Universitat Oberta de Catalunya

uoc.edu

Sistema de monitorización de aves con tecnología Smart Cities

Emiliano Ortega Montoliu
Máster de Ingeniería de Telecomunicación (plan 2017)
Smart Cities

Rubén Molina Casanovas
Carlos Monzo Sánchez

Junio 2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Sistema de monitorización de aves con tecnología Smart Cities*
Nombre del autor:	*Emiliano Ortega Montoliu*
Nombre del consultor/a:	*Rubén Molina Casanovas*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega (mm/aaaa):	16/6/2022
Titulación:	*Máster en Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Smart Cities*
Idioma del trabajo:	*Castellano*
Palabras clave	*Smartcities, IoT, Machine Learning, Ave, Rapaz*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Muchas zonas urbanas se encuentran en contacto directo con áreas con un elevado valor natural, donde ambos entornos conviven de forma integrada. Tal es el caso de las aves rapaces que, proviniendo de poblaciones en zonas protegidas, vuelan inevitablemente sobre zonas urbanizadas las cuales se encuentran dentro de su área de campeo habitual.

En este sentido, surge la necesidad de desarrollar nuevas aplicaciones en el ámbito de las Smart Cities, que tengan como objetivo velar por una convivencia sostenible con dichos ecosistemas.

El presente trabajo pretende desarrollar un prototipo de monitorización de aves rapaces en la ciudad de Collado Villalba (situada en la sierra norte de Madrid), a través de la captura de imágenes que permitan reconocerlas y registrar su presencia, mediante tecnologías de Internet de las Cosas.

Se trata de un sistema que presenta varios retos a superar importantes, tales como la captura de imágenes con IoT, el ancho de banda de las comunicaciones necesario, o el sistema de reconocimiento a utilizar. Por ello, el trabajo analizará las diferentes opciones disponibles, con sus ventajas y limitaciones, antes de definir cada subsistema.

Por ello, el prototipo a desarrollar será también una prueba de concepto, del alcance actual de las opciones disponibles en el mercado, para construir aplicaciones en el ámbito de Smart Cities, con una perspectiva "*maker*" con presupuesto limitado.

El resultado final, deberá de servir para estimar las implicaciones de su implementación como producto final, y obtener conclusiones sobre nuevas posibilidades de desarrollo futuro.

Abstract (in English, 250 words or less):

Many urban areas are in direct contact with zones that are of significant natural value, where both environments integrally coexist. Such is the case for birds of prey, which, coming from populations in protected areas, inevitably fly over urban areas that are located within their customary home ranges.

In this sense, the need arises to develop new applications within the scope of Smart Cities, for the purpose of overseeing a sustainable coexistence for these ecosystems.

The current project intends to develop a monitoring prototype for the birds of prey in the city of Collado Villalba (located in the mountains north of Madrid), through the capture of images that will enable recognising them and recording their presence, using technologies from the Internet of Things.

This is a system that presents substantial challenges, such as the capture of images with IoT, the bandwidth needed for the communications, or the recognition system to be used. To this end, the project will analyse the different options available, with their advantages and limitations, prior to defining each subsystem.

As a result, the prototype to be developed will also be a *proof of concept* of the current scope of the options available on the market, to build applications within the area of Smart Cities, with a *maker* perspective and a limited budget.

The end result should serve to estimate the implications of its implementation as a final product, and to obtain conclusions regarding the new possibilities for development in the future.

Índice

Contenido

Lista de figuras

Lista de tablas

1. Introducción

En este primer capítulo se introducirá el contexto y justificación del trabajo, los objetivos, el enfoque y método, así como la planificación y una breve descripción de los productos obtenidos, así como de los capítulos de la memoria.

1.1 Contexto y justificación del Trabajo

Para contextualizar el Trabajo, antes efectuaré una pequeña explicación de tres conceptos relevantes:

- El concepto de Smart City normalmente se asocia a el uso de tecnología y dispositivos (*Information and communication Technology*) conectados a través de IoT (*Internet of Things* – Internet de las Cosas) para la obtención de datos los cuales se utilizan para mejorar las operaciones y servicios de la ciudad, con un mejor uso de los recursos y de forma sostenible hacia el medioambiente (Smart City - Wikipedia, 2022)

- De forma paralela, el concepto de ingeniería ecológica (*green / greening engineering*) (Habash, 2018) incide en una visión multidisciplinar de la ingeniería para ser capaz de resolver problemas complejos. Este modelo, define al ingeniero según cuatro dimensiones: científico, diseñador, conocedor de la sociedad y realizador práctico.

- En tercer lugar, mencionar el surgimiento de la denominada “cultura maker” que aboga por el aprendizaje a través del constructivismo, con unos principios basados en un “*Manifesto*” (Hatch, 2013) que consiste en: hacer, compartir, dar, aprender, utilizar herramientas, jugar, participar, apoyar y cambiar.

En este contexto, este Trabajo trata de aunar los tres conceptos en cuanto a:

- Construir un sistema de tipo Smart City, en entorno de una ciudad de tamaño medio como es Collado Villalba, la cual conecta el entorno urbano residencial, con un entorno natural, ya que se encuentra a los pies del Parque Nacional de la Sierra de Guadarrama y dentro del Parque Regional de la Cuenca Alta del Manzanares, en donde habitan gran cantidad de especies, entre ellas una numerosa y diversa población de aves rapaces.

- Para llevar a cabo dicho sistema, se requiere además de una visión multidisciplinar en el campo de la ingeniería: sistemas de telecomunicación, electrónica, servicios en la nube y *machine learning*, de otros conocimientos como la biología y el medioambiente.

- Por último, la solución a construir se desarrollará bajo una perspectiva “*maker*”, con dispositivos disponibles en el mercado.

La justificación de este sistema es la de controlar el impacto que en la población de aves rapaces pueda tener el hecho de residir de forma integrada a una ciudad como Collado Villalba. Este control, se logrará mediante la monitorización del tráfico de la población de aves rapaces, a través de su observación mediante imágenes, de forma que puedan construirse series de datos que permitan extraer conclusiones sobre sus hábitos.

Se trata de un tema relevante, porque el Parque Nacional de la Sierra de Guadarrama se encuadra dentro de las “Zonas de Especial Protección para las Aves” (ZEPA) a partir de la Directiva 79/409/CEE (Parque Nacional Sierra de Guadarrama - Aves, 2022) y cuenta con

135 especies inventariadas, entre las cuales se encuentran: el buitre negro (siendo la mayor colonia nidificante de la península) y el águila imperial ibérica (es una de las más amenazadas de nuestra fauna).

Por su parte el Parque Regional de la Cuenca Alta del Manzanares (Parque Regional de la Cuenca Alta del Manzanares, 2022) forma parte de la Red Internacional de Reservas de la Biosfera, comprende 18 términos municipales, entre ellos Collado Villalba. Además de contar con una población significativa de águila imperial ibérica, en aquellas ubicaciones más cercanas a las zonas urbanas de Collado Villalba (Peña del Águila y el Canto Hastial, 2022) se pueden apreciar otras rapaces como el buitre leonado, el águila calzada, el milano real y el milano negro.

Actualmente, los sistemas de monitorización y censado de aves se basan en la observación humana siguiendo guías metodológicas como el muestreo de campo (Las rapaces forestales en España, 2011).

Este trabajo trata de aportar un avance hacia la automatización de dicha labor en aquellas urbes muy cercanas a las poblaciones de aves rapaces, a través de un prototipo de observación, registro y análisis, de bajo consumo y operación continuada, bajo el paradigma de Smart City e IoT. Este prototipo podría asimismo evolucionar a una aplicación de actuación, cuando se observen presencias de estas aves cercanas a lugares potencialmente peligrosos como, por ejemplo, las torres de alta tensión.

La inclusión de este tipo de aplicaciones en el ámbito de las Smart Cities deberá de contribuir a conocer y mejorar el impacto medioambiental de la actividad humana en ciudades próximas a espacios de alto valor natural.

1.2 Objetivos del Trabajo

El objetivo fundamental del trabajo consiste en desarrollar un prototipo “prueba de concepto” de un sistema completo (*end to end*) que comprenda los siguientes subsistemas:

- La captura de información a través de imágenes mediante un dispositivo de bajo consumo compuesto al menos por un microcontrolador y una microcámara.
- El envio de dichas imágenes a Internet a través de alguno de los sistemas de telecomunicación disponibles para Smart Cities.
- La recepción de dichas imágenes en Internet a través de un servicio en la nube (*cloud*).
- El reconocimiento de imágenes mediante *machine learning* que identifique la presencia de un ave rapaz.
- El registro de dichos eventos con el fin de explotar la información a través de un graficado, mapa de calor, u otros.
- Adicionalmente se podría incluir un retorno desde el reconocimiento del evento para accionar algún sensor o actuador en el punto de captura de imágenes.

Un segundo objetivo del trabajo consistirá en evaluar y diagnosticar el estado de la tecnología actual en cada uno de los diferentes subsistemas mencionados, en cuanto su capacidad para la realización de proyectos “completos” con una perspectiva *maker*.

- Estado de las tecnologías IoT comprendiendo: microcontroladores, sensores y microcámaras.
- Estado de las tecnologías de sistemas de comunicación IoT.
- Estado de los servicios *cloud* y *machine learning*.

1.3 Enfoque y método seguido

El enfoque y método serán modulares, es decir a partir del estudio realizado para cada subsistema se establecerá la mejor solución en cada caso, considerando aquel que mejor se ajuste a los requerimientos, así como las limitaciones existentes.

En relación con el subsistema de captura de información mediante microcontroladores, microcámaras y sensores, analizaremos los módulos más utilizados en el mercado, tales como Arduino, Raspberry Pi y ESP32/8266. Asimismo, se analizará la problemática específica de utilizar cámaras y sus implicaciones. El resultado en este caso será una aplicación específica que utilice una combinación y configuración de dichos componentes.
En cuanto los sistemas de comunicación, se resumirán las diferentes opciones tales como: WiFi, Zigbee, LoRa, Sigfox y 4G, con sus características y limitaciones. La elección final deberá de cumplir los requerimientos del prototipo, es decir, será capaz de enviar fotografías en tiempo real.

Por último, analizaremos las herramientas disponibles para integrar los dispositivos de captura de datos con Internet, para su almacenamiento, visualización, reconocimiento, así como las interfaces que utilizaremos para su programación.

El enfoque priorizará alcanzar una solución en todos los subsistemas con el fin de asegurar una solución final "*end to end*".

1.4 Planificación del Trabajo

A continuación, incluimos un diagrama Gantt con la planificación del trabajo prevista.

- La primera fase consiste en desarrollar la idea, su justificación y objetivos.
- La segunda analizará el estado del arte, lo cual nos servirá para tomar decisiones de qué opción utilizar en cada subsistema del prototipo.
- La tercera consiste en desarrollar la solución con sus esquemas de circuitos y sistemas, así como los códigos de programación.
- La cuarta es donde razonaremos las principales conclusiones y aprendizajes, así como posibles vías de desarrollo futuro.

Dicho plan queda reflejado en el siguiente Gantt:

Diagrama Gantt

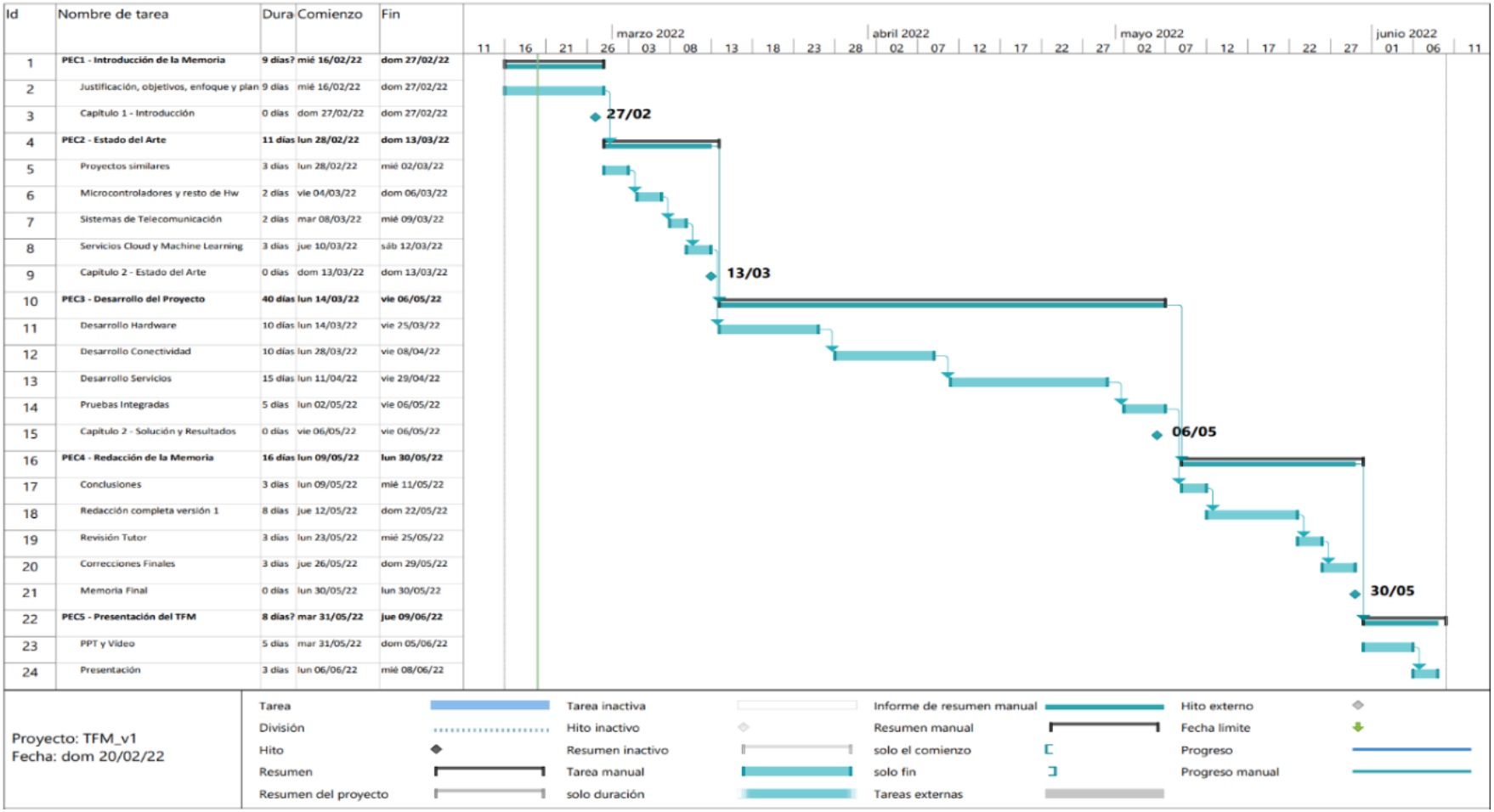


Ilustración 1- Diagrama Gantt

1.5 Breve sumario de productos obtenidos

A continuación, enumeramos los productos obtenidos:

- Desarrollo software en Arduino IDE para subir por wifi imágenes a un servidor a través de la tarjeta con cámara integrada ESP32 CAM. Código: **Sketch_TFM.ino**
- Desarrollo software para subir por wifi imágenes a un servidor web PHP a través de la tarjeta con cámara integrada ESP32 CAM. Código: **Upload.php**
- Desarrollo software para subir imágenes a un servidor local Node.js a través de la tarjeta con cámara integrada ESP32 CAM. Código: **Upload.js**
- Desarrollo software para utilizar el API AutoML desarrollado a medida en Google Cloud Platform para realizar predicciones de clasificación y clasificar las imágenes en una determinada carpeta del servidor. Código: **Indexreco.php**
- Modelo personalizado de reconocimiento en la nube en función de una colección de imágenes de entrenamiento: **projectId = 'averapaz-87893', location = 'us-central1', modelId = 'ICN5993585217565097984'**

En el siguiente esquema, podemos ver la relación de los productos obtenidos como parte de una única solución:

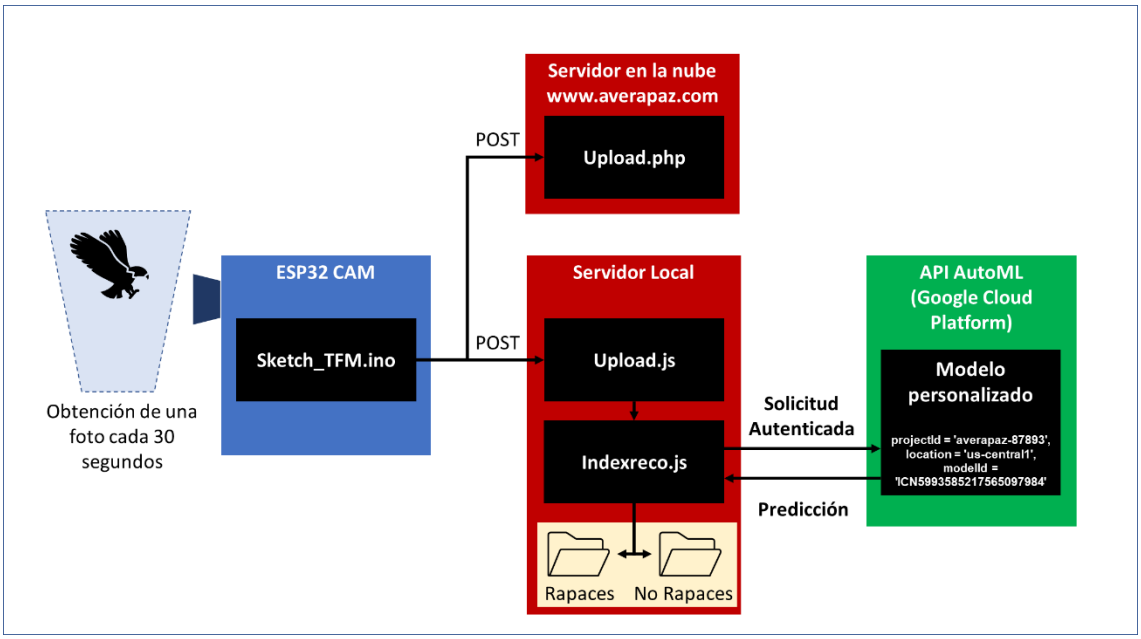


Ilustración 2 Resumen de productos obtenidos

1.6 Breve descripción de los otros capítulos de la memoria

Una vez vista la introducción, la memoria se compone de los siguientes capítulos:

Estado del Arte. - En este capítulo revisaremos algunos de los estudios más relevantes para el reconocimiento de aves, desde el uso de radares, a la extracción automática de características de imágenes para aplicar inteligencia artificial.

Posteriormente haremos un recorrido por el concepto y arquitectura actual de Internet de las Cosas, desde los sensores hasta la nube, para finalmente, analizar algunas de las aplicaciones más relevantes para utilizar *machine learning* como una de las partes fundamentales del trabajo.

Desarrollo de la Solución. – Este apartado presenta y desarrolla el problema del reconocimiento de aves, como un problema de clasificación de un vector de características. A continuación, analizaremos las diferentes alternativas de implementación de cara a tomar la mejor decisión para el proyecto.

Una vez elegida la solución a desarrollar, se explicarán las tres partes diferenciadas de las que consta: desarrollo del hardware y software sobre la placa y cámara para subir imágenes al servidor, desarrollo del software del servidor que recibe las imágenes y, por último, el desarrollo del algoritmo de *machine learning* en la nube y su utilización mediante un API.

Conclusiones. - En este aparado analizaremos las lecciones aprendidas del proyecto y haremos una reflexión sobre los objetivos alcanzados, así como idoneidad de la metodología seguida en el mismo. Finalmente, enumeraremos las líneas de trabajo a futuro más relevantes dentro de esta área de desarrollo.

Glosario, Bibliografía y Anexos. – Estos últimos capítulos, además de contener todas las referencias utilizadas en la memoria, y los acrónimos más relevantes, incluye los códigos fuente de los desarrollos y productos obtenidos a lo largo del proyecto, salvo el modelo *machine learning* personalizado, el cual reside en el host de Google con el identificador indicado en la memoria.

2. Estado del Arte

En el segundo capítulo analizaremos el estado del arte sobre los temas tratados en el trabajo. Para lo cual, se ha estructurado en tres secciones: contexto y métodos utilizados en el reconocimiento de aves, Internet de las cosas, su arquitectura y panorama actual, y aplicaciones en *machine learning* y procesamiento de imágenes.

2.1 Contexto y métodos utilizados en el Reconocimiento de Aves

El reconocimiento de aves a través de la tecnología es un ámbito de investigación que ha sido abordado desde diferentes perspectivas, y con diferentes aplicaciones. Si nos basamos en la situación de observación sobre el ave, podemos encontrarnos con dos casos: en vuelo o en reposo. Dependiendo de dicha situación, será posible aplicar diferentes tecnologías para su reconocimiento. A continuación, hacemos un breve repaso del estado del arte:

- **Identificación de aves en vuelo**. Dentro de estos estudios, se encuentran:

    - **Los basados en radares**. Este tipo de métodos se remontan a hace casi 60 años (Eastwood, 1967), y permiten identificar la altitud y distancia de aves que se encuentran desde unos pocos metros hasta 100km de distancia. Así, por ejemplo, en el siguiente trabajo (Zaugg, Saporta, Loon, Schmaljohann, & Liechti, 2008) se diferencia entre señales producidas por pájaros, insectos y objetos difusos (nubes o lluvia), basados en los patrones de intensidad de señal que producen debido al aleteo de las alas.

    - **Los basados en imágenes**. En ese apartado, encontramos estudios orientados a la detección para evitar la colisión con aerogeneradores (R. Yoshihashi, 2017) en donde se compara la aplicación de métodos clásicos de clasificación (Freund Y, 1995) como el *Adaboost*, que clasifica dos clases en función de la suma ponderada de características, frente a los más novedosos de *deep learning* basados en CNN, (redes neuronales de convolución) (LeCun Y, 1998). Es interesante ver cómo llega a la conclusión de que ambos métodos se comportan de forma similar.

      Por otro lado, cuando lo que se trata de analizar son áreas amplias, se suelen utilizar imágenes tomadas desde aeronaves; en la siguiente referencia (Dominique Chabot, 2016) encontramos un resumen sobre algunas de las diferentes técnicas y algoritmos aplicados.

      De forma más reciente, (Marin & Bodgan, 2019) efectuó un estudio con observación desde la superficie mediante varias cámaras apuntando al cielo, para identificar las aves como un primer paso para la clasificación de estas. Se trató de una prueba de concepto en la que se grabaron vídeos de aves para posteriormente efectuar una extracción de datos con OpenCV (Opencv.org, 2022) para su análisis con Keras (Keras.io, 2022) (ambas herramientas software las veremos más adelante). En este caso, se utilizó un algoritmo de detección de movimiento, para en caso positivo, identificar si se trataba de un pájaro u otro objeto como por ejemplo un avión. En este estudio las siluetas de las aves detectadas fueron guardadas para estudios posteriores cuyo objetivo consiste en la identificación de especies.

    - **Los basados en extracción de características del vuelo**. En este caso destaca el trabajo según la modelización de su esqueleto (Tianhang WU1, 2017). Este novedoso método, obtiene un vector de características basado en 4 ejes del ave, e identifica diversos patrones de dicho vuelo o poses: vuelo con aleteo, vuelo en saltos, y vuelo en deslizamiento. Este estudio se utilizó para la identificación de aves y prevención de accidentes en zonas de tráfico aéreo.

- **Identificación de aves en reposo**. El objetivo es el de reconocer la especie de ave concreta, por lo que se trata de un proceso que obtienen muchas más características y más precisas que cuando se trata de aves en vuelo. Aquí podemos encontrar las siguientes técnicas:

    - **Las basadas en el reconocimiento de imágenes.** El ejemplo más relevante de *machine learning* es el caso de la aplicación Merlin (Cornell, 2022). Esta aplicación permite a través de un software específico denominado *eBird* que los usuarios puedan clasificar la especie observada con ayuda de los algoritmos de dicha aplicación. Además, podrán también subir las fotos obtenidas, ampliando y mejorando la base de datos a nivel mundial, generando así una sinergia entre las personas que participan y la inteligencia artificial (Kelling, y otros, 2013).

    - **Las basadas en el análisis del canto.** Aquí podemos encontrar diversos tipos de análisis. En este caso, el método de representación de la señal permite dividir los métodos entre los que utilizan la transformada de Fourier de los que no lo hacen, en especial los métodos de predicción lineal (Dan Stowell, 2011).

      En cuanto a la aplicación y método, encontramos desde las basadas en redes neuronales para su reconocimiento (Incze, Jancso, Szilagyi, & Sulyok, 2018), a los estudios de sonogramas para identificar diferentes dialectos dentro de una determinada especie de ave (Baker, 2001). En ese caso también encontramos aplicaciones disponibles como el caso de BirdNet (Cornell-birdnet, 2022).

A partir de los casos estudiados, podemos concluir que cuando se trata de clasificar a un ave según su especie, la observación en reposo resulta más precisa. Mientras que cuando se encuentran en vuelo, la principal aplicación consiste en identificar su presencia, ya que su clasificación resulta más difícil.

2.2 Internet de las cosas, su arquitectura y panorama actual

A diferencia de los casos estudiados, nuestro proyecto pretende efectuar un prototipo de observación y reconocimiento de aves rapaces basado en un modelo de arquitectura IoT. Para ello comenzaremos por efectuar una breve descripción de IoT y la arquitectura que lo caracteriza.

El término IoT fue acuñado por Kevin Ashton en 1999 (Thorsten Kramp, 2013), y dispone de diferentes definiciones que podemos encontrar en (Sharma, Madhavi, & Inderjit, 2018):

- El IAB (*Internet Architecture Board*) lo define como un servicio de comunicaciones donde IoT representa una gran cantidad de dispositivos incorporados denominados "*Smart Objects*" que se pueden comunicar sin necesidad de intervención humana.

- IEEE *Communications Magazine*, lo define como un marco en donde cada objeto se identifica de forma única en Internet, eliminando el "gap" entre el mundo real y virtual a través de comunicaciones M2M (*Machine to machine*)

- Por su parte, el IETF (*Internet Engineering Task Force*) define "Smart Object" como un dispositivo con limitación de potencia, memoria y ancho de banda, capaz de interoperar con la red y otros *smart objects*.

- Por último, según (Atzori, 2010) IoT combina tres ideas clave: el middleware orientado a Internet, sensores orientados a "cosas" y el conocimiento orientado a la semántica. El potencial de IoT se basa en la interacción de estas tres ideas clave.

En cuanto a su arquitectura, (Syed, Sierra-Sosa, Kumar, & Elmaghraby, 2021) establece que Internet de las Cosas unifica las operaciones de sensorización de datos, transmisión y recepción de la información, así como el procesado y almacenamiento a través de una serie de servicios en la nube, lo cual supone una arquitectura de 5 niveles:

- **Capa de sensores**: También denominada "capa de percepción", la cual se compone de una serie de sensores que obtiene los datos de interés y los actuadores que actúan sobre los objetos físicos. En nuestro caso, en esta capa se encontrará la microcámara (como sensor principal), sensores y microcontroladores.

- **Capa de red**: La información de los sensores es transmitida mediante tecnologías de red inalámbricas denominadas tipo LPWAN (Low Power Wide Area Networks) que ofrecen un alcance variable tanto de velocidad de datos como de alcance. La elección de la tecnología a aplicar dependerá de la carga de datos que sean necesarios transmitir, así como de la distancia respecto el *gateway* de acceso a Internet.

- **Capa de software intermedio (middleware)**: Esta capa facilita una interfaz genérica hacia el hardware de la capa de sensores y facilita los datos a la capa de aplicación.

- **Capa de aplicación**: Esta capa utiliza los datos a través de APIs y servicios de bases de datos sobre la capa de software intermedio (*middleware*). Actualmente encontraremos una oferta amplia de servicios en la nube orientados a IoT.

- **Capa de negocio**: Está unida a la capa de aplicación con el objetivo de desarrollar estrategias y establecer políticas para gestionar el sistema. En esta capa encontraremos al menos, los análisis y representación de los datos obtenidos en el sistema.

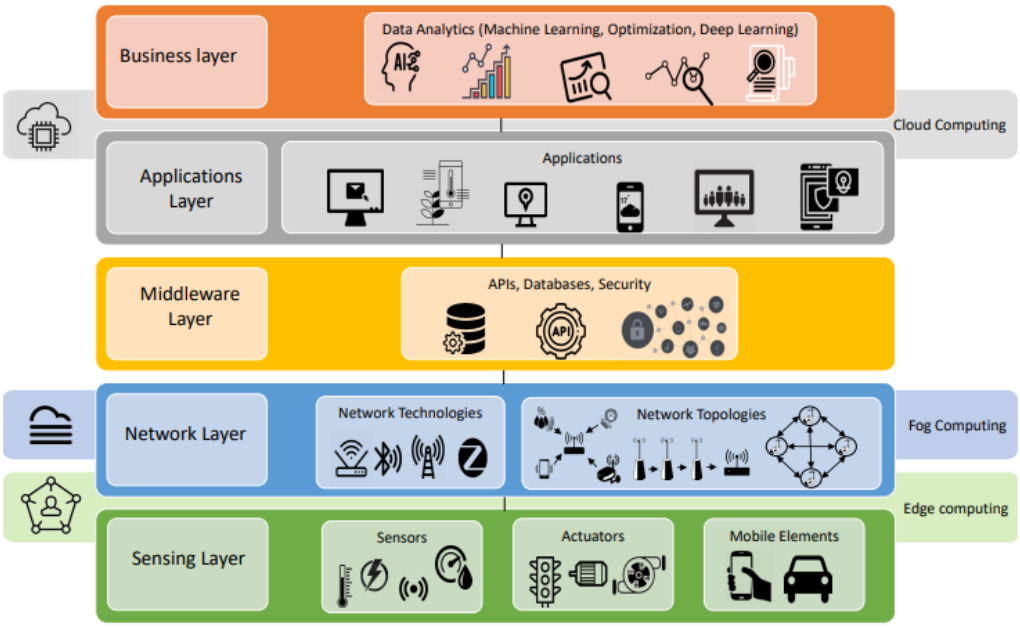


Ilustración 3 Arquitectura IoT (Syed, Sierra-Sosa, Kumar, & Elmaghraby, 2021)

Una clasificación de tipos de arquitectura interesante es aquella que la categoriza en función de las responsabilidades de operación en cada capa del sistema, así pues, podemos hablar de tres tipos de arquitectura:

- **Cloud**: Este modelo se basa en la premisa de que el procesamiento principal de los datos se realizará en la nube. Este es un punto importante en nuestra aplicación, ya que, si el reconocimiento de las imágenes se efectúa en la nube, es necesario transmitir dicha información, lo cual determina una elevada carga de datos en la capa de red. Esto reduce el número de opciones a la hora de elegir una tecnología LPWAN.

- **Edge**: Este modelo consiste en realizar el procesado y tomar las decisiones en los propios nodos del sistema o a nivel de "cosa". De esta forma se reduce enormemente la utilización de recursos de red. En nuestro caso, si el procesamiento de imágenes y la decisión se tomase en el microcontrolador, estaríamos en un sistema tipo *edge*, donde toda la lógica estaría incorporada (en inglés encontramos el término *embedded*) en el microcontrolador y no sería necesario transmitir todos los datos capturados, sino sólo los resultados del su proceso. En este modelo, sería posible utilizar una tecnología LPWAN con una velocidad de datos mucho más baja.

- **Fog**: Este modelo es intermedio sobre los dos anteriores. En este caso, el procesamiento de la información se realizaría a nivel de red local de sensores, normalmente en el *router* de la misma. En nuestro caso podría aplicar si se utiliza un dispositivo intermedio para el acceso a Internet, por ejemplo, un móvil al que se conecte el microcontrolador.

En los siguientes apartados, analizaremos el estado del arte de las tecnologías de IoT siguiendo las diferentes capas del modelo de arquitectura aplicados a nuestro caso.

2.2.1 Capa de sensores: microcontroladores y microcámaras.

Microcontroladores

La parte fundamental que conecta los sensores con la red es el microcontrolador, el cual, a diferencia del microprocesador, está orientado a operaciones más sencillas y repetitivas.

No obstante, en los casos donde queramos implementar arquitecturas de tipo Edge, será necesario contar con microcontroladores más potentes capaces de ejecutar programas, por ejemplo, de *machine learning* (TinyML, 2022).

Para revisar el estado del arte en este apartado, realizaremos un breve análisis sobre el panorama actual de placas con microcontrolador, enfocado al colectivo *maker*, a través de algunos de los fabricantes más conocidos como Arduino, Espressif y Raspberry Pi.

Arduino

Se trata de la empresa más conocida en crear placas con microcontroladores de hardware libre dirigidos al gran público. Sus productos tienen Licencia Pública, de forma que cualquier otra empresa puede fabricar y distribuir placas similares, debido a lo cual han surgido clones tales como Elegoo, DFRobot, AZDelivery entre otros.

A nivel histórico, hay que indicar que surgió como proyecto en 2005 dirigido a estudiantes del *Interaction Design Institute Ivrea* (IDII) en Italia, de hecho, el nombre de Arduino se corresponde a un rey de Italia (Arduino de Ivrea) del año 1002 (Arduino, 2022). Sus placas están segmentadas en cuatro grupos principales

Core (Core, 2022): Aquí encontramos las placas más básicas y originales, como el Arduino UNO. Son ideales para comenzar a trabajar en dicho entorno dada su sencillez. Estas placas utilizan por lo general un procesador ATmega328 u otras variedades superiores. Se trata de un procesador de tan solo 8 bits a 16MHz, con 32KB de memoria Flash y una SRAM de 2KB, con lo cual nos podemos dar una idea de su reducida capacidad.

A continuación, mostramos una tabla con las diferentes opciones en dicha familia.

Placa	Factor de Forma	Procesador	Tipo	Memoria	Pines	Radio	Conexión
UNO	69X53mm	ATmega328p	8 bits 16 MHZ	32KB SRAM 2KB	14digital / 6analógicos	-	USB
UNO WIFI	69X53mm	ATmega4809	8 bits 16 MHZ	48KB 6KB	14digital (5PWM) / 6analógicos	WIFI Bluetooth&BLE	USB
MEGA 2560	102X53MM	ATmega2560	8 bits 16 MHZ	256KB 8KB	54digital(15PWM) /16analógicos/ 4UARTs	-	USB
DUE	102X53MM	AT91SAM3X8E	32 bits 84MHZ	512KB 96KB	54digital(12PWM) /12analógicos/ 4UARTs	-	USB
MICRO	48X18MM	ATmega32u4	8 bits 16 MHZ	32KB SRAM2,5KB	20digital (7PWM) / 12analógicos	-	USB
NANO	45x18mm	ATmega328	8 bits 16 MHZ	32KB SRAM 2KB	22digital(6PWM) / 8analógicos	-	USB

Tabla 1 Arduino Core. Elaboración propia a partir de (Core, 2022)

Nano (Nano, 2022): Se trata de una familia de placas con un factor de forma y consumo muy reducidos. Por lo general cuentan con conectividad WIFI, Bluetooth y Bluetooth Low Energy. Además, incorporan algunos sensores de forma directa, lo cual las hace muy interesantes para su uso directo en algunas aplicaciones. El microprocesador también es más avanzado que en el caso de Core, ya que parten del ARM M0 SAM D21, de 32 bits y 48MHz, con 256KB de memoria Flash, disponen de un ARM M4 NRF52840, que amplía la capacidad hasta 1MB y finalizan con un RP2040, es decir el mismo que el de una Raspberry Pi “Pico” con 16MB de memoria Flash.

Quizás lo más interesante de estas placas es que ofrecen la posibilidad de realizar proyectos con *Tiny Machine Learning* (tinyML, 2022) en varios ámbitos, entre ellos la visión artificial.

Placa	Factor de Forma	Procesador	Tipo	Memoria	Pines	Radio	Conexión	Otras Características
NANO 33 IoT	45x18mm	ARM M0 SAM D21	32 bits 48MHz	256KB SRAM32KB	14digital(11PWM) /8analógicos/ 1UARTs	WIFI Bluetooth&BLE	USB mini	6 axis IMU
NANO 33 BLE	45x18mm	ARM M4 NRF52840	32 bits 64MHz	Flash 1MB SRAM 256KB	14digital(14PWM) / 8analógicos/1UART	Bluetooth&BLE	micro USB	9 axis IMU versión SENSE: incorpora MIC, movimiento, temperatura, humedad y presión, y color, brillo, proximidad y gestos
NANO RP2040	45x18mm	RP2040	32 bits 133MHz	Flash 16MB SRAM 264KB	20digital(20PWM) / 8analógicos/1UART	WIFI Bluetooth&BLE	micro USB	mic 6 axis IMU OpenMV license

Tabla 2 Arduino Nano. Elaboración propia a partir de (Nano, 2022)

MKR (Mkr, 2022): Es una familia de placas orientada a la rápida creación de prototipos y proyectos IoT por parte de ingenieros y *makers*, todas ellas con el mismo factor de forma. Estas placas están basadas en el microprocesador ARM M0 SAM D21 antes mencionado, y suelen disponer de conectividad integrada según el modelo: WIFI, Bluetooth/BLE, LoRa, Sigfox, GSM y LTE M1/NB1.

Además, existe un modelo que incorpora una FPGA para su programación. A continuación, vemos un resumen comparativo de los diferentes modelos:

Placa	Factor de Forma	Procesador	Tipo	Memoria	Pines	Radio	Conexión	Otras Características
MKR WIFI 1010	62x25mm	ARM M0 SAM D21	32 bits 48MHz	256KB	8digital(13PWM) /7analógicos/ 1UARTs	WIFI Bluetooth&BLE	Full speed USB	Orientado a servicios IoT
MKR WAN 1300/1310	68x25mm	ARM M0 SAM D21	32 bits 48MHz	Flash256KB / SRAM32KB 1310: external2MB	8digital(13PWM) /7analógicos/ 1UARTs	LoRa	Full speed USB	Orientado a servicios IoT
MKR FOX 1200	68x25mm	ARM M0 SAM D21	32 bits 48MHz	Flash256KB / SRAM32KB	8digital(13PWM) /7analógicos/ 1UARTs	Sigfox	Full speed USB	Orientado a servicios IoT SigFox Learning Program
MKR GSM 1400	68x25mm	ARM M0 SAM D21	32 bits 48MHz	Flash256KB / SRAM32KB	8digital(13PWM) /7analógicos/ 1UARTs	GSM	Full speed USB	Orientado a servicios IoT
MKR NB 1500	68x25mm	ARM M0 SAM D21	32 bits 48MHz	Flash256KB / SRAM32KB	8digital(13PWM) /7analógicos/ 1UARTs	LTE M1/NB1	Full speed USB	Orientado a servicios IoT
MKR VIDOR 4000	68x25mm	ARM M0 SAM D21 FPGA INTEL CYCLONE	32 bits 48MHz	Flash256KB / SRAM32KB 8MB SRAM (FPGA)	8digital(13PWM) /7analógicos/ 1UARTs	WIFI Bluetooth&BLE	Full speed USB	FPGA programable Orientado a servicios IoT

Tabla 3 Arduino Mkr. Elaboración propia a partir de (Mkr, 2022)

PRO (Pro, 2022): Es una familia orientada a aplicaciones industriales, y consta con la placa Portenta H7 con un doble procesador capaz de ejecutar en tiempo real, código a alto nivel como MicroPython y JavaScript. Asimismo, soporta TensorFlow Lite para sistemas de visión artificial. Esta placa dispone de una *shield* de visión, compuesta por una cámara de 324x324 píxeles, orientada al reconocimiento e identificación de objetos y formas.

Además, encontramos la placa Nicla Vision, con el mismo procesador que Portenta y además incorpora una cámara de 2MP y diversos sensores. Soporta programación en TinyML y MicroPython, orientada a las aplicaciones de visión artificial.

Placa	Factor de Forma	Procesador	Tipo	Memoria	Pines	Radio	Conexión	Otras Características
Portenta H7	62x25mm	STM32H747 incluye ARM M7 y ARM M4	32 bits 480 y 240MHz	Flash 16-128MB SRAM 8-64MB	80 pines 4UARTs Interface Camara	WIFI Bluetooth&BLE	Display Port - USB C	Orientado a industria, visión artificial, robótica, etc.
Nicla Vision	23x23mm	STM32H747 incluye ARM M7 y ARM M4	32 bits 480 y 240MHz	Flash 16-128MB SRAM 8-64MB	17 pines y 8 fins	WIFI Bluetooth&BLE	Micro USB	Cámara 2MP IMU 6 ejes Sensor Distancia Micrófono

Tabla 4 Arduino PRO. Elaboración propia a partir de (Mkr, 2022)

Espressif

Se trata de una compañía china que ha desarrollado una serie de placas basadas en procesadores Xtensa, fundamentalmente con altas prestaciones a un precio más económico que en el caso de Arduino.

Inicialmente comenzó con el microcontrolador ESP8266 que posteriormente evolucionó al ESP32 en diversas opciones que vemos en la tabla resumen adjunta (Espressif, 2022)

Existen varias placas que utilizan este procesador bajo diferentes marcas: Ardafruit, Sparkfun. Asimismo, presenta numerosas presentaciones como: ESP32 SX1278 LoRa, ESP 32 con OLED entre otras.

Placa	Procesador	Tipo	Memoria	Pines	Radio
ESP8266	Xtensa LX106 - Single Core	hasta 160MHz	RAM160KB	32	WIFI
ESP32-S2	Xtensa LX7 - Single Core	hasta-240MHz	SRAM320KB ROM128KB RTC-SRAM16kb	56	WIFI
ESP32-C3	RISC-V - Single Core	hasta-160MHz	SRAM400KB ROM384KB RTC-SRAM8KB	32	WIFI Bluetooth&BLE
ESP32-WROOM	Xtensa LX6 - Dual Core	hasta-240MHz	SRAM520KB ROM448KB RTC-SRAM16kb	48	WIFI Bluetooth&BLE
ESP32-S3	Xtensa LX7 - Dual Core	240MHz	SRAM512KB ROM384KB RTC-SRAM16kb	56	WIFI Bluetooth&BLE

Tabla 5 Espressif. Elaboración propia a partir de (Espressif, 2022)

Raspberry Pi

Esta placa, se ideó para contener todo lo necesario para ser un ordenador de bajo coste, ya que puede ser conectada a un monitor, teclado y ratón. Por tanto, más que un microcontrolador se considera una computadora o miniordenador. Fue creada en 2012 por la Raspberry Pi Foundation, con un enfoque educativo en el Reino Unido (Raspberry_Pi, 2022). Los modelos iniciales (Pi-1 y Pi2) fueron sustituyéndose por modelos más actualizados (Pi3 y Pi4) que potencian las prestaciones en cuanto capacidad del microprocesador, memoria RAM y conexiones.

A continuación, mostramos un resumen de todas estas placas con sus características principales (Raspberry_Pi_Org, 2022):

Placa	Factor de Forma	Procesador	Tipo	Memoria	Pines	Radio	Conexión	Otras Características
PICO	51x21mm	ARM M0 RP2040	32 bits 133MHz	Flash 2-16MB SRAM 264KB	26digital(16PWM) / 3analógicos/2UART /2SPI/2I2C		micro USB	
ZERO 2 W	65x30mm	ARM A53 BCM2710	64 bits, 1GHz	SRAM 512MB	40 a través de HAT	Versión W incluye: Bluetooth&BLE WIFI	2 micro USB (1 power)	1 Conexión HDMI CSI Camara Port MicroSD
PI 3 A+	65x57mm	ARM A53 BCM2837	64 bits 1,4GHz	SRAM 512MB	40 a través de HAT	Bluetooth&BLE WIFI	1 USB 2.0	1 Conexión HDMI DSI Display Port CSI Camara Port Video port Micro SD
PI 3 B+	86x57 mm	ARM A53 BCM2837	64 bits 1,4GHz	SRAM 1GB	40 a través de HAT	Bluetooth&BLE WIFI	4 USB 2.0 y GigaEthernet (USB 2.0)	1 Conexión HDMI DSI Display Port CSI Camara Port Video port Micro SD
PI 4 B	86x56 mm	ARM M72 BCM2711	64 bits 1,5GHz	SRAM 2-8GB	26digital(16PWM) / 3analógicos/2UART /2SPI/2I2C	Bluetooth&BLE WIFI	4 USB (2:3.0 y 2:2.0) GigaEthernet 1 USB-C (alimentación)	2 Conexión HDMI 4k DSI Display Port CSI Camara Port Video port Micro SD

Tabla 6 Raspberry Pi. Elaboración propia a partir de (Raspberry_Pi_Org, 2022)

En general, utilizan dos fatores de forma: el B, que se corresponde a un tamaño de placa *standard*, y el A, donde se logra un factor de tipo “compacto”. Además de estas opciones, se creó la versión ZERO como una versión “ultra compacta” respecto a las anteriores.

Sin embargo, en 2021 se lanzó la versión PICO, que sí podríamos considerar un microcontrolador, de forma similar que el Arduino NANO RP2040, si bien en el caso de PICO no dispone de conectividad radio, lo cual supone una limitación para aplicaciones IoT.

Microcámaras

Las microcámaras digitales no dejan de ser sensores con la particularidad de capturar una gran cantidad de datos en función de su resolución (Megapíxeles). Los tres aspectos que analizaremos serán: la interfaz de la cámara, los sensores y las lentes.

La interfaz

La interfaz con el microcontrolador es un aspecto muy importante ya que limita la utilización de cámaras con mayor resolución según la capacidad del microcontrolador. Así pues, las cámaras con mayor resolución / fps (*frames per second*) utilizarán interfaces más complejos (MIPI_CSI_2, 2022), siendo necesario recurrir a miniordenadores tipo Raspberri Pi. Mientras que la mayor parte de los microcontroladores analizados soportan interfaces de cámaras con menor resolución / fps (SPI_Arduino, 2022)

Interfaz MIPI. - Es una interfaz serie entre el sensor y el procesador anfitrión definido por el *Mobile Industry Processor Interface Alliance*. Esta interfaz permite elevadas tasas de transmisión para video de alta resolución (vídeo 4k y 8k), presentando además bajo consumo y bajo nivel de interferencia electromagnética (MIPI_CSI_2, 2022).

Interfaz SPI. - Los microcontroladores vistos anteriormente cuentan con un interfaz serie denominado SPI, cuya velocidad de transmisión viene limitada por la velocidad de reloj (por ejemplo, en Arduino UNO, es de 16MHz). Dado que interfaces de los sensores son en paralelo, es necesario adaptar dicha señal a los interfaces serie (SPI) de los microcontroladores en cada caso.

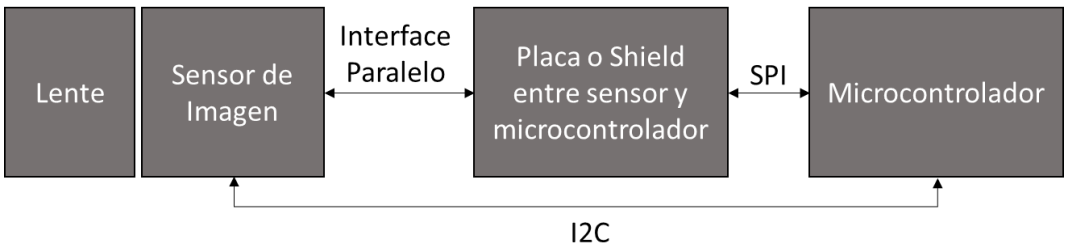


Ilustración 4 Interfaz SPI. Elaboración propia a partir de (Arducam, 2022).

Por este motivo, estas cámaras suelen venir acompañadas de una placa o un *shield*, transforman la interfaz paralela a SPI. Además, hemos de ser conscientes que estos microcontroladores tienen una memoria SRAM muy limitada, de forma que no es posible almacenar una imagen, al menos en los más básicos. Por este motivo, las placas que acompañan a estas cámaras disponen de una memoria buffer para almacenar la imagen.

Sensores y lentes de cámara

Los sensores de imagen orientados a microcámaras pueden variar desde una resolución de 0,1MP (megapíxel) hasta 13MP o más. A continuación, mostramos una tabla resumen con los fabricantes más destacados (Arducam, 2022).

Omnivision						
Modelo	Resolución	Pixel	Matriz	Color	Frame Rate	Aplicación
OV7675	0,3MP	2,5µm	640x470	RGB	30fps	Multimedia
OV7251	0,3MP	3µm	640x480	Mono	100fps	Machine Vision
OV9281	1MP	3µm	1280x800	Mono	60fps	Machine Vision
OV2640	2MP	2,2µm	1600x1200	RGB	15fps	IoT/Wildlife
OV5640/42	5MP	1,4µm	2592x1944	RGB	15fps	IoT/Wildlife
OV13550	13MP	1,12µm	4224x3136	RGB	30fps	PC Multimedia
Aptina						
Modelo	Resolución	Pixel	Matriz	Color	Frame Rate	Aplicación
MT9V034/22	0,36MP	6µm	752x480	RGB	60fps	Machine Vision
MT9M001	1,3MP	5,2µm	1280x1024	RGB/Mono	30fps	Industrial/Medical
MT9D111	2MP	2,8µm	1600x1200	RGB	15fps	IoT
MT9P031	5MP	2,2µm	2592x1944	RGB	14fps	Wide FOV
MT9N001	9MP	1,75µm	3488x2616	RGB	9,7fps	Industrial/Medicina
MT9J001/3	10MP	1,67µm	3856x2764	RGB/Mono	7,5fps	Industrial/Medicina
MT9N001	14MP	1,4µm	4608x3288	RGB	13,7fps	Industrial/Medicina
Sony						
Modelo	Resolución	Pixel	Matriz	Color	Frame Rate	Aplicación
IMX477	12MP	1,55µm	4056x3040	RGB	60fps	Industrial/Medicina
IMX135	13MP	1,12µm	4208x3120	RGB	24fps	Consumer
IMX298	16MP	1,12µm	4656x3496	RGB	30fps	Consumer

Tabla 7 Sensores de Imagen. Elaboración propia a partir de (Arducam, 2022)

Además del sensor, es muy importante la lente que se aplique y su distancia focal, ya que esta determinará el FOV (Field of View) de la imagen.

Así pues, las fotos normales, suelen tener una distancia focal entre 40mm y 60mm, mientras que por encima de 60mm se considera telefoto, que es la más apropiada para fotografiar un objetivo lejano como pudiera ser un ave (Length, 2022).

Otro aspecto interesante, son las fotos en ángulo amplio, las cuales utilizan lentes entre 24mm y 35mm. Estas fotos tienen la ventaja de abarcar una un ángulo / FOV mayor, por lo que son capaces de capturar la presencia de aves en un espacio más amplio, en cuyo extremo tendríamos las lentes *fish-eye* (ojo de pez).

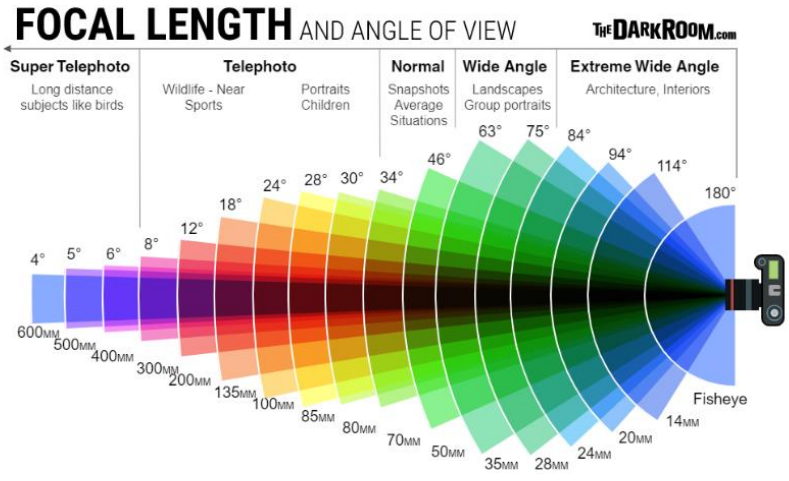


Ilustración 5 Distancia Focal. Elaboración propia a partir de (Length, 2022).

2.2.2 Capa de red: LPWAN

Uno de los aspectos de estas tecnologías de radio inalámbricas es el adaptarse a las aplicaciones IoT con unos requerimientos de bajo consumo energético, y el poder operar en áreas geográficamente amplias (Qadur, y otros, 2018). Dentro de estas tecnologías, tenemos aquellas que utilizan frecuencias libres de licencia (*Industrial, Scientific and Medical*) (ISM, 2022) o con licencia. En función de su alcance encontramos:

Corto alcance (todas en frecuencias libres de licencia ISM 5 GHz / 2,5GHz / 900 MHz).

- **ZigBee**. - Está basada en la especificación IEEE 802.15.4, y es adecuada para cubrir áreas pequeñas (en torno a 100 metros) y unas velocidades de transmisión en torno a 250kbps. Se trata de una tecnología capaz de soportar diferentes topologías de nodos (malla, árbol y estrella) con buenos niveles de escalabilidad.

- **Bluetooth/BLE**. - El estándar es IEEE 802.15.1, el cual opera en rangos de hasta 30 metros y velocidades de 30Mbps. Dentro de esta tecnología, encontramos Bluetooth de baja energía BLE, especialmente adecuado para operar con muy bajo consumo energético.

- **Wifi**. - Con diferentes versiones del estándar IEEE 802.11 permite mayores velocidades de que en los casos anteriores, con mejores niveles de latencia, y operando en rangos superiores a 100 metros con un incremento importante del consumo energético. En cambio, una de sus últimas versiones, IEEE 802.11 ah **WiFiHaLow**, permite conectividad con bajo consumo y largo alcance (hasta 1 km), operando a una frecuencia de 900MHz. En la siguiente tabla observamos las diferentes versiones de WiFi (802.11ah, 2022) y (WiFi, 2022).

Acrónimo	WiFi1	WiFi2	WiFi3	WiFi4	WiFi5	WiFi6	HaLow
Versión	802.11b	802.11a	802.11g	802.11n	802.11ac	802.11ax	802.11ah
Banda	2,4GHz	5GHz	2,4GHz	2,4GHz	5Ghz	2,4/5Ghz	900MHz
Velocidad Máxima	11Mbps	54Mbps	54Mbps	600Mbps	6,9Gbps	9,6Gbps	347MBps
Alcance externo	140m	120m	140m	250m	300m	300m	1km

Tabla 8 Tipos de WiFi. Elaboración propia a partir de (WiFi, 2022)

Largo alcance (en frecuencias ISM de 900 MHz o licenciadas LTE).

- **LoRa**. - Se trata de una tecnología propietaria de Semtech que cubre entornos de hasta 5km en entornos urbanos y 15km en entornos rurales. Utiliza la banda ISM de 868MHz en Europa, y presenta unos niveles de velocidad limitados entre 0,3 y 50kpbs. Su topología es estrella y es muy indicada para redes de sensores con bajo consumo energético y bajo tráfico de datos.

  Las redes que utilizan LoRa se denominan LoRaWAN, y una de las redes más importantes es la construida por la comunidad de contribuidores voluntarios TheThingsNetwork, la cual dispone actualmente en España (thethingsnetwork.org, 2022) 55 comunidades y 393 *gateways* disponibles para su uso gratuito, bajo unas determinadas normas en la red. Así, por ejemplo, existen 4 *gateways* disponibles en la zona de Collado Villalba y el Parque Nacional de la Sierra de Guadarrama.

- **SigFox**. - Es también una solución propietaria que opera en una banda ISM similar a la anterior, si bien es capaz de alcanzar distancias aún mayores, entre 10km y 50km para entornos urbanos y rurales respectivamente, aunque con velocidades más bajas (subida entre 100 y 600bps, y bajada 600bps) con un nivel de latencia elevado. Tiene la ventaja de operar en una red privada ya instalada, aunque existe un límite de mensajes diarios de subida y bajada. Se trata de una muy buena opción para aplicaciones en zonas rurales.

- **NB-IoT**. - Esta tecnología denominada Narrow Band IoT, ha sido definida por 3GPP está basada en LTE, siendo accesible a través de operadores con licencia. Permite coberturas de hasta 10km y velocidades (NB2) 127kbps en bajada y 159kbps en subida (Narrowband_IoT, 2022). Su principal ventaja es que al utilizar una banda licenciada hace que sea más fiable que en los casos anteriores.

- **LTE-M**.- Se trata de otra tecnología definida por 3GPP accesible asimismo por operadores con licencia. En este caso, la velocidad alcanza (LTE M2) 4MBps en bajada y 7Mbps en subida.

En el siguiente gráfico (Van Linh Nguyen, 2019) observamos las diferentes tecnologías en función de la velocidad de transmisión y el alcance. También podemos observar la diferenciación de las tecnologías con y sin licencia, en donde las ofrecidas por los operadores presentan ventajas significativas (NB-IoT y LTE M) sobre sus equivalentes.

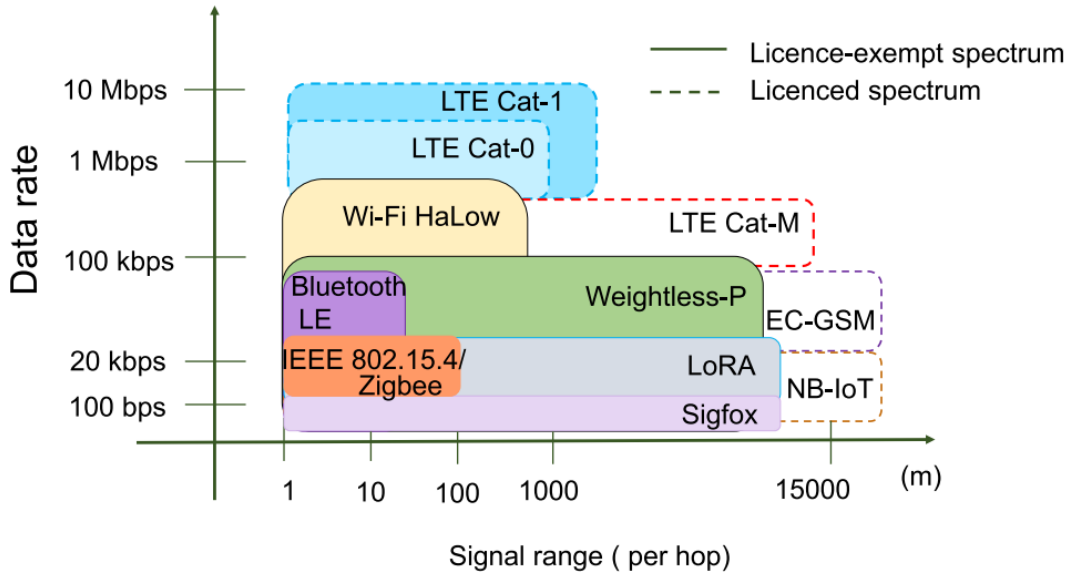


Ilustración 6 Tecnologías LPWAN (Van Linh Nguyen, 2019)

2.2.3 Capa de middleware, aplicaciones y negocio

En este apartado analizaremos algunas de las plataformas más conocidas.

Arduino Cloud. - (Arduino_Cloud, 2022) Se trata de una plataforma orientada a microcontroladores tipo Arduino. Su funcionamiento se basa en crear representaciones de "cosas" (*things*) y las variables que se desean monitorizar. Esta plataforma permite la sincronización de variables entre dispositivos conectados, dispone de un planificador de trabajos y es capaz de subir código a los dispositivos además de poder integrarse con otros servicios. Asimismo, permite efectuar cuadros de mando de forma visual sobre las variables que estamos monitorizando.

Thinger.io. - (thinger_io, 2022) Es una plataforma de código abierto que permite gestionar una serie de dispositivos, y almacenar una serie de parámetros denominados *data buckets*. Estos datos pueden ser actualizados con una frecuencia máxima de un minuto y ser visualizados a través de un cuadro de mandos.

ThingSpeak. - (ThingSpeak, 2022) Es una plataforma capaz de ejecutar código Matlab, por lo que permite efectuar análisis de datos avanzado y visualización a través de dicha herramienta sin necesidad de pagar la licencia. Utiliza APIs RESTful y el protocolo MQTT (MQTT, 2022) desarrollado para aplicaciones M2M, pudiendo ser sus datos utilizados desde otras APIs.

Dispone de un planificador y alertas. En este caso, permite almacenar datos con una limitación de 8 campos de cualquier tipo, 3 para localización y 1 de estado. La tasa de actualización es cada 15 segundos.

Blynk. - (Blynk_io, 2022) Se trata de una aplicación móvil y un servidor en la nube. La aplicación móvil consiste en un entorno visual para poder interactuar con el microcontrolador de forma virtual mediante una serie de *widgets* como botones o controladores. Para poder conectarse con la aplicación, se utiliza en servidor Blynk en la nube, si bien, también da la opción de descargarse el código del servidor, ya que es de código abierto, y ejecutarse desde un pc o incluyo una Raspberry Pi (Blynk, 2022).

Cayenne my devices. - (Cayenne, 2022) Es una aplicación móvil con un entorno visual basado en *widgets*, que permite monitorizar los sistemas desarrollados con el microcontrolador. Esta aplicación presenta la ventaja de poder lanzar notificaciones cuando suceden los eventos detectados en la placa. Funciona mediante el protocolo MQTT (MQTT, 2022)

Propuestas de las grandes empresas digitales. - Las soluciones más destacadas para ofrecer servicios IoT en la nube son: Google Cloud, Amazon Web Services IoT (AWS) o Azure IoT Hub (Microsoft). Estas propuestas tienen la ventaja de integrar con otros servicios ofrecidos, incluyendo los de *machine learning* que luego veremos.

2.3 Aplicaciones en *machine learning* y procesamiento de imágenes

2.3.1 Conceptos previos

Sin entrar en profundidad, el término Inteligencia Artificial es un concepto bastante amplio que se acuñó en una conferencia de Dartmouth en 1956 (Dartmouth, 2022) y normalmente se asocia a tareas de razonamiento realizadas por computadoras que normalmente realizarían humanos (Artificial_Intelligence, 2022).

Como subconjunto de la Inteligencia Artificial, surge el *machine learning*, que consiste en que los humanos entrenan a las máquinas para que éstas sean capaces de reconocer ciertos patrones y establecer predicciones. En este apartado podemos encontrar diferentes técnicas tales como: clasificadores lineales, árboles de decisión, técnicas K-NN (vecino más próximo) y redes neuronales (Machine_Learning, 2022).

Por último, *deep learning*, se refiere a técnicas donde la máquina es capaz por sí misma de extraer las características a partir de los datos. Normalmente utiliza técnicas basadas en redes neuronales convolucionales (CNN), es decir cuando una capa toma como input la salida de una capa previa. En este caso el aprendizaje puede ser supervisado o no supervisado (Vargas, 2017)

A continuación, mostramos los pasos necesarios a seguir por un sistema de *machine learning*, desde la captura de una imagen hasta su clasificación:

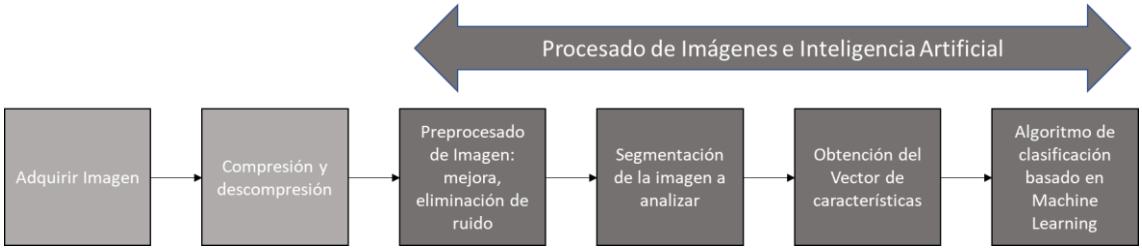


Ilustración 7 Proceso de clasificación de imágenes. Elaboración propia a partir de (Machine_Learning, 2022)

2.3.2 Aplicaciones de procesamiento de imágenes y *machine learning*

A continuación, vamos a realizar un breve repaso de las herramientas software de código abierto más significativas en la actualidad:

OpenCV. - (OpenCV, 2022) Es una serie de librerías para llevar a cabo el preprocesamiento de imágenes, segmentación y extracción del vector de características para posteriormente aplicar un algoritmo de inteligencia artificial.

OpenCV integra entornos de desarrollo de *machine learning* tales como: TensorFlow, PyTorch y Caffe. Además, incorpora librerías propias basadas tanto en redes neuronales como en algoritmos de reconocimiento de patrones, como por ejemplo el algoritmo k-NN. Estas librerías soportan los lenguajes de programación más comunes, C++, Java, Python y también tiene un interfaz con Matlab.

TensorFlow. - (TensorFlow, 2022) Es un entorno de *machine learning* creado por Google. Esta herramienta cuenta con una serie de librerías que nos permiten entrenar modelos

basados en redes neuronales, soportando los lenguajes de programación más comunes (C++, Python, Java). Este entorno se utiliza fundamentalmente para, a partir del vector de características de la imagen, clasificarla sobre una serie de categorías predefinidas, reconocer si se trata o no de un determinado patrón, o segmentar dicha imagen.

TensorFlow suele ser elegida por personas que requieren del uso de redes neuronales sin tener gran conocimiento de éstas y empresas centradas en mejorar sus servicios, más que en el conocimiento científico que haya detrás.

TensorFlow Lite. - (TensorFlow_Lite, 2022) Es una versión de TensorFlow para poder ser ejecutada en el propio dispositivo donde se realiza la captura de la imagen. Dicha versión se encuentra preparada para ser utilizada tanto en dispositivos móviles como en miniordenadores o en microcontroladores. El poder disponer de la lógica de reconocimiento incorporada en el microcontrolador presenta una serie de ventajas muy importantes:

- Menores requerimientos de conectividad, pues no es necesario enviar la imagen.
- Consumo de energía reducido por el menor uso de las comunicaciones.
- Mejor latencia, ya que no es necesario procesar en el servidor.
- Menores requerimientos de privacidad, ya que las imágenes no son enviadas.
- Tamaño más reducido y simplicidad del modelo

Keras. - (Keras.io, 2022) Es una librería de código abierto en Python para crear modelos de *deep learning*. Se trata de una herramienta para facilitar el rápido prototipado en diferentes modelos de redes neuronales. Está construida con TensorFlow e integrada en dicho entorno.

PyTorch. - (PyTorch, 2022) Es un conjunto de librerías enfocadas a Python desarrolladas por Facebook y suponen la principal alternativa a TensorFlow. Además, tienen capacidad de ejecutarse en GPU (unidad de proceso gráfico).

En comparación con TensorFlow, PyTorch dispone de un interfaz y librería más sencillos y permite utilizar redes neuronales a bajo nivel más fácilmente. Esta opción es elegida por investigadores o empresas cuyo objetivo se centra en la Inteligencia Artificial (DeepLearning, 2022).

En 2017 PyTorch incorporó el entorno Caffe (Caffe, 2022), cuyas siglas indican la aplicación de dicho entorno, es decir, aplicaciones basadas en redes neuronales convolucionales (CNN) para su rápida incorporación (*convolutional architecture for fast feature embedding*)

Además de las plataformas de servicios IoT que vimos antes de las grandes empresas digitales, éstas ofrecen sus propios servicios de *machine learning*:

- **Vertex AI y AutoML Image**. - Dentro de Google Cloud, encontramos Vertex AI como una plataforma integrada para preparar, almacenar, entrenar, probar y gestionar modelos de aprendizaje desarrollados por el usuario. Por su parte AutoML Image es otra herramienta de Google que permite a través de un API (tipo REST o RPC) detectar objetos y clasificar imágenes.

- **SageMaker Neo**. - Se trata de la herramienta de AWS para aprendizaje automático. Presenta compatibilidad con los modelos de desarrollo vistos anteriormente, ya que los puede transformar a un formato común ejecutable para el hardware específico que se trate, incluidos dispositivos Edge. Además, se encuentra integrado con los servicios de AWS IoT.

- **Azure Machine Learning**. - Es la solución de Microsoft para este tipo de aplicaciones. Combina herramientas que facilitan la preparación de datos y construcción de modelos sin necesidad de programación.

3. Desarrollo de la solución

3.1 Presentación del problema de reconocimiento

Aunque a priori, la idea es utilizar un algoritmo de *machine learning*, he visto adecuado realizar un análisis previo de la naturaleza de las imágenes que vamos a utilizar con el fin de razonar el comportamiento del sistema de reconocimiento. En este caso las imágenes que se prevén analizar serán siempre obtenidas desde una base en vertical, lo cual implica que se espera que la mayor parte de las aves sean fotografiadas como una silueta en vuelo.

La forma de dichas siluetas serán la información de base para entrenar el modelo, por lo que realmente el tamaño de estas no debe de ser un factor relevante, ya que el tamaño dependerá de la distancia a la cámara. De esta forma, la resolución de la cámara no debería de ser el factor que mejorase mucho el comportamiento salvo para capturas muy lejanas.

Es importante entender cuáles son las características principales por la que el modelo tomará la decisión de clasificar en una u otra categoría. Nuestro problema es aparentemente sencillo ya que sólo hay dos posibles decisiones: rapaz o no rapaz, aunque también cabe una tercera que sería: ninguna de las dos.

A priori, cuando se estudia una muestra representativa de las especies que van a representar cada clase, y desde un punto de vista heurístico, la primera característica por la que clasificaríamos si un ave es rapaz sería la envergadura de sus alas en vuelo.

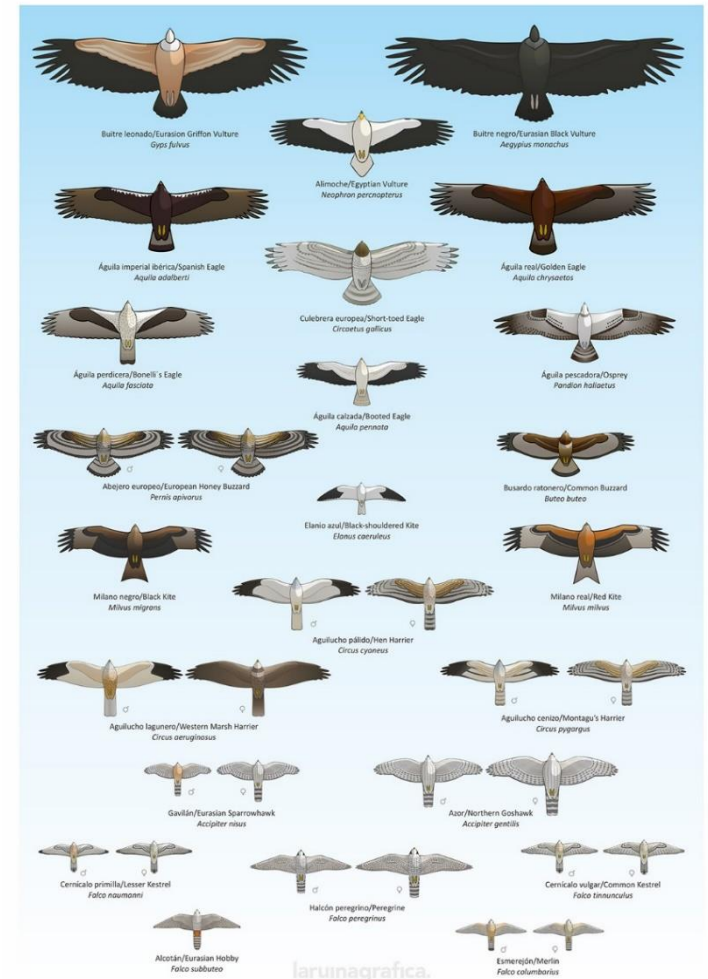


Ilustración 8 Siluetas de aves rapaces (Rapaces diaurnas, 2022)

En la ilustración adjunta (Rapaces diaurnas, 2022), se observa como todas estas aves poseen una silueta con una relación entre el ancho de las alas y el largo del cuerpo ciertamente significativa, en particular las de mayor tamaño. A esta característica la vamos a denominar “relación rectangular”.

En este estudio no se van a incluir todas las aves rapaces sino una muestra de las más significativas en la zona prevista de observación (Red Natura, 2022). Asimismo, para el resto de las aves, también las hemos acotado a las más frecuentes de la zona. Por este motivo será importante comprender en qué medida la “relación rectangular” (RR) es determinante para clasificar en las categorías “rapaz” o no “rapaz”.

En la siguiente fotografía podemos observar dicha característica a través de un rectángulo amarillo girado que rodea la silueta del ave (RR: 3,45).

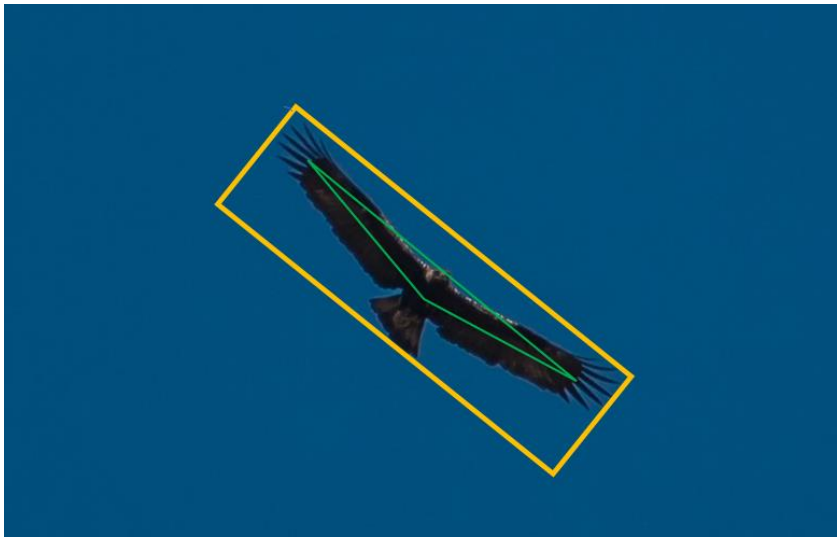


Ilustración 9 Ejemplo de Segmentación de "relación rectangular" y "ángulo de planeo"

Si continuamos con el razonamiento heurístico, y en búsqueda de una segunda característica, se observa que las aves rapaces tienen unas alas muy rectas y con un ángulo hacia el exterior. Esta característica la podemos ver en la foto anterior, donde se aprecia un par de líneas verdes que nacen en el centro del cuerpo del ave, y van hasta el extremo de las alas por el punto medio. Finalmente se unen estas dos líneas formando un triángulo hacia el exterior.

A esta segunda característica, representada por el ángulo de cada lado del triángulo verde respecto el lado mayor del rectángulo amarillo, la he dominado “ángulo de planeo”. En el caso de la foto sería de +11º (AP: +11º)

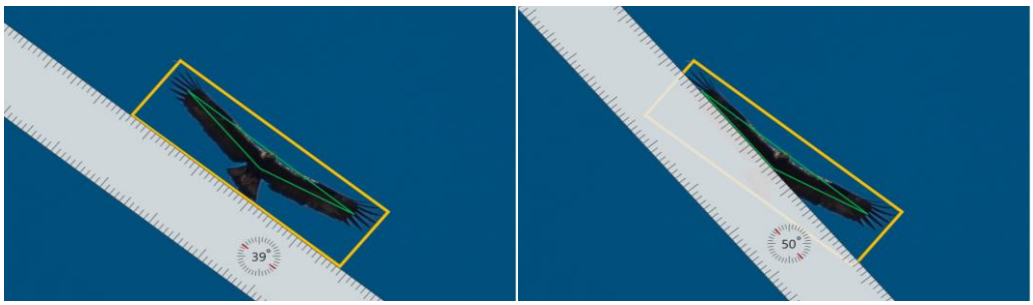
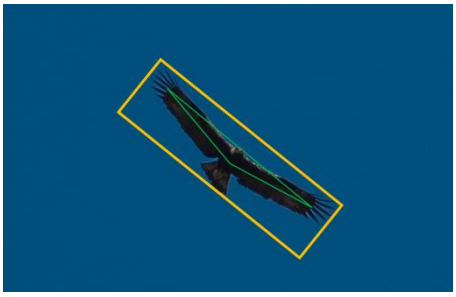
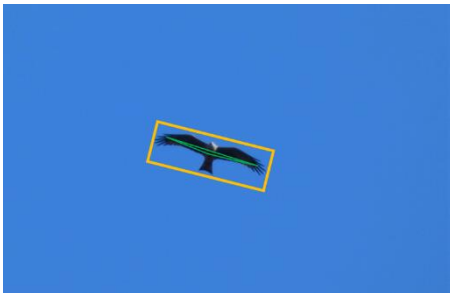
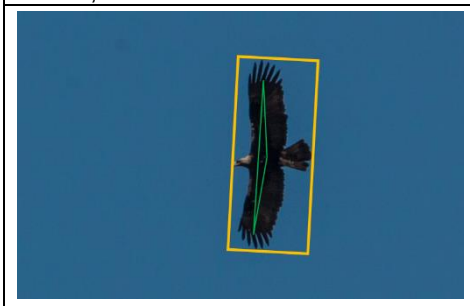
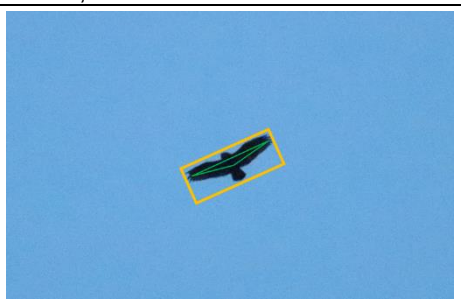
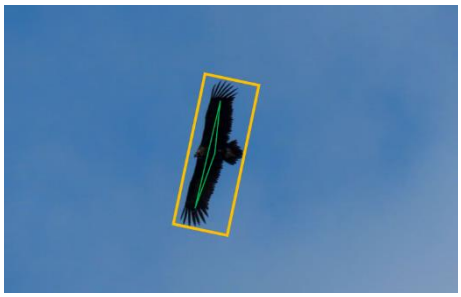
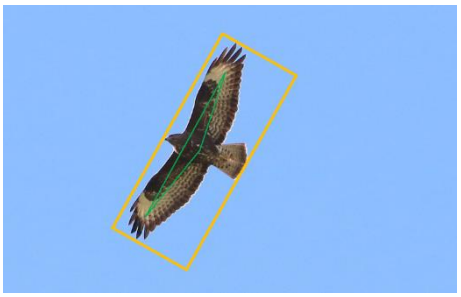
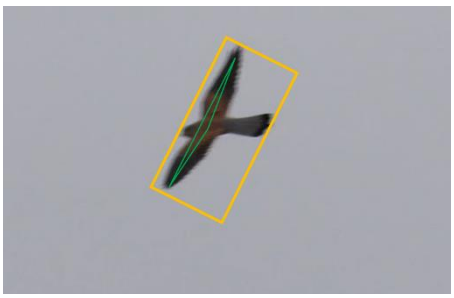
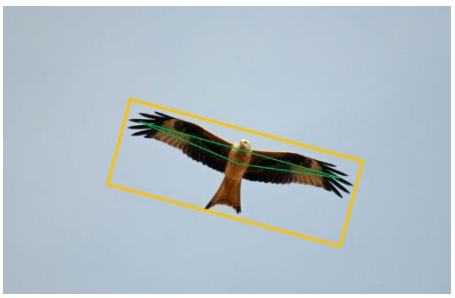
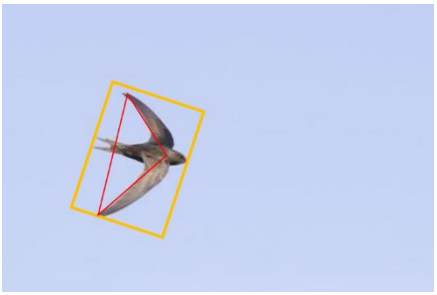
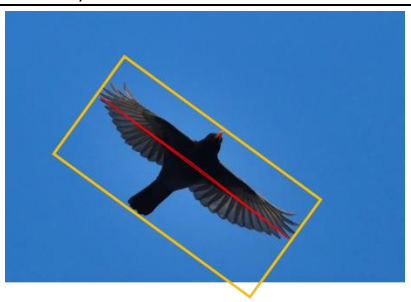
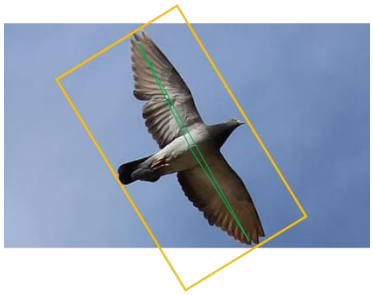
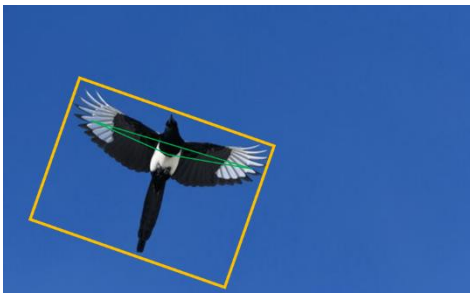


Ilustración 10 Ejemplo de medición del "ángulo de planeo" con una regla digital

A continuación, mostramos estas características en la muestra de aves rapaces:

Aguila Imperial Ibérica RR: 3,45. AP:+11º	Milano Negro RR: 2,92. AP:+6º
	
Águila Real RR: 2,97. AP:+6º	Buitre Leonado RR: 2,74. AP:+12º
	
Águila Culebrera RR: 2,41. AP:+6º	Búho Real RR: 2,54. AP:+6º
	
Buitre Negro RR: 2,78. AP:+7º	Busardo Ratonero RR: 2,63. AP:+8º
	
Cernícalo RR: 2,11. AP:+6º	Milano Real RR: 2,73. AP:+7º
	

A continuación, mostramos estas mismas dos características en la muestra de aves no rapaces:

Cigüeña RR: 2,02. AP:-6º	Golondrina RR: 1,38. AP:-12º
	
Gorrión Común RR: 1,34. AP:+6º	Mirlo Común RR: 2,02. AP:+0º
	
Paloma Común RR: 1,63. AP:+2º	Urraca RR: 1,37. AP:+6º
	
Vencejo RR: 2,11. AP:-6º	
	

En estos casos, contamos con aves cuyo “ángulo de planeo” es hacia la cola, es decir, negativo. Destacando la golondrina y el vencejo cuyas alas son muy redondeadas. Esta característica tiene una amplia variabilidad entre las aves no rapaces, pero sin embargo nos puede ser de gran ayuda cuando exista duda con la “relación rectangular”.

Si representamos estos datos en un gráfico de dos ejes, podemos diferenciar el área donde se encuentra cada tipo de ave en función de dichas características:

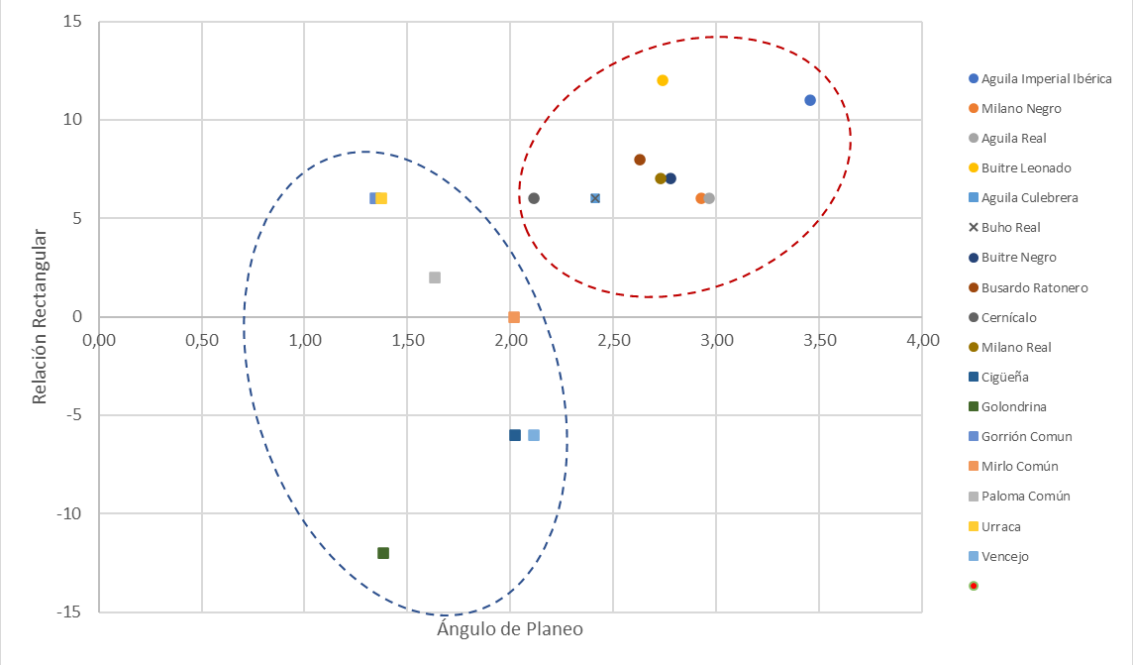


Ilustración 11 Agrupaciones de aves según sus características

En este gráfico se ve como el ave no rapaz con mayor "relación rectangular" es el vencejo con 2,11, por lo que podría ser confundida con un ave rapaz. Sin embargo, al presentar un ángulo de planeo negativo, quedará bien clasificada dentro del área de las aves no rapaces.

De forma equivalente, entre las rapaces, el cernícalo presenta una menor "relación rectangular", si bien presenta asimismo un ángulo de planeo positivo de +6º, por lo que también quedará correctamente clasificada según estas dos características.

Esta gráfica nos permite implementar, de una forma muy sencilla, el algoritmo de K-NN o de vecino más próximo visto en el capítulo 2, el cual consiste en, ante la aparición de un ave a clasificar, hacerlo según las k distancias euclídeas más cercanas, siendo k un número impar para evitar los empates.

Así pues, hallaríamos la distancia del nuevo elemento x con respecto los "p" elementos (y_p) de la muestra.

$$\forall\, y_p \;;\; d_p(x, y_p) = \sqrt{\sum_{i=1}^{n} (x_i - y_{pi})^2}$$

Una vez tenemos todas las distancias, seleccionamos los k primeros con menor distancia, para determinar la pertenencia al de la mayoría más cercana.

A continuación, veremos un ejemplo donde aparece un ave desconocida justo en mitad de las dos distribuciones, por lo que si queremos aplicar el método K-NN procederemos a calcular las distancias sobre los vecinos más cercanos.

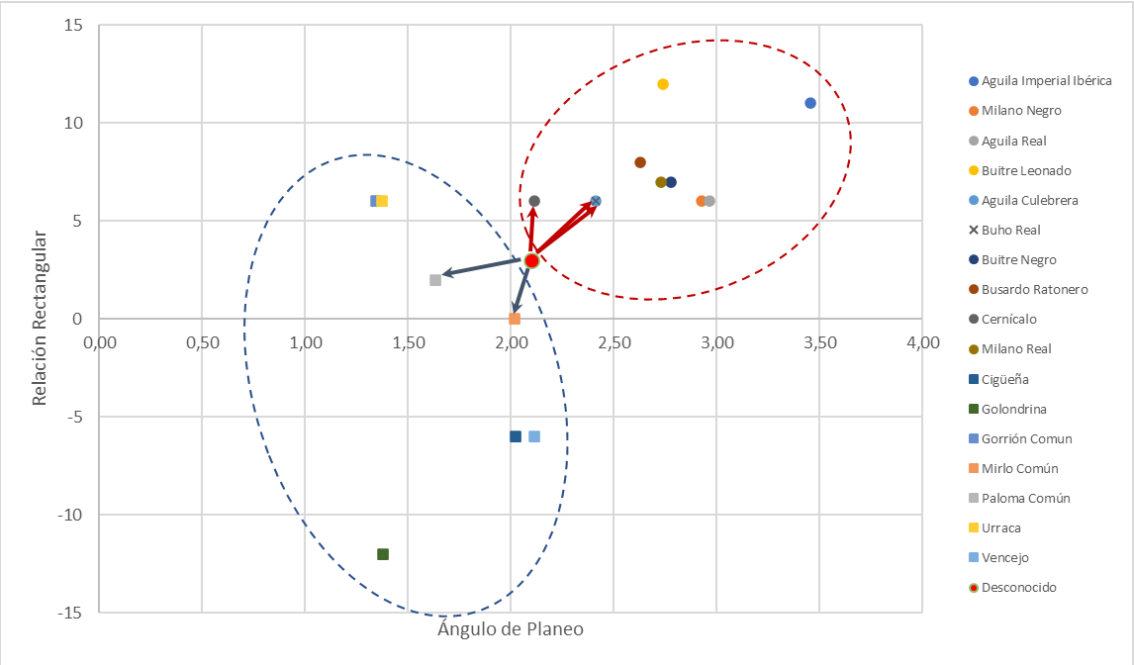


Ilustración 12 Representación gráfica del método K-NN con distancia euclídea

En este caso, dependiendo del valor de K, clasificaremos el ave como no rapaz (K=1 y K=3) o como rapaz (K=5). Aquí se puede apreciar la importancia de que K sea impar.

Ave	Relación Rectangular	Ángulo Planeo	Tipo	Distancia	K=1	K=3	K=5
Desconocido	**2,10**	**3**	**?**	**0,00**	**no rapaz**	**no rapaz**	**rapaz**
Paloma Común	1,63	2	no rapaz	1,10	no rapaz	no rapaz	no rapaz
Cernícalo	2,11	6	rapaz	3,00		rapaz	rapaz
Mirlo Común	2,02	0	no rapaz	3,00		no rapaz	no rapaz
Águila Culebrera	2,41	6	rapaz	3,02			rapaz
Búho Real	2,54	6	rapaz	3,03			rapaz
Urraca	1,37	6	no rapaz	3,09			
Gorrión Común	1,34	6	no rapaz	3,09			
Milano Negro	2,92	6	rapaz	3,11			
Águila Real	2,97	6	rapaz	3,12			
Milano Real	2,73	7	rapaz	4,05			
Buitre Negro	2,78	7	rapaz	4,06			
Busardo Ratonero	2,63	8	rapaz	5,03			
Águila Imperial Ibérica	3,45	11	rapaz	8,11			
Vencejo	2,11	-6	no rapaz	9,00			
Cigüeña	2,02	-6	no rapaz	9,00			
Buitre Leonado	2,74	12	rapaz	9,02			
Golondrina	1,38	-12	no rapaz	15,02			

Tabla 9 Ejemplo práctico de la aplicación del algoritmo K-NN

Uno de los problemas de trabajar con las distancias, es que no se tiene en cuenta la variabilidad de cada característica, por lo que en ocasiones se normaliza con la desviación típica con el fin de dar mayor o menor peso según la confiabilidad de la característica. Esta normalización se denomina Distancia Euclídea Ponderada (Clasificador de mínima distancia, 2022):

$$\forall\, y_p \;;\; d_{wp}(x, y_p) = \sqrt{\sum_{i=1}^{n} \left(\frac{1}{\sigma_i} (x_i - y_{pi}) \right)^2}$$

Otra cuestión para tener en cuenta es cuando no hay homogeneidad entre las características, es decir, cuando son de diferente naturaleza y por tanto tengan diferentes escalas. En nuestro caso es así, si bien al no haber mucha diferencia de escala, este aspecto no se nota significativamente.

Sin embargo, además de dicho efecto puede que las características estén relacionadas entre sí. Para tener todo esto en cuenta, utilizaremos la Distancia de Mahalanobis, la cual, partiendo de la distancia ponderada vista anteriormente en formato vectorial, donde S es la matriz diagonal que contiene la varianza de cada variable:

$$d_w(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})\, \mathrm{S}^{-1} (\vec{x} - \vec{y})}$$

Esta distancia resuelve el problema de las escalas, pero no de la interrelación de variables, para lo cual, utilizaremos la matriz de covarianzas ∑.

$$d_m(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})\, \Sigma^{-1} (\vec{x} - \vec{y})}$$

En el siguiente gráfico observamos los efectos de distribuir los elementos a clasificar en tres tipos de espacio. En primer lugar, tenemos un espacio euclídeo, donde las distancias son iguales en todas las direcciones. En segundo lugar, un espacio normalizado por la diagonal de desviaciones típicas, lo cual normaliza esta variabilidad de las características.

Por último, vemos un espacio donde hemos aplicado la distancia de Mahalanobis, en la que al incluir la matriz de covarianza estamos incluyendo tanto la ortogonalidad, rotando los ejes, como la escala, donde las líneas de distancia son elipses, pero rotadas (ComputaciónClasificación, 2022).

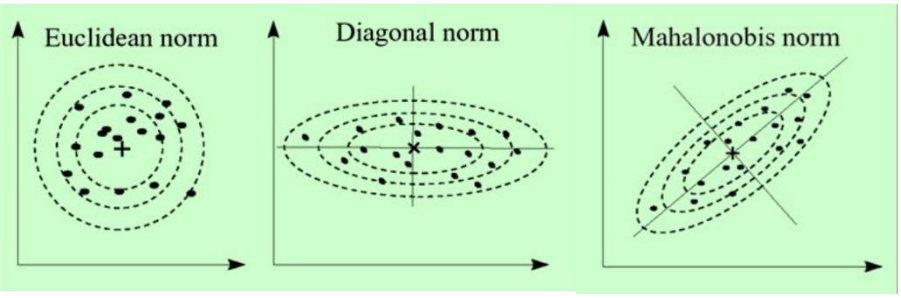


Ilustración 13 Representación gráfica de los diferentes tipos de distancia (ComputaciónClasificación, 2022)

Estas medidas permiten implementar un clasificador de mínima distancia de forma sencilla a través de fórmulas matemáticas sobre los valores de las características obtenidas de la imagen. A continuación, veremos cómo podemos extraer dichas características de forma automática.

3.1.1 Implementación del vector de características

Normalmente la extracción de características ha de hacerse mediante un software que permita el tratamiento de imágenes como el mencionado en el capítulo anterior denominado OpenCV, el cual puede funcionar en un entorno Python. En nuestro caso, OpenCV ofrece gran cantidad de funciones útiles para extraer estas y otras características necesarias para llevar cabo la clasificación (Contour Features, 2022).

Dada la alternativa de implementación elegida en este trabajo fue la de un servicio API, finalmente no fue necesario programar en OpenCv la extracción de características, por lo que a continuación sólo mostraremos una muestra de las funciones más relevantes para obtener las características de las imágenes comentadas anteriormente:

- Rectángulo rotado: **cv.minAreaRect()**: dibuja una "caja" en dos dimensiones de mínima área sobre una determinada forma. Nos devuelve sus coordenadas centrales, su anchura, altura y ángulo de rotación. Esta función nos permite extraer la "relación rectangular" del ave.

- Encaje en una línea: **cv.line()**: nos devuelve una línea que se aproxima a una determinada forma. Dicha línea nos devuelve su ángulo de rotación. Esta función puede aplicarse por separado a la división simétrica de las alas para determinar el "ángulo de planeo".

- Casco convexo: **convex.hull()**: nos devuelve una forma que envuelve a la imagen la cual corrige las convexidades hacia el interior. Esta función es muy interesante pues nos permitiría, por ejemplo, corregir el contorno externo de las aves rapaces, las cuales presentan un contorno de plumas separadas. Siendo sin duda una de las características más importantes de dichas aves, que podríamos utilizar como tercera característica de reconocimiento.

3.2 Elección de alternativas de Implementación

Uno de los aspectos más decisivos del trabajo consistió en la elección de alternativas de implementación. En este sentido, las decisiones principales fueron:

- Dónde se llevaría a cabo el algoritmo de procesado e identificación de la imagen, surgiendo dos opciones posibles que condicionarían todo el diseño posterior: algoritmo en la nube y algoritmo en el microcontrolador.
- Qué solución utilizar para desarrollar el algoritmo, encontrando también dos opciones: desarrollo programado o llamada a un servicio mediante un API.

3.2.1 Algoritmo en la nube

Esta opción implica "enviar" la imagen capturada, lo cual requiere de un microcontrolador con conectividad radio suficientemente elevada como para transmitir imágenes. Esta opción supone renunciar a las tecnologías de baja tasa de transmisión y mayores distancias, como SigFox y Lora.

En este caso, a falta de una oferta LTE-M de operadores para el público residencial, nos queda la opción de WiFi como única opción práctica, o en su caso BLE con acceso a un Gateway con acceso móvil a algún operador comercial.

Esta opción, sin embargo, tiene la ventaja de desacoplar el hardware del microcontrolador de la solución de reconocimiento. De esta forma, si en una segunda fase decidimos cambiar

o mejorar el algoritmo mediante entrenamiento, con más imágenes o más etiquetas, no será necesario cambiar nada en el hardware.

3.2.2 Algoritmo en el microcontrolador

Esta segunda opción, igualmente válida, consiste en realizar todo el tratamiento y reconocimiento en el propio controlador. Dicha alternativa, está siendo una tendencia a medida que los microcontroladores van ganando cada vez más en capacidad de proceso (nicla-vision, 2022).

La gran ventaja que ofrece este modelo es que, una vez realizado el procesado y reconocimiento, únicamente es necesario transmitir los resultados del algoritmo y no la imagen, utilizando menos recursos y ampliando el alcance.

Sin embargo, las tarjetas con mayor capacidad computacional vienen con conectividad estándar tipo WiFi/BLE ya que están orientadas a entornos industriales, y por tanto, requieren de *shields* para poder utilizar sistemas de radio de mayor alcance, lo cual encarece la inversión.

Además, esta solución suele utilizarse con algoritmos “programados” en entornos especializados para microcontroladores, los cuales requieren cierta curva de aprendizaje, en torno a 3-4 meses, para dominar estas tecnologías denominadas Tiny ML, para poder programar en, por ejemplo, TensorFlow Lite para microcontroladores (EDX Certificate, 2022).

Otra ventaja de esta opción es que existen muchos servicios IoT en la nube concebidos para recibir datos con un significado de gestión y no imágenes. El subir únicamente los resultados hace que nos podamos centrar más en la gestión haciendo un uso más directo de dichos servicios.

3.2.3 Desarrollo programado

Consiste en utilizar un software extractor de características del tipo OpenCV más alguna solución de inteligencia artificial como TensorFlow o PyTorch. Este método implica, además de aprender a utilizar el software, del tiempo necesario para desarrollar y depurar el código, así como de realizar suficientes pruebas y ajuste de parámetros, para llegar a obtener resultados óptimos.

Se trata sin duda, de un método donde se tiene total control sobre el problema de reconocimiento hasta el punto de que, si las características son suficientemente discriminantes, se pueden utilizar técnicas de clasificación más sencillas que *machine learning*.

Este tipo de desarrollo cuenta con la versatilidad de poderse alojar tanto en un servidor en la nube, como estar integrado en el propio controlador. Sin embargo, su principal desventaja es la curva y el tiempo de aprendizaje y desarrollo del algoritmo.

3.2.4 Servicio mediante API

Esta segunda opción, es sin duda mucho más rápida que el desarrollo programado ya que no es necesario ni extraer las características ni programar el algoritmo de *machine learning*.

Además, está basado en software totalmente testado y probado que están utilizando otros usuarios.

Sin embargo, se trata de una solución de “caja negra” donde se pierde la pista de cuáles son las características discriminantes del problema de clasificación, ni tan siquiera se conoce cuáles son dichas características. En esta solución los esfuerzos deben de focalizarse en realizar un buen entrenamiento, esto es, con una muestra suficientemente representativa del problema a abordar.

El modelo de servicio, por lo general asume que se desarrollará en la nube, aunque existen soluciones que permiten exportar en clasificador resultante para su implementación en el microcontrolador. En el primer caso conlleva un coste que, en alta disponibilidad, puede encarecer significativamente la solución.

Por último, no hay que olvidar que la llamada al API también requiere de desarrollo software desde el servidor, con una serie de parámetros y requerimientos que también implican tiempo y curva de aprendizaje.

3.2.5 Elección las mejores alternativas

Tras considerar todas estas opciones hemos efectuado una matriz de decisión basada en factores con diferentes pesos, para las dos decisiones analizadas con sus cuatro alternativas. Hay que destacar que entre dichos factores hemos ponderado más aquellos dependientes del tiempo, como curva de aprendizaje y tiempo de desarrollo:

		Algoritmo en la Nube	Algoritmo en el Microcontrolador
Alcance Radio de la solución	25%	2	5
Desacople del software de reconocimiento	20%	5	2
Coste hardware	20%	5	2
Centrarse en otros servicios IoT	20%	2	5
Versatilidad del software de reconocimiento	15%	5	3
		3,65	3,5
		Desarrollo Programado	Servicio mediante API
Curva de aprendizaje	20%	1	3
Tiempo de desarrollo	40%	1	3
Control sobre las características	10%	5	2
Coste de la solución software	20%	4	2
Versatilidad donde alojarse	10%	4	3
		2,3	2,7

Tabla 10 Matriz de ponderación de factores de decisión de alternativas

Tras este análisis de alternativas, la solución a implementar fue desarrollar el **Algoritmo en la Nube** mediane el uso de un **Servicio mediante API**. Los factores relevantes de la decisión fueron el tiempo limitado de desarrollo y el tratar de minimizar la curva de aprendizaje sobre el software de programación.

3.3 Esquema de la solución desarrollada

A partir de las alternativas elegidas de realizar el algoritmo de reconocimiento en la nube mediante el uso de un servicio mediante API, a continuación, mostramos el esquema general de la solución:

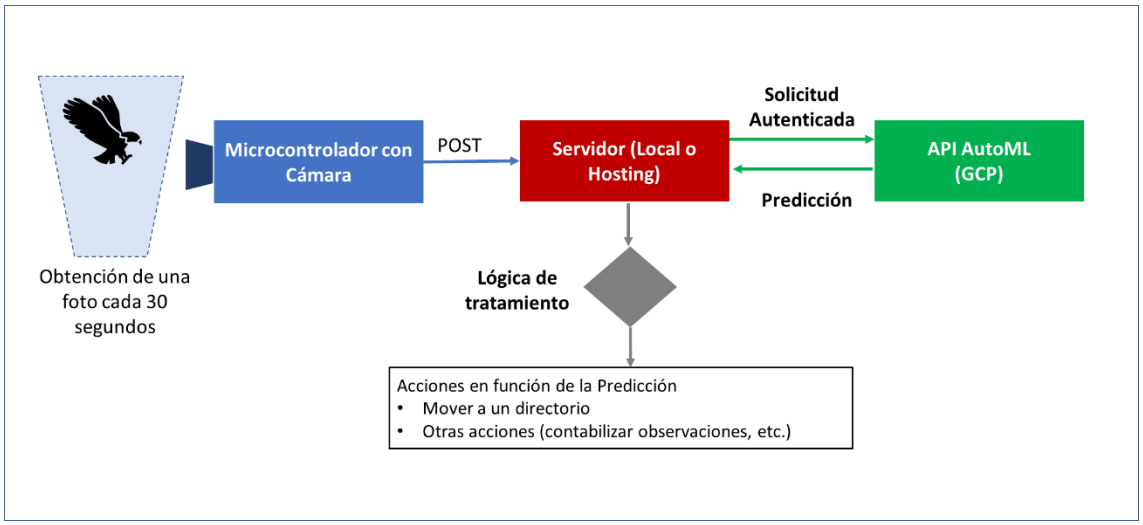


Ilustración 14 Esquema general de la solución

El funcionamiento consta de tres partes:

- Desarrollo hardware y software para subir imágenes a un servidor. Esta parte del desarrollo presenta el reto de que la mayor parte de las placas están orientas a subir pequeños paquetes de datos de sensores y no imágenes que pueden ocupar más que la propia memoria del microcontrolador. El propósito de este desarrollo es obtener una foto de forma periódica, cada 30 segundos, y enviarla al servidor donde será recibida y analizada.

- Desarrollo del servidor que recibe las imágenes. En este caso, el reto es de recibir peticiones HTML POST de imágenes y almacenarlas en el servidor. A partir de ahí aplicar una lógica en función del servicio del reconocimiento.

- Desarrollo del servicio de *Machine Learning* y lógica de actuación. En esta fase lo que haremos es entrenar un modelo de *Machine Learning* de *Google Cloud Platform*, para su posterior utilización desde el servidor mediante una llamada al API que realiza solicitudes autenticadas dentro de dicho servicio.

  En cuanto a la lógica de actuación, en principio ésta fue tan sencilla como almacenar la imagen en una carpeta u otra dependiendo de la clasificación obtenida. Aunque realmente está lógica se puede ampliar a otro tipo de acciones como el registro y tratamiento estadístico de la información.

3.4 Desarrollo Hardware y Software para subir imágenes a un servidor

La selección de hardware comenzó con la placa básica de Arduino UNO y una cámara OV2640 de 2M píxeles. Esta cámara presenta la ventaja de disponer de un soporte para probar diferentes tipos de lentes, podría ser muy adecuado para la observación en la naturaleza.

Estas primeras pruebas, realizadas con los *sketches* de prueba encontrados en la librería (githubArduino, 2022), y sirvieron únicamente para verificar el funcionamiento de la cámara, ya que la placa de Arduino UNO no dispone de conexión inalámbrica. A continuación, vemos el esquema de conexión entre la cámara y la placa (CamArduinoUNO, 2022)

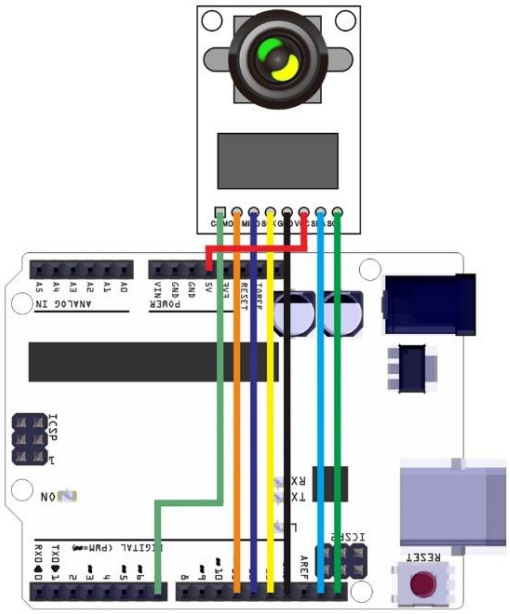


Ilustración 15 Esquema de conexión de la cámara OV2440 a Arduino UNO (CamArduinoUNO, 2022)

Por lo general los ejemplos funcionan a través de la interfaz serie USB de Arduino o ESP8266. El ejemplo más significativo lo podemos encontrar en (GirhubCapturaHTML, 2022), orientado a ESP8266 con capacidad de capturar una foto y enviarla mediante HTTP_GET a una carpeta HTML. Este método es muy similar al utilizado en la solución final, con la diferencia que en mi caso utilice un método POST.

Dado que el objetivo era efectuar una conexión inalámbrica, utilicé una placa Arduino MKR1010, que hemos visto en el capítulo 2, con una memoria de 256kb y conectividad WiFi, gracias al módulo u-blox NINA-W102. Sin embargo, la mayor parte de los ejemplos encontrados, para esta placa estaban orientados el envío de datos de sensores y no a capturas de imágenes.

En este sentido traté de modificar los ejemplos mencionados anteriormente para MKR1010, pero no funcionaban adecuadamente, por lo que de forma paralela analicé la posibilidad de utilizar una placa de tipo ESP, reorientando así los diseños hacia estas placas que ofrecían mayores prestaciones.

En la siguiente imagen muestro las dos placas utilizadas inicialmente de la familia Arduino, donde puede apreciarse la diferencia de tamaño, así como la cámara OV2640 con soporte para lentes.

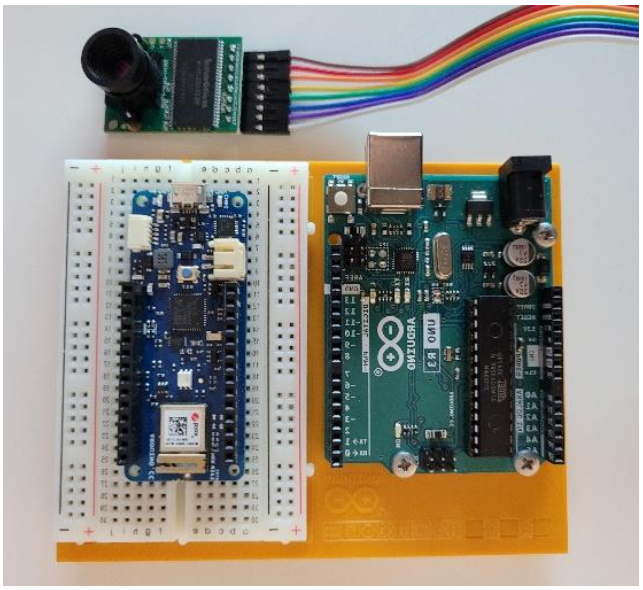


Ilustración 16 Placas y cámara utilizadas en las pruebas iniciales

De esta forma, dentro del proceso de investigación, pude observar que existían varios *scketches* orientados a la tarjeta ESP32 (algo superior a la ESP8266) con una camara incorporada similar a la de las primeras pruebas, (Randomnerdtutorials, 2022), por lo que adquirí un par de módulos ESP32-CAM con el módulo de conversión serie. Gracias a dicho módulo no es necesario efectuar un cableado entre la cámara y el adaptador FTDI (ESP32 CAM FTDI, 2022).

A continuación, muestro unas imágenes de placa adquirida con la cámara integrada junto con su conversor, separados y ensamblados. En la foto podemos apreciar el pequeño tamaño de dicho módulo, lo cual lo hace ideal para un proyecto de estas características.

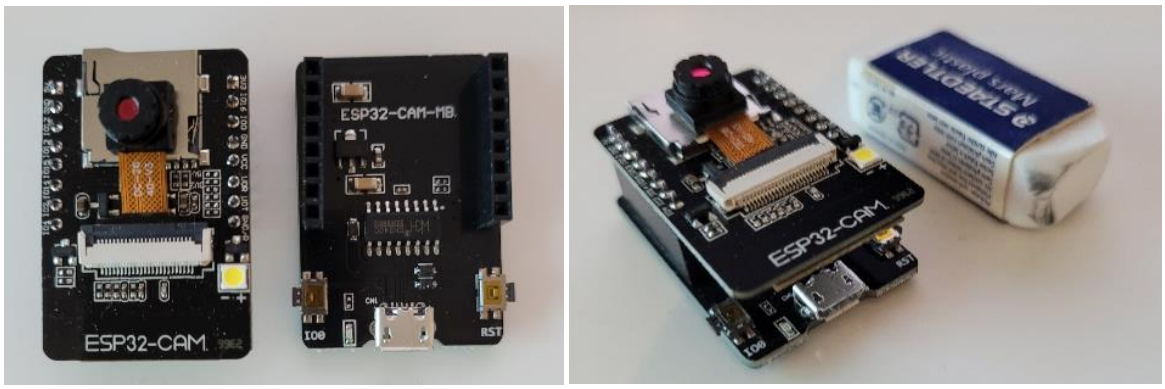


Ilustración 17 Placa ESP32 CAM con su conversor a USB-Mini

La conexión con el ordenador es mediante un cable usb-mini, y la conectividad es la WIFI-BLE de la placa ESP32. Dispone de una memoria de 520KB SRAM y externa de 4M PSRAM. Aunque las librerías del IDE de Arduino más comunes son las del fabricante AI-Thinker, podemos encontrar gran cantidad de clones con prestaciones similares como la de DiY More, que fue la que adquirí (Mini ESP32-CAM, 2022).

En cuanto al esquema de pines, son los siguientes (ESP32 CAM pinout, 2022):

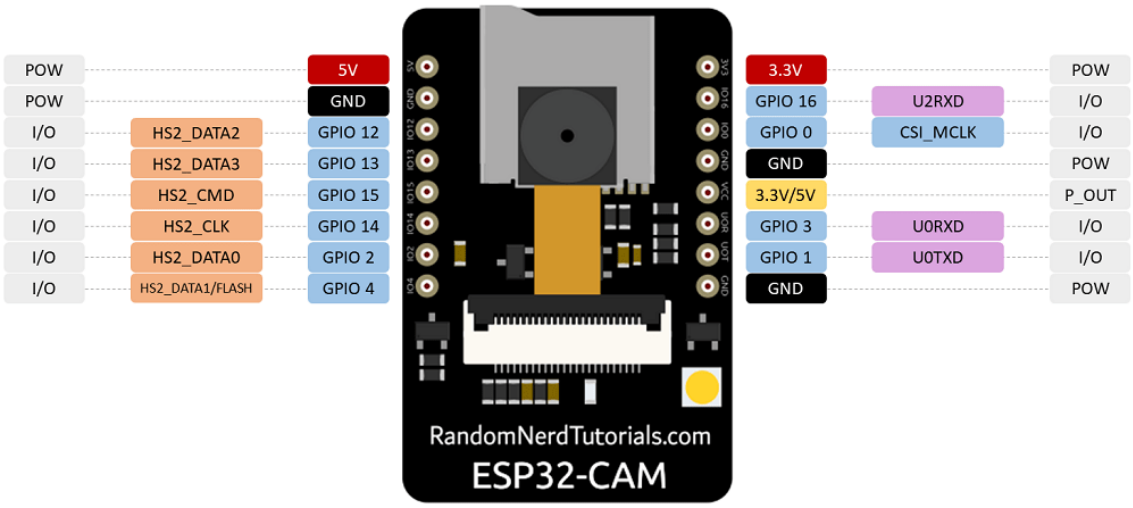


Ilustración 18 Esquema de pines del ESP32 CAM (ESP32 CAM pinout, 2022)

- Como se puede apreciar, permite alimentación a 5V y 3,3V (en rojo). Los pines serie son los GPIO 1 y 3, que se utilizan para subir el código hacia la tarjeta y se conectan al módulo FTDI (pines TX y RX).

- Esta cámara puede utilizar una luz *flash*, para lo cual es necesario conectar GPIO0 a GND (internamente está conectado a una resistencia en pull-up).

- Se dispone de una tarjeta microSD, cuya utilización requiere el uso de los siguientes pines: GPIO14: reloj, GPIO15: CMD, GPIO 2, 4, 12 y 3 a DATA.

- GPIO4, es el que se conecta al *flash*. GPIO33, conectado a un led rojo incorporado, que se enciende, por ejemplo, cuando el WIFI está conectado.

En cuanto la conexión de pines a la cámara es la que se muestran en la siguiente ilustración.

OV2640 CAMERA	ESP32	Variable name in code
D0	GPIO 5	Y2_GPIO_NUM
D1	GPIO 18	Y3_GPIO_NUM
D2	GPIO 19	Y4_GPIO_NUM
D3	GPIO 21	Y5_GPIO_NUM
D4	GPIO 36	Y6_GPIO_NUM
D5	GPIO 39	Y7_GPIO_NUM
D6	GPIO 34	Y8_GPIO_NUM
D7	GPIO 35	Y9_GPIO_NUM
XCLK	GPIO 0	XCLK_GPIO_NUM
PCLK	GPIO 22	PCLK_GPIO_NUM
VSYNC	GPIO 25	VSYNC_GPIO_NUM
HREF	GPIO 23	HREF_GPIO_NUM
SDA	GPIO 26	SIOD_GPIO_NUM
SCL	GPIO 27	SIOC_GPIO_NUM
POWER PIN	GPIO 32	PWDN_GPIO_NUM

Ilustración 19 pines de conexión entre cámara y tarjeta (ESP32 CAM pinout, 2022)

Esta definición será la que posteriormente deberemos de replicar en la definición del código de definición de constantes en el IDE de Arduino.

En cuanto a las características de la cámara, existen una serie de parámetros relacionados con el brillo, contraste, saturación etc., que pueden utilizarse a través de una serie de funciones disponibles en (OC2640 Camera Settings, 2022).

Además, se puede determinar diferentes formatos de imagen (YUV422, Grayscale, RGB565 y JPG), para lo cual se utiliza la siguiente función para el caso de “jpg”:

```
config.pixel_format = PIXFORMAT_JPEG;
```

En cuanto el tamaño del marco soporta desde QVGA de 320x240 hasta SXGA de 1280x1024.

Una de las grandes ventajas de esta placa con cámara integrada, es que no es necesario realizar ningún esquema de conexión. Lo único necesario es conectar una fuente de alimentación.

3.4.1 Alimentación de la placa

La placa ESP32 funciona a voltajes entre 2,55 y 3,6 voltios. Y utiliza una corriente que puede ir desde los 7µA en modo “*deepsleep*” si se alimenta con batería externa o 11 mA con una fuente de alimentación a la conexión USB, hasta los 400mA si utilizamos el WIFI.

En este sentido, existen varias posibilidades a la hora de alimentar dicha placa que mencionamos a continuación (AlimentaciónESP32, 2022):

- Utilizar un banco de energía estándar: aparentemente es una opción interesante debido a la elevada capacidad de estas baterías que normalmente trabajan a 3,7 voltios que luego transforma a 5 v. Sin embargo, este factor y el hecho de tener que volver a transformar a 3,3 v del ESP32 mediante un regulador de voltaje hace que tengamos pérdidas de eficiencia. Además, hay algunos modelos que pueden llegar a desconectarse al no detectar el bajo consumo de energía.

- Utilización de pilas estándar. Este caso no es posible ya que las normales de Ni-MH aunque teóricamente son de 1,5 v, en la práctica suministran solo 1,2 v, por lo que dos pilas no alcanzaría el mínimo y tres lo supera.

- Pilas de litio. Este caso es totalmente aplicable ya que con dos de 1,5 v tendríamos una tensión constante dentro del margen, durante el 90% de la vida útil de la batería. Además, estas baterías soportan altos requerimientos de la operación wifi, y aguantan en reposo más de 5 años. Además, pueden funcionar en un amplio rango de temperatura ambiente.

- Pilas recargables $LiFePO_4$. Estas pilas tienen un buen rendimiento, aunque algo inferior que el de las pilas de litio. Además, presentan la gran ventaja de poderlas recargar. Sin embargo, utilizan un polímero que puede llegar a incendiarse si su uso no es adecuado.

- Baterías recargables de polímero de litio. Estas baterías generan suficiente energía para un funcionamiento óptimo, sin embargo, también requieren de una bajada de tensión de 3,7 v - 4,2v a 3,3v, con la consiguiente pérdida de eficiencia por el uso de un regulador de tensión. De esta forma su duración es más corta.

A continuación, vemos un cuadro comparativo de las diferentes opciones.

1.- Banco de Energía. Eficiencia: - Duración: + Fiabilidad: -- Seguridad: = Coste: -- Recarga: ++		**2.- Pilas de litio.** Eficiencia: ++ Duración: ++ Fiabilidad: ++ Seguridad: ++ Coste: + Recarga:--	
3.- Pilas recargables. Eficiencia: + Duración: + Fiabilidad: ++ Seguridad: - Coste: + Recarga: ++		**4.- Batería recargable.** Eficiencia: - Duración: - Fiabilidad: ++ Seguridad: - Coste: - Recarga: ++	

Tabla 11 Tabla comparativa de alternativas de alimentación externa de la placa

En la implementación realizada, al tratarse de un piloto no llegamos a utilizar alimentación externa, por lo que la alimentación se producía directamente a través del puerto mini-USB conectado al ordenador. En el caso de tener que utilizar alguna de las opciones habría utilizado la opción 2 en una primera instancia por la sencillez y fiabilidad.

3.5 Desarrollo Software para subir imágenes a un servidor

En este apartado veremos cómo hemos utilizado nuestra placa para jugar un papel de cliente y realizar peticiones HTTP a un servidor (HTTP Request, 2022).

El método HTTP POST, es un método que envía datos al servidor que utilizamos a través de un formulario HTML. A diferencia del método GET que envía los datos en la propia URL, el método POST lo hace en el cuerpo de la petición. La elección del tipo de contenido es muy importante ya que tenemos dos opciones:

- Enviarlo en un único bloque (application/x-www-form-urlencoded) lo cual tiene una limitación de tamaño.

- Enviarlo en varios bloques (multipart/form-data) que permite enviarlo en varias partes, cada una de las cuales tiene una parte de los datos. Este será el tipo de contenido que debemos utilizar ya que, al enviar una imagen, el tamaño puede ser de gran tamaño.

Dentro del cuerpo de un contenido multipart/form-data, tenemos asimismo una serie de parámetros:

- Content-Disposition: en este campo se especifica si el contenido se mostrará en el navegador o si se trata de una descarga. En el caso específico de POST deberemos especificar “form-data” ya que indica que los datos están divididos en partes separadas por un “boundary”. Asimismo, se define un parámetro “name”, para identificar la parte del formulario que se trate.

- Content-Type: se refiere al tipo y formato del contenido. Este punto es realmente interesante ya que nos ofrece una gran amplitud de posibilidades. A continuación, mostramos un listado con todos los posibles tipos de contenido a enviar (HTTP headers ContentType, 2022). En nuestro caso aplicará el tipo **image/jpeg**.

Type	Values	Type	Values
Application	application/EDI-X12	Multipart	multipart/mixed
	application/EDIFACT		multipart/alternative
	application/javascript		multipart/related (using by MHTML (HTML mail).)
	application/octet-stream		multipart/form-data
	application/ogg	Text	text/css
	application/pdf		text/csv
	application/xhtml+xml		text/html
	application/x-shockwave-flash		text/javascript (obsolete)
	application/json		text/plain
	application/ld+json		text/xml
	application/xml	Video	video/mpeg
	application/zip		video/mp4
	application/x-www-form-urlencoded		video/quicktime
Audio	audio/mpeg		video/x-ms-wmv
	audio/x-ms-wma		video/x-msvideo
	audio/vnd.rn-realaudio		video/x-flv
	audio/x-wav		video/webm
Image	image/gif	VND	application/vnd.oasis.opendocument.text
	image/jpeg		application/vnd.oasis.opendocument.spreadsheet
	image/png		application/vnd.oasis.opendocument.presentation
	image/tiff		application/vnd.oasis.opendocument.graphics
	image/vnd.microsoft.icon		application/vnd.ms-excel
	image/x-icon		application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
	image/vnd.djvu		application/vnd.ms-powerpoint
	image/svg+xml		application/vnd.openxmlformats-officedocument.presentationml.presentation
			application/msword
			application/vnd.openxmlformats-officedocument.wordprocessingml.document
			application/vnd.mozilla.xul+xml

Ilustración 20 Listado de los diferentes tipos de content-type en el método POST (HTTP headers ContentType, 2022)

De este modo, la sintaxis de la petición tendría que seguir el siguiente formato:

```
POST serverPath HTTP/1.1
Host: serverName
Content-Length: longitud del contenido
Content-Type: alguno de los dos comentados anteriormente; bundary="boundary"
se trata de la etiqueta con la que delimitamos cada parte del mensaje

--boundary
    Content-Disposition: form-data; name=\"imageFile\";
    filename=\"esp32-cam.jpg\"\r\n
    Content-Type: image/jpeg\r\n\r\n


    // aquí enviamos el fichero .jpg en paquetes de 1024 bytes.


    \r\n--Final--\r\n
--boundary
```

Ilustración 21Formato del método POST para enviar una imagen

Para poder implementar esta función en el entorno de desarrollo de Arduino, será necesario además conocer una serie de funciones imprescindibles que se encuentran en la librería de Arduino WiFi:

- **Client.connect(ip, port)**. Donde “ip” es la dirección IP donde el cliente se conectará (un array de 4 bytes), también puede indicarse una URL con el nombre de un dominio. “Port”, el puerto donde el servidor está escuchando para poderse conectar. Esta función retorna un 1 o “true” si la conexión fue exitosa y un valor negativo o “false” si no lo fue.
    - En nuestro caso, el nombre del servidor en el caso de ser local, utilizamos la dirección 192.168.0.11. Es importante señalar que si esta variable la almacenamos en un *string*, a la hora de pasarla a la función hay que convertirla a formato C (es decir acabado en un null), mediante la función c_str(). Asimismo, el puerto será el que hayamos incluido en la escucha en el código del servidor.

- **Client.println(data).** Envía datos seguidos por un retorno y nueva línea al servidor al que nos hemos logrado conectar. Imprime una secuencia de dígitos en formato ASCII.
    - En nuestro caso, utilizamos esta función para construir un texto concatenando cadenas de caracteres que componen el envío del HTTP POST al servidor, tal y como hemos visto anteriormente.

- **Client.write (buffer, size)**. Envía un array de bytes de datos al servidor especificados en buffer, del tamaño indicado en el campo *size* en bytes. La principal diferencia entre esta función y Client.println, es que la segunda envía los datos utilizando código ASCII para cada carácter.

- **Client.available()**. Es una función que nos devuelve la cantidad de bytes que han sido escritos en el cliente por el servidor al cual está conectado. En nuestro caso, cuando esperamos respuesta del POST, utilizaremos esta función para ver si tenemos bytes disponibles para poder leer.

- **Client.read()**. Consiste en leer el siguiente byte recibido del servidor desde la última vez que se leyó. En nuestro programa lo utilizamos para leer si la respuesta al método POST fue satisfactoria o no.

Además, utilizaremos estas otras funciones más típicas para la conexión WiFi:

- **WiFi.begin(ssid, pass)**. Es la función utilizada para para iniciar la conexión WiFi. El "Service Set Identifier" es el nombre de la red Wifi a la que nos conectemos, por ejemplo "vodafone4330".

Asimismo, vamos a utilizar una serie de funciones de la cámara que se encuentran en la librería "**esp_camera.h**" (github_EspCam.h, 2022)

- **camera_config_t**. Es un tipo de datos que se define como una estructura de variables que responden a las diferentes entradas, frecuencia de reloj, timers, formato de píxel, tamaño de la foto, calidad de la foto, número de buffer de fotos a utilizar, localización del buffer de fotos.

- **Esp_camera_init(&config).** Se utiliza probar que la cámara funciona correctamente, Asimismo inicializa la interface I2C, el/los buffer de fotos, otros buffers de memoria así como las entrada paralelas I2S. Utiliza una variable de tipo camera_config_t que le pasemos, para configurar los parámetros de la misma

- **Esp_camera_fb_get().** Captura una foto y devuelve el puntero al buffer de fotos donde se encuentra dicha captura.

- **Esp_camera_fb_return().** Libera el buffer de fotos para ser utilizado de nuevo.

Otras funciones de la ESP32:

- **WiFi.mode().** Se refiere al tipo de acceso que utilizaremos con el ESP32. Tenemos tres opciones:
  - **WIFI_STA**: modo estación, ESP32 se conectará al punto de acceso.
  - **WIFI_AP**: modo punto de acceso, donde otras estaciones se podrán conectar al ESP32
  - **WIFI_STA_AP**: modo estación y punto de acceso.

- **PsramFound()**. Esta función nos indica si nuestra tarjeta cuenta con memoria externa PSRAM. De esta forma podemos configurar la resolución del tamaño de nuestra foto.

- **ESP.restart():** reinicia la tarjeta.

Adicionalmente a dichas funciones, hemos incluido funciones de cálculo de hora incluidas en NTPClient.h por si se fueran a utilizar en una posterior versión:

- **WiFiUDP udp**: se habilita comunicación UDP para utilizar con el cliente NTP (Network Time Protocol)

- **NTPClient ntp(udp, "europe.pool.ntp.org", -3 * 3600, 60000)**; se conecta al servidor NTP para obtener poder obtener la hora de dicha zona

- **Ntp.getFormattedTime()**: obtiene la hora de dicho servidor

- **Ntp.getDay()**: obtiene el día de dicho servidor

Algoritmo desarrollado en el Sketch_TFM.ino

Este código, se basa en algunos ejemplos encontrados en Internet (Randomnerdtutorial Post Image, 2022) (Kaduzi.us, 2022) y es el responsable de enviar la imagen capturada al servidor (en ese caso node.js, pero es prácticamente similar en caso de PHP).

El código tiene un primer bloque de definiciones de constantes necesarias para establecer la conexión wifi, establecer la ruta y nombre del servidor, definir los parámetros de configuración de la cámara e inclusiones de las bibliotecas utilizadas, las cuales son las recomendadas para el ESP32 CAM en especial la “esp_camera.h”. Asimismo, adicionalmente hemos incluido un servicio NTP por si se quisiera obtener la hora.

El segundo bloque es el de setup() consiste en inicializar la WiFi y la cámara con la función esp_camera_init(&config); en donde se envían todos los parámetros de configuración, para a continuación llamar a la función enviar foto por primera vez. A partir de ahí pasamos a la función loop() la cual, cada 30 segundos enviará una nueva foto y actualizará la hora y el día.

La función donde se concentra la mayor complejidad es la de enviar foto: sendPhoto() cuya lógica resumimos a continuación:

1. En primer lugar se obtiene la foto a través de una función de la cámara que nos devuelve un puntero a la imagen obtenida: fb = esp_camera_fb_get() (Github ESP32 CAM, 2022)

2. Posteriormente, seguiremos los pasos para construir la petición POST al servidor tal y como explicamos anteriormente (ESP32 IO, 2022)

    - Me conecto al servidor mediante la función client.connect();
    - Efectúo la petición escribiendo en el servidor la cabecera del método POST con client.println()
    - Escribo en el servidor mediante la función client.write() los bloques de datos en los que envío la foto y el final con client.println()
    - Liberamos memoria a través de esp_camera_fb_return(fb);
    - Leo el byte de respuesta del servidor client.read() hasta llegar al final para ver si ha habido error. En esta parte utilizo un contador con un límite de tiempo para evitar que quede bloqueado.
    - Vuelvo al bloque loop().

La siguiente figura muestra las secciones principales del código Stketch_TFM.INO que podemos encontrar en el anexo.

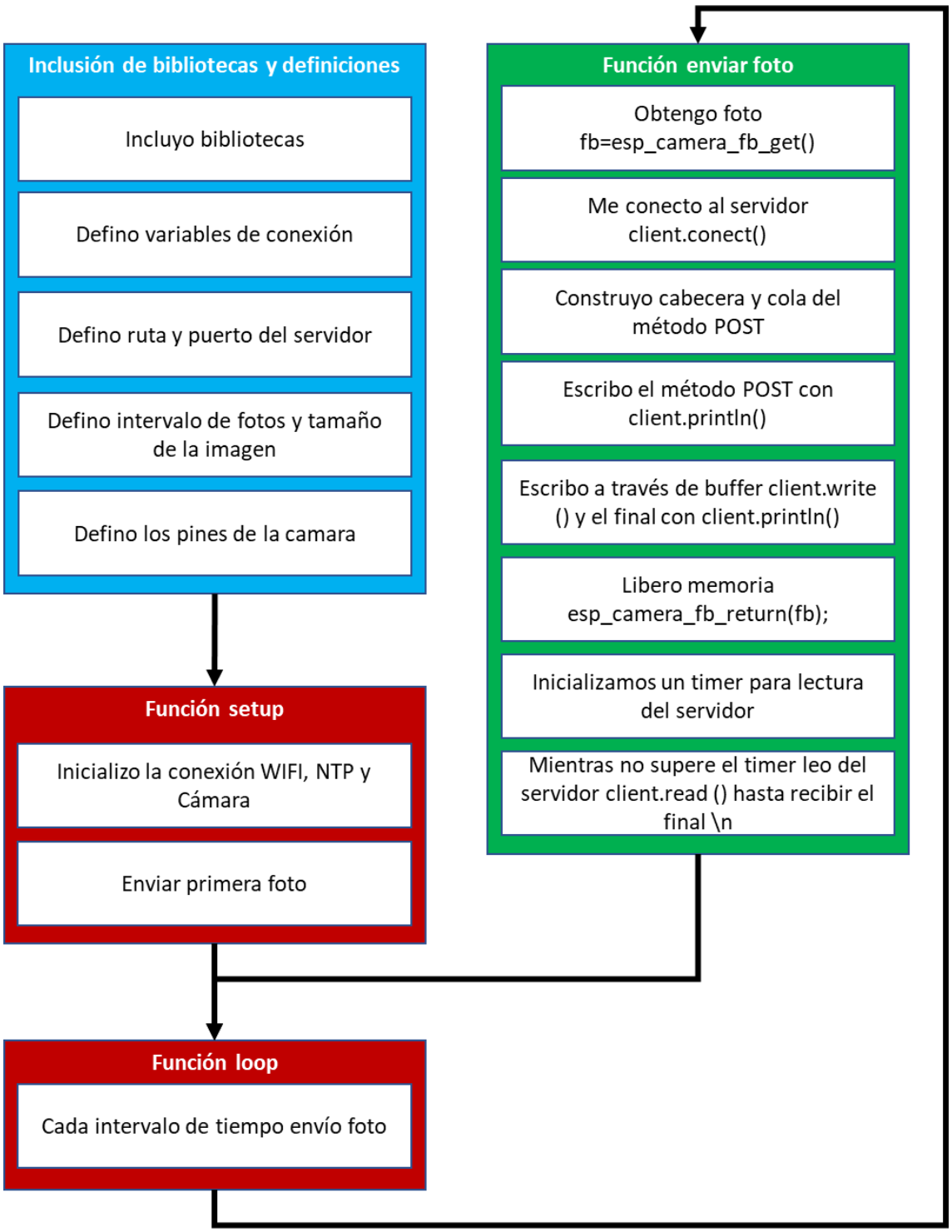


Ilustración 22 Esquema del código Sketch_TFM.ino subido al ESP32 CAM desde el IDE de Arduino

3.6 Desarrollo del servidor que recibe las imágenes

Uno de los principales retos del sistema consiste en recibir la imagen capturada en un servidor que puede situarse en Internet o bien ser un servidor local.

3.6.1 Subida a un servidor web en un hosting con un dominio propio con tecnología PHP

En nuestro caso efectué un primer desarrollo en el que el ESP32 CAM sube una imagen cada 30 segundos a un servidor web situado en un hosting, cuyo dominio es www.averapaz.com. Este desarrollo cuenta con la ventaja de que puede ser accedido de forma externa y abierta al público en general, y las fotos pueden ser publicadas si así de desea como parte del proyecto de investigación.

Este código, desarrollado en PHP y basado en (w3schools php upload, 2022) (Randomnerdtutorial Post Image, 2022) utiliza una serie de variables predefinidas propias del método POST en PHP que pasamos a resumir (PHP NET POST method, 2022):

Cuando el método POST recibe un fichero utiliza la variable $_FILES que consiste en un array de elementos asociados al fichero subido en el método POST. Los elementos más importantes son:

- $_FILES['userfile']['name']: se refiere al nombre original del fichero en el cliente.
- $_FILES['userfile']['type']: este campo se denomina “mime type” y se refiere al tipo que especificamos en la petición. En nuestro caso fue “image/jpeg”.
- $_FILES['userfile']['size']: es el tamaño en bytes.
- $_FILES['userfile']['tmp_name']: es el nombre temporal que el fichero tiene cuando se almacena en el servidor.
- $_FILES['userfile']['error']: es el Código de error asociado a la subida de ficheros.
- $_FILES['userfile']['full_path']: es la ubicación que se envió en la petición.

Asimismo, utilizamos la variable predefinida $_POST: que también es un array de variables enviadas en el script mediante HTTP POST. En nuestro caso, únicamente vamos a utilizarla para ver si está definida como método de comprobar que se subió el fichero, esto lo hacemos mediante la siguiente función:

- Isset — Determina si la variable está declarara y es diferente a *null*, devolviendo un valor booleano.

En nuestro caso se aplica isset($_POST["submit"], en este caso “submit” tiene el significado de que se envió el formulario.

Otras funciones utilizadas en el desarrollo fueron:

- Move_uploaded_file(): utilizada cuando tenemos una petición HTTP POST, la cual mueve el fichero subido a una nueva ubicación en el servidor.

- File_exist(): que utilizamos para ver si el fichero temporal almacenado en el servidor existe o no.

- PATHINFO_EXTENSION: la cual, se puede utilizar como función o propiedad para averiguar la última extensión de un fichero.

La siguiente figura muestra las secciones principales del código UPLOAD.PHP que podemos encontrar en el anexo.

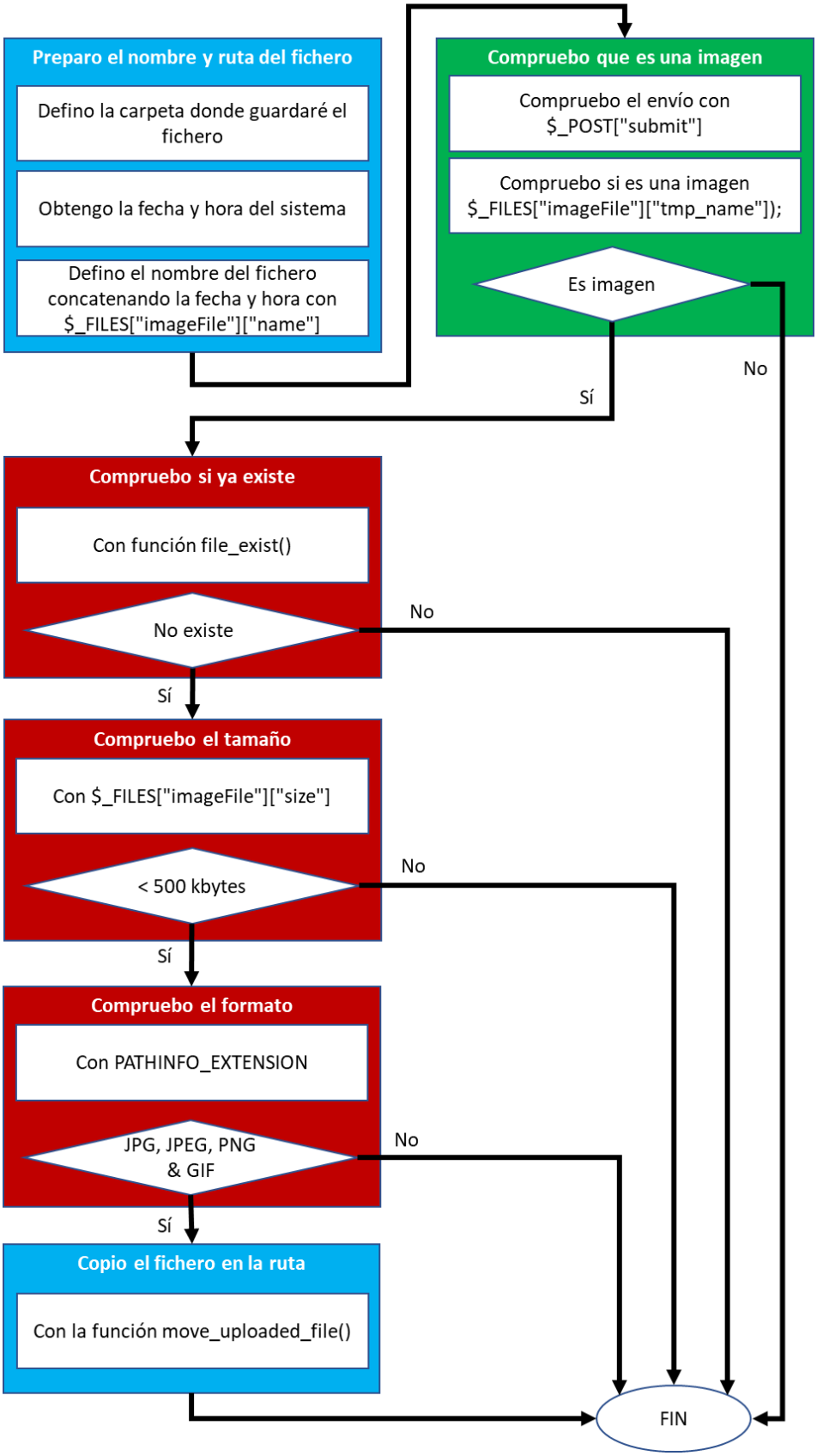


Ilustración 23 Esquema del código Upload.php que recibe la petición HTML POST

Este código será el que haya que subir a la carpeta pública del sitio web www.averapaz.com como podemos ver en la pantalla adjunta del panel de gestión del hosting en webempresa.

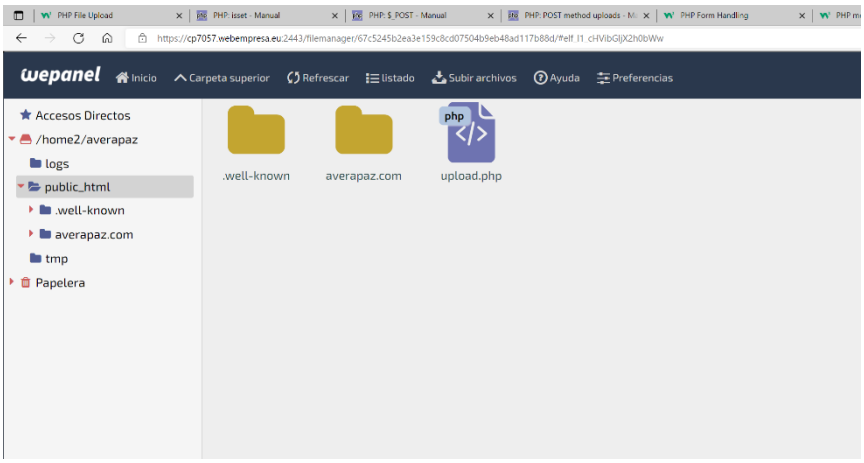


Ilustración 24 Panel de control del hosting donde se aloja el código Upload.php

Las fotos son almacenadas en la carpeta /uploads y pueden ser accedidas de forma abierta mediante http://www.averapaz.com/galery.php, (este último código es un desarrollo sencillo para poder visualizar los ficheros cargados en formato imagen)

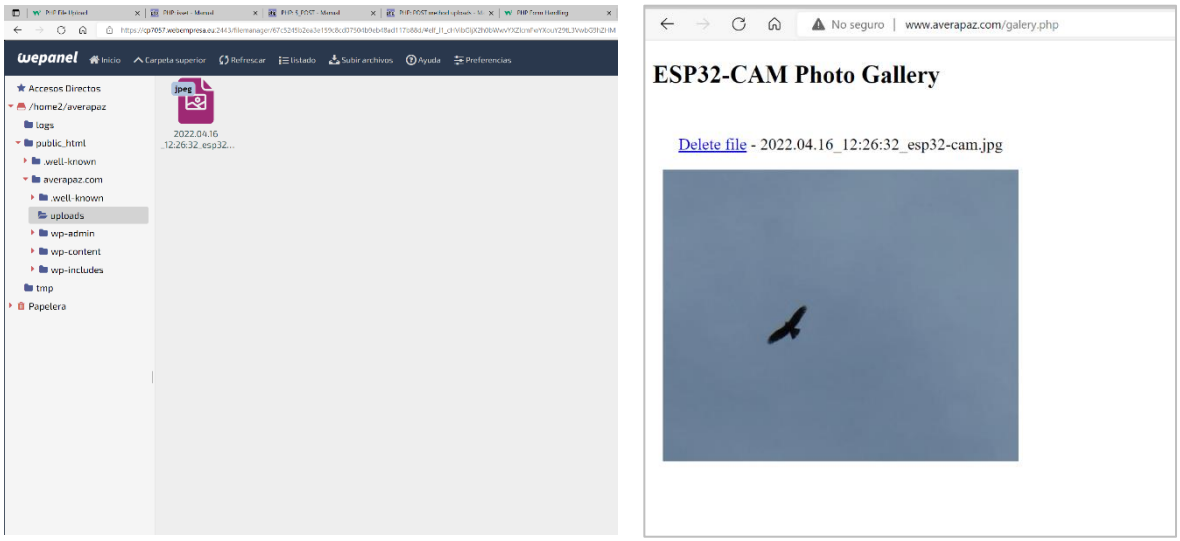


Ilustración 25 Galería de fotos de la web con dominio adquirido http://www.averapaz.com

Este desarrollo, sin embargo, presenta el inconveniente de que se ha realizado en PHP, dado que es la tecnología que presenta el Hosting utilizado (WebEmpresa). Mas tarde, cuando desarrollé la implementación del servicio a través del API AutoML Vision, comprobé que la llamada al API era más adecuada realizarla en otros lenguajes como Python, Java o Nodejs, por lo que tuve que cambiar esta opción de desarrollo por otra consistente en desarrollar un servidor en Nodejs, el cual me aseguraba el acceso al API y ofrecía una mayor facilidad y versatilidad en la lógica del proceso.

Es interesante ver cómo el servicio del API puede condicionar los desarrollos de fases previas a llegar al mismo. Este factor de diseño podría haberme servido para elegir un servicio de Hosting con capacidad de utilizar otras tecnologías como Nodejs.

3.6.2 Subida a un servidor local con tecnología nodejs

Según lo comentado en el apartado anterior, teníamos que escoger una tecnología en el servidor que fuera compatible con la llamada al API de reconocimiento de imágenes. La opción más rápida encontrada fue hacerlo con tecnología nodejs (Lucas, 2019) (Capan, 2022), en este caso a un servidor local:

Características de Node.js

A continuación, haré un breve resumen de las características de nodejs, las cuales le convierten en muy buena opción para este proyecto:

- Se trata de un entorno de ejecución en tiempo real para ejecutar un código en javascript en el servidor, y se ejecuta con el motor de javascript V8 de Google Chrome, de esta forma no es necesaria la compilación previa.

- Utiliza un modo de entrada y salida "sin bloqueo", controlado por eventos, orientado a solicitudes y respuestas, como escribir o leer ficheros.

- El servidor se compone de un subproceso que va procesando una sucesión de eventos, cuando hay una operación de entrada o salida, no espera a su finalización, sino que comienza a procesar otro evento.

- Al compartir un único subproceso todas las solicitudes, existe un riesgo de fallo general cuando el cálculo del código sea pesado o exista un error. Por ello, se transfiere los errores a la llamada como si fueran parámetros.

- Utiliza una herramienta denominada NPM (*node package manager*) que permite instalar componentes a través de un repositorio en línea mediante la línea de comandos. En nuestro proyecto hemos utilizado algunos como por ejemplo *express*.

Express utiliza un concepto denominado middleware (Using Middleware, 2022) que consiste en una función que realiza peticiones a un objeto y devuelve respuestas. Estas funciones de middleware si no finalizan, pueden dejar el flujo de ejecución bloqueado, para evitarlo pueden llamar a la siguiente función a través de next().

Disponemos de varios tipos de middleware pueden estar orientados a: nivel de aplicación, nivel de enrutamiento, manejo de errores, de gestión interna o de terceros. La forma de crear una instancia de un objeto de nivel de aplicación (app object) es mediante app.use() y app.method(), donde method() es el método de la petición realizada desde HTTP, es decir get o post.

A continuación, explicamos las funciones de middleware utilizadas:

- app.post('/upload', function(req, res). Con esta llamada, recuperamos la información del objeto subido, referenciándolo a través del campo de entrada, en nuestro ejemplo imageFile. La información se devuelve a través de "req" con los siguientes campos:
    - req.files.imageFile.name: el nombre del fichero (esp32_cam.jpg)
    - req.files.imageFile.mv: mueve el fichero a otra ubicación
    - req.files.imageFile.mimetype: el tipo mime del fichero (en nuestro caso image/jpeg)
    - req.files.imageFile.data: es un buffer del fichero cuando se crea temporalmente.
    - req.files.imageFile.tempFilePath: es la ubicación temporal del fichero.
    - req.files.imageFile.truncated: indica si ha superado el tamaño límite.
    - req.files.imageFile.size: es el tamaño del fichero en bytes.
    - req.files.imageFile.md5: es la suma checksum en MD5.

- app.use(fileUpload). Es una opción recomendada para utilizar ficheros temporales en lugar de memoria para gestionar el proceso de subida

A continuación, mostramos el esquema general del código del servidor local nodejs que recibe el POST de la imagen. Dicho código utiliza el *framework* express específico para nodejs. Con este componente podemos utilizar el método app.post() que dirige todas las peticiones HTTP POST para ser tratadas (App.post, 2022). Este código está basado en las siguientes referencias (Express, 2022) (Kaduzi.us, 2022) y podemos consultarlo en el anexo como upload.js.

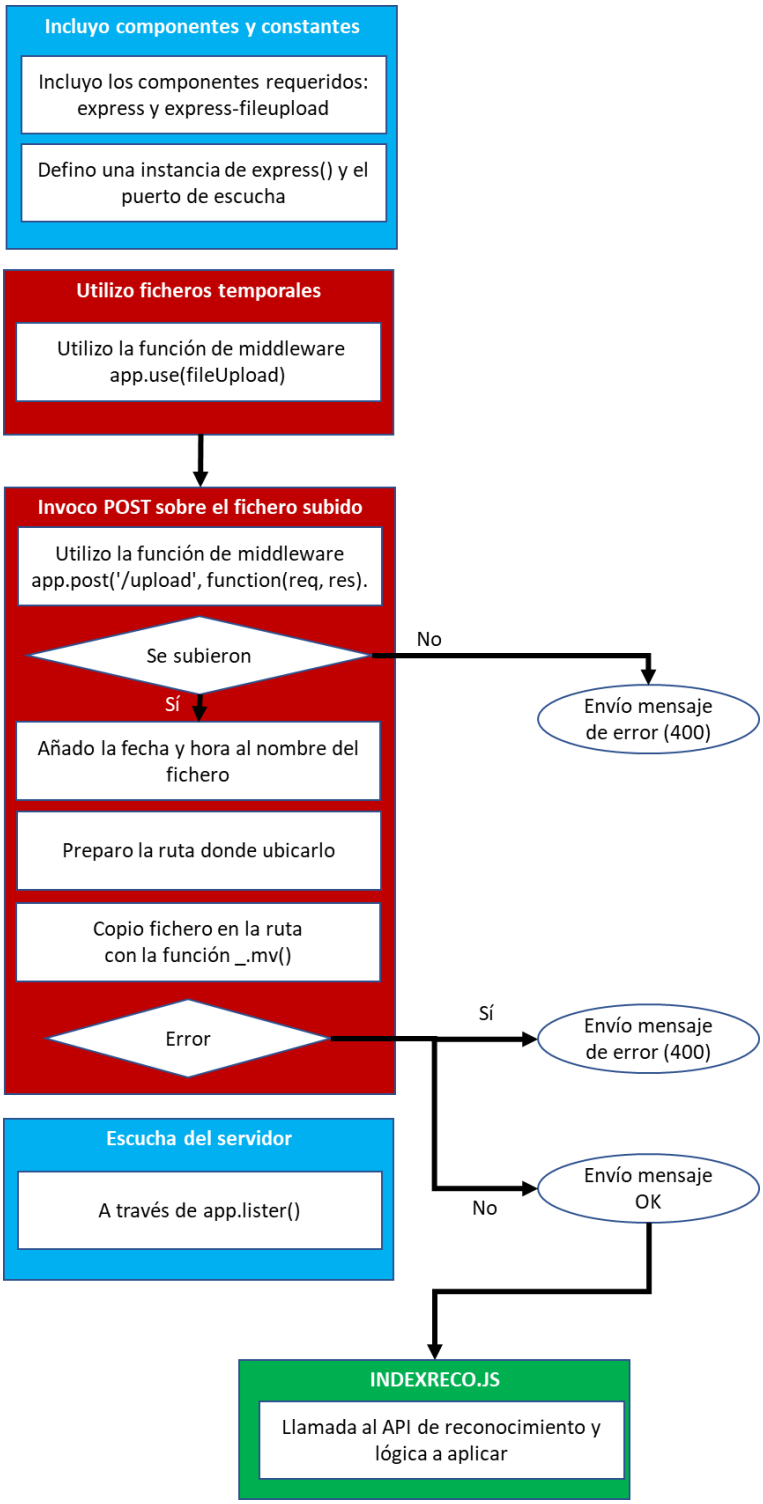


Ilustración 26 Esquema del código Upload.js que recibe la petición HTML POST

3.7 Desarrollo del servicio de Machine Learning y lógica de actuación

Considerando la solución elegida del servicio en la nube y a partir de las soluciones propuestas en el capítulo 2, decidí probar con el servicio proporcionado por Google, denominada AutoML Vision.

3.7.1 Servicios de Google de Visión

En la actualidad, dentro de la plataforma Google Clooud Platform (GCP) se ofrecen 2 APIs de visión.

1. **Modelos previamente entrenados**, cuyo uso de puede realizar o bien a través del API de Visión o mediante el servicio *Vision Product Search*.

2. **Modelos que pueden ser entrenados y personalizados** mediante *AutoML Vision* que también cuenta con un API.

En nuestro caso optamos por dicho modelo personalizado ya que el API general no era capaz de reconocer y etiquetar correctamente un ave en vuelo, y por tanto no nos sirve para determinar si es o no es rapaz.

Antes de comenzar a utilizar este servicio de Google, es necesario darse de alta con una cuenta gratuita en GCP (la mencionada plataforma de Google) y crear un proyecto, en nuestro caso de denominó “averapaz 87893”, y ahora sí, comenzamos con los pasos para poder construir un modelo desde cero.

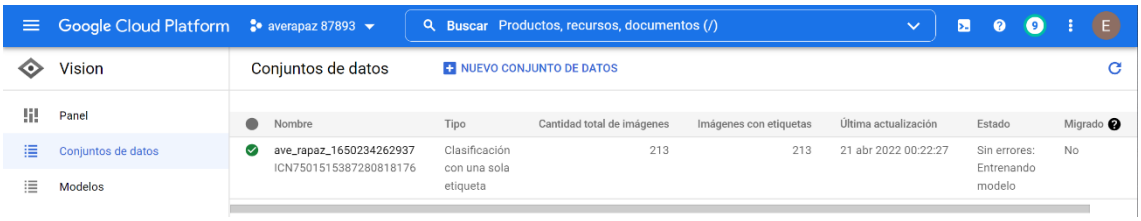


Ilustración 27 Proyecto ave_rapaz en al interfaz Google Cloud Platform

3.7.2 Creación de un modelo en AutoML Vision

Para poder comenzar a construir un modelo, lo primero es crear un conjunto de datos que será la base del entrenamiento.

En el proyecto, utilicé fundamentalmente una base de fotografías (Observation.org, 2022) con observaciones de todo el mundo sobre todo tipo de especies por usuarios que contribuyen a través de una comunidad sin ánimo de lucro.

Este portal cuenta con un buscador por especie, indicando la localización, fecha y el nombre del observador. Además, permite utilizar su contenido a través de la correcta cita de su referencia.

Gracias a dicha fuente de información pude construir una base de datos del conjunto de aves rapaces y no rapaces más comunes en la zona prevista de observación (Red Natura, 2022).

A continuación, listamos las especies de aves seleccionadas:

Especies de aves rapaces seleccionadas

- Águila Imperial Ibérica: 22 fotos
- Milano Negro: 22 fotos
- Águila Real: 13 fotos
- Buitre Leonado: 15 fotos
- Águila Culebrera: 6 fotos
- Búho Real: 4 fotos (que en realidad es una rapaz nocturna, por lo que sería muy extraña su observación durante el día)
- Buitre Negro: 13 fotos
- Busardo Ratonero: 10 fotos
- Cernícalo: 4 fotos
- Milano Real: 5 fotos

Especies de aves no rapaces seleccionadas

- Cigüeña: 16 fotos (una de las cuales una es negra)
- Golondrina: 15 fotos
- Gorrión Común: 13 fotos
- Mirlo Común: 16 fotos
- Paloma Común: 14 fotos
- Urraca: 12 fotos
- Vencejo: 15 fotos

En esta selección, me aseguré de que dichas fotos tuvieran suficiente variedad de ángulos, distancias, luminosidad y otros factores que faciliten el aprendizaje del sistema.

Además, reservé otro grupo de fotos de las mismas especies para pruebas posteriores. Una vez, construida esta base de imágenes, procedimos con los pasos necesarios para entrenar el modelo.

Ilustración 28 Fases para crear un modelo de Machine Learning

A.- Importación de Imágenes.

Para importar las imágenes es necesario subirlas a la plataforma GCP en alguno de los formatos soportados (en mi caso JPG) a un contenedor denominado *cloud storage*, organizado en carpetas que representen las diferentes categorías que vamos a clasificar.

Además, de cara al almacenamiento de los ficheros, es muy importante seleccionar la localización del servidor es “us-central1 (Iowa)” ya que por el momento es la única ubicación de Google donde ejecuta el servicio *AutoML Vision*.

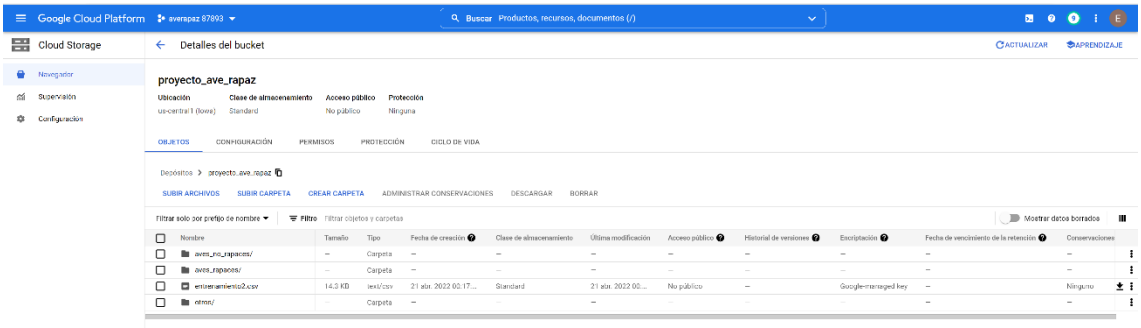


Ilustración 29 Conjunto de datos utilizados para el proyecto ave rapaz

Una vez subidas las imágenes, es necesario crear un fichero “csv” donde se indica la su ubicación dentro de cada carpeta. Dicho fichero será utilizado con posterioridad en la fase de entrenamiento.

La sintaxis utilizada será del tipo:

gs:// Destino en Cloud Storage, Etiqueta

para lo cual, existen diferentes estrategias para su construcción (AIAdventures, 2018). En mi caso, la opción más cómoda fue realizarlo a través de un Excel que concatenase toda la información y luego exportarlo como texto. A continuación, se indica una muestra del resultado de dicho fichero visualizado con el block de notas en formato UTF-8.

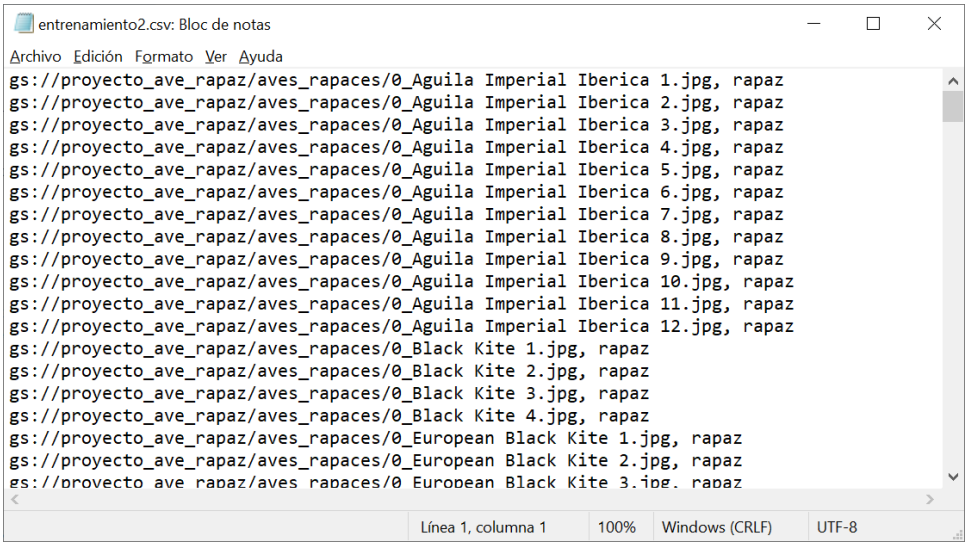


Ilustración 30 Aspecto del fichero “csv” para entrenamiento del modelo

Como puede observarse, cada foto de ave es etiquetada con únicamente dos posibles opciones: rapaz y no rapaz. Existía la opción de añadir una tercera categoría: “ninguno de los dos” en referencia a la no aparición de ave (cielo abierto) o cualquier otro evento (aviones o insectos).

Sin embargo, dado que el modelo de predicción establece un parámetro numérico, no vi necesario en ese momento añadir una tercera categoría pues en caso de no estar en ninguno de los dos casos, la predicción, sea cual fuere daría un parámetro numérico bajo en relación con un determinado umbral de certidumbre.

B.- Validar Etiquetado.

Una de las ventajas significativas de utilizar la interfaz de *Google Cloud Platform*, es que permite visualizar las imágenes etiquetadas, añadir nuevas etiquetas o cambiar las ya existentes. En mi caso no fue necesario, ya que iban todas etiquetadas desde el momento que preparé los datos en carpetas, sin embargo, es una opción muy útil, sobre todo para modelos con múltiples etiquetas.

En la pantalla adjunta, podemos ver esta funcionalidad sobre un total de 213 imágenes etiquetadas: 113 rapaces y 100 no rapaces.

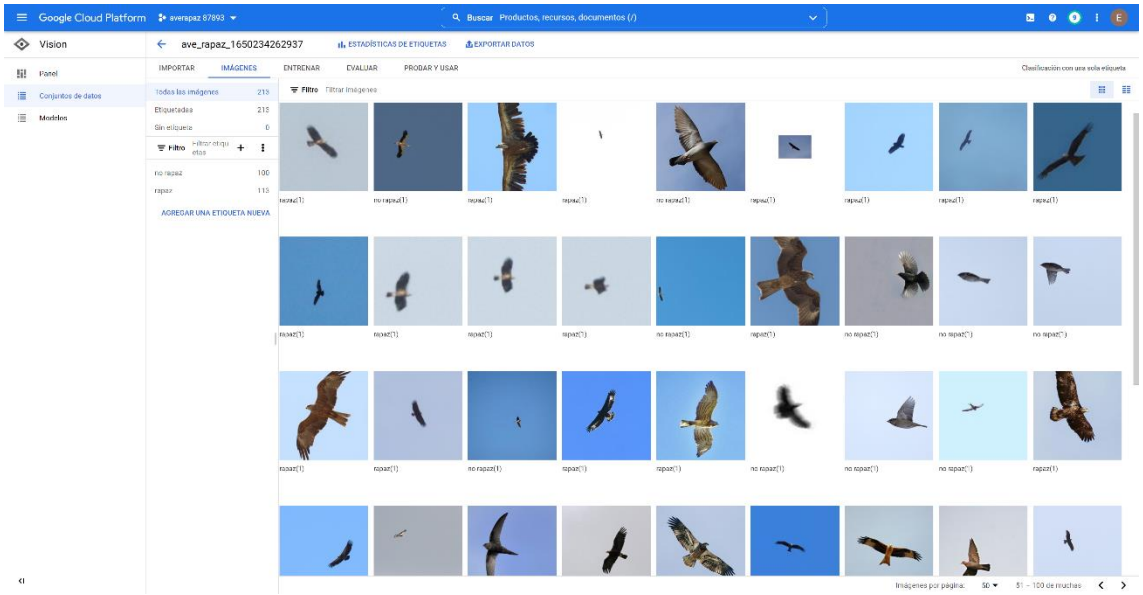


Ilustración 31 Imágenes etiquetades con el fichero "csv" vistes en la interfaz de GCP

C.- Entrenar.

En entrenamiento es una de las fases críticas del diseño, ya que consume recursos de procesamiento valiosos que implican un coste (en mi caso se incluyó dentro del crédito que concede Google por el uso de prueba de su plataforma), utilizando 16 horas de procesamiento.

En este sentido es importante antes de entrenar un modelo, estar bien seguros de que nuestra muestra es lo suficientemente amplia, con el fin de no tener que repetir el proceso, con el coste adicional que conlleva.

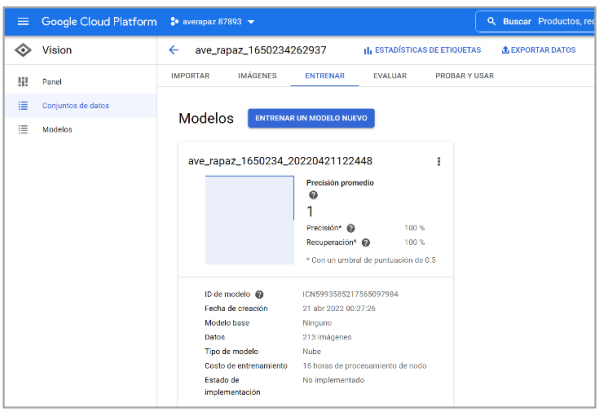


Ilustración 32 Características del modelo entrenado

D.- Evaluar.

El entrenamiento, ofrece como resultado dos parámetros de calidad del modelo:

- **Precisión**: La frecuencia de las predicciones que fueron correctas (positivas). Cuanto mayor sea la precisión, habrá menos predicciones con falsos positivos.

- **Recuperación**: Cuanto mayor es la recuperación, menos falsos negativos (o predicciones erradas) habrá.

Un modelo con alta precisión y baja recuperación, casi todas las clasificaciones en cada categoría serán correctas, pero dejará alguna si clasificar. Mientras que un modelo con alta recuperación y baja precisión, no se dejará ningún caso de una categoría sin clasificar, pero se incluirán algunos casos erróneos. Por ello se utiliza un parámetro ajustable denominado "límite de confianza" que permita maximizar la precisión y la recuperación.

En nuestro modelo, el nivel de "límite de confianza" del 50% logra un 100% de cumplimiento en ambos parámetros de calidad. De esta forma es capaz de clasificar correctamente todas las imágenes utilizadas en el entrenamiento incluidas las de prueba.

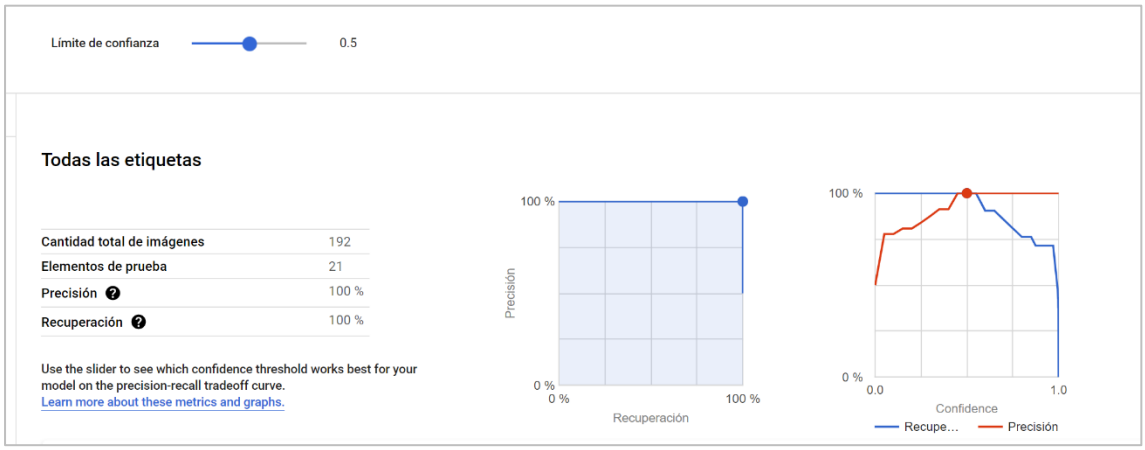


Ilustración 33 Resultado del entrenamiento del modelo en cuanto a precisión y recuperación

Adicionalmente se introduce una "matriz de confusión" en donde se muestra la frecuencia con la que el modelo clasificó cada etiqueta de manera correcta (en color azul) y las etiquetas que se confundieron con dicha etiqueta (en color gris), en nuestro caso no hubo ninguna clasificación errónea.

Ilustración 34 Matriz de confusión del modelo entrenado

Esta matriz es muy útil en modelos más complejos, ya que permite identificar los pares de etiquetas más difíciles de diferenciar, así como las imágenes que generan dicha confusión. Los resultados de dicha matriz se exportan a través de un archivo en formato CSV con el fin de poder analizar dicha información.

E.- Probar y Usar.

Una de las partes más interesantes del proceso, es la prueba del modelo entrenado, para lo cual pude utilizar una serie de fotos reservadas para tal fin, y así valorar su uso posterior. El testeo se puede realizar subiendo hasta un máximo de 10 imágenes y observando los resultados de la clasificación.

A continuación, muestro un par de ejemplos correspondientes a fotos realistas de baja resolución (un milano negro: “rapaz” y una cigüeña: “no rapaz”):

Ilustración 35 Ejemplos probados en la interfaz de GCP

La predicción es correcta en ambos casos, con unos índices superiores al 0,9 que nos permiten validar la viabilidad práctica del modelo. Sin embargo, con el objetivo de identificar sus límites, continué haciendo pruebas en búsqueda de errores, y pude encontrar la siguiente clasificación errónea:

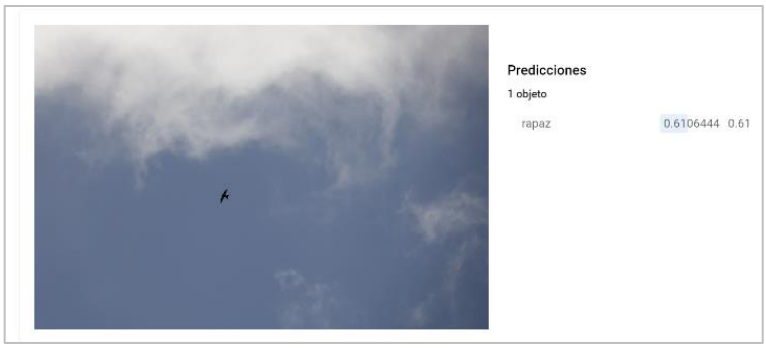


Ilustración 36 Ejemplo de clasificación errónea

Este caso es especialmente interesante ya que se trata de un vencejo volando en altura con una extensión de alas de planeo que confunde al sistema como si de un ave rapaz se tratase.

Este ejemplo, nos ilustra uno de los inconvenientes que mencionamos en el capítulo 2 sobre los sistemas de “*Deep learning*” en cuanto que hay una parte que permanece oculta para nosotros, que son las características que utiliza para realizar la clasificación. Es decir, no sabemos por qué decide.

De haber abordado el sistema mediante un clasificador K-NN o uno basado en reglas, con un vector de características basado en el conocimiento “heurístico” del problema, posiblemente la clasificación hubiera sido correcta.

Así pues, la primera característica: “relación rectangular”, ofrece por sí un valor bastante amplio en comparación con otras aves no rapaces (2,11 como vimos anteriormente) y puede llevarnos al error en la clasificación.

Sin embargo, al introducir la segunda característica: “ángulo de planeo”, la cual, según se aprecia en la foto, resulta en contra de la dirección del vuelo (podría estar en torno a -6º como vimos anteriormente), el mencionado clasificador K-NN, no habría tenido dudas de clasificarla como “no rapaz”. Tampoco la habría tenido dudas un clasificador basado en reglas pues es muy difícil que un ave rapaz tenga un “ángulo de planeo negativo”.

Posiblemente el clasificador “*Deep learning*”, de alguna manera esté dando un peso superior a la primera característica de “relación rectangular”, sin contar con que, en este caso, la curvatura de alas medida a partir del “ángulo de planeo”, aun siendo más excepcional en las observaciones, cuando aparece, anula la primera hipótesis de tratarse de un ave rapaz (regla heurística), pues dichas aves no muestran tal curvatura.

Esta reflexión nos sirve, por un lado, para conocer las limitaciones del sistema y por otro, para plantear un sistema que apoye con reglas heurísticas y distancias el clasificador de “*Deep learning*” implementado. Este sistema, podría realizar una extracción de posibles características: envergadura a través de la “relación rectangular”, curvatura de alas a través del “ángulo de planeo”, y otras como la “relación convexa de los extremos de las alas”, mencionadas anteriormente.

Dicho modelo de apoyo utilizaría su capacidad discriminante, mediante distancias o reglas entre categorías, para corregir si aplicase, las predicciones del sistema “*Deep learning*”.

En cuanto, el uso del sistema, la forma de hacerlo es a través del API AutoML Vision accesible desde varias tecnologías (API de Cloud Vision, 2022) tales como Python, Java y Node.js. Como antes comentamos, este aspecto condicionó la tecnología del servidor que recibe el HTTP POST. En nuestro caso, optamos por el desarrollo a través de Node.js, dada la sencillez y facilidad de desarrollo y uso.

3.7.3 Utilización del API de AutoML Vision

Para poder utilizar un modelo, es necesario seleccionar un proyecto de nuestra cuenta en Google y habilitar sus APIs que son 3:

- Cloud Storage
- Cloud AutoML API
- Google Cloud Storage JSON API

Es importante tener en cuenta que desde que se activa el modelo, Google nos va a empezar a facturar por el servicio incluso si no se realizan predicciones. En este sentido Google ofrece una detallada guía de precios, así como una calculadora, en función de los requerimientos de Hardware y disponibilidad, pudiendo llegar a ser un coste muy significativo si no se parametriza adecuadamente este uso del servicio.

En mi caso, seleccioné los siguientes parámetros:

- Sistema operativo: Debian, CentOS, CoreOS, Ubuntu or BYOL (Bring Your Own License)
- Serie: E2
- Tipo de máquina: e2-standard-2 (vCPUs: 2, RAM: 8GB)
- Tipo de nodo: n1-node-96-624 (vCPUs: 96, RAM: 624 GB)
- 2 hilos por núcleo
- Disco persistente estándar
- 24 horas 7 días (para poder utilizarlo por las noches también para hacer pruebas)

Una vez activados los APIs, hay que crear una cuenta de servicio y descargar un archivo con claves. Cada cuenta de servicio está asociada a un par de claves RSA pública/ privada, las cuales serán utilizadas por la API de credenciales que nos permitirá realizar la autenticación. La cuenta de servicio genera la siguiente información:

- Un id. alfanumérico *SA_NAME*. En nuestro caso: averapaz-432
- Una dirección de correo: SA_NAME@PROJECT_ID.iam.gserviceaccount.com. En nuestro caso: averapaz-432@averapaz-87893.iam.gserviceaccount.com
- Un id. único y permanente: 118129315167334332294
- Una descripción *SA_DESCRIPTION*. En nuestro caso: academic.
- Un nombre mostrado *SA_DISPLAY_NAME*. En nuestro caso: averapaz.
- El ID del proyecto *PROJECT_ID*. En nuestro caso: averapaz-87893.
- Además, se deben de establecer los usuarios que pueden tener acceso al servicio, los cuales serán la propia cuenta de correo creada y la del usuario principal. Se permiten varios roles, en este caso hemos elegido propietario.

A continuación, en la máquina donde vayamos a ejecutar el código de llamada al API (indexreco.js) incluimos la variable de entorno GOOGLE_APPLICATION_CREDENTIALS en la ruta de acceso al archivo de claves. Esta operación la podemos hacer desde la línea de comandos según el sistema operativo, en el caso de Windows:

```
set GOOGLE_APPLICATION_CREDENTIALS=C:\Users\emili\Desktop\nodejs\averapaz-
87893-bfb777f6e151.json
```

Una de las posibilidades de la cuenta de servicio, es la capacidad de crear el modelo desde cero mediante programación. Para habilitar dicha capacidad será necesario instalar CLI (interfaz de línea de comandos de Google), y posteriormente ejecutar los siguientes comandos:

```
gcloud auth login

gcloud projects add-iam-policy-binding averapaz-87893 --
member="user:emiliano1440@gmail.com" --role="roles/automl.admin"


gcloud projects add-iam-policy-binding averapaz-87893 --
member=serviceAccount:averapaz-432@averapaz-87893.iam.gserviceaccount.com --
role="roles/automl.editor"
```

Dado que ya tenemos el modelo, iremos directamente al código necesario en node.js necesario para llamar al API, si bien, antes nos tenemos que asegurar que las siguientes variables de entorno están bien configuradas:

```
set PROJECT_ID=averapaz-87893
set REGION_NAME=us-central1
```

Así como instalar la siguiente librería a utilizar por el código de nodejs:

```
npm install --save @google-cloud/automl
npm install --save fs-extra
```

Para poder realizar el código de llamada al API, tendremos que utilizar una serie de nuevas funciones que a continuación detallo.

Las específicas de la librería AutoML (Prediction Service Client, 2022):

- **PredictionSercice Client():** se trata del servicio del automl.v1 para el cual es necesario crear una instancia del mismo. Este servicio valida las credenciales que nos enviaron cuando creamos la cuenta de cliente y que tenemos almacenadas en un archivo “JSON”.

- **Predict():** cada vez que realizo una predicción utilizo esta función en donde envío la identificación del modelo de clasificación, la localización y el proyecto. Asimismo, le envío la variable que referencia el fichero con la imagen a predecir.

- **Await()**: espera la respuesta del cliente de predicción. Esto lo podemos hacer cuando el cliente nos avisa que la operación puede llevar un tiempo de operación.

- **Payload:** es una estructura de datos donde la función de predicción devuelve los valores y parámetros devueltos por el API, en este caso hemos utilizado:
    - La categoría clasificada: annotationPayload.displayName
    - El nivel de certidumbre: annotationPayload.classification.score

Y las correspondientes a la librería fs-extra (Geeks FileSync, 2022)

- **Fs.readFileSync(filePath):** es un método que permite leer un fichero de forma asíncrona y no bloqueante. El parámetro especifica la ruta donde se sitúa dicho fichero

- **Fs.move():** es una función que permite mover un fichero de una fuente a un destino especificado por el usuario.

Este proceso de llamada al API, deberá de hacerse cada vez que se suba un fichero al servidor. A continuación, mostramos el esquema del código **Indexreco.js** capaz de clasificar una imagen situada en la carpeta recursos y moverla a otra carpeta dependiendo de la clasificación en rapaz o no rapaz.

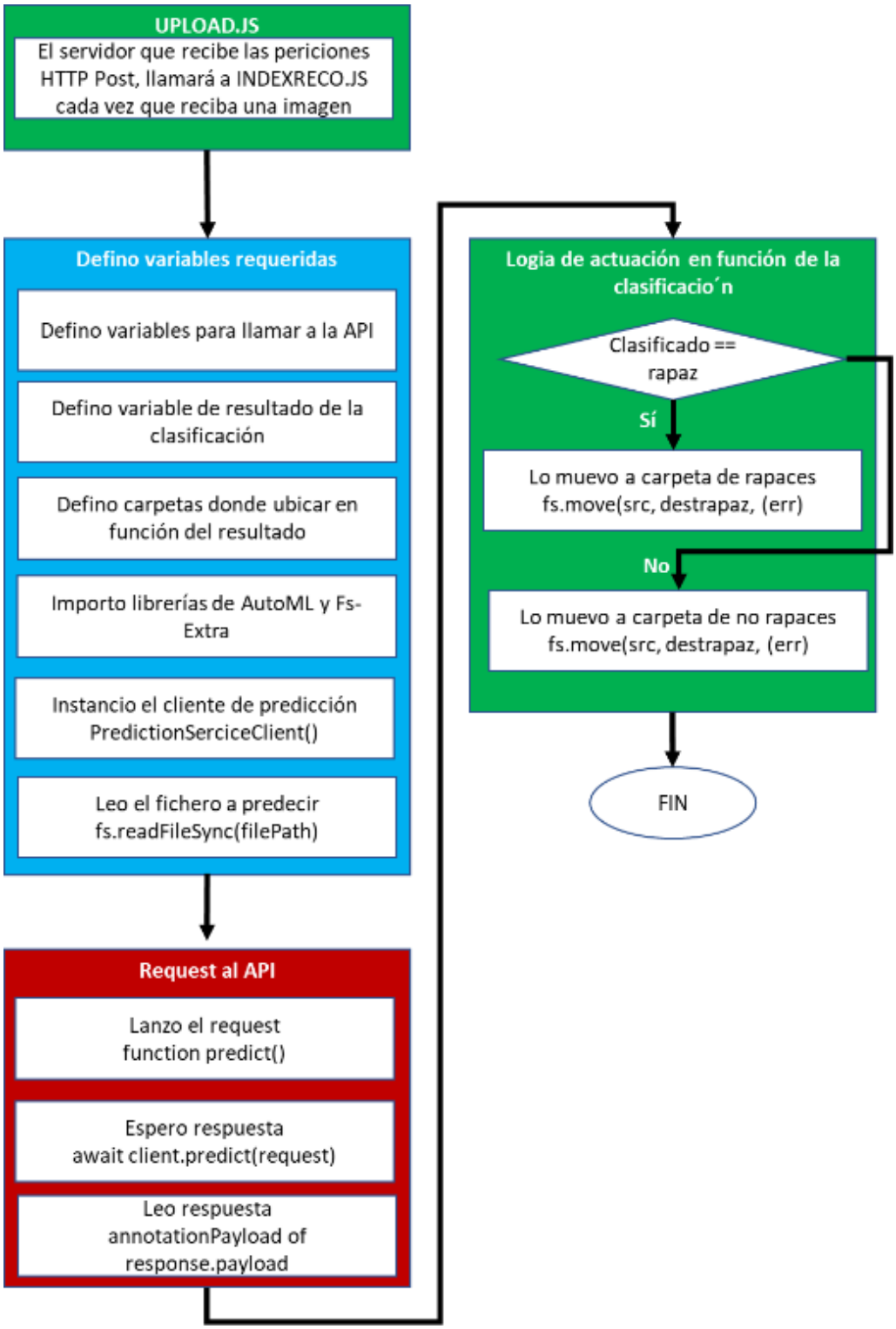


Ilustración 37 Esquema del código Indexreco.js que envía la petición al API de VisionML

Prueba del código

Para asegurarnos que el código de reconocimiento funciona correctamente hemos efectuado dos pruebas. Una primera situando una imagen sin clasificar denominada “upload.jpg” en el directorio /recursos respecto al código indexreco.js. Se trata de la misma foto de la cigüeña que anteriormente identificamos con el testeo de prueba. Posteriormente repetiremos la prueba, pero con una imagen correspondiente al milano negro utilizada previamente.

Imagen sin clasificar “upload.jpg”:

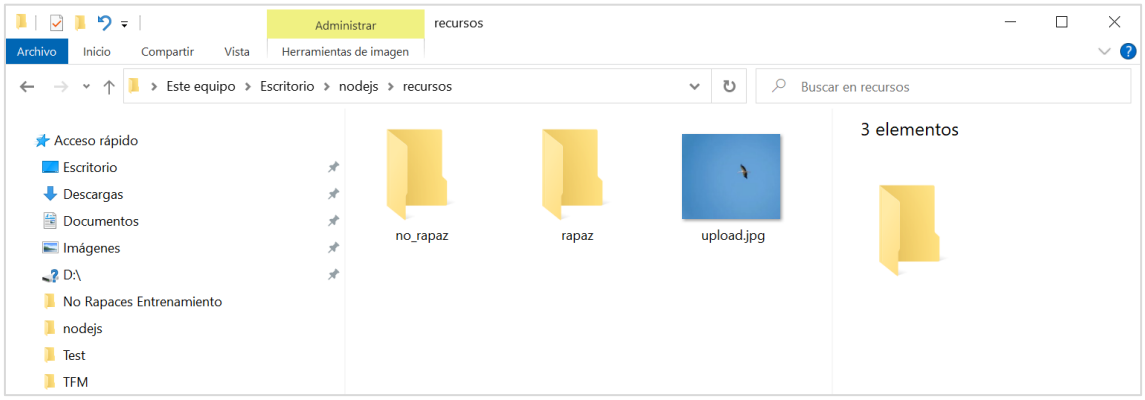


Ilustración 38 Escenario inicial con una fotografia subida sin clasificar

Llamada al API a través de la línea de comandos ejecutando node indexreco.js

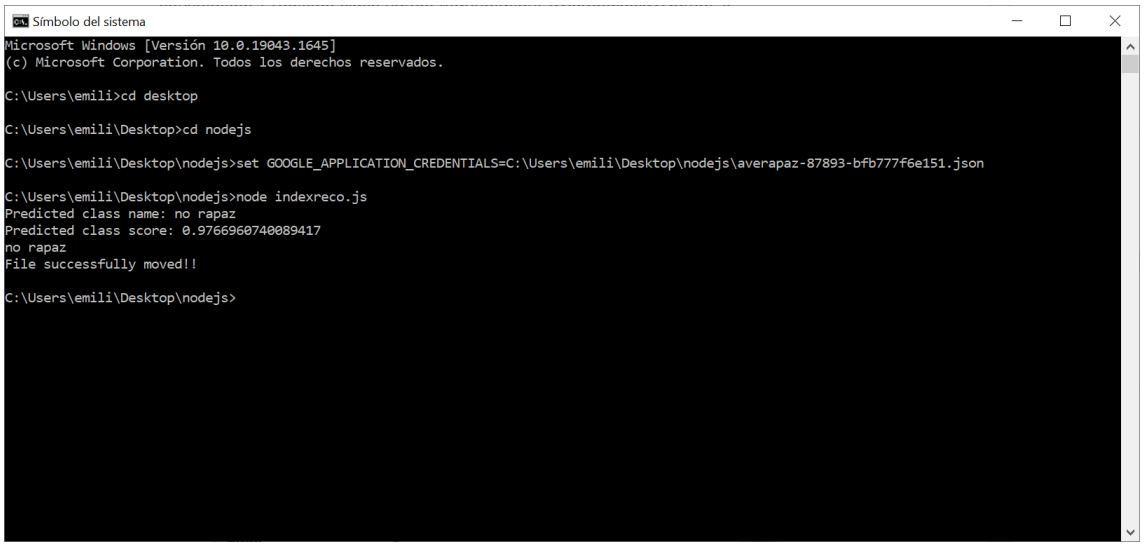


Ilustración 39 Resultado de la clasificación del API de AutoML desarrollado

Observamos que nos devuelve el nombre: no rapaz, con un score de 0,9766, es decir, el mismo que obtuvimos con la clasificación en el testeo en la interfaz gráfica. Por último, inspeccionamos la carpeta “recursos”, y vemos que se ha movido la imagen a la carpeta “no rapaz”

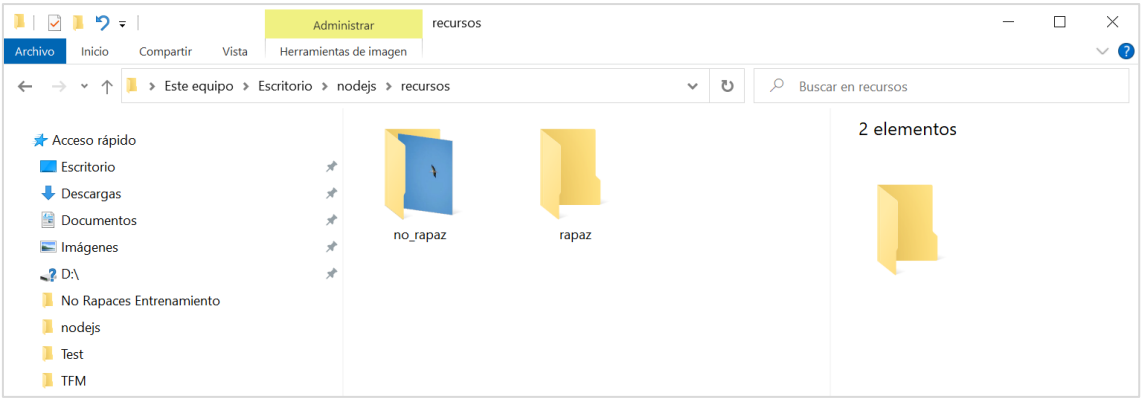


Ilustración 40 Resultado de la acción tomada de reubicación en la carpeta de aves no rapaces

A continuación, vamos a repetir la operación, pero ahora con la imagen del “milano negro” que situaremos en la carpeta /recursos con el nombre “upload.jpg”, ya que a priori, no sabemos cuál es su categoría.

Imagen sin clasificar “upload.jpg”

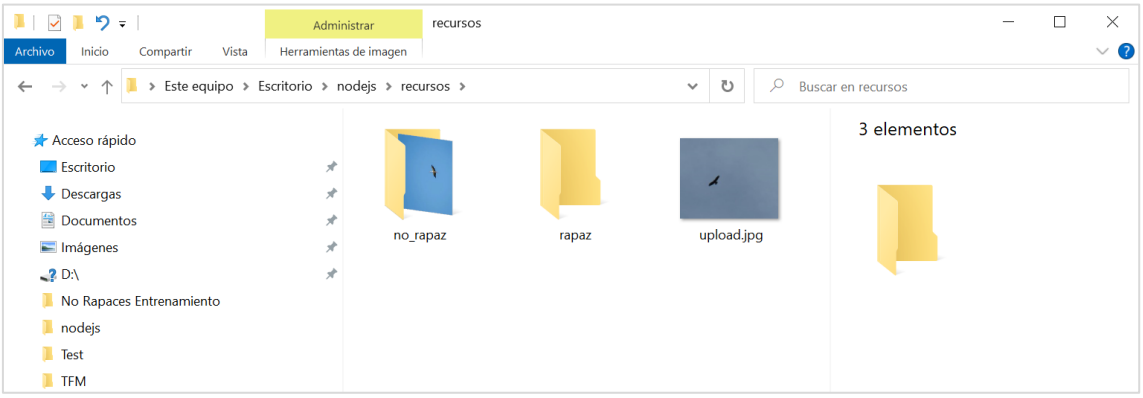


Ilustración 41 Nuevo escenario inicial con una fotografia subida sin clasificar

A continuación, volvemos a llamar al API a través de la línea de comandos ejecutando node indexreco.js

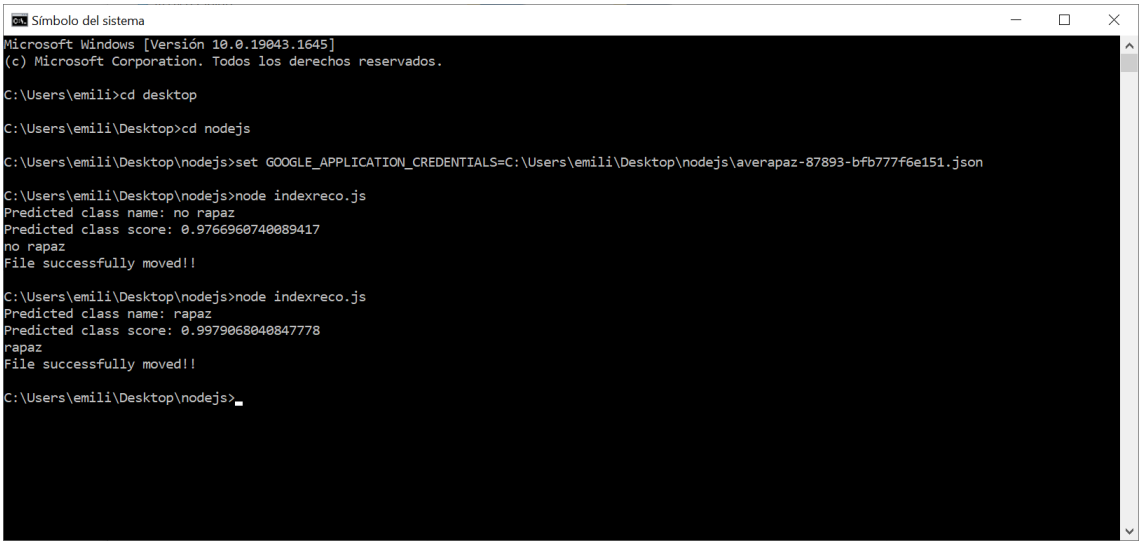


Ilustración 42 Resultado de la clasificación del API de AutoML desarrollado

En este caso, nos devuelve el valor: rapaz, con un score de 0,9979, que es también, el mismo que obtuvimos con la clasificación en el testeo en la interfaz gráfica. Asimismo, vemos que el fichero se ha movido la imagen a la carpeta “rapaz”.

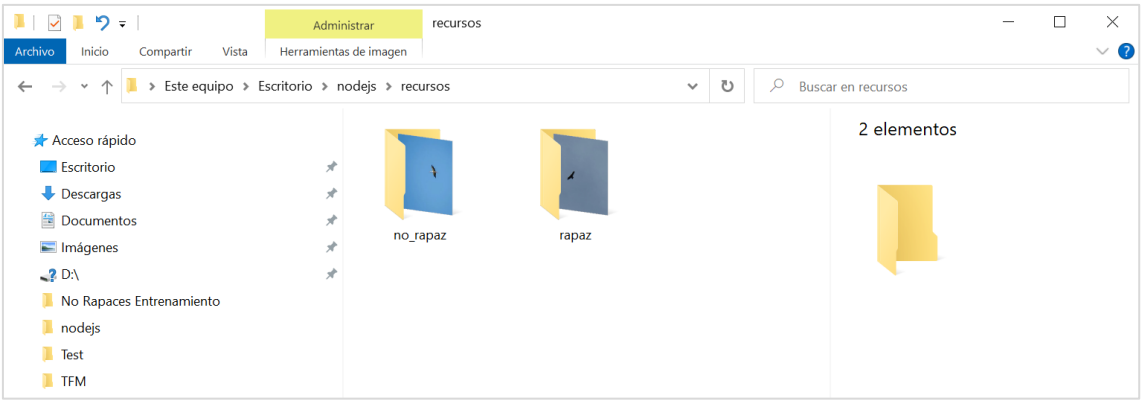


Ilustración 43 Resultado de la acción tomada de reubicación en la carpeta de aves rapaces

De esta forma verificamos el uso del API AutoML y el código capaz de leer una imagen situada en la carpeta origen y moverla a una carpeta destino según el resultado de la predicción del API, lo cual era el uno de nuestros objetivos principales.

4. Conclusiones

4.1 Lecciones aprendidas

En todo proyecto, es importante hacer una reflexión sobre las lecciones aprendidas cuando dicho proyecto se acerca a su fin, en ese sentido a continuación menciono las más relevantes:

- Se trata de un proyecto con varios retos difíciles de alcanzar en su conjunto. Así, por ejemplo, la conectividad de IoT fue diseñada para aplicaciones que utilizan bajas tasas de transmisión a través de dispositivos con una reducida capacidad de cálculo. El tratar de hacer una aplicación con mayores prestaciones, como transmitir imágenes, supone ir en contra de dicha limitación.

- No he sido el único desarrollador que se ha encontrado con esta situación. Vistos los proyectos realizados por otras personas en Internet y como tratan de superar dichas limitaciones, en general se percibe una necesidad latente a crear aplicaciones con mayor contenido multimedia y/o capacidad de procesamiento, por lo que también ser aprecia una tendencia de los fabricantes a ofrecer dispositivos con mejores características, como por ejemplo placas con cámaras integradas, con cada vez mayor capacidad.

- En principio me esperaba que en un proyecto IoT predominase la parte electrónica, la programación en C/C++ y el uso de servicios en la nube predefinidos que no requiriesen programación. Sin embargo, cuando nos salimos del estándar de las aplicaciones disponibles en la nube para el manejo y gestión de “cosas”, es igualmente importante, incluso más, los conocimientos de programación web: HTML cliente servidor, peticiones GET-POST, programación en PHP, JS, llamada a servicios API, etc.

- La inteligencia artificial es una herramienta fundamental hoy en día con múltiples utilidades. En este proyecto la hemos usado para reconocer imágenes, pero podríamos haberla utilizado igualmente para interpretar los datos de las observaciones, incluso hacer predicciones. Por ello, la perspectiva de diseño de un proyecto es conveniente hacerla siempre en cuanto a “para qué se quiere desarrollar algo” con qué finalidad, y en ese sentido, ver como la inteligencia artificial nos puede ayudar a hacerlo y podemos sacar el mejor partido de ella.

- El tiempo es un recurso fundamental para dimensionar correctamente el alcance de un proyecto y su posterior dedicación. En este caso, fue fundamental hacer una óptima gestión de los esfuerzos pues se trataba de abarcar diferentes retos, y tratar de buscar una solución suficientemente viable, que, sin ser óptima, nos permitiera pasar al siguiente reto.

4.2 Reflexión sobre el logro de objetivos

Podríamos decir que hemos alcanzado en buena cantidad los objetivos planteados en un principio, de hecho, los más difíciles, como el de subir una imagen a un servidor o utilizar un API de un sistema de *Machine Learning* desarrollado a medida de nuestras necesidades.

Sin embargo, algunos objetivos no los hemos podido alcanzar como el utilizar algún servicio ya existente en la nube para llevar la estadística de los datos, o el haber dotado a nuestro dispositivo de una carcasa y batería para utilizarlo en entornos más realistas.

En este sentido hubo que priorizar los esfuerzos a aquellos más necesarios relacionados con los objetivos fundamentales del proyecto como “prueba de concepto” de la aplicación perseguida.

4.3 Análisis crítico sobre la planificación y la metodología seguida

En principio la planificación planteaba tres fases secuenciales de desarrollo: hardware, conectividad y servicios. Y por último pruebas integradas. Tras finalizar estas fases y aun considerando que en general fueron correctas, podemos decir que:

- La fase de desarrollo de servicios (la 3ª) condicionó la tecnología de las anteriores, ya que la llamada al API eficaz se podía hacer con tecnologías tipo Java o Python y no así con PHP que fue la tecnología inicialmente utilizada en el servidor. Por tanto, tal vez tendría que haber empezado por dicha fase del desarrollo del servicio o haberla abordado en paralelo con las otras fases.

- La fase de hardware (la 1ª) tal vez consumió excesivo tiempo por el hecho de tener que probar varias placas de Arduino previamente. De haber estudiado todo el problema en su conjunto, podría haberme decantando antes por una placa de la familia ESP mucho antes.

- La fase de conectividad (la 2ª) es en realidad la del servidor que recibe la imagen y la almacena. En este caso y por falta de tiempo, fue muy ligada a complementar la fase anterior de hardware. Sin embargo, tendría que haber abordado desde un principio el enlace a otros servicios en la nube, así como con qué tecnología de servidor pueden ser llamados.

El análisis crítico llega a la conclusión que tal vez las fases tendrían que haber tenido un orden inverso, o al menos de forma paralela.

4.4 Líneas de trabajo futuro

Este trabajo, como prueba de concepto, resuelve la viabilidad de llevar a cabo la idea inicial, si bien deja abiertas una serie de líneas de trabajo que permitan su mejora e implementación práctica.

- En la parte del **servidor**:

    - Se podría implementar el método que llamara a un servicio que contabilizara la estadística de registro de datos subidos y elaboración de un mapa de calor. Este servicio se podría desarrollar en un tiempo razonable a través de la plataforma Thinger io (thinger_io, 2022). Además, de cara a aumentar la riqueza de datos, podríamos incorporar información del entorno tal como temperatura y humedad, las cuales son relativamente más sencillas de obtener que en el caso de las imágenes.

- En la parte de **reconocimiento** de imágenes:

    - A nivel del servicio de aprendizaje utilizado, se podría aumentar el número de categorías a clasificar, creando la categoría “desconocida” ya que estos casos, sólo los podemos discriminar en función del nivel de certidumbre recibida (por ejemplo, menor a 0,8).

- Otra mejora en la misma línea sería crear grupos de especies con formas similares, es decir en lugar de utilizar una clasificación tan sencilla como rapaz y no rapaz, ampliar el abanico a grupos comparables. En este sentido, para poderlo llevar a cabo, podríamos utilizar el “aprendizaje no supervisado” y así conocer cuántos grupos surgen de forma natural en base únicamente a su semejanza.

- Asimismo, también sería interesante abordar el problema con técnicas de inteligencia artificial más clásicas, analizando las características más relevantes a la hora de discriminar las categorías definidas tal y como hicimos en la presentación del problema de reconocimiento. El desarrollar un clasificador basado en K-NN con las diferentes distancias comentadas, o bien basado en reglas de decisión, nos permitiría mejorar las predicciones del clasificador de *deep learning* desarrollado en la nube.

- Una de las líneas más interesantes a explorar es la de realizar el mismo sistema con la lógica de reconocimiento en el propio microcontrolador, lo cual nos ahorraría entre otros, el uso del servicio en la nube y podríamos optimizar el recurso radio a utilizar. Ya hemos hablado previamente de dicha opción, y la curva de aprendizaje que supone aprender los entornos de programación Tiny ML.

- En la parte de **hardware**:

  - El poder llevar a cabo la lógica en el microcontrolador, nos permitiría cambiar el tipo de conexión radio, a opciones de mayor alcance como SixFox y Lora, ya que disminuiría enormemente el flujo de datos hacia el servidor.

  - En la parte hardware, pese a funcionar de forma inalámbrica mediante WiFi, el sistema no dispone de alimentación autónoma, lo cual sería necesario para pasar de “prueba de concepto en laboratorio” a un prototipo preparado para ser utilizado en un entorno real. Asimismo, sería necesario construir una carcasa, para lo cual podemos recurrir a implementarla mediante un diseño imprimible 3d disponible en el siguiente enlace (ESP32 CAM Case, 2022).

  - Asimismo, en la parte de hardware, se podría implementar algún tipo de accionamiento en caso de reconocimiento enviado por el servidor. Por ejemplo, en caso de detectar una rapaz, se podría iniciar una grabación en vídeo hasta que se dejase de detectar, que podría quedar registrada en una tarjeta de memoria en la placa.

- Por último, y en cuanto a aportación a la sociedad

  - El poder llegar a desarrollar un producto totalmente operativo, permitiría aportar imágenes y conocimiento a grupos de interés y comunidades como la mencionada (Observation.org, 2022) u otras preocupadas por el impacto del desarrollo urbanístico tiene en la conservación de las especies protegidas y fauna en general.

5. Glosario

- 4G: 4ª generación de tecnología de telefonía móvil. Esta tecnología desarrolla el estándar *long term evolution* (LTE) alcanzando velocidades de 100Mbps
- API: representa las siglas en inglés de *Applicarion Programming Interface*, se refiere a un conjunto de funciones y procedimientos que se ofrecen para ser utilizados mediante otro software como una capa de abstracción.
- AWS: Es una empresa subsidiaria de Amazon que ofrece servicios de computación en la nube a demanda de los clientes.
- BLE: Son las siglas en inglés del estándar de comunicación *Bloetooth Low Enegy* diseñado para utilizarse con un bajo consumo de energía
- CNN: Son las siglas en inglés para referirse a *convolutional neural network*, o redes neuronales convolucionales en referencia a redes neuronales que utilizan el operador “convolución”, o capacidad de filtrar datos según el entrenamiento previo.
- FOV: Son las siglas en inglés para referirse a *field of view* o el ángulo de detección de un sensor de radiación electromagnética, como por ejemplo de luz visible.
- FPGA: Son las siglas en inglés de *field programable gate* array para referirse a una matriz de puertas lógicas programable
- FPS: Son las siglas en inglés de *frame per second*, para referirse al número de fotogramas por segundo.
- FTDI: se refiere a un cable para, junto con sus drivers, convertir estándares RS232 o Transmisión serie TTL hacia o desde USB. El acrónimo viene del nombre de la empresa denominada *Future Techonology Devices Internatinoal Limited*.
- GCP: son las siglas de Google Cloud Platform, o plataforma de Google donde se ofrecen servicios en la nube a demanda de clientes
- GPIO: sirve para identificar una patilla de un microprocesador o una placa como propósito general de entrada o salida, en inglés *general-purpose input/output*
- HaLow: se refiere al estándar WiFi 802.11 ah para trabajar a baja potencia y largo alcance.
- HTML: Son las siglas en inglés de hyper text markup language, es el estándar de documentos con etiquetas o marcas, para poder ser accedidos mediante un navegador
- IAB: Son las siglas en inglés de Internet Architecture Board, el cual es el comité responsable de monitorizar y desarrollar Internet a través de una serie de grupos de trabajo
- IDE: son las siglas en inglés de *integrated development environmet*, o entorno para desarrollar programas.
- IEEE: son las siglas en inglés de *Institute of Electrical and Electronics Engineers* es una asociación dedicada a la normalización y promover el desarrollo tecnológico.
- IETF: son las siglas en inglés de Internet *Engineering Task Force*, como el grupo de trabajo más conocido del IAB.
- IoT: son las siglas en inglés de Internet of Things, o Internet de las cosas, que hemos visto en la memoria
- JS: son las siglas en inglés de *JavaScript*, el cual es un lenguaje de programación interpretado orientado principalmente a cliente, pero también a servidor.

- JSON: es un estándar en formato texto sencillo para el intercambio de datos. El acrónimo es *JavaScript object notation*.
- K-NN: es un algoritmo de clasificación denominado *k-nearest neighbors* o los k vecinos más cercanos.
- LoRa: es el acrónimo de *long range*, o largo alcance referencia a la tecnología de modulación propietaria por Cycleo, Semtech. El protocolo de comunicación se define en LoRaWAN a través de una alianza sin ánimo de lucro.
- LPWAN: son las siglas en inglés de *Low Power Wide Area Networks*, o redes de área local de baja potencia.
- LTE-M: dentro del estándar LTE, se define LTE-MTC o *machine type communication*, como su aplicación LPWAN.
- M2M: se refiere a la expresión en inglés *machine to machine*, o comunicación entre dos máquinas
- MIPI: se refiere a la asociación *Mobile Industry Processor Interface* que especifican interfaces para dispositivos como cámaras o pantallas.
- NB-IoT: se refiere a narrow band IoT, en relación a un estándar LPWAN de banda estrecha.
- PHP: es un lenguaje de programación dirigido a la programación web. Inicialmente significaba *personal home page*, per actualmente se denomina *hypertext preprocessor*.
- REST: son las siglas en inglés de representacional state transfer, o transferencia de estado representacional, lo cual se utiliza en la actualidad como cualquier interfaz que utilice HTTP para obtener datos o indicar operaciones en los datos.
- RPC: son las siglas en inglés de *remote procedure call* o llamada a un procedimiento remoto,
- SigFox: es un proveedor global de servicios de IoT que utiliza una tecnología de banda estrecha.
- SPI: son las siglas en inglés de *serial pheripheral interface*, en referencia a un estándar de comunicaciones or un interfaz entre dispositivos electrónicos.
- SRAM: se refiere a una memoria de tipo RAM estática o basada en semiconductores.
- WiFi: es una familia de protocolos de comunicación inalámbrica definidos en los estándares IEEE 802.11
- ZEPA: zona de especial protección de aves
- Zigbee: se reviere a un protocolo que de comunicación inalámbrica definido en el estándar IEEE 802.15.4

6. Bibliografía

Trabajos citados

802.11ah, I. (12 de 3 de 2022). *https://es.frwiki.wiki/wiki/IEEE_802.11ah*.

AIAdventures, A. V. (2018). *AutoML Vision*. Obtenido de https://www.youtube.com/watch?v=kgxfdTh9lz0

AlimentaciónESP32. (2022). Obtenido de https://www.radioshuttle.de/es/medias-es/informaciones-tecnicas/esp32-alimentado-por-bateria/

API de Cloud Vision. (2022). Obtenido de https://cloud.google.com/vision/automl/docs/tutorial?hl=es-419

App.post. (2022). Obtenido de https://www.tutorialspoint.com/express-js-app-post-method

Arducam. (13 de 3 de 2022). *https://www.arducam.com/compact-camera-module-oem-odm/*.

Arduino. (13 de 3 de 2022). *https://en.wikipedia.org/wiki/Arduino*.

Arduino_Cloud. (13 de 3 de 2022). *https://docs.arduino.cc/cloud/iot-cloud*.

Artificial_Intelligence. (13 de 3 de 2022). *https://www.britannica.com/technology/artificial-intelligence*.

Atzori, I. M. (2010). The Internet of things: a survey. *Computer Network Sci Direct*, 2787-2805.

Baker, M. C. (2001). Bird Song Research: The Past 100 Years. *Bird Behavior*, 3-50.

Blynk, D. s. (1 de 3 de 2022). *https://es.ephesossoftware.com/articles/diy/getting-started-with-blynk-simple-diy-iot-devices.html*.

Blynk_io. (13 de 3 de 2022). *https://blynk.io/*.

Caffe. (13 de 3 de 2022). *https://caffe.berkeleyvision.org/*.

Caffe2. (13 de 3 de 2022). *https://caffe2.ai/*.

CamArduinoUNO. (2022). Obtenido de https://www.arducam.com/wp-content/uploads/2017/12/ArduinoConnection-.jpg

Capan, T. (2022). *Tutorial Node.js*. Obtenido de https://www.toptal.com/nodejs/por-que-demonios-usaria-node-js-un-tutorial-caso-por-caso

Cayenne. (13 de 3 de 2022). *https://developers.mydevices.com/cayenne/features/*.

Clasificador de mínima distancia. (2022). Obtenido de https://www.fceia.unr.edu.ar/dip/Clasificador_minima_distancia.pdf

ComputaciónClasificación. (2022). Obtenido de https://slideplayer.es/amp/3497975/

Contour Features. (2022). Obtenido de https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html

Core. (13 de 3 de 2022). *https://store.arduino.cc/collections/core-family*.

Cornell. (Marzo de 2022). *https://merlin.allaboutbirds.org/*. Obtenido de https://merlin.allaboutbirds.org/.

Cornell-birdnet. (12 de 3 de 2022). *https://birdnet.cornell.edu/*.

Dan Stowell, M. D. (2011). Birdsong and C4DM: A survey of UK birdsong and machine recognition for music researchers. *Centre for Digital Music*.

Dartmouth. (3 de 2022). *https://en.wikipedia.org/wiki/Dartmouth_workshop*.

DeepLearning. (13 de 3 de 2022). *https://rubialesalberto.medium.com/deep-learning-qu%C3%A9-librer%C3%ADa-escoger-tensorflow-o-pytorch-fe26ad892ac7*.

Dominique Chabot, C. M. (2016). Computer-automated bird detection and counts in high-resolution aerial images: a review. *Journal of field ornithology*, Vol.87 (4), p.343-359.

Eastwood, E. (1967). *Radar Ornithology.* London: Methuen.

EDX Certificate. (Mayo de 2022). Obtenido de https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning

ESP32 CAM Case. (2022). Obtenido de https://www.yeggi.com/q/esp32+cam+case/

ESP32 CAM FTDI. (2022). Obtenido de https://randomnerdtutorials.com/program-upload-code-esp32-cam/

ESP32 CAM pinout. (2022). Obtenido de https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/

ESP32 IO. (2022). Obtenido de https://esp32io.com/tutorials/esp32-http-request
Espressif. (13 de 3 de 2022). *https://www.espressif.com/*.
Express, G. (2022). *Express file upload*. Obtenido de https://github.com/richardgirges/express-fileupload/tree/master/example
fileupload. (2022). Obtenido de https://sebhastian.com/express-fileupload/
Freund Y, S. R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Computat Learn Theory*, 23-27.
Geeks FileSync. (2022). Obtenido de https://www.geeksforgeeks.org/node-js-fs-readfilesync-method/
GirhubCapturaHTML. (2022). Obtenido de https://github.com/ArduCAM/ArduCAM_ESP8266_UNO/blob/master/libraries/ArduCAM/examples/ESP8266/ArduCAM_ESP8266_UNO_Capture/ArduCAM_ESP8266_UNO_Capture.ino
Github ESP32 CAM. (2022). Obtenido de https://github.com/espressif/esp32-camera
github_EspCam.h. (2022). Obtenido de https://github.com/espressif/esp32-camera/blob/master/driver/include/esp_camera.h
githubArduino. (2022). Obtenido de https://github.com/ArduCAM/Arduino
Habash, R. (2018). *Green Engineering: Innovation, Entrepreneurship and Design 1st Edición.* Boca Raton, FL: CRC Press. Obtenido de https://www.amazon.com/-/es/Riadh-Habash/dp/1138035882/ref=sr_1_4?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=34AHWPTIGUDK&keywords=green+engineering&qid=1645299321&s=books&sprefix=green+engineering%2Cstripbooks%2C148&sr=1-4&asin=1138035882&revisionId
Hatch, M. (2013). *The Maker Movement Manifesto: Rules for Innovation in the New World of Crafters, Hackers, and Tinkerers.* McGraw-Hill.
HTTP headers ContentType. (2022). Obtenido de https://www.geeksforgeeks.org/http-headers-content-type/
HTTP Request. (2022). Obtenido de https://arduinogetstarted.com/tutorials/arduino-http-request
Incze, A., Jancso, H.-B., Szilagyi, Z., & Sulyok, C. (2018). Bird Sound Recognition Using a Convolutional Neural Network. *IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY).*
ISM. (13 de 3 de 2022). *https://aprendiendoarduino.wordpress.com/tag/banda-ism/*.
Kaduzi.us. (2022). Obtenido de https://www.kaduzi.us/esp32cam-timelapse-nodejs-server/
Kelling, S., Gerbracht, J., Fink, D., Lagoze, C., Wong, W.-K., Jun Yu, T. D., & Gome, C. (2013). eBird: A Human/Computer Learning Network for Biodiversity Conservation and Research. *Ai Magazine*.
Keras.io. (Marzo de 2022). *https://keras.io/*. Obtenido de https://keras.io/.
Las rapaces forestales en España. (2011). Obtenido de https://www.miteco.gob.es/es/biodiversidad/temas/inventarios-nacionales/36-rapacesforestales_tcm30-207977.pdf
LeCun Y, B. L. (1998). Gradient-based learning applied to document recognition. *Proc IEEE.*
Length, U. C. (12 de 3 de 2022). *https://thedarkroom.com/focal-length/*.
Lucas, J. (2019). *NodeJS*. Obtenido de https://openwebinars.net/blog/que-es-nodejs/
Machine_Learning. (13 de 03 de 2022). *https://programarfacil.com/blog/vision-artificial/que-es-machine-learning/*.
Marin, M., & Bodgan, F. (2019). Automatic Identification of Flying Bird Species Using Computer Vision Technique for Ecological Data Analysis. *The Annals of "Dunarea de Jos" University of Galati Fascicle IX Metallurgy and Materials Science*, 46-49.
Mini ESP32-CAM. (2022). Obtenido de https://www.diymore.cc/products/esp32-cam-wifi-wireless-module-esp32-serial-to-wifi-esp32-cam-spi-flash-bluetooth-development-board-with-ov2640-camera-module
MIPI_CSI_2. (13 de 3 de 2022). *https://www.mipi.org/specifications/csi-2*.
Mkr. (13 de 3 de 2022). *https://store.arduino.cc/collections/mkr-family*.
MQTT. (12 de 3 de 2022). *https://mqtt.org/*.
Nano. (13 de 3 de 2022). *https://store.arduino.cc/collections/nano-family*.

Narrowband_IoT. (12 de 3 de 2022). *https://en.wikipedia.org/wiki/Narrowband_IoT*.
nicla-vision. (Mayo de 2022). Obtenido de https://store.arduino.cc/collections/pro-family/products/nicla-vision
Observation.org, F. O. (Mayo de 2022). *Observation.org*. Obtenido de https://observation.org/
OC2640 Camera Settings. (2022). Obtenido de https://randomnerdtutorials.com/esp32-cam-ov2640-camera-settings/
OpenCV. (13 de 3 de 2022). *https://opencv.org/*.
Opencv.org. (Marzo de 2022). *https://opencv.org/*. Obtenido de https://opencv.org/.
Parque Nacional Sierra de Guadarrama - Aves. (20 de 02 de 2022). Obtenido de https://www.parquenacionalsierraguadarrama.es/es/naturaleza/fauna/119-aves
Parque Regional de la Cuenca Alta del Manzanares. (20 de 02 de 2022). Obtenido de https://www.comunidad.madrid/servicios/urbanismo-medio-ambiente/parque-regional-cuenca-alta-manzanares
Peña del Águila y el Canto Hastial. (20 de 02 de 2022). Obtenido de http://www.colladovillalba.es/es/servicios-municipales/empleo-formacion-y-empresas/turismo/rutas-por-la-montana.html
PHP NET POST method. (2022). Obtenido de https://www.php.net/manual/en/features.file-upload.post-method.php
Prediction Service Client. (2022). Obtenido de https://cloud.google.com/python/docs/reference/aiplatform/latest/google.cloud.aiplatform_v1.services.prediction_service.PredictionServiceClient
Pro. (13 de 3 de 2022). *https://store.arduino.cc/collections/pro-family*.
PyTorch. (13 de 3 de 2022). *https://pytorch.org/*.
Qadur, Q. M., Rashid, T. A., Al-Salihi, N. K., Ismael, B., Kist, A. A., & Zhang, Z. (2018). Low Power Wide Area Networks: A Survey of Enabling Technologies, Applications and Interoperability Needs. *IEEE*.
R. Yoshihashi, R. K. (2017). Bird detection and species classification with time-lapse imagesaround a wind farm: Dataset construction and evaluation. *Wind energy (Chichester, England)*, Vol.20 (12), p.1983-1995.
Randomnerdtutorial Post Image. (2022). Obtenido de https://randomnerdtutorials.com/esp32-cam-post-image-photo-server/
Randomnerdtutorials. (2022). Obtenido de https://randomnerdtutorials.com/
Rapaces diaurnas. (2022). Obtenido de https://www.laruinagrafica.com/blog/identificando-rapaces-diurnas-no-solo-machos-1
Raspberry_Pi. (13 de 3 de 2022). *https://en.wikipedia.org/wiki/Raspberry_Pi*.
Raspberry_Pi_Org. (13 de 3 de 2022). *https://www.raspberrypi.org/*.
Red Natura, C. d. (Mayo de 2022). *Red Natura 2000*. Obtenido de https://www.comunidad.madrid/servicios/urbanismo-medio-ambiente/espacios-protegidos-red-natura-2000
Sharma, N., Madhavi, S., & Inderjit, S. (2018). The History, Present and Future with IoT. En *Internet of Things and Big Data Analytics for Smart Generation* (págs. 27-51).
Smart City - Wikipedia. (20 de 02 de 2022). Obtenido de https://en.wikipedia.org/wiki/Smart_city
SPI_Arduino. (13 de 3 de 2022). *https://www.e-tinkers.com/2020/03/do-you-know-arduino-spi-and-arduino-spi-library/*.
Syed, A., Sierra-Sosa, D., Kumar, A., & Elmaghraby, A. (2021). IoT in Smart Cities: A Survey of Technologies, Practices and Challenges. *MDPI*, 429–475.
TensorFlow. (13 de 3 de 2022). *https://www.tensorflow.org/*.
TensorFlow_Lite. (13 de 3 de 2022). *https://www.tensorflow.org/lite*.
thethingsnetwork.org. (10 de Marzo de 2022). *https://www.thethingsnetwork.org/country/spain/*.
thinger_io. (13 de 3 de 2022). *https://thinger.io/*.
ThingSpeak. (13 de 3 de 2022). *https://thingspeak.com/*.
Thorsten Kramp, R. v. (2013). Introduction to the Internet of Things. En *Enabling Things to Talk.*

Tianhang WU1, X. L. (2017). A new skeleton based flying bird detection method for low-altitude air traffic management. *Chinese Journal of Aeronautics*.

tinyML. (13 de 3 de 2022). *https://www.tinyml.org/*.

TinyML, D. (13 de 3 de 2022). *https://learning.edx.org/course/course-v1:HarvardX+TinyML3+1T2021/home*.

Using Middleware. (2022). Obtenido de https://expressjs.com/en/guide/using-middleware.html

Van Linh Nguyen, P.-C. L.-H. (2019). Energy Depletion Attacks in Low Power Wireless Networks. *IEEE Access*.

Vargas, M. R. (2017). DEEP LEARNING: A REVIEW . *Advances in Intelligent Systems and Computing*.

w3schools php upload. (2022). Obtenido de https://www.w3schools.com/php/php_file_upload.asp

WiFi, D. b. (12 de 3 de 2022). *https://www.rfwireless-world.com/Terminology/difference-between-wifi-6-and-wifi-5.html*.

Zaugg, S., Saporta, G., Loon, E. v., Schmaljohann, H., & Liechti, F. (2008). Automatic identification of bird targets with radar via patterns produced by wing flapping. *Journal of The Royal Society*, 1041-1053.

7. Anexos

Código: Sketch_TFM.ino

// Incluyo las bibliotecas

```
#include <Arduino.h>
#include <WiFi.h>
#include "soc/soc.h" // biblioteca necesaria para el uso de la ESP32 CAM
#include "soc/rtc_cntl_reg.h" // biblioteca necesaria para el uso de la ESP32 CAM
#include "esp_camera.h" // se incluye la biblioteca de la ESP32 CAM
#include <NTPClient.h> // se incluye esta biblioteca para utilizar el horario
```

// Defino variables de la conexión wifi

```
const char* ssid     = "vodafone4330";
const char* password = "LK3FPBDWS*****";
```

// Utilizo esta funcionalidad NTP para poder obtener la hora

```
WiFiUDP udp;
NTPClient ntp(udp, "europe.pool.ntp.org", -3 * 3600, 60000);
String hora;
```

// Defino la dirección, ruta y puerto del servidor

```
String serverName    = "192.168.0.11";   // Dirección del servidor local
String serverPath    = "/upload";     // Ruta por defecto del servidor
const int serverPort = 3000;          // Puerto del servidor
```

// Defino el intervalo entre fotos y configuro el tamaño de la imagen

```
const int timeIntervalInMinutes = 0.5; //una foto cada 30 segundos
const int timerInterval = timeIntervalInMinutes*60000; // tiempo entre cada HTTP POST
unsigned long previousMillis    = 0;  // tiempo desde que la última imagen se envió
framesize_t picSize= FRAMESIZE_SVGA;              // tamaño de la imagen

WiFiClient client; // se crea el cliente para conectar a la dirección IP
```

// Defino los pines de la cámara AI_THINKER o similar vistos anteriormente

```
#define PWDN_GPIO_NUM   32
#define RESET_GPIO_NUM  -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
#define Y2_GPIO_NUM      5
#define VSYNC_GPIO_NUM  25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22
```

// Función enviar Foto

```
String sendPhoto() {
```

```
  String getAll;
  String getBody;
  camera_fb_t * fb = NULL;
  fb = esp_camera_fb_get(); //función para obtener la foto de la cámara devuelve en fb
  if(!fb) {
    Serial.println("Camera capture failed");
    delay(1000);
    ESP.restart(); // en caso de error la reinicia
  }

Serial.println("Connecting to server: " + serverName);
```

// Conectarse al sevidor

```
  if (client.connect(serverName.c_str(), serverPort)) {
    Serial.println("Connection successful!");
```

//Construimos una cabecera y una cola a la imagen

```
    String head = "--boundary\r\nContent-Disposition: form-data; name=\"imageFile\"; filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";
    String tail = "\r\n--Final--\r\n";

    uint32_t imageLen = fb->len;
    uint32_t extraLen = head.length() + tail.length();
    uint32_t totalLen = imageLen + extraLen;
```

//Escribo el método POST en el servidor con print

```
    client.println("POST " + serverPath + " HTTP/1.1");
    client.println("Host: " + serverName);
    client.println("Content-Length: " + String(totalLen));
    client.println("Content-Type: multipart/form-data; boundary=boundary");
    client.println();
    client.print(head);
```

// Escritura servidor a través del buffer con write y finalizo con la cola

```
    uint8_t *fbBuf = fb->buf;
    size_t fbLen = fb->len;
    for (size_t n=0; n<fbLen; n=n+1024) {
      if (n+1024 < fbLen) {
        client.write(fbBuf, 1024); //vamos enviando la imagen en paquetes de 1024
        fbBuf += 1024;
      }
      else if (fbLen%1024>0) {
        size_t remainder = fbLen%1024;
        client.write(fbBuf, remainder);
      }
    }
    client.print(tail);
    esp_camera_fb_return(fb); //libero memoria
```

// Inicializo timer para lectura del servidor

```
    int timoutTimer = 10000;
    long startTimer = millis();
```

```
    boolean state = false;

// Mientras no supere el timer leo del servidor client.read ()
// hasta recibir el final \n

      while ((startTimer + timoutTimer) > millis()) {
      Serial.print(".");
      delay(100);
      while (client.available()) {
        char c = client.read();
        if (c == '\n') {
          if (getAll.length()==0) { state=true; }
          getAll = "";
        }
        else if (c != '\r') { getAll += String(c); }
        if (state==true) { getBody += String(c); }
        startTimer = millis();
      }
      if (getBody.length()>0) { break; }
    }
    Serial.println();
    client.stop();
    Serial.println(getBody);
  }
  else {
    getBody = "Connection to " + serverName +  " failed.";
    Serial.println(getBody);
  }
  return getBody;
}

// Función setup ()

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Serial.begin(115200);
  pinMode(4, OUTPUT);
  digitalWrite(4, HIGH);

// Inicializo la conexión WiFi, NTP y Camara

WiFi.mode(WIFI_STA);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  //digitalWrite(4, LOW);
  Serial.println();
  Serial.print("ESP32-CAM IP Address: ");
  Serial.println(WiFi.localIP());
  ntp.begin();
  ntp.forceUpdate();
  camera_config_t config;
```

```
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  // init with high specs to pre-allocate larger buffers
  if(psramFound()){
    config.frame_size = picSize;  // RESOLUTION
    config.jpeg_quality = 10;  //0-63 lower number means higher quality
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_CIF;
    config.jpeg_quality = 12;  //0-63 lower number means higher quality
    config.fb_count = 1;
  }
  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    delay(1000);
    ESP.restart();
  }
  sendPhoto();
}
```

// Función setup ()

```
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= timerInterval) {
    sendPhoto();
    previousMillis = currentMillis;
    hora = ntp.getFormattedTime();
    ntp.getDay();
  }
}
```

Código: Upload.php

```
<?php
$target_dir = "uploads/";
$datum = mktime(date('H')+0, date('i'), date('s'), date('m'), date('d'), date('y'));
$target_file = $target_dir . date('Y.m.d_H:i:s_', $datum) . basename($_FILES["imageFile"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

// Comprobación de que se trata de una imagen
if(isset($_POST["submit"])) {
  $check = getimagesize($_FILES["imageFile"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  }
  else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}

// Compruebo si ya existía el fichero
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = 0;
}

// Compruebo el tamaño del fichero
if ($_FILES["imageFile"]["size"] > 500000) {
  echo "Sorry, your file is too large.";
  $uploadOk = 0;
}

// Compruebo que pertenece a alguno de los formatos de imagen
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
  echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
  $uploadOk = 0;
}

// Compruebo la variable $uploadOk está a 0 por algún error
if ($uploadOk == 0) {
  echo "Sorry, your file was not uploaded.";
// Si todo está bien trato de subir el fichero
}
else {
  if (move_uploaded_file($_FILES["imageFile"]["tmp_name"], $target_file)) {
    echo "The file ". basename( $_FILES["imageFile"]["name"]). " has been uploaded.";
  }
  else {
    echo "Sorry, there was an error uploading your file.";
  }
}
?>
```

Código: upload.js

```
// Estos son los componentes necesarios instalar previamente a través de NPM
const express = require('express');
const fileUpload = require('express-fileupload');

const app = express();
const port = process.env.PORT || 3000;
function addZero(number){
  if (number <= 9)
     return "0" + number;
  else
     return number;
}

// Función de middleware para utilizar ficheros temporales
app.use(fileUpload ({
    useTempFiles : true,
    tempFileDir : '/tmp/',
    debug: false
}));

// Función Post que recibe el fichero
app.post('/upload', function(req, res) {
  let sampleFile;
  let uploadPath;
  if (!req.files || Object.keys(req.files).length === 0) {
    return res.status(400).send('No files were uploaded.');
  }

// Preparo el formato del nombre del fichero
  let date = new Date();
  let formatatedDate = (addZero(date.getDate() )) + "-" + (addZero(date.getMonth() + 1)) + "-
" + date.getFullYear() + "_" + addZero(date.getHours()) + addZero(date.getMinutes()) +
addZero(date.getSeconds());
  console.log(formatatedDate);

  // El nombre del campo de entrada ("sampleFile") es utilizado para recuperar el fichero
subido
  sampleFile = req.files.sampleFile;
  uploadPath = __dirname + '\uploads\' + formatatedDate+ sampleFile.name;

 // Utilizamos este método mv() para ubicar el fichero dentro del servidor
  sampleFile.mv(uploadPath, function(err) {
    if (err)
      return res.status(500).send(err);
    res.send('File uploaded!');
  });
});
 //Escucha
app.listen(port);
```

Código: indexreco.js

```
'use strict';

// se definen las variables requeridas para llamar al API

function main(
  projectId = 'averapaz-87893',
  location = 'us-central1',
  modelId = 'ICN5993585217565097984',
  filePath = './recursos/upload.jpg'
) {

// Esta variable almacena el resultado de la clasificación
  var clasificado = "no sabemos";

// Estas constantes indican la carpeta donde destino en función de la predicción
  const destrapaz = './recursos/rapaz/upload.jpg';
  const destnorapaz = './recursos/no_rapaz/upload.jpg';

  //Se importan las librerías requeridas de AutoML y fs-extra
  const {PredictionServiceClient} = require('@google-cloud/automl').v1;
  const fs = require('fs-extra');

  // Se instancia un cliente
  const client = new PredictionServiceClient();

  // Se lee el contenido del fichero a predecir
  const content = fs.readFileSync(filePath);

  // Esta function contiene el request hacia el API
async function predict() {
    // Construct request
    // params is additional domain-specific parameters.
    // score_threshold is used to filter the result
    const request = {
      name: client.modelPath(projectId, location, modelId),
      payload: {
        image: {
          imageBytes: content,
        },
      },
    };
// Se espera a la respuesta que se almacena en dicha variable
    const [response] = await client.predict(request);

// Leemos la respuesta: el nombre y el score, lo mostramos por pantalla
    for (const annotationPayload of response.payload) {
      console.log(`Predicted class name: ${annotationPayload.displayName}`);
      console.log(`Predicted class score: ${annotationPayload.classification.score}`);
      clasificado = annotationPayload.displayName;
      console.log(clasificado)

     // origen donde está el fichero
      const src = filePath;

      // en función de la clasificación lo moveré a un destino u otro
```

```
      if (clasificado==='rapaz')
      {
      // Movimiento a carpeta rapaz con función call back
      fs.move(src, destrapaz, (err) => {
        if (err) return console.log(err);
        console.log(`File successfully moved!!`)})
      }
      else
      {
      // Movimiento a carpeta no rapaz con función call back
      fs.move(src, destnorapaz, (err) => {
        if (err) return console.log(err);
        console.log(`File successfully moved!!`)})
      }
    }
  }

  // Llamada a la función asíncrona predict
  predict();
}

// llamada al proceso
process.on('unhandledRejection', err => {
  console.error(err.message);
  process.exitCode = 1;
});
main(...process.argv.slice(2));
```