es que consume una cantidad considerable de recursos, por lo que no es adecuado para el desarrollo en ordenadores más básicos.

Existen otros entornos de desarrollo, pero no alcanzan todas las características y beneficios que disponen las anteriores. También hay alternativas como “Andromo” [47] o “AppGeyser” [48], que prometen desarrollo sencillo sin ningún tipo de programación y fácilmente adaptable a dispositivos iOS. Pero ya que sabemos programar, no tendría mucho sentido hacer uso de estas herramientas, ya que limitan mucho lo que se puede hacer con ellas.

Se ha seleccionado Android Studio para realizar la aplicación móvil, ya que es la herramienta oficial de desarrollo para este sistema operativo y además está basado en el entorno de IntelliJ, que es muy cómodo y tiene muchas funcionalidades, aparte de que ya tenemos cierta familiaridad con él. El lenguaje a utilizar será Java, ya que se ha utilizado mucho a lo largo de la carrera y tenemos experiencia y conocimientos adquiridos durante estos años.

5.3.6 Servicios *cloud*

Para la implantación del servidor necesitamos algún servicio de máquinas virtuales en la nube, también conocidas por sus proveedores como “instancias”, que nos permita ejecutar nuestro servidor sin ningún tipo de interrupción, manteniendo una IP fija para su comunicación con la aplicación móvil y que tenga un costo nulo preferiblemente; y en caso de no ser posible, que sea económico.

En primer lugar, *Microsoft Azure Virtual Machines* [49] permite la creación de una máquina virtual gratuita durante 30 días, con un coste permitido de hasta un máximo de 200 dólares estadounidenses donde se puede ejecutar el servidor; después de este límite se empieza a cobrar dependiendo de las horas de ejecución mensuales de la máquina y de las características hardware de la misma. Consta de una gran cantidad de tipos distintos de instancia según las necesidades, incluyendo algunas con GPU para poder acelerar el cálculo de las operaciones necesarias para la detección de las aves en las fotos y su posterior clasificación. Permite también la asignación de una IP estática, pero no es de uso gratuito y ronda los $35 anuales, dependiendo también del área geográfica donde se desee ejecutar el servidor. Es una plataforma muy segura y cuenta con herramientas para ayudar a desarrollar aplicaciones de forma más sencilla.

Amazon Web Services (AWS) [50] posee un servicio de instancias llamado “EC2” que también tiene las mismas funcionalidades que Azure; los precios son muy similares y también se cobra por horas mensuales según las características del sistema elegido. La diferencia principal en lo que a este proyecto se refiere es la asignación de IP estática, pues para este servicio es gratuita mientras esté en funcionamiento la máquina virtual. Es muy sencillo escalar también el servidor debido a que se pueden crear máquinas

virtuales que ejecutan lo mismo y luego simplemente se redirige la petición a una o a otra dependiendo de la carga actual.

La última alternativa es *Google Compute Engine* [51]; tiene un gran rendimiento y tiene varias herramientas centradas en aprendizaje automático e inteligencia artificial. La gestión de datos es más sencilla en esta plataforma que en las anteriormente mencionadas, pero esto tampoco es de necesidad en nuestro proyecto. Asignar una IP estática no es gratuito y tiene coste por hora. Las comunicaciones funcionan de extremo a extremo con cifrado con sus propias redes privadas, por lo que un ciberataque es poco probable.

La elección es difícil, ya que las tres plataformas ofrecen servicios prácticamente idénticos, y están apoyadas por tres de las compañías tecnológicas más grandes actualmente. Cualquiera de ellas nos puede permitir subir nuestro servidor sin problema alguno, y los precios son similares. Ya que nuestro uso va a ser simple y sin necesitar herramientas adicionales, la ventaja que presenta AWS es no tener que pagar por la IP estática siempre y cuando el servidor esté en funcionamiento, y que al tener la máquina virtual precios muy similares a sus competidores, estaríamos ahorrando dinero por este servicio. Permite además un cambio muy sencillo de plan en caso de querer cambiar a una máquina más potente.

6. Implementación, pruebas y resultados

El primer paso tomado para la realización de este proyecto ha sido la selección de las distintas especies de pájaros a usar. Para ello se ha utilizado un artículo de la Universitat Politècnica de València donde indica varios de los tipos de aves que se pueden encontrar en el campus de Vera [52]. A su vez se han aprovechado los conocimientos del tutor de este proyecto, que ha sugerido dos especies más observadas con frecuencia en la ciudad de Valencia. Así se tiene un total de diez clases distintas que pueden ser observadas en la figura 10, donde de izquierda a derecha por filas son:

- Fila 1:
  - Lavandera blanca
  - Mirlo común
  - Estornino pinto
- Fila 2:
  - Cotorra de Kramer
  - Colirrojo común
  - Abubilla
  - Golondrina común
- Fila 3:
  - Gorrión común
  - Paloma bravia
  - Tórtola turca

Figura 10: Especies de pájaros utilizadas para la aplicación

Tras haber acordado los grupos adecuados a clasificar se ha procedido a recolectar imágenes de cada una de las especies con el propósito de entrenar el clasificador y poder comprobar su precisión posteriormente. Con el fin de acelerar este propósito se ha hecho uso de un conjunto de datos de la plataforma *Kaggle* [53], que contiene imágenes de 400 especies de pájaros distintas con resolución de 224x224 y 3 canales. De éstas hemos podido obtener datos para la paloma bravia, abubilla, estornino pinto y golondrina común. Pero a pesar de haber obtenido imágenes de estas aves, ha sido necesario eliminar algunas de ellas debido a que eran dibujos de la misma sin demasiado realismo o primeros planos de sus ojos, pico o cabeza, que solo empeorarían el rendimiento de nuestro clasificador al incluirlas en el entrenamiento. Se han buscado para suplir esta falta de datos imágenes adicionales a través de motores de búsqueda, y se han descargado con un *script* Python creado por nosotros que descarga todas las imágenes de la página. Se ha conseguido con esto una gran cantidad de imágenes, de las que ha habido que descartar muchas por baja resolución, pájaro equivocado, malas condiciones de la fotografía u otras en las que el pájaro ocupa una parte minúscula de la imagen.

La siguiente tarea ha sido filtrar las imágenes y seleccionar las adecuadas. El siguiente paso ha sido recortar una a una cada una de las muestras obtenidas y convertirlas en imágenes de primer plano de cada una de las aves seleccionadas. En este proceso se han descartado aún más imágenes debido a no cumplir con una resolución aceptable al ser recortadas. La resolución deseada es de 224x224, al igual que las muestras obtenidas de *Kaggle*, ya que con esto se homogenizan las imágenes y se entrenará el clasificador con esta resolución, que es una muy común para este uso. En la tabla 1 se muestra el número de imágenes tras recortar (primer plano) y las imágenes con un plano general del pájaro. Las fotografías de plano general serán utilizadas para comprobar la precisión del detector de objetos; de éstas se han obtenido muchas de las imágenes de primeros planos. En las muestras obtenidas de *Kaggle* el número de fotografías de planos generales es menor dado que se han tenido que recolectar menos de forma manual para recortar.

Nombre de la especie	**Nº de imágenes primer plano**	**Nº de imágenes plano general**
Lavandera blanca	122	156
Mirlo común	145	174
Estornino pinto	145	89
Cotorra de Kramer	140	161
Colirrojo común	146	180
Abubilla	127	105
Golondrina común	132	113
Gorrión común	127	141
Paloma bravia	125	81
Tórtola turca	145	203
Total	**1354**	**1403**

Tabla 1: Imágenes recolectadas por cada grupo de ave

Posteriormente, se han aislado aproximadamente 15 imágenes de plano general de cada especie que no han sido recortadas. El propósito es poder comprobar qué precisión se puede obtener con el sistema completo; para ello se usan muestras distintas a las usadas para entrenamiento con el propósito de obtener unos resultados fiables.

Para la implementación del entrenamiento del clasificador se ha hecho uso de Google Colab, ya que proporciona tarjetas gráficas de uso gratuito y permite la ejecución de libretas de código Python, con lo que cubre nuestras necesidades. Se ha utilizado el libro “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems” [26] de Aurélien Géron para refrescar conceptos, profundizar en nociones teóricas y aprender a implementar un clasificador a través de *transfer learning* con las librerías Python “Keras” y “Tensorflow”, ya que hasta ahora solo teníamos experiencia trabajando con GNU Octave [54] y siempre entrenando desde cero el clasificador.

Este proceso de entrenamiento ha empezado subiendo las imágenes recolectadas a Google Drive para poder utilizarlas en Google Colab, ya que permite montar el espacio de almacenamiento *cloud* como si de un disco duro se tratase, con lo que permite el acceso a estos datos de forma sencilla, no diferenciándose de un entrenamiento realizado en local. El problema de esta aproximación es su coste temporal asociado a la importación de datos, que es mucho más lento que su contraparte en local. Se empieza la ejecución clasificando cada una de las imágenes en su especie correspondiente; este proceso resulta sencillo ya que se encuentran en carpetas distintas y no hay que realizar trabajo adicional. Luego se convierte cada una de las imágenes a forma matricial con resolución 224x224 para poder trabajar con ellas, tardando así 12 minutos únicamente en importar todas las imágenes y convertirlas en este espacio de trabajo de libreta Python; esto es algo que en local tarda unos pocos segundos, pero este tiempo se gana luego en la fase de entrenamiento. Las imágenes son normalizadas al dividir sus valores entre 255 para que se sitúen entre 0 y 1 los valores de color de cada píxel. Se asocia también un vector con las etiquetas de clase de cada una de las muestras, que permitirá conocer si se ha acertado o no en tiempo de clasificación y prueba. Luego dividimos las imágenes en las que se utilizarán como conjunto de entrenamiento y las de conjunto de test, donde hemos decidido hacer una división de 80% y 20% respectivamente. Se usa un conjunto de test de tamaño estándar; por un lado, debe ser relativamente grande ya que no contamos con muchas muestras totales, pero al mismo tiempo al estar usando un clasificador con muy pocos parámetros, reduce la necesidad de aumentar mucho el tamaño del conjunto de test.

El siguiente paso es importar el modelo de MobileNetv3 y configurar su entrada a imágenes de resolución 224x224. Le añadimos una capa final densa, es decir, que cada neurona de esta capa conecta con todas de la anterior capa. El tamaño de salida de esta nueva capa es igual al número de clases de las que disponemos, 10 distintas. Esta capa usa la función de activación *Softmax*, que da como salida un vector donde la suma de sus componentes es 1, de forma que cada componente es la confianza que se tiene en la clase correspondiente, siendo la clase elegida finalmente la que mayor confianza posea. Esta función de activación es normalmente la más utilizada para la capa final de

clasificadores donde las etiquetas sean mutuamente exclusivas, que es nuestro caso, ya que un pájaro no puede pertenecer simultáneamente a dos especies. Con esta capa, el modelo cuenta en total con 2.744.426 parámetros, de los que 12.810 pertenecen a la última capa agregada.

Una vez seleccionado el modelo, se deben elegir los parámetros de entrenamiento. En este caso se ha seleccionado el algoritmo de optimización *Adam*, que será el encargado de ajustar parámetros de la red como los pesos o la tasa de aprendizaje para obtener mejor precisión. La función de pérdida a minimizar es entropía cruzada [55], usada a menudo en problemas de clasificación; describe la distancia entre dos distribuciones de probabilidad, la de las predicciones actuales del clasificador y la que representan las respuestas verdaderas y que buscamos aproximar; cuanto menor sea el número resultante, más se aproximarán las dos distribuciones y es lo que se busca minimizar. Para el entrenamiento se realizan 10 *epochs*; por tanto, se recorre completamente el conjunto de entrenamiento 10 veces en total. El primer *epoch* obtiene una pérdida de 1,412 y una precisión de 68,3% respecto al conjunto de entrenamiento y cuando se llega al décimo la pérdida se sitúa en 0,050 y la precisión es de 99,6% mientras que en el conjunto de validación comienza con una pérdida de 0,725 y una precisión de 91,1% y en el décimo epoch cuenta con una pérdida de 0,128 y 96,7% de precisión. En principio se había probado a hacer uso de 25 *epochs,* pero no mejoraba la precisión, tardaba más en entrenar y puede generar problemas de sobreajuste, donde el clasificador aprende los casos particulares enseñados y será incapaz de reconocer nuevos datos proporcionados, clasificando incorrectamente. Por ese motivo se ha decidido reducir el número de *epochs* a 10 con el propósito de evitar el sobreajuste. En la figura 11 se puede observar la evolución de la precisión a medida que se va entrenando el clasificador y cómo a partir del séptimo *epoch* aproximadamente, aumenta mínimamente en ambos conjuntos la precisión. Se puede ver también que a partir del *epoch* 18 el clasificador ya nunca falla en el conjunto de entrenamiento al haber aprendido los datos.

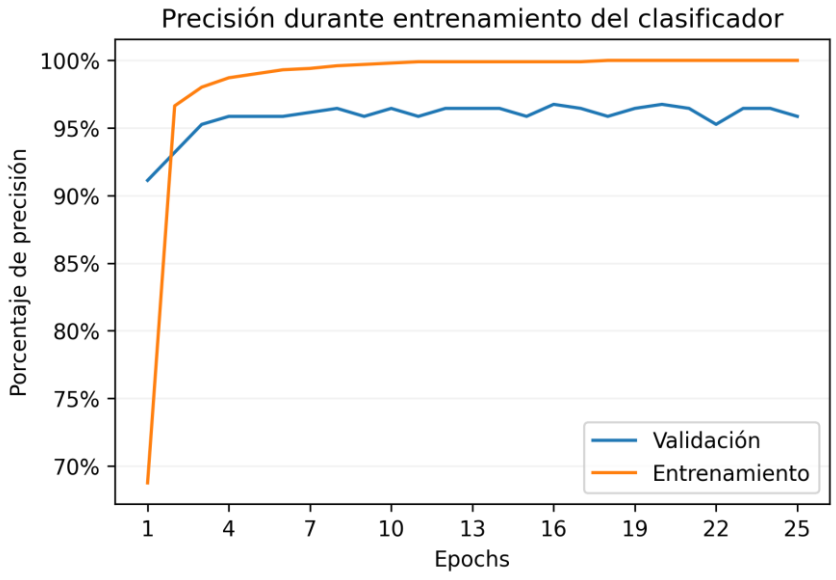

Figura 11: Precisión durante entrenamiento

En la figura 12 se muestra el mismo proceso de entrenamiento, pero esta vez con los valores de la función de pérdida, que está relacionada con la precisión, pues también es un indicador de qué tan correcto es el funcionamiento del clasificador; sin embargo, un valor alto de precisión no implica una baja pérdida ni viceversa, ya que no miden lo mismo. Se observa un rápido descenso de la función en validación hasta aproximadamente el séptimo u octavo *epoch.*

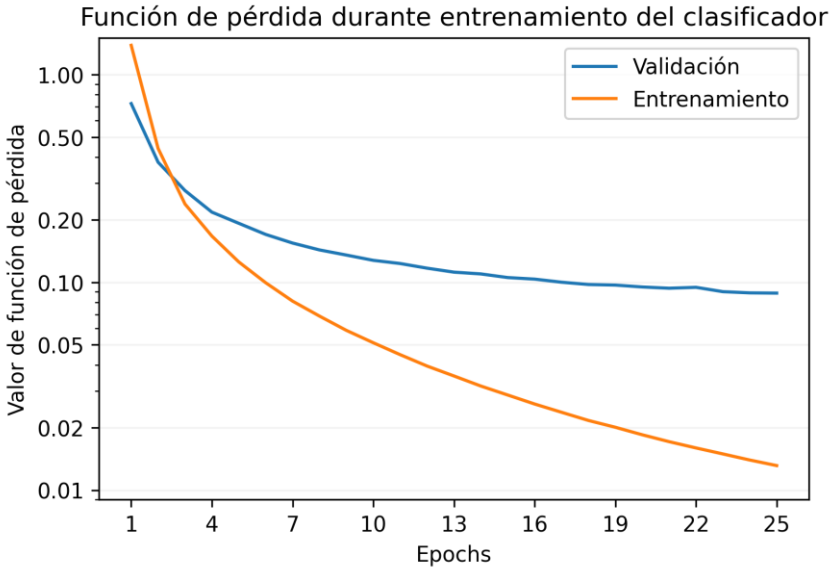

Figura 12: Función de pérdida durante entrenamiento

El tiempo de entrenamiento ha sido substancialmente menor a lo esperado, haciendo uso de la GPU asignada, que es totalmente aleatoria según disponibilidad de Google Colab. En nuestro caso ha sido una Nvidia Tesla K80, con la que se ha obtenido un tiempo total de ejecución de 10 minutos y 8 segundos, donde el primer *epoch* ha tardado 5 minutos y los nueve posteriores han rondado todos los 34 segundos de ejecución. Esto demuestra que es un modelo muy ligero y con pocos parámetros, que será de utilidad para luego poder ejecutarlo en una CPU no tan potente, por lo que no habrá demora en las respuestas a las peticiones de los clientes.

Para que las métricas de error de las pruebas del modelo ya entrenado sean correctas, se debe utilizar el intervalo de confianza del 95%, que es una manera de obtener una estimación media sin tener que utilizar una cantidad infinita de datos, pues es imposible obtener el porcentaje de error de forma exacta sin una cantidad masiva de datos, ya que los usados pueden no ser del todo representativos; por tanto, se obtiene un intervalo de error acotado. Se calcula de la siguiente forma:

$$p(\hat{p}_{error} - \epsilon \leq p_{error} \leq \hat{p}_{error} + \epsilon) = 0{,}95$$

$$\epsilon = 1{,}96\sqrt{\frac{\hat{p}_{error}(1 - \hat{p}_{error})}{N}}$$

Donde *N* es el número de muestras totales usadas en la prueba, $\hat{p}_{error}$ es la probabilidad de error calculada dividiendo el número de errores en la prueba entre *N* y p_{error} es el verdadero valor del error que deseamos acotar.

Ya entrenado el clasificador con un gran resultado en las muestras de entrenamiento, se debe probar con las muestras de test, donde obtiene un 96,7% de precisión, con un error de 3,3 ± 2,3%, equivalente a un intervalo de error de [1,0%-5,6%]. Consideramos estos resultados como aptos para su uso y no requiere la modificación de los parámetros del clasificador. Una vez comprobada la aptitud del modelo, se exporta para poder utilizarlo ya entrenado, pues si no se hace esto, se perderán los datos obtenidos en Google Colab al cabo de unas horas o desde que se cierre la sesión.

El siguiente paso es comprobar si hay que realizar alguna modificación del detector de objetos. Para ello comprobamos su funcionamiento a través de darle como entrada 700 imágenes de pájaros de distintas especies, de forma que se observa el pájaro claramente pero no en un primer plano. Esto corresponde con aproximadamente la mitad de las muestras totales de planos generales obtenidas. En estas pruebas se ha hecho uso de la GPU Nvidia Tesla T4, que ha sido la asignada en el momento. En primer lugar, se ha probado a utilizar la resolución por defecto de YOLOv4 recomendada, 416x416. Con esta resolución, el algoritmo detector de objetos ha sido capaz de localizar al menos un ave en 643 imágenes, un 91,9% del total, aplicando el intervalo de confianza de 95% proporciona un error comprendido en el intervalo [6,1%-10,2%] y se obtiene un tiempo de inferencia total de 28 segundos, que equivale a poder analizar 25 imágenes por segundo o un tiempo de inferencia de cada imagen individual de 40 milisegundos. Aumentando la resolución de entrada de las imágenes a 608x608 se detectan aves en 674 imágenes, siendo esto un 96,3% del total; en este caso el intervalo de error sería [2,3%-5,1%], con un tiempo de inferencia de 48 segundos, o 14,6 imágenes procesadas por segundo, unos 69 milisegundos por imagen. Esta es la resolución máxima en la que se detallan pruebas de YOLOv4 por parte de sus creadores, aunque se puede aumentar más siempre y cuando el lado de la imagen sea múltiplo de 32. Se ha probado también con una resolución de 864x864, pero se detectan aves en 656 imágenes, 93,7% de las totales; en este último caso el intervalo de error es de [4,5%-8,1%], siendo ligeramente peor que la prueba anterior pese a tener más resolución de entrada. Esto se puede deber a varios factores, como que algunas de las imágenes empleadas tienen una resolución total menor a ésta o que el modelo no ha sido entrenado para tratar con imágenes de tanta resolución, con lo que las predicciones no son tan precisas. Además, el tiempo de inferencia total de esta prueba aumenta a 87 segundos, por lo que no se obtiene ningún tipo de beneficio.

Una vez comprobados los modelos de detección de objetos y clasificación de imágenes por separado, llega el momento de unificarlos y probar el algoritmo completo. Para ello se usarán las imágenes aisladas anteriormente, unas 15 por cada clase (en total 146). El algoritmo obtendrá como entrada la imagen, la que será entregada a YOLO, sin normalizar, ya que YOLO cuenta con capas encargadas de este proceso; si se detecta un ave, se recorta la misma de la imagen original, se reescala a 224x224 y se normaliza para el clasificador dividiéndola entre 255. En total, de las 146 imágenes se han detectado aves en 139 de ellas, y de esas 139, se ha detectado correctamente

la especie un 90,7% de las veces, es decir, en 126 imágenes, por lo que el sistema unificado cuenta con un 86,3% de precisión en esta prueba. El error en este caso sería de 13,7 ± 5,6%, obteniendo un intervalo de error de [8,1%-19,3%]. Es un rango de números amplio, pero en el peor de los casos el modelo tiene un error menor al 20%, que resulta aceptable con las limitaciones con las que se han contado; podría acotarse más este intervalo de utilizar muchas más imágenes en las pruebas. Este número es ligeramente menor a las pruebas realizadas únicamente del clasificador debido a que las imágenes pueden no contar con una resolución suficiente como para que la imagen del pájaro una vez recortado cuente con resolución de 224x224, y por tanto habría que redimensionar la imagen a una resolución mayor, lo que no es del todo preciso. Además, algunas aves pueden detectarse en la imagen con una proporción más alta que ancha o viceversa, pues no son perfectamente cuadradas, lo que lleva a una deformación de la imagen entregada al clasificador, ocasionando a una pérdida de precisión al clasificar.

La siguiente parte a realizar es crear un servidor implementando el algoritmo desarrollado anteriormente. Para ello se ha realizado un *script* Python con *sockets* que recibe peticiones, las atiende y devuelve una respuesta. Este proceso empezó creando un servidor que simplemente recibía cualquier mensaje a través de un *socket* e imprimía un mensaje por pantalla para comprobar el funcionamiento. Con el propósito de comprobar la comunicación y recepción de mensajes, se creó un cliente que mandaba mensajes de texto al puerto del servidor, todo en la misma máquina *host* para no tener que preocuparse por abrir puertos ni direcciones IP de momento. El siguiente paso fue integrar el algoritmo creado anteriormente en este servidor para procesar las imágenes. En este momento el servidor recibía una imagen, detectaba todas las aves que pudiese en la misma y guardaba en la carpeta del *script* una copia de la imagen, pero con las *Bounding Boxes*, sus etiquetas correspondientes y su confianza. Esto permitió observar que poseía un funcionamiento correcto. El último paso antes de empezar con la aplicación Android fue que devolviese al cliente un código numérico asociado a la especie detectada o 99 en caso de no haber detectado una.

La creación de la aplicación fue un proceso largo, pues se ha requerido estudiar el ciclo de vida de las aplicaciones Android para entender por qué se borran variables, se inicializan de nuevo y en qué parte del código se debe gestionar cada parte de la aplicación, ya que inicializar código donde no es adecuado puede llevar a cuelgues en la aplicación, valores de variables erróneos o funcionamiento poco consistente, como hemos podido comprobar durante su desarrollo. Tras haber entendido un poco mejor estos conceptos, se ha empezado a implementar el uso de la cámara en la aplicación, en nuestro caso haciendo uso de la biblioteca CameraX de Google, que es una biblioteca sencilla de usar pero con un gran número de funcionalidades. Se ha creado un menú con dos pantallas diferentes y la navegación correspondiente entre ellas, e ideado un sistema mediante el uso de *SharedPreferences* para poder guardar las preferencias del usuario con respecto a los pájaros a espantar.

Hemos implementado también una forma de hacer que la pantalla no se apague mediante el uso de un *flag* en la aplicación; esto es necesario ya que si la pantalla se apaga no podrá seguir sacando fotos. Se intentó implementar una manera de hacer que la aplicación funcione incluso cuando el dispositivo se ponga en modo suspensión, pero

no tuvimos éxito en ello; se debe a que Google considera que el uso de la cámara mientras Android no está en uso supone un riesgo de seguridad, ya que se podrían crear aplicaciones maliciosas de esta forma. Hay maneras para evitar estas restricciones, pero son complejas y no hemos conseguido implementarlas. Se implementó en esta fase la solicitud de permisos al usuario, ya que hasta ahora había que manualmente conceder los permisos a la aplicación desde los ajustes del sistema cada vez que se creaba una nueva versión.

Otro problema que pudimos observar al empezar a tomar fotografías con la cámara del dispositivo de prueba es que algunos dispositivos Android rotan la imagen automáticamente. Esto depende totalmente del dispositivo y su versión Android, pues en el dispositivo de pruebas físico y una de las máquinas virtuales se daba este problema, pero en otra de ellas no. Tras investigar el problema se descubrió que las librerías de cámara de Android vienen preparadas para estos casos, ya que poseen una función que permite obtener el número de grados que debe ser girada la imagen para llegar a su posición original. Se intentó rotar la imagen en el propio dispositivo tras su captura, y aunque se consiguió, el uso de distintas estructuras de datos para la conversión de la imagen provocaba algunos problemas en la máquina virtual de menos prestaciones que usábamos para pruebas, que contaba únicamente con 256 megabytes de memoria RAM. Por tanto, se decidió pasar la carga de esta tarea al servidor a través del envío del número de grados que debe girar la imagen, dato que podemos obtener relativo a la orientación del dispositivo, con lo que es obligatorio el desbloqueo de la orientación. Realizar el giro de la imagen resulta trivial para el servidor y solo supone para la aplicación móvil enviar un mensaje adicional con esta información.

Ya decidida la solución al problema se procede a establecer una conexión al servidor, que en este momento se estaba ejecutando en un ordenador personal, pero no conectaba de ninguna forma. Se llegó a creer que era un problema de implementación, pero tras probar con el cliente Python desarrollado anteriormente ejecutándolo desde otro dispositivo en la misma red local tampoco funcionaba. Sin embargo, cuando se ejecutaban en la misma máquina no había problema. Tras investigar posibles causas, se detectó que el *router* del proveedor de internet donde está situada la máquina usada no permite conexiones de bucle invertido, que son las conexiones entre dos dispositivos de la misma red local, pero usando para conectarse entre sí la IP externa pública de la red en la que se encuentran los dispositivos. Por tanto, lleva a la obligación de tener que subir el servidor a un servicio *cloud* un poco antes de lo esperado. Se elige para ello la nube de Amazon Web Services y se crea y configura para su uso una instancia gratuita *t3.micro*, que será ejecutada en la región de París, que es la más cercana geográficamente a Valencia entre las existentes y cuenta con la menor latencia, entre todas las regiones posibles de ejecución situadas en Europa, obteniendo Paris una latencia de 104 milisegundos de media. Esta instancia posee dos hilos de ejecución de CPU y un único gigabyte de memoria RAM. Se asocia una dirección IP estática a la instancia para conectar de forma sencilla y se suben los archivos que componen el servidor, siendo éstos el modelo del clasificador entrenado, un archivo de configuración de YOLO con los parámetros a usar y un fichero de nombres que utiliza YOLO para poder conocer la clase detectada. Esto se realiza a través del protocolo FTP (*File*

Transfer Protocol), que sirve para transferencia de archivos como su nombre indica, mediante la aplicación FileZilla, un cliente para el uso de este protocolo. Una vez subidos los archivos necesarios a la máquina virtual, se procede a poner en ejecución el servidor, que parece funcionar sin ningún problema aparente, pero tras conseguir conectarlo a la aplicación Android y enviar una imagen, se da un cuelgue del proceso del servidor en la instancia. Al leer los *logs* del sistema se observa que es debido a falta de recursos, en este caso de memoria RAM, por lo que se mejora la instancia a una *t3.small*, que cuenta con el doble de memoria RAM que la anterior, en este caso 2 gigabytes, suficiente para soportar el sistema y el servidor sin ningún problema.

Solucionado este problema y comprobando el correcto funcionamiento del servidor en la nube se desarrolla el reproductor de sonidos en la aplicación Android, que se activa cuando el pájaro detectado está incluido en la lista de especies a espantar, y se desarrolla también una versión alternativa de la interfaz para la disposición horizontal de los dispositivos, pues es lógico que la interfaz se adapte si el desbloqueo de la orientación es un requisito de la aplicación. Con esto se da por concluido el desarrollo general del sistema y solo queda realizar pruebas finales del mismo.

Para comprobar la precisión del sistema se ha utilizado un teléfono móvil personal. Su modelo es Redmi Note 9 Pro, que posee una cámara y prestaciones suficientemente buenas como para no haber problemas de rendimiento en las mediciones de tiempo de ejecución y se cuenta con una conexión estable de fibra óptica a la que está conectado el dispositivo. El primer paso de la prueba es comprobar el tiempo de procesamiento de una imagen en el servidor sin tener en cuenta el tiempo de envío de la misma; de esta forma se incluye en el programa de forma temporal una medición de tiempos que imprime por la pantalla de la instancia el tiempo desde que se tiene la imagen ya en el *buffer* de la aplicación hasta que se ha obtenido un resultado tras procesarla. Tras procesar 15 imágenes se obtiene una media de 3,7 segundos; todas estas imágenes se han tomado con el dispositivo móvil apuntando a imágenes de distintos pájaros en un monitor de ordenador, con el propósito de que aumente el tiempo de inferencia al tener que ejecutar también el modelo de clasificación. Incluyendo el tiempo de envío de la imagen y recepción de la respuesta esta media pasa a ser de 4,1 segundos, que ha sido medida con las 15 imágenes anteriores al mismo tiempo. El cálculo se ha realizado mediante Android Studio, imprimiendo por pantalla el tiempo desde que se va a empezar la transmisión de datos al servidor hasta que llega una respuesta del mismo.

Para comprobar la precisión del sistema final, se usan las imágenes que habíamos aislado previamente para comprobar el sistema unificado, y se utilizan 10 de cada especie, 100 imágenes en total. Se usa el mismo dispositivo móvil y las imágenes se visualizan en el monitor de un ordenador, de donde el dispositivo toma la imagen e intenta detectar el pájaro, de forma que la previsualización de la cámara cubre únicamente la imagen para tener una comparación adecuada. Se ha obtenido un resultado de 88% de precisión, clasificando correctamente 88 veces. Hay una muy ligera mejora de precisión con respecto al sistema probado sin la aplicación móvil, siendo la diferencia de 1,7% a favor de esta última prueba, pues antes se obtenía un 86,3% de precisión. El intervalo de error final es de [5,6%-18,4%]. Esta diferencia puede deberse

al bajo número de imágenes utilizadas, que no acota tanto los resultados, pero ambas pruebas producen unos resultados bastante similares.

7. Conclusiones

7.1 Conclusión

El proyecto realizado puede ser considerado un éxito, pues se ha logrado construir un sistema final que es capaz de distinguir entre las especies de pájaros más comunes en la ciudad de Valencia, espantando a las no deseadas con precisión y tiempos de respuesta aptos. La aplicación desarrollada ha cumplido con el objetivo de tener una alta compatibilidad con la gran mayoría de dispositivos Android, pues sólo con tener cámara, altavoz y una mínima potencia de cómputo es capaz de ejecutar la aplicación correctamente, ya que los cálculos se realizan en el servidor. Es además personalizable, pues permite a cualquier usuario adaptar su funcionamiento a su gusto gracias a la opción de poder seleccionar los pájaros espantados por la aplicación. Desgraciadamente no se ha cumplido con el objetivo de ser una aplicación disponible en Google Play; esta decisión se ha tomado debido a que creemos que la aplicación tendrá un atractivo mayor para un usuario cuando posea una cantidad mayor de funcionalidades y más especies distintas que poder clasificar, ya que actualmente solo podría ser usada en Valencia ciudad. Los objetivos de error y tiempo de respuesta también se han podido cumplir; tal y como se ha observado en el apartado de implementación, pruebas y resultados, obtiene más del 75% de precisión y tiempos de respuesta menores a 8 segundos, que eran los objetivos planteados para este sistema al principio de este proyecto. Consideramos por tanto que los resultados finales son buenos y estamos satisfechos con ellos, pero nos gustaría mejorar la aplicación más antes de permitir que el público general acceda a ella. Creemos que es importante la percepción inicial que puede haber sobre la aplicación y continuaremos realizando mejoras gradualmente sobre ella hasta que sea apta.

A lo largo del desarrollo de este sistema se han encontrado muchos problemas, la gran mayoría de ellos debido al desconocimiento sobre las tecnologías a usar, pues muchas de ellas eran novedosas para nosotros. El hecho de que Android rote las imágenes en algunos dispositivos de forma automática al capturarlas ha sido sobre todo el problema que más tiempo ha costado resolver, pues en las pruebas la imagen salía rotada en algunos dispositivos y en otros no, y no conseguíamos entender el por qué hasta haber hecho una extensa investigación en el funcionamiento de las librerías de cámara Android. El otro problema principal ha sido un error al importar el modelo del clasificador ya entrenado, que daba error en uno de los ordenadores de desarrollo y sin embargo en otro no, llevándonos a pensar que podría haber algún tipo de corrupción en el archivo del modelo. Tras probar muchas alternativas, se tomó la solución extrema de reinstalar por completo el sistema operativo, que solucionó este problema, con lo que pensamos que puede haber sido alguna incompatibilidad de programas instalados o de las librerías usadas.

En general ha sido un proyecto difícil de implementar, ha conllevado el aprendizaje de tecnologías y teoría sobre las distintas partes del sistema a implantar para poder conocer las alternativas disponibles y cuáles se adaptan mejor a nuestras necesidades.

En concreto ha sido la primera vez que se ha trabajado con la librería keras de Python, pero por suerte disponemos de un libro [26] que ha facilitado en gran medida el proceso de aprendizaje y desarrollo. Android Studio y el desarrollo de aplicaciones Android en general era una materia que desconocíamos por completo hasta ahora, al igual que la gestión e integración de aplicaciones en la nube. Esto ha representado una experiencia de aprendizaje, donde al crear un sistema que se compone de tantas partes distintas no sólo se ha comprendido un poco sobre cada componente sino la interconexión entre los mismos, y en global ha supuesto el desarrollo de un proyecto mayor a cualquiera realizado anteriormente, ya sea personal o educativo.

7.2 Trabajos futuros

Debido a la limitación temporal existente, algunas ampliaciones y mejoras deseadas no han sido posibles de implementar. La aplicación es completamente funcional, pues cumple su cometido, pero no significa que no sea mejorable o que no se le puedan agregar funcionalidades.

La primera ampliación que nos gustaría poder haber realizado es la inclusión de más especies de pájaros distintas en el clasificador de imágenes. Esto no ha sido posible debido a que para cada ave distinta se deben recolectar una gran cantidad de imágenes distintas con una buena resolución y donde el pájaro se vea correctamente, y posteriormente recortarlas manualmente una a una para utilizar en el entrenamiento y pruebas del clasificador. Consideramos que diez tipos distintos es un número razonable, pero siempre existe posibilidad de mejora, principalmente porque es una aplicación que podría adaptarse a muchos otros lugares, es decir, no sólo a la ciudad de Valencia sino a otras ciudades españolas, donde habitan otras razas de aves diferentes. De esta forma sería una aplicación más completa y con mayor capacidad de distinción entre aves, pues actualmente el clasificador solo dispone de diez clases, por lo que si el detector de objetos detecta un ave que no pertenezca a ninguna de estas clases la clasificará forzosamente en una de estas, dando así un dato erróneo y espantando un pájaro que posiblemente no se deseaba espantar.

Otra posible mejora a la aplicación es la ampliación de la distancia de detección y actuación. Ahora mismo la aplicación tiene como entrada una imagen de 608x608 píxeles. Esta resolución es la que trata como máximo YOLO antes de empezar a empeorar considerablemente el tiempo de inferencia sin mejorar significativamente la precisión de la detección. Si se hiciese uso de otro algoritmo de detección de objetos más preciso, como alguno de los de dos fases mencionado en la sección del estado del arte, se podría lograr una mejor detección de los pájaros que se encuentren más lejos de la cámara del dispositivo, permitiendo así espantar pájaros desde una distancia mayor. El mismo proceso es aplicable al clasificador, pues, aunque tiene una gran precisión, como se ha observado en las pruebas realizadas, siempre es mejorable con más muestras de cada pájaro y un modelo con más parámetros a entrenar mientras se hagan los ajustes adecuados. El motivo por el que no se han realizado estas ampliaciones ha sido el tiempo de procesamiento de la imagen: hemos tenido que tomar un compromiso de precisión a cambio de tener un rendimiento aceptable, pues

recordamos que actualmente se ejecuta en CPU todo este proceso. Sin embargo, de ser ejecutado en GPU cambiaría por completo la forma de verlo; no habría que preocuparse por estos tiempos, ya que es mucho más rápida la ejecución de estos algoritmos, permitiendo también el uso por parte de más clientes, ya que las peticiones se procesan y responden con más agilidad. No se ha realizado la ejecución con GPU por coste económico, pues como se ha mencionado en el apartado de presupuesto, resulta en un gasto considerable mensualmente.

Otra mejora que también estaba planeada es la adaptación a entornos más rurales. Esta mejora va de la mano con la primera descrita, ya que, si se añaden más especies distintas, es posible añadir algunas que no pertenezcan a ciudades sino a lugares abiertos y fuera de la urbe. El problema que impide el uso de la aplicación tal y como es ahora es la falta de cobertura móvil o de una fuente de acceso a internet con una conexión fiable. Esta dificultad se puede sobrepasar realizando la detección y clasificación en el propio dispositivo. La parte más compleja ya está implementada, pues la detección y clasificación de objetos con YOLOv4 y MobileNetv3 son idóneas para un dispositivo móvil debido a su rapidez de cómputo. Así mismo, le proporcionaríamos al usuario una manera de seleccionar si desea procesar la imagen en su propio dispositivo, permitiendo ejecutar la aplicación en un entorno sin acceso fiable a internet, o si en cambio prefiere mandar al servidor y ahorrar batería y obtener unos resultados más fiables, ya que es posible que hubiese que reducir la resolución de detección de YOLO para móviles. Esta posibilidad de seleccionar da además la ventaja de mantener todo en un entorno local, ya que hay muchos usuarios que pueden preferir no enviar sus imágenes por motivos de privacidad o porque pueden pensar que las imágenes se almacenan de alguna manera. Recalcamos que las imágenes que procesa el servidor se guardan en *buffer* y no se hace ningún tipo de uso de ellas que no sea el necesario para esta aplicación, es decir, no se guardan. Pero algunos usuarios pueden preferir, aun sabiendo esto, ejecutar el algoritmo en local, con lo que así dispondrían de esta opción.

7.3 Relación del trabajo desarrollado con los estudios cursados

Los conocimientos adquiridos a lo largo del grado de ingeniería informática han proporcionado una base estable con la que realizar este proyecto. Y aunque no se hayan visto todas las tecnologías utilizadas o cómo implementar algunas partes, resultan de gran ayuda para conocer lo suficiente de cada apartado del proyecto como para comprender cómo y dónde se debe buscar la información pertinente para continuar el desarrollo, reduciendo en gran medida el tiempo empleado para estudiar conceptos teóricos que deben ser implementados.

Para la parte de elección, entrenamiento y pruebas del clasificador, ha sido de gran ayuda haber cursado las asignaturas de *Percepción* y *Aprendizaje automático*, donde se ha trabajado con múltiples clasificadores de imágenes, aprendiendo de esta forma a entrenarlo, dividir las imágenes en conjuntos adecuados y, tras esto, probar un clasificador final con los ajustes que hayamos encontrado óptimos. En las prácticas de ambas asignaturas se ha realizado este proceso al completo un gran número de

ocasiones, lo que ha permitido no tener que investigar adicionalmente cómo realizar el proceso de entrenamiento y comprobación.

El desarrollo de la comunicación entre la aplicación Android y el servidor hubiese sido mucho más complicado sin los conocimientos adquiridos durante las asignaturas de *Redes* y *Tecnologías de sistemas de información en la red*. En concreto, en *Redes* se aprende todo lo relacionado con los distintos protocolos de comunicación, cómo funcionan y el envío y recepción de datos, tanto de forma teórica como práctica. En esta asignatura se ha realizado como ejercicio de prácticas un *chat* en Java donde se comunican entre múltiples clientes a través de un servidor, lo que proporciona una base para conocer cómo se dividen los mensajes en el protocolo TCP y poder particionarlos correctamente para su lectura desde el servidor. En *Tecnologías de sistemas de información en la red* se ha impartido la teoría correspondiente a servidores y *sockets* más al completo; en esta asignatura se explican los distintos tipos de *sockets*, además de hacer hincapié en la programación de servidores, ya que en las prácticas de la asignatura se realizan varias aplicaciones cliente/servidor y las comunicaciones asociadas. No es lo único que se ha aprendido de esta materia, pues la asignatura también trata con la creación de contenedores y sus imágenes asociadas. Esto ha sido de gran ayuda a la hora de implementar el servidor en los servicios *cloud* de Amazon, pues, aunque no es exactamente la misma tecnología, este servicio también funciona a través de imágenes y máquinas virtuales.

Por último, los conocimientos obtenidos sobre programación con el lenguaje Java a lo largo de la carrera han sido de gran utilidad para la programación en Android Studio. Estos conocimientos se han ido adquiriendo a lo largo de una gran cantidad de asignaturas del grado empezando desde el primer curso.

8. Referencias

[1] **Michael A. Davis, Gary D. Butcher, and F. Ben Mather.** *Avian Diseases Transmissible to Humans.* s.l. : University of Florida, 2015.

[2] **Ali H, Khattab S, Al-Mukhtar M.** *The Effect of Biodeterioration by Bird Droppings on the Degradation of Stone Built*. Engineering Geology for Society and Territory - Volume 8. 2014. pp. 515-520.

[3] **Murray A.** The Complete Software Project Manager: Mastering Technology from Planning to Launch and Beyond. Hoboken, NJ: Wiley; 2016.

[4] **Girshick R, Donahue J, Darrell T, Malik J.** Rich feature hierarchies for accurate object detection and semantic segmentation. IEEE Conference on Computer Vision and Pattern Recognition. 2014. pp. 580-587.

[5] **Lee S, Kwak S, Cho M.** Universal bounding box regression and its applications. Computer Vision – ACCV 2018. pp. 373–387.

[6] **Girshick R.** Fast R-CNN. IEEE International Conference on Computer Vision (ICCV). 2015. pp. 1440-1448.

[7] **Ren S, He K, Girshick R, Sun J.** Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017;39(6). pp. 1137–1149.

[8] **Pramanik A, Pal SK, Maiti J, Mitra P.** Granulated RCNN and Multi-Class Deep SORT for Multi-Object Detection and Tracking. IEEE Transactions on Emerging Topics in Computational Intelligence. 2021 Enero; 6(1). pp. 171-181.

[9] **Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg A.** SSD: Single Shot Multibox Detector. 14th European Conference on Computer Vision (ECCV), Amsterdam 2016. pp. 21-37.

[10] **Simonyan K, Zisserman A.** Very Deep Convolutional Networks for Large-Scale Image Recognition. International Conference on Learning Representations 2015 (ICLR2015). pp. 1-14.

[11] **Red Redmon J, Divvala S, Girshick R, Farhadi A.** You Only Look Once: Unified, real-time object detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. pp. 779-788.

[12] **Yuan, L.; Chen, D.; Chen, Y.-L.; Codella, N.; Dai, X.; Gao, J.; Hu, H.; Huang, X.; Li, B.; Li, C.; Liu, C.; Liu, M.; Liu, Z.; Lu, Y.; Shi, Y.; Wang, L.; Wang, J.; Xiao, B.; Xiao, Z.; Yang, J.; Zeng, M.; Zhou, L.; Zhang, P. Florence** A New Foundation Model for Computer Vision. [*arXiv preprint arXiv:2111.11432*].; 2021.

[13] **Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L.** Imagenet: A large-scale hierarchical image database. IEEE conference on computer vision and pattern recognition. 2009. pp. 248–55.

[14] **Tan M, Le QV.** EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. [*arXiv preprint arXiv:1905.11946*].; 2019.

[15] **Xie Q, Luong MT, Hovy E, Le QV, GRBT.** Self-training with Noisy Student improves ImageNet classification. Carnegie Mellon University. [*arXiv preprint arXiv:1911.04252*].; 2020.

[16] **Pham H, Dai Z, Xie Q, Luong MT, Le QV.** Meta Pseudo Labels. [*arXiv preprint arXiv:2003.10580*].; 2020.

[17] **Howard GA, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H.** MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. [*arXiv preprint arXiv:1704.04861*].; 2017.

[18] **Huff S.** Android Police. [Online].; 2022 [citado 2022 Mayo]. Disponible desde: https://www.androidpolice.com/google-chrome-drops-right-click-reverse-image-search-in-favor-of-lens/

[19] **Chennapragada A.** Google Blog. [Online].; 2018 [citado 2022 Mayo] Disponible desde: https://www.blog.google/perspectives/aparna-chennapragada/google-lens-one-year/.

[20] **Yang Q, Zhang Y, Dai W, Pan SJ.** Transfer learning. Cambridge: Cambridge University Press; 2020.

[21] Talent. [Online]. [citado 2022 Mayo] Disponible desde: https://es.talent.com/salary?job=programador.

[22] Jobted. [Online]. [citado 2022 Mayo] Disponible desde: https://www.jobted.es/salario/programador.

[23] Google Colab planes. [Online]. [citado 2022 Mayo] Disponible desde: https://colab.research.google.com/signup.

[24] Amazon Web Services t3 Instances. [Online]. [citado 2022 Mayo] Disponible desde: https://aws.amazon.com/es/ec2/instance-types/t3/.

[25] Amazon Web Services G4 Instances. [Online]. [citado 2022 Mayo] Disponible desde: https://aws.amazon.com/es/ec2/instance-types/g4/.

[26] **Géron A.** Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools, and techniques to build Intelligent Systems. Sebastopol, CA: O'Reilly Media, Inc.; 2019.

[27] **Redmon J.** YOLO: Real Time Object Detection [Online]. [citado 2022 Mayo] Disponible desde: https://pjreddie.com/darknet/yolov2/

[28] **Kedar P, Chinmay P, Sukrut A.** A Convolutional Neural Network based Live Object Recognition System as Blind Aid; 2018. https://arxiv.org/abs/1811.10399

[29] **Mikolajczyk A, Grochowski M.** Data augmentation for improving deep learning in image classification problem. International Interdisciplinary PhD Workshop (IIPhDW). 2018. pp. 117-122.

[30] **Mehta K, Kobti Z, Pfaff K, Fox S.** Data augmentation using CA evolved GANs. IEEE Symposium on Computers and Communications (ISCC). 2019. pp. 1087-1092.

[31] **Redmon J, Farhadi A.** Yolo9000: Better, faster, stronger. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. 248-255.

[32] **Redmon J, Farhadi A.** YOLOv3: An Incremental Improvement [Online]. 2018. Disponible desde: http://arxiv.org/abs/1804.02767

[33] **Bochkovskiy A., Wang C., Liao H.M.** YOLOv4: Optimal Speed and Accuracy of Object Detection. ArXiv, abs/2004.10934. 2020

[34] **Jocher G.** ultralytics/yolov5: v6.1. Zenodo. *10.5281/zenodo.4154370*. 2020

[35] Android Lifecycle. [Online]. [citado 2022 Junio] Disponible desde: https://developer.android.com/guide/components/activities/activity-lifecycle

[36] MS COCO. [Online]. [citado 2022 Junio] Disponible desde: https://cocodataset.org/

[37] **Wang C-Y, Bochkovskiy A, Liao H-YM.** Scaled-YOLOv4: Scaling cross stage partial network. IEEE/CVF Conference on Computer Vision and Pattern

Recognition (CVPR). 2021. pp. 13029-13038. pp. 1-6, doi: 10.23919/OCEANS44145.2021.9705695.

[38] **Howard A, Sandler M, Chen B, Wang W, Chen L-C, Tan M, et al.** Searching for MobileNetV3. IEEE/CVF International Conference on Computer Vision (ICCV). 2019. pp. 1314-1324.

[39] **Ye X, Zhang W, Li Y, Luo W.** MOBILENETV3-YOLOv4-Sonar: Object Detection Model based on lightweight network for forward-looking sonar image. OCEANS: San Diego – Porto. 2021. pp. 1-6.

[40] **Kleppmann M.** Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. Sebastopol, CA: O'Reilly; 2021.

[41] **Erl T.** Cloud Computing Concepts, Technology & Architecture; 2013.

[42] Master IT. IaaS vs paas vs SAAS: Qué son Y Cuál Utilizar [Online]. OpenWebinars.net. OpenWebinars; 2021 [citado 2022 Mayo] Disponible desde: https://openwebinars.net/blog/iaas-vs-paas-vs-saas-que-son-y-cual-utilizar/.

[43] Javascript. [Online]. [citado 2022 Junio] Disponible desde: https://developer.mozilla.org/es/docs/Web/JavaScript.

[44] Java. [Online]. [citado 2022 Junio] Disponible desde: https://www.oracle.com/es/java/.

[45] Visual Studio. [Online]. [citado 2022 Junio] Disponible desde: https://visualstudio.microsoft.com/es/.

[46] Android Studio. [Online]. [citado 2022 Junio] Disponible desde: https://developer.android.com/studio.

[47] Andromo. [Online]. [citado 2022 Junio] Disponible desde: https://www.andromo.com.

[48] AppGeyser. [Online]. [citado 2022 Junio] Disponible desde: https://appsgeyser.com.

[49] Microsoft Azure Virtual Machines. [Online]. [citado 2022 Junio] Disponible desde: https://azure.microsoft.com/en-us/free/virtual-machines.

[50] Amazon Web Services. [Online]. [citado 2022 Junio] Disponible desde: https://aws.amazon.com/es/free/.

[51] Google Compute Engine. [Online]. [citado 2022 Junio] Disponible desde: https://cloud.google.com/compute/.

[52] Conoce las aves del campus de Vera (UPV). [Online]. [citado 2022 Junio] Disponible desde: http://www.upv.es/noticias-upv/noticia-12572-conoce-las-ave-es.html

[53] Kaggle – Birds 400. [Online]. [citado 2022 Junio] Disponible desde: https://www.kaggle.com/datasets/gpiosenka/100-bird-species

[54] GNU Octave. [Online]. [citado 2022 Junio] Disponible desde: https://octave.org

[55] **Good, I. J.** Rational Decisions. Journal of the Royal Statistical Society. Series B (Methodological), vol. 14, no. 1. 1952, pp. 107-114.

Anexo 1 – Objetivos de desarrollo sostenible

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

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

Este proyecto en gran medida puede contribuir a diferentes objetivos de desarrollo sostenible, y aunque es subjetiva la aportación que puede tener la aplicación desarrollada a estos objetivos, hay algunos factores a los que es capaz de ayudar de forma directa o indirecta para cumplir los objetivos de la Agenda 2030.

- ODS 2: **Hambre cero.**
  El presente proyecto es de gran utilidad para este objetivo, ya que es un espantapájaros, y estos han sido utilizados a lo largo de la historia para cuidar las cosechas y mantener aves no deseadas fuera de los cultivos. Nuestra aplicación es capaz de reconocer aves urbanas, con lo que los huertos urbanos se pueden beneficiar en gran medida de su uso. Y como a medida que crecen las ciudades se dispone de más población y menos espacio para cultivar es de vital importancia aprovechar cada metro cuadrado posible.

- ODS 3: **Salud y bienestar.**
  Como ya se mencionó en la introducción, muchas aves son capaces de portar y transmitir distintas enfermedades a humanos, siendo varias de ellas muy graves; esto, en lugares donde no se disponga de un fácil acceso a medicinas o doctores, puede resultar mortal. Además, colabora hacia la meta 3.9, ya que también podría espantar a pájaros que contaminasen el agua y el suelo con sus excrementos.

- ODS 4: **Educación de calidad.**
  Al poder seleccionar los pájaros no deseados y tener una alerta sonora al detectar uno, los usuarios pueden aprender sobre los distintos tipos de pájaros de su entorno en caso de que no conozcan las especies de aves de su alrededor; por ello se le otorga un nivel bajo de vinculación a este objetivo.

- ODS 6: **Agua limpia y saneamiento.**
  En muchas regiones con menos desarrollo no se dispone de sistemas de tratamiento y saneamiento de agua, y es comúnmente obtenida de pozos u otras fuentes de agua que pueden estar contaminadas. Las aves son un factor adicional en esta contaminación, ya que muchas veces residen en las cercanías de fuentes de agua y pueden transmitir enfermedades y dañar la calidad del suministro de agua de esta forma.

- ODS 11: **Ciudades y comunidades sostenibles.**
  La aplicación desarrollada permite que las aves no se acerquen a ciertos puntos, con lo que si se implementa en zonas públicas como parques o plazas se pueden evitar ciertos tipos de pájaros no deseados o dañinos mientras que se mantienen otros que pueden ser alimentados u observados por la población

como entretenimiento y como forma de conexión con la naturaleza. Esto colabora con la meta 11.7, que ayudaría a proporcionar espacios verdes seguros donde podamos controlar las especies dañinas.

- ODS 15: **Vida de ecosistemas terrestres.**

  Muchas especies de pájaros, como los gorriones, se ven cada vez menos en entornos urbanos; sin embargo, otras como las palomas son incluso consideradas plagas. Al tener una aplicación que permita expulsar a las aves no deseadas y mantener a las que tienen más peligro de desaparición de nuestros entornos, podemos darles un espacio seguro donde comer, descansar y beber, al mismo tiempo que se evita que otras aves mucho más comunes, tomen estos recursos para ellas, promoviendo así la diversidad de especies en entornos urbanos.

- ODS 17: **Alianzas para lograr objetivos.**

  El espantapájaros selectivo es un claro ejemplo de una aplicación que beneficia al medioambiente, se aplica de una forma moderna y con herramientas actuales. Con lo que se está desarrollando una tecnología ecológicamente racional, lo que está estrechamente vinculado con la meta 17.7.