Proyecto Fin de Carrera
Ingeniería Electrónica Robótica y Mecatrónica

Implementación de técnicas de aprendizaje automático en sistemas de cámara trampa

Autor: Jesús Gamero Borrego
Tutor: Begoña Chiquinquira Arrue Ulles

Dpto. de Automatización y Control
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Carrera
Ingeniería Electrónica Robótica y Mecatrónica

Implementación de técnicas de aprendizaje automático en sistemas de cámara trampa

Autor:
Jesús Gamero Borrego

Tutor:
Begoña Chiquinquira Arrue Ulles
Profesor titular

Dpto. de Automatización y Control
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Carrera: Implementación de técnicas de aprendizaje automático en sistemas de cámara trampa

Autor: Jesús Gamero Borrego

Tutor: Begoña Chiquinquira Arrue Ulles

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

"There will be times in life where you need to stand up for yourself. Times where the right thing is actually to back down and apologize. Times when the right thing is to fight. Times when the right thing is to turn and run. Times to hold on with all you have and times to let go with grace."

- Taylor Swift -

Agradecimientos

Quiero comenzar expresando mi más profundo agradecimiento a todas las personas que han sido parte fundamental de este viaje académico y que han dejado una huella imborrable en mí.

En primer lugar, gracias, Claudia, Tomas, Andrea y Javi, por estar ahí tanto en los buenos como en los malos momentos, por vuestra compañía en este tortuoso camino que ha sido esta etapa en nuestras vidas. Por los grandes momentos que hemos vivido estos años, todo lo que hemos crecido juntos tanto en lo personal, como en lo profesional. Siempre os llevaré conmigo.

Agradecer a mi familia todo el apoyo que me han proporcionado durante toda mi etapa académica, apostándolo todo por mí y ofreciéndome su amor incondicional. Gracias también a Agro, mi perro, por su compañía durante días eternos de estudio.

A Javi, quien se ha convertido en una parte fundamental de mi familia. Por su amor, paciencia y todas las tardes que hemos compartido al borde de la demencia mientras navegábamos por nuestras respectivas carreras, tu presencia ha sido un bálsamo en los momentos de estrés y una fuente constante de inspiración.

Jesús Gamero Borrego

Sevilla, 2023

Resumen

La humanidad está experimentando una revolución sin precedentes gracias a tecnologías emergentes como la *robótica*, el *big data* y la *inteligencia artificial*. Estas innovaciones están transformando sectores que antes se consideraban inamovibles, y es fundamental estudiar y comprender estas tecnologías que ya forman parte integral de nuestra sociedad. Su potencial para el desarrollo de nuevas aplicaciones es inmenso, y su correcto uso puede generar mejoras significativas en la calidad de vida de las personas.

En este proyecto, nos enfocaremos en explorar una forma específica de poner la *inteligencia artificial* al servicio de la sociedad. La *inteligencia artificial* ofrece una serie de herramientas y capacidades que tienen el potencial de abordar problemas complejos que necesitan de la intervención humana por lo que pueden brindar soluciones innovadoras en diversos campos.

Nuestro objetivo es investigar y desarrollar una aplicación práctica de inteligencia artificial mediante la implementación de un modelo de detección de objetos en una Raspberry Pi 3B. Esta combinación nos permite aprovechar los avances de la inteligencia artificial en un dispositivo accesible y de bajo costo.

En particular, nos centraremos en un desafío concreto: mejorar los sistemas de fototrampeo utilizados para el seguimiento y control de las poblaciones de lince ibérico. El lince ibérico es una especie en peligro de extinción, y su supervivencia depende de un monitoreo preciso y efectivo de su población. El uso tradicional de los sistemas de fototrampeo presenta limitaciones ya que estos recogen una cantidad muy abundante de datos que debe ser procesada por una persona para clasificar las diferentes capturas y poder obtener conclusiones. Al implementar un modelo de inteligencia artificial en la Raspberry Pi 3B, se analizarán automáticamente las imágenes capturadas, clasificando las diferentes detecciones para poder obtener datos relevantes sobre su comportamiento y distribución incluso en tiempo real.

Para cumplir este objetivo se hará uso del marco de trabajo Tensorflow Lite, así como del modelo de detección SSD MobileNetV2 FPN. Para poder implementar la aplicación se construirá un conjunto de datos que permita entrenar un modelo capaz de detectar la presencia de diferentes especies, entre ellas el lince. Se buscará un buen rendimiento del modelo en las condiciones más límites que presentan los medios naturales, entre ellas la escasez de luz.

En resumen, este proyecto se enfoca en estudiar una forma concreta de poner la inteligencia artificial al servicio de la sociedad. Mediante su implementación en una Raspberry Pi 3B, buscamos mejorar los sistemas de fototrampeo utilizados para el seguimiento y control de las poblaciones de lince ibérico. Este enfoque no solo tiene el potencial de preservar esta especie en peligro crítico de extinción, sino también de sentar las bases para aplicaciones futuras de inteligencia artificial en beneficio de la sociedad en su conjunto.

Abstract

Humanity is experiencing an unprecedented revolution thanks to emerging technologies such as robotics, big data, and artificial intelligence. These innovations are transforming industries that were once considered immovable, and it is crucial to study and understand these technologies that have already become an integral part of our society. Their potential for developing new applications is immense, and their proper use can generate significant improvements in people's quality of life.

In this project, we will focus on exploring a specific way to harness the power of artificial intelligence for the benefit of society. Artificial intelligence offers a range of tools and capabilities that have the potential to address complex problems that typically require human intervention, thus providing innovative solutions in various fields.

Our objective is to investigate and develop a practical application of artificial intelligence by implementing an object detection model on a Raspberry Pi 3B. This combination allows us to leverage the advancements in artificial intelligence on an accessible and low-cost device.

We will concentrate on a specific challenge: improving the phototrapping systems used for monitoring and controlling Iberian lynx populations. The Iberian lynx is an endangered species, and its survival depends on accurate and effective population monitoring. Traditional phototrapping systems have limitations as they collect a large amount of data that needs to be manually processed for classification and drawing conclusions. By implementing an artificial intelligence model on the Raspberry Pi 3B, captured images will be automatically analyzed, enabling the classification of different detections to obtain relevant data on lynx behavior and distribution, even in real-time.

To achieve this objective, we will utilize the Tensorflow Lite framework and the SSD MobileNetV2 FPN detection model. To implement the application, a dataset will be constructed to train a model capable of detecting the presence of different species, including the lynx. The model's performance will be optimized for the challenging conditions encountered in natural environments, such as low light availability.

In summary, this project focuses on studying a specific approach to harness artificial intelligence for the benefit of society. By implementing it on a Raspberry Pi 3B, we aim to improve the phototrapping systems used for monitoring and controlling Iberian lynx populations. This approach not only has the potential to preserve this critically endangered species but also lays the foundation for future applications of artificial intelligence for the benefit of society.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

s	Sesgo
w_i	Peso iésimo de una neurona
$\vec{w}$	Vector de pesos
x_i	Entrada iésima de una neurona
y_i	Salida de una neurona jésima
η	Ratio de aprendizaje
∇	Gradiente
∂y ∂x	Derivada parcial de y respecto
$<$	Menor o igual
$>$	Mayor o igual

1 INTRODUCCIÓN

"The way to succeed is to double your failure rate."

- Thomas J. Watson -

Este capítulo sirve como antesala para presentar el proyecto de Implementación de un sistema de detección de linces ibéricos mediante inteligencia artificial en una Raspberry Pi 3B, para su control y seguimiento. Se justificará su elección presentando el problema que el proyecto busca abarcar, se describirán los objetivos y la finalidad de este.

1.1 Preámbulo

Desde los orígenes de la civilización se ha intentado entender la inteligencia humana, podemos remontarnos a la antigua Grecia, Aristóteles formalizó la estructura de razonamiento que son los silogismos [1]. Desarrolló una teoría de la lógica que se basa en la idea de que el conocimiento se puede obtener a través de la razón y la observación del medio natural. Tras la semilla que Aristóteles plantó no se produjeron avances significativos hasta mediados del siglo XIX cuando George Boole [2] estableció la conocida lógica booleana o lógica de orden cero mucho más compleja que los silogismos de Aristóteles. Posteriormente esta seria extendida por Gottlob Frege [3] dando lugar a la llamada Lógica de primer orden.

Pero no sería hasta el comienzo de la revolución informática a mediados del siglo XX cuando los investigadores, entre ellos Alan Turing [4], comenzaron a preguntarse si las maquinas pueden llegar a pensar. Como contexto en 1941 Konrad Zuse creó la primera computadora programable y tres años después Warren McCulloch y Walter Pitts publicaron el articulo *«A Logical Calculus of Ideas Immanent in Nervous Activity »* [5], el cual presentaba el primer modelo matemático para la creación de una red neuronal, siendo considerado el primer trabajo del campo de la inteligencia artificial aunque para entonces el termino aun no estaría acuñado.

El termino no sería normalizado hasta 1956 en la conferencia «Dartmouth Summer Research Project on Artificial Intelligence» [6] de John McCarthy, donde se presentaron todos los avances de la época. Es considerado el nacimiento de la inteligencia artificial.

Tras todos los avances que surgieron a mediados de siglo, en la década de los setenta las dudas surgieron sobre el campo de las IA. Los primeros proyectos habían generado muchas expectativas, pero muchos de ellos no alcanzaron los resultados esperados, lo cual generó una pérdida de interés por parte de los investigadores, así como un abandono por parte de las fuentes de financiación [7]. Tuvimos que esperar hasta comienzos de la década de los noventa, para ver un aumento significativo en el campo, gracias a los avances en las tecnologías de computación, así como el nacimiento de internet.

Hoy en día el sueño de crear maquinas inteligentes está más cerca que nunca, gracias a la inversión tanto del

sector privado como público, las IA han podido dar solución a numerosos problemas complejos, así como automatizar ciertas tareas que hasta el momento necesitaban de la capacidad humana.

Con este proyecto se busca aportar nuevas soluciones a problemas actuales haciendo uso de las ventajas que proporcionan las inteligencias artificiales. Nos centraremos en ayudar a preservar y proteger a las especies en peligro de extinción, en particular nos centraremos en lince ibérico, ya que es una de las especies que mayor riesgo de extinción tiene en nuestros ecosistemas.

1.2 Contexto y justificación del problema

El lince ibérico (Lynx Pardinus) es una especie única de la península ibérica, la cual, debido a la acción humana en los ecosistemas, como por diferentes enfermedades se ha llegado encontrar en peligro crítico de extinción, la especie en estado silvestre estuvo a punto de desaparecer a principios de siglo.

Gracias a los avances tecnológicos, se desarrollaron técnicas más sensibles para el seguimiento de las poblaciones; el fototrampeo y el análisis genético de excrementos. Estas son técnicas que proporcionan estimaciones basadas en indicios directos, permitiendo detectar cualquier población de lince ibérico, por baja que sea su densidad. Hasta entonces las técnicas usadas se basaban en búsqueda de huellas y excrementos, las cuales son poco precisas y cuentan con una gran dificultad a la hora de asignar a una especie sin un análisis genético previo. Estas eran estimas basadas en indicios indirectos por lo cual no proporcionaban buenos resultados a la hora de estimar poblaciones pequeñas de linces.

En el año 2004, Guzmán y col. publican los resultados de un trabajo de prospección basado en la combinación de estas dos técnicas, gracias al cual se evidenció el estado de pre-extinción del lince, se estimó que solo quedaban poblaciones estables de linces en Sierra Morena oriental y en Doñana, unos 160 ejemplares distribuidos en 500 km2. Gracias a estas investigaciones se evidenció el estado de peligro de la especie y se pusieron en marcha acciones de conservación para evitar la extinción definitiva. [8]

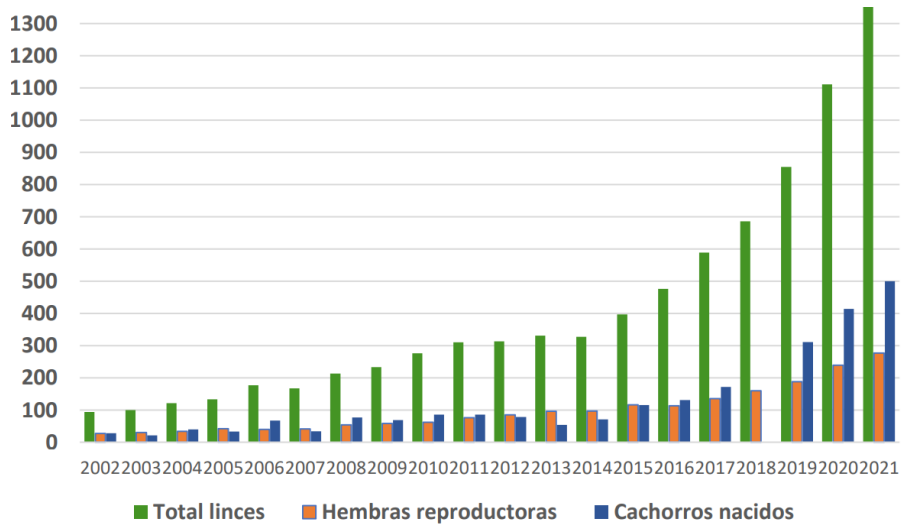


Figura 1-1. Evolución de la población de linces a lo largo de las últimas décadas.

Gracias a las acciones de conservación desde comienzos de siglo hasta la actualidad, se han conseguido grandes avances. Aumentando los ejemplares silvestres consiguiendo sacar a la especie del peligro crítico de extinción, actualmente se encuentra en peligro de extinción.

Para conseguir estos avances la herramienta principal usada por los investigadores ha sido la fototrampeo, para seguir la evolución poblacional de la especie y realizar censos anuales, así como para conocer los comportamientos del lince.

1.3 Objetivos y alcance del proyecto

Con este proyecto se busca mejorar las técnicas de fototrampeo para hacerlas más automáticas y eficientes mediante el uso de inteligencia artificial. Se busca dotar al sistema de la capacidad de detectar un lince en una imagen, así como otras especies relacionadas con el cómo pueden ser el zorro y el conejo.

Se implementará en una Raspberry Pi 3B, ya que este es un sistema compacto, barato y potente que permite un rendimiento aceptable, así como la portabilidad necesaria para colocar el sistema en enclaves estratégicos.

En resumen, se definen los siguientes objetivos:

- Instalación de TFlite, así como comprobar el correcto funcionamiento del modelo en la Raspberry pi3.
- Entrenar el modelo para conseguir detectar y diferenciar especies distintas en diferentes condiciones de luminosidad.
- Automatizar el proceso de censado de especies.
- Esbozar posibles líneas futuras de investigación.

1.4 Estructura

Tras esta introducción se estudiarán diferentes capítulos, para así entender el funcionamiento del sistema, así como su implementación.

- **Estado del arte**: Se revisarán proyectos ya realizados en el campo del el seguimiento y control poblacional de especies en peligro.
- **Fundamentos teóricos**: Donde se buscará entender los conceptos básicos de la inteligencia artificial, llegando a las redes neuronales convolucionales. Además de explorar los conceptos de visión por computador.
- **Implementación del sistema**: Se profundizará en el sistema a desarrollar aclarando sus componentes hardware y software.
- **Pruebas y validación**: Se mostrarán los resultados con objeto de mostrar las fortalezas debilidades del sistema.
- **Conclusiones**: Se reunirán todos los resultados obtenidos en los capítulos anteriores para si poder enunciar una resolución sobre la viabilidad del sistema, así como las posibles líneas de trabajo futuras.

Tras esta introducción al proyecto daremos paso a la parte bibliográfica del proyecto en la cual analizaremos las soluciones existentes hasta ahora al problema del rastreo de especies.

2 ESTADO DEL ARTE

En este capítulo se llevará a cabo una revisión bibliográfica para conocer principales enfoques, métodos y resultados de investigaciones previas relevantes. Además, se identificarán lagunas en el conocimiento actual para así entender la necesidad y justificación del trabajo de investigación que se desarrolla. En este capítulo se abordarán las investigaciones más relevantes en el área de estudio y se explicará cómo el proyecto aporta novedades y avances significativos en el campo.

2.1 Métodos de rastreo de especies mediante técnicas convencionales

El estudio del movimiento y comportamiento de los organismos en la naturaleza ha sido una preocupación constante del ser humano a lo largo de la historia. Desde tiempos prehistóricos, las razones para comprender estos fenómenos han variado en función del contexto histórico. En la prehistoria, una de las principales razones para estudiar el movimiento y comportamiento de la fauna era la necesidad de controlar a los depredadores, ya que representaban una amenaza para la supervivencia. Además, el conocimiento sobre el movimiento y comportamiento de los animales también resultaba esencial para la caza, permitiendo a los cazadores obtener información valiosa sobre las rutas migratorias y hábitos alimenticios de sus presas. Con el paso del tiempo, el interés por comprender el movimiento y comportamiento de la fauna se ha ampliado y diversificado, incluyendo razones como la conservación de la biodiversidad, la comprensión de los efectos del cambio climático y la evaluación del impacto humanos en los ecosistemas naturales.

A continuación, revisaremos algunos de los métodos más extendidos en el ámbito.

2.1.1 Radio seguimiento

El radio seguimiento o radio tracking, es una técnica basada en la localización de animales utilizando ondas de radio. Para ello el animal debe portar un dispositivo transmisor de frecuencias cercanas a los 150 MHz y que estas sean interceptadas por un receptor.

El dispositivo receptor suele tomar la forma de un collar debe ser poco pesado, no debe superar el 5% del peso del animal, para evitar interferir con su movilidad y comportamiento natural. Además, el receptor debe estar ubicado a una distancia adecuada de los emisores, para garantizar una buena recepción de las señales de radiofrecuencia emitidas por los collares.

Es importante destacar que los dispositivos emisores de radiofrecuencia deben ser configurados en frecuencias específicas, generalmente alrededor de los 150 MHz, como ya se ha comentado, para asegurar una señal clara y distinguir entre distintos individuos de la misma especie. Es decir, cada collar debe emitir una frecuencia única y reconocible para poder ser identificado por el receptor de radio.

Figura 2-1. Lince con collar emisor.

El radio seguimiento es una técnica ampliamente utilizada en la investigación y monitoreo de la vida silvestre. Esta técnica es particularmente útil en el seguimiento de especies que tienen rangos de hogar limitados y no realizan migraciones o grandes desplazamientos, ya que esto puede resultar en la pérdida de la señal por el receptor. Además, el radio seguimiento es una técnica especialmente valiosa para el seguimiento de ejemplares reintroducidos o de especies en peligro de extinción, ya que permite a los investigadores monitorear su ubicación y comportamiento para evaluar la efectividad de los programas de reintroducción y conservación. [9]

Esta técnica permite conocer el área de campeo de una especie, detectar la muerte de un animal y obtener información sobre su comportamiento en tiempo real. Aunque presenta limitaciones, es una herramienta valiosa para la conservación y gestión de especies en peligro de extinción.

2.1.2 Análisis genético

Los métodos basados en el análisis genético buscan mediante el muestreo de material genético y el procesado de este, extraer información valiosa para el entendimiento del comportamiento de una especie. Estos métodos son especialmente útiles para el seguimiento de poblaciones y la identificación de individuos en especies poco conocidas o difíciles de estudiar.

La extracción de información a partir del ADN presenta numerosas ventajas en comparación con otras técnicas, ya que puede proporcionar una gran variedad de información que no se puede obtener con otros métodos. Por ejemplo, permite el diagnóstico de enfermedades genéticas y estudios de filiación o parentesco, lo que resulta especialmente útil en especies en peligro extremo con poblaciones muy reducidas, permitiendo identificar la endogamia dentro de una población. Además, el análisis de ADN también se puede utilizar para identificar diferentes especies y para estudios de biogeografía.

Una de las técnicas es el ADN ambiental [10] (eDNA, Environmental DNA), consiste en la obtención de material genético de organismos que habitan en el medio ambiente, como suelo, aire o agua, para el estudio de las especies o poblaciones presentes en dicho medio. Es una técnica muy útil para identificar la presencia de especies en áreas donde su observación directa puede ser difícil, costosa o invasiva para el habitad. Esta es una de las principales ventajas de los métodos que buscan hacer uso del material genético, no necesariamente necesitan contacto con la especie lo que la hace una técnica adecuada para ciertas situaciones.

2.1.3 Métodos acústicos

Los métodos acústicos se basan en el registro y análisis de las señales acústicas emitidas por los animales para obtener información relevante sobre su comportamiento, distribución y diversidad. Estas técnicas pueden incluir el uso de grabaciones de sonidos emitidos por las especies, así como el análisis en tiempo real de las señales sonoras a través de sistemas de detección acústica automatizados. También pueden utilizarse métodos de telemetría acústica para el seguimiento y localización de animales a través de sus emisiones sonoras.

Las técnicas de rastreo acústico son muy versátiles y permiten el seguimiento de especies en diferentes entornos, incluyendo tierra y océanos, sin necesidad de invadir su espacio vital. Esto las convierte en una herramienta valiosa para el estudio de comportamientos, patrones de movimiento y distribución espacial de las especies.

Dentro de las técnicas más extendidas para el rastreo de especies mediante el uso de ondas sonoras, encontramos el uso de hidrófonos. Estos son micrófonos sumergibles que se utilizan para detectar los sonidos emitidos por los animales acuáticos, como ballenas, delfines y tiburones, entre otros. Permiten estudiar la distribución, comportamiento y comunicación de estas especies en su hábitat natural. Otra técnica muy utilizada es la telemetría acústica, que consiste en la colocación de receptores en el fondo marino que reciben una señal acústica emitida por una marca electrónica situada en el cuerpo de los individuos bajo estudio. Esta técnica permite el seguimiento de la ubicación, la profundidad y la temperatura de los animales marinos [11]. Cabe destacar los detectores de ultrasonidos frecuentemente utilizados para analizar los sonidos emitidos por murciélagos, así como muchos otros animales marinos (más información [12]).

2.1.4 Métodos ópticos

Son técnicas que se basan en el uso de la luz como variable de estudio por lo que se trabajará con cámaras,

tanto las que usen el espectro visible como que no, así como sensores ópticos como pueden ser los infrarrojos para poder capturar la información necesaria para monitorear la presencia y el movimiento de los animales en su entorno natural. Estas técnicas son muy versátiles y se aplican en diferentes contextos, desde estudios de comportamiento hasta conservación y manejo de especies. Además, la incorporación de nuevas tecnologías, como los vehículos aéreos no tripulados (UAVs) y la inteligencia artificial, está ampliando el horizonte para obtener resultados de manera más automática y eficiente.

Existen diferentes técnicas entre las que se encuentran la teledetección mediante la cual haciendo uso de sensores situados en plataformas espaciales, que hacen uso de su interacción electromagnética con el terreno para obtener información sobre la Tierra, lo que supone una forma efectiva de obtener información sobre las características del terreno y las condiciones ambientales. Permitiendo detectar patrones en el hábitat y distribución de las especies [13]. También encontramos técnicas que usan tecnologías que se centran el espectro infrarrojo lejano medio, captando el calor que emiten los seres vivos. Proporcionando información muy útil para monitorear la actividad y presencia de animales en ambientes nocturnos o de baja visibilidad. Por último, centrándonos en el espectro visible de la luz encontramos diferentes técnicas entre las que se encuentran, el rastreo o seguimiento vía satélite o el uso de UAVs para tomar imágenes de áreas de difícil accesibilidad. También contamos con el llamado **fototrampeo** o cámaras trampa, esta técnica consiste en situar cámaras en sitios estratégicos para capturar imágenes de animales en su hábitat natural.

Profundizando más en la tecnología que usan estas cámaras trampa, estas forman un sistema compacto cuentan con una batería que les permite tener una gran autonomía, ya que el sistema permanece en modo de bajo consumo hasta que detecta un movimiento ya que cuentan con sensores presencia que les permiten activarse. Además, algunos modelos incluyen sensores de temperatura, humedad y presión atmosférica para obtener información adicional del entorno en el que se encuentran los animales.

Figura 2-2. Cámaras trampa.

Estas cámaras pueden ser programadas para tomar imágenes o videos en momentos específicos del día o cuando detectan movimiento, lo que permite obtener información detallada sobre los patrones de actividad de las especies. También se han desarrollado cámaras trampa con tecnología infrarroja para capturar imágenes nocturnas sin perturbar el comportamiento natural de los animales. [14]

Por lo general, las cámaras trampa no cuentan con métodos de comunicación incorporados, lo que significa que los datos se deben recuperar directamente de la tarjeta de memoria de la cámara. Sin embargo, algunas cuentan con comunicación por internet haciendo uso de diferentes tecnologías como el 4G, que permiten obtener los resultados de un avistamiento en el momento que se produce. En la referencia [15], podemos ver más información sobre el catálogo disponible.

El fototrampeo se ha convertido en una de las técnicas más usadas por los expertos ya que cuenta con muchas ventajas, entre ellas la capacidad de obtener información de especies elusivas en su hábitat sin la necesidad de la invasión humana.

Como podemos ver, el campo del rastreo de especies es una disciplina esencial para la gestión y conservación de la fauna. Estos métodos permiten alcanzar objetivos importantes como la mejora de las poblaciones de especies en peligro, la evaluación del impacto humano y la delimitación de zonas de acceso restringido para la protección de especies y sus hábitats.

Actualmente gracias a los numerosos avances tecnológicos se están poniendo en marcha técnicas novedosas para mejorar las acciones de rastreo. Una de las novedades más destacadas es el uso de inteligencia artificial. La IA permite procesar grandes cantidades de datos de manera rápida y precisa, lo que puede resultar en una mayor eficiencia y precisión a la hora de procesar la información. A continuación, revisaremos algunas de sus aplicaciones más útiles.

2.2. Métodos de rastreo de especies mediante Inteligencia Artificial

La una de las grandes revoluciones de nuestro tiempo es la revolución de la inteligencia artificial, gracias a su versatilidad está siendo aplicada en gran variedad de campos, nosotros no centraremos en el uso que se hace de ella para controlar y estudiar la fauna.

La inteligencia artificial en este campo busca automatizar tareas que anteriormente necesitaban de la capacidad humana para ser realizadas. Por ejemplo, clasificar 2000 imágenes de fototrampeo o identificar diferentes sonidos obtenidos por un hidrófono.

A continuación, revisaremos los distintos métodos de rastreo en los cuales la Inteligencia Artificial (IA) se utiliza con mayor frecuencia, así como sus diversas aplicaciones en cada uno de ellos.

2.2.1. Métodos acústicos

Como ya mencionamos anteriormente, los métodos acústicos para rastrear especies son muy versátiles y permiten una amplia gama de aplicaciones. Entre ellas se puede estudiar el comportamiento de las especies, analizando los diferentes patrones en el sonido que emiten. Por ejemplo, el análisis de los patrones de sonido puede proporcionar información valiosa sobre los patrones de alimentación, migración y apareamiento de las especies. Otra aplicación importante de estos métodos es su capacidad para evaluar la biodiversidad de un área determinada. Al identificar las diferentes especies presentes en un área a partir de las señales acústicas que emiten, los investigadores pueden obtener una comprensión más completa de la diversidad de especies en un ecosistema determinado.

Para la detección y clasificación de diferentes especies, se ha desarrollado la plataforma **Multi Species Bioacoustic Classification de Microsoft** [16]. Esta plataforma forma parte de la iniciativa *IA for Earth* de Microsoft, cuyo objetivo es utilizar la IA para abordar algunos de los problemas más graves de la tierra, incluyendo la conservación de la biodiversidad.

El modelo de aprendizaje automático que implementa utiliza un proceso en dos etapas para procesar las señales acústicas. En la primera etapa, se extraen características acústicas de la señal mediante técnicas de procesamiento de señales digitales, como la transformada de Fourier. En particular, se utiliza el Espectrograma de Mel para codificar la información (más información [17]) debido a su utilidad con señales no periódicas, como el sonido capturado en la naturaleza.

En la segunda etapa, se utiliza una red neuronal convolucional (*Convolucional Neural Network CNN*, se aclara el funcionamiento de estas en **apartado 3.3.1.**), para detectar patrones en los espectros generados en la etapa anterior y así detectar y clasificar los diferentes emisores. Aunque las CNN se utilizan típicamente en modelos especializados en imágenes, pueden aplicarse eficazmente al procesamiento de señales acústicas debido a la naturaleza similar de las representaciones de datos de imágenes y espectros de sonido.

También podemos señalar el modelo, **Automatic Bioacoustic Source Separation with Deep Neural Networks,** desarrollado por **Earthspecies** [18], este modelo tiene el objetivo de separar señales de audio mixtas, que provienen de varias fuentes, en el contexto del medio natural. Este problema se presenta cuando se registran múltiples especies animales emitiendo sonidos al mismo tiempo, lo que hace difícil separar y analizar cada una de las señales por separado.

En la **figura 2-3**, podemos ver los resultados que se obtienen. Podemos ver la mezcla de ambas señales tanto

en el dominio de la frecuencia como en el tiempo, podemos observar que la separación se consigue satisfactoriamente.

Por último, veremos el proyecto **Belugasounds** [19], este también forma parte de *IA for Earth* de Microsoft. Este busca automatizar el proceso de separación y análisis de los sonidos emitidos por las belugas. Debido a que los mares cuentan con una gran cantidad de contaminación acústica y las belugas son una especie escasa en los océanos, revisar grabaciones puede ser una tarea tediosa. Permitiendo así utilizar para analizar grabaciones de sonidos de belugas en tiempo real, clasificando los diferentes tipos de sonidos emitidos y generando reportes automáticos de análisis de sonido. Esta tecnología es útil para el monitoreo y conservación de las poblaciones de belugas, ya que permite detectar y analizar las señales acústicas emitidas por estas especies en su hábitat natural de manera más eficiente y precisa.

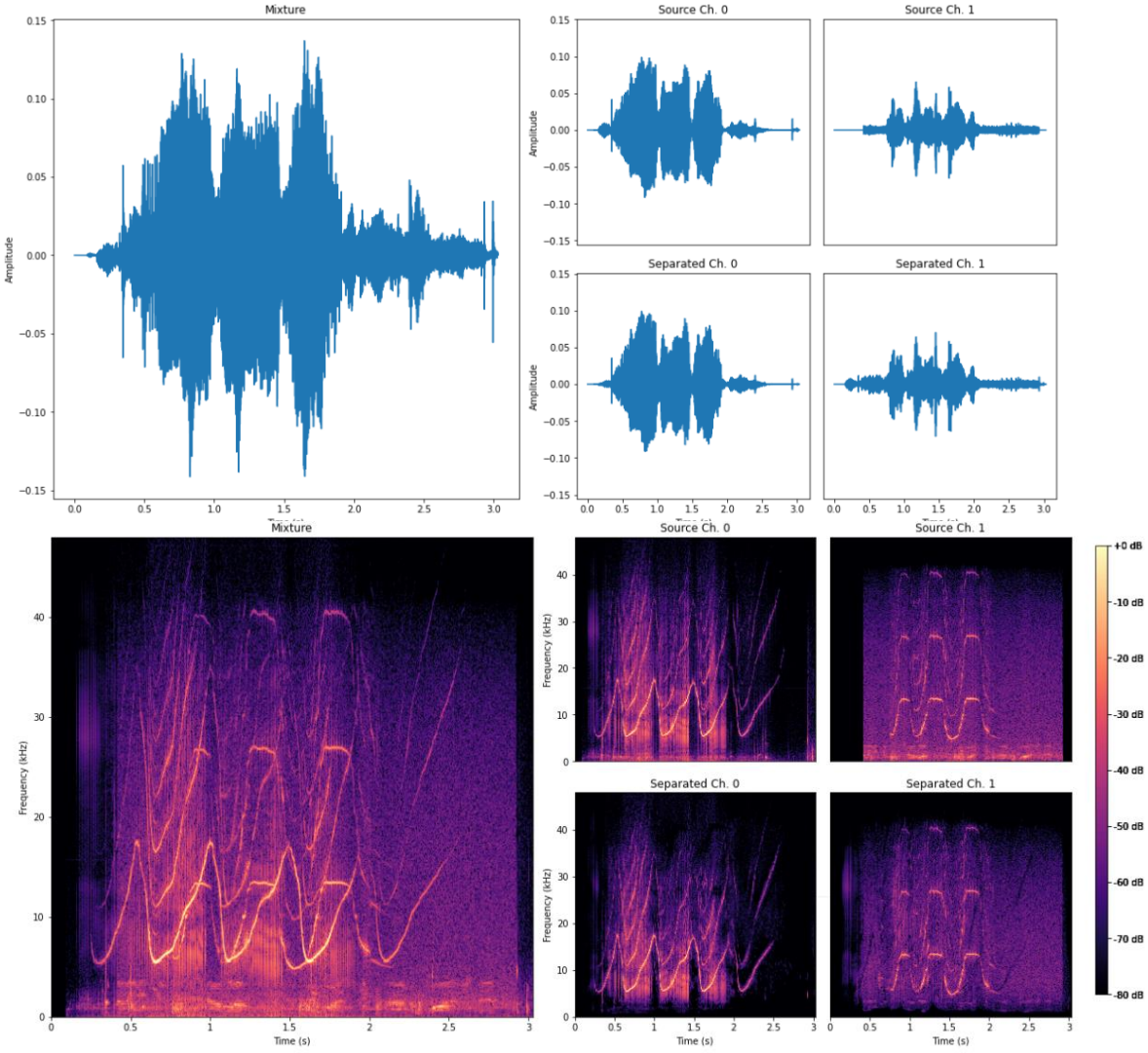


Figura 2-3. Señales de entrada y salida de la red separadora. (obtenida de [18])

2.2.2. Métodos ópticos

Como ya se vio, los métodos ópticos son una herramienta valiosa para obtener información sobre el comportamiento y las características físicas de las especies en su hábitat natural, lo que permite una mejor comprensión de la biodiversidad de un área y la protección de las especies que la habitan. Sin embargo, una desventaja importante de estos métodos es el gran volumen de datos que se generan en el proceso de observación y captura de imágenes.

Es aquí donde la inteligencia artificial juega un papel clave, ya que tiene la capacidad de procesar grandes

cantidades de datos de manera eficiente y precisa. Los algoritmos de aprendizaje automático pueden analizar imágenes y datos de comportamiento de las especies, identificando patrones y relaciones que serían difíciles o imposibles de detectar mediante el análisis manual.

Además, la inteligencia artificial también puede ayudar en la identificación y monitoreo de actividades humanas que amenacen la supervivencia de las especies, como la caza furtiva. Los algoritmos de detección de patrones pueden analizar imágenes de cámaras de vigilancia y alertar a los investigadores sobre posibles amenazas a las especies.

Entrando en detalle encontramos plataformas para la detección de especies, como puede ser **SpeciesClassification** [20], este sería la versión análoga para imágenes del **Multi Species Bioacoustic Classification**, visto en el apartado anterior. Este también forma parte de *IA for Earth* de Microsoft, su arquitectura se basa en una red neuronal convolucional (CNN), en particular usa *faster-RCNN* (se desarrolla en el **apartado 3.3.2.**). Cuenta con un modelo pre-entrenado que cuenta con la capacidad de detectar alrededor de 5000 plantas y especies animales, lo que lo hace una herramienta útil para aplicaciones en las que se necesite analizar gran cantidad de datos.

Puede ser una herramienta útil, pero cabe destacar que usa una arquitectura un tanto antigua ya que existen mejores alternativas a *faster-RCNN* en el ámbito de la detección, como pueden ser YOLO o SSD. Se compararán los modelos nombrados en el **apartado 3.3.2.**

También contamos con dos plataformas de detección especializadas en diferentes aplicaciones de detección como por ejemplo **CameraTraps** y **ArticSeals,** de igual forman parte de *IA for Earth* de Microsoft.

CameraTraps es esta especializado en imágenes procedentes del fototrampeo. Para ello hace uso de un modelo de detección llamado **MegaDetector**, permite su uso en aplicaciones en tiempo real por lo que se pueden realizar detecciones a través de una cámara que trasmita en directo.

MegaDetector [21] es un modelo de detección de objetos desarrollado por Microsoft utilizado en la plataforma **CameraTraps** [22], para identificar la presencia de animales en imágenes tomadas por cámaras de fototrampeo. Este modelo ha sido entrenado en una amplia variedad de especies animales, incluyendo mamíferos, aves y reptiles, y ha demostrado una alta precisión en la detección de animales en entornos naturales.

Utiliza una arquitectura basada en redes neuronales convolucionales (CNN), concretamente usa en su última versión usa YOLOv5 y ha sido entrenado con un conjunto de datos de más de 3 millones de imágenes. Además, esta versión también incluye mejoras en la precisión de la detección de animales y en la detección de especies raras o difíciles de identificar.

Este detector gracias a su buen rendimiento está siendo utilizado por gran cantidad de instituciones alrededor del mundo entre ellas la **Estación Biológica de Doñana.**

Por otro lado, **ArticSeals** [23] busca el estudio y la conservación de las focas en el ártico. Busca detectar y clasificar focas árticas en fotos aéreas. El objetivo principal es ayudar a los investigadores a recopilar información valiosa sobre la población de focas, su distribución y comportamiento. El proyecto utiliza un modelo de detección y clasificación basado en la arquitectura de redes neuronales convolucionales (CNN) llamado RetinaNet.

Por último, debemos hablar sobre la identificación de individuos de una especie, ya que es una tarea crucial en el seguimiento y conservación de la biodiversidad. La técnica más comúnmente utilizada para tal fin se basa en la comparación de patrones en la piel o pelaje de los animales, sin embargo, este proceso puede ser tedioso y consumir mucho tiempo. Para abordar esta problemática, se han desarrollado plataformas como **WildBook** [24], que automatizan el proceso de identificación utilizando la información obtenida de los patrones de piel o pelaje. De esta forma, se logra una estimación poblacional más precisa y se reduce significativamente el tiempo y la carga de trabajo necesarios para realizar la identificación de individuos en grandes poblaciones animales.

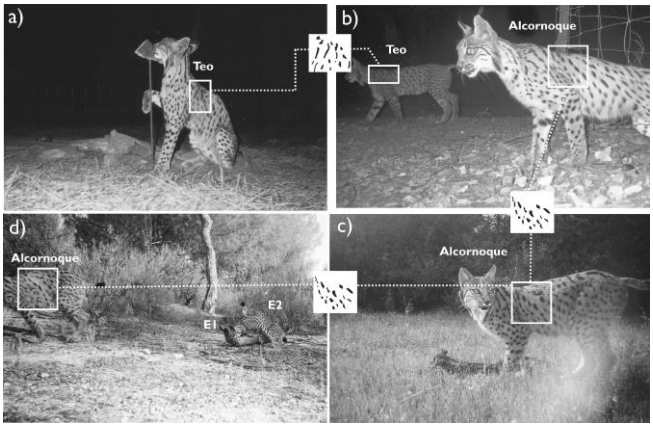


Figura 2-4. Identificación de linces mediante patrones en el pelaje. (obtenida de [30])

Wildbook-IA [25] es el modelo que usa la plataforma, como ya hemos comentado se basa en la comparación de patrones de la piel o pelaje de los individuos de una especie para identificarlos de forma automática. Serán necesarias imágenes de la piel o pelaje de los individuos, ya sea tomadas de forma manual o mediante cámaras trampa, se cargarán en la plataforma Wildbook. Luego, el modelo analiza los patrones en estas imágenes y los compara con una base de datos de patrones ya conocidos, para determinar la identidad de los individuos fotografiados.

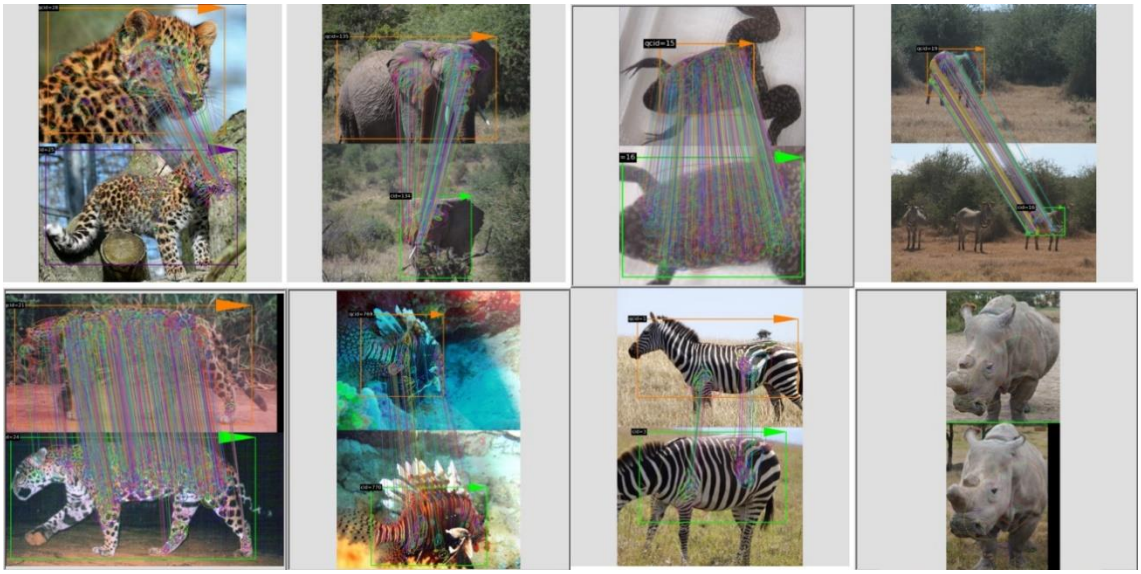


Figura 2-5. Funcionamiento de Wildbook-IA

Para llevar a cabo este proceso, Wildbook-IA utiliza algoritmos de aprendizaje automático basados en redes neuronales convolucionales (CNN). Estos algoritmos se entrenan con una gran cantidad de imágenes de diferentes individuos de la especie en cuestión, para que aprendan a reconocer y comparar patrones en las imágenes. Una vez entrenado, el modelo puede analizar nuevas imágenes y determinar la identidad de los individuos de la especie que aparecen en ellas, en función de los patrones detectados.

Además de la identificación automatizada, Wildbook-IA también permite a los usuarios realizar anotaciones manuales en las imágenes para corregir errores en la identificación, y añadir información adicional sobre los individuos (como su edad, sexo, etc.), así como información sobre el hábitat y las condiciones meteorológicas, para su posterior análisis. [24]

Actualmente Wildbook es una herramienta clave para la conservación y seguimiento de especies, ya que permite automatizar el proceso de identificación, una tarea que puede ser tediosa y consumir muchos recursos en términos de tiempo y dinero para los investigadores. Con Wildbook, se pueden llevar a cabo análisis más

eficientes y precisos de las poblaciones de especies, lo que ayuda a los investigadores y conservacionistas a tomar decisiones y a implementar estrategias efectivas de conservación y manejo de poblaciones. Esta plataforma ha sido utilizada con éxito para el seguimiento de numerosas poblaciones de especies, lo que demuestra su eficacia y versatilidad. Entre ellas encontramos tiburones [26], grandes felinos [27], incluso hasta el lince ibérico [28].

Además de Wildbook, existen otras herramientas de identificación. Una de ellas es **BearID** [29], la cual se enfoca específicamente en la identificación de osos utilizando una red neuronal convolucional (CNN). Esta técnica permite extraer características únicas del oso, como la forma de su cabeza, orejas y cuerpo, lo que permite identificar individuos con un alto grado de precisión.

Como se puede ver el uso de la inteligencia artificial está transformando la forma en que se aborda la conservación de la fauna, ya que permite a los investigadores procesar grandes cantidades de datos de manera eficiente y enfocarse en desarrollar tácticas efectivas para la conservación de especies y la investigación de ecosistemas.

A continuación, se presentarán algunos de los estudios más relevantes sobre el seguimiento del lince ibérico, que permitirán tener una visión más amplia y detallada de las estrategias utilizadas para proteger esta especie y asegurar su supervivencia. Se analizarán los resultados obtenidos, las técnicas empleadas y las limitaciones encontradas, con el fin de contextualizar el proyecto y justificar su importancia.

2.3. Trabajos previos sobre el rastreo y seguimiento del lince ibérico

Dedicaremos este apartado al análisis de dos trabajos relacionados con la detección y rastreo del lince ibérico. Gracias a los cuales podremos entender la utilidad del fototrampeo en el contexto de las estimaciones poblacionales, para así entender su utilidad y las ventajas que aportan a estos sistemas los algoritmos de aprendizaje automático.

Los proyectos en los que profundizáremos son, *"Change in demographic patterns of the Doñana Iberian lynx (Lynx pardinus): management implications and conservation perspectives"*, publicado en la revista Oryx, analiza los cambios en los patrones demográficos del lince ibérico en Doñana y sus implicaciones para la conservación de la especie. Por otro lado *"The use of camera trapping for estimating Iberian lynx (Lynx pardinus) home ranges"*, [31] publicado en la revista Wildlife Biology, se centra en la utilización de cámaras trampa para estimar los rangos de hogar del lince ibérico en la región de Sierra Morena, Andalucía.

Ambos artículos comparten metodologías y técnicas similares, por lo que es relevante destacar las estrategias empleadas de forma conjunta.

Durante el período de 2002 a 2006, el Ministerio de Medio Ambiente regional del gobierno de Andalucía, en el proyecto LIFE de la Unión Europea "Recuperación de las poblaciones de lince ibérico en Andalucía", estableció una metodología estandarizada para el rastreo del lince, utilizando la búsqueda de indicios y cámaras trampa. Esta metodología es una de las más utilizadas debido a su eficacia en la detección de la presencia del lince ibérico, en la estimación de su densidad poblacional, así como su baja invasión en los ecosistemas, lo que minimiza su impacto en la fauna y flora nativa.

Para llevar a cabo este método, es fundamental conocer los trabajos previos que demuestren la presencia del lince ibérico para si poder definir áreas de estudio.

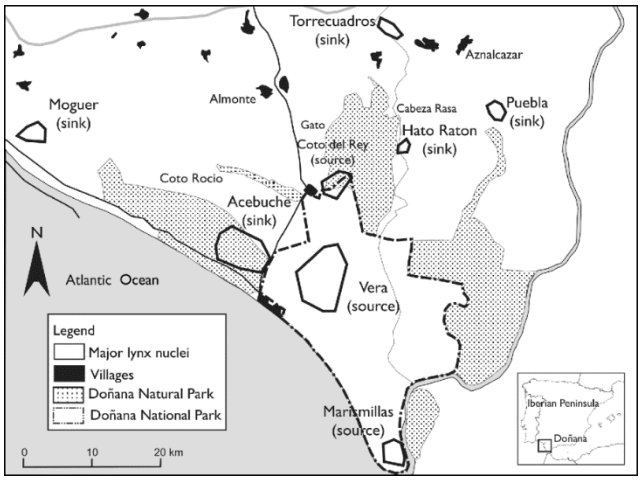


Figura 2-6. Zonas con presencia de linces en Doñana. (obtenida de [30])

La búsqueda de indicios por parte de los investigadores es fundamental para determinar si una zona es frecuentada por los linces, lo que la convierte en un lugar propicio para la colocación de cámaras trampa. De esta manera, se puede aumentar la eficacia del método y obtener datos más precisos sobre la presencia y actividad del lince en el área de estudio. En particular, en Sierra Morena, el área de estudio se dividió en cuadrículas de 1x1 km^2. Se consideró una celda como positiva si se encontraba al menos un indicio de la presencia del lince, y negativa si después de dos horas de búsqueda no se encontraron evidencias. Este proceso se repitió cada año para adaptar el mapa a los movimientos de la especie y obtener una imagen más precisa de las áreas en las que los linces son más frecuentes.

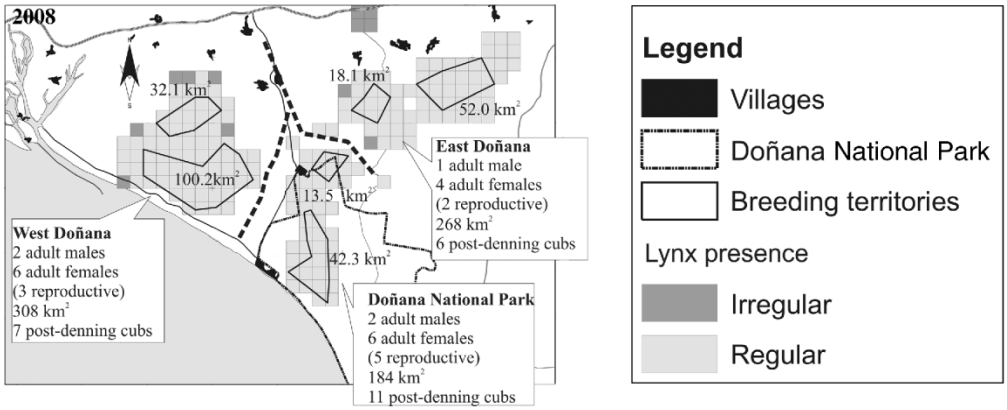


Figura 2-7. División en cuadrícula del territorio bajo estudio. (obtenida de [30])

De esta forma haciendo de esta versión discretizada del mapa por celdas de pueden determinar las zonas de campeo de la especie para así poder estudiar su comportamiento. Podemos ver en la **figura 2-6**, un ejemplo un estudio realizado en Doñana. Donde sí se detectaba la presencia de un lince en una celda por un periodo de seis meses consecutivos se marcaban como zonas regularmente usadas, en caso contrario se determinan como zonas irregulares o de paso.

Por lo que podemos ver, la detección de la presencia del lince es vital para conocer los patrones de comportamiento de estos y el uso de fototrampeo abunda como técnica en los proyectos realizados para su seguimiento. Así el uso de una herramienta como la inteligencia artificial que acelere el procesado de datos que estas cámaras producen es algo esencial para optimizar los procesos de investigación.

A continuación, en el siguiente capitulo revisaremos los fundamentos teóricos de la inteligencia artificial para así poder tener una mejor comprensión del proyecto y de los objetivos que se buscan cumplir.

3 FUNDAMENTOS TEÓRICOS

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

- Alan Turing -

Para una mejor comprensión y análisis del proyecto, resulta fundamental tener un conocimiento teórico que permita contextualizar y profundizar en los conceptos a tratar. En este capítulo, se abordarán los fundamentos teóricos necesarios para comprender los temas clave del proyecto, comenzando por los conceptos básicos de la inteligencia. Además, se analizarán en profundidad conceptos esenciales como la visión por computador y las redes neuronales convolucionales, que son vitales en la detección de objetos. Por último, se profundizará en el modelo utilizado en esta aplicación y se explorarán sus características relevantes.

3.1. Inteligencia artificial y machine learning

Definir que es la inteligencia artificial, es complejo por lo que podemos encontrarnos con diferentes respuestas a esta pregunta. Podemos simplificalo como «la habilidad de los ordenadores para hacer actividades que normalmente requieren inteligencia humana» [32] pero si queremos más profundidad podemos decir que es el «estudio y diseño de dispositivos capaces de percibir su entorno y tomar acciones que maximicen la probabilidad de éxito en diferentes objetivos y tareas».

Es cierto que existen maquinas con la capacidad de percibir su entorno y tomar decisiones, sin involucrar ningún tipo de inteligencia artificial, pero a los dispositivos que nos referimos son los que implementan los llamados *agentes inteligentes.* Estos se diferencian de los sistemas convencionales en que deben ser capaces de percibir su entorno, adaptarse a los posibles cambios, aprender de la experiencia y tomar la decisión optima, sin la necesidad de intervención humana. En resumen, los *agentes inteligentes* son sistemas de inteligencia artificial que están diseñados para actuar y tomar decisiones de manera autónoma en entornos complejos y dinámicos [33].

Es importante destacar que, coloquialmente el termino *inteligencia artificial* se aplica a maquinas con la capacidad de imitar las funciones cognitivas humanas como el aprendizaje y la resolución de problemas [34].

El *machine learning* o *aprendizaje automático*, es una disciplina dentro de la *inteligencia artificial*. Para entenderla podemos hacer una analogía con el proceso de aprendizaje humano. Al nacer, contamos con un cerebro que tiene la capacidad de aprender, pero aún no poseemos conocimientos adquiridos más allá de los instintivos. De manera similar, las máquinas tienen la habilidad de adquirir nuevos conocimientos sin la necesidad de ser programadas explícitamente. Esto se debe a la implementación de algoritmos que les permiten aprender y tomar decisiones a partir de datos externos [35].

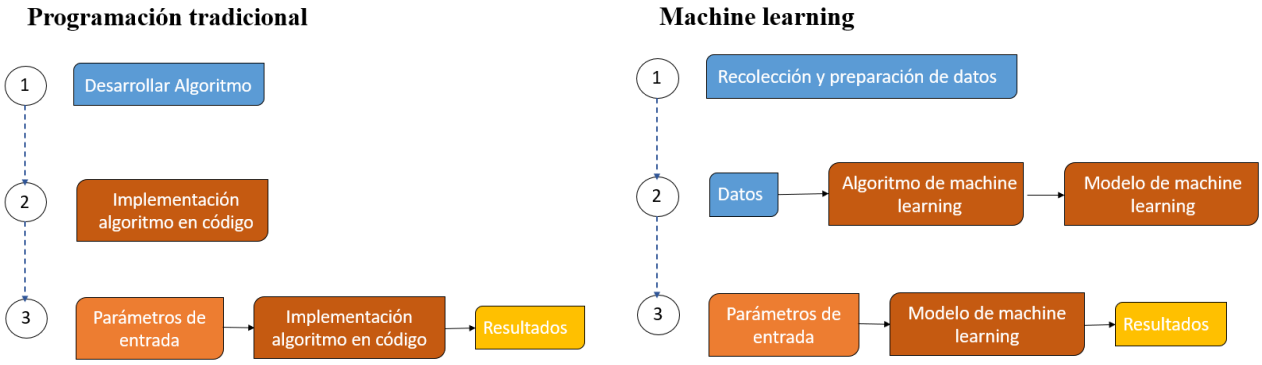


Figura 3-1. Comparación entre las técnicas de aprendizaje automático y la programación tradicional

Los *algoritmos de aprendizaje automático*, como se puede apreciar en la **Figura 2-6**, reciben datos de entrada y generan un programa o modelo que se ajusta a dichos datos. A diferencia de la programación tradicional, en el *aprendizaje automático* el ordenador es capaz de aprender a realizar nuevas tareas sin la necesidad de ser programado por un usuario. De esta forma, el ordenador es autónomo en su proceso de aprendizaje y puede "adquirir nuevos conocimientos" para realizar nuevas tareas [36].

Cabe destacar que, nos estamos enfocando únicamente en la posibilidad de aplicar *aprendizaje supervisado,* ya que es el que usaremos para generar nuestro modelo de detección. El *aprendizaje supervisado* consiste en entrenar mediante el uso de *algoritmos de aprendizaje supervisado* haciendo uso de ejemplos de las entradas y salidas correspondientes del problema a resolver. Este tipo de datos se conoce como *datos etiquetados* [37]. Los algoritmos de aprendizaje supervisado actúan como un bucle de control con realimentación negativa, toman la salida del modelo y la comparan con la salida esperada calculando así un error gracias al cual podrá ajustar el modelo y *aprender.* Podemos verlo con detalle en la **Figura 3-2**.

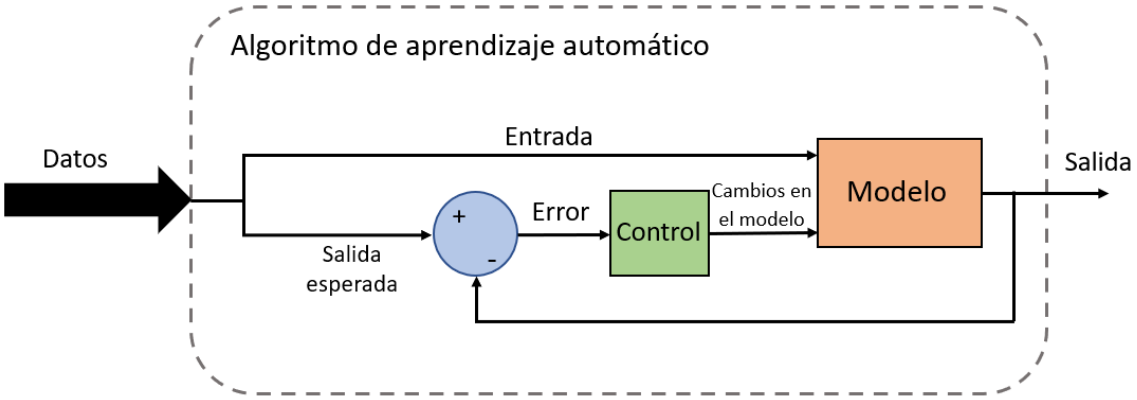


Figura 3-2. Analogía de los algoritmos de aprendizaje supervisado con control feedback.

Existen otros métodos como; el *aprendizaje no supervisado* donde no es necesaria la salida correspondiente para las entradas dadas, o el *aprendizaje por refuerzo,* ampliamente usado en robótica, el sistema interactúa con el medio con el fin de maximizar una recompensa numérica.

Una forma sencilla de entender el *aprendizaje automático* es comparándolo con la *programación tradicional* a través de un ejemplo. Queremos diseñar un detector que capaz de reconocer y detectar los dígitos de una matrícula de coche para aplicaciones de controles viales.

Si decidiéramos hacer uso de la *programación convencional* nuestro programa debería realizar unas instrucciones similares a las siguientes.

Partiríamos de imágenes similares a la de la **figura 3-3.** para extraer los caracteres del 0 al 9, en diferentes posiciones y resoluciones.

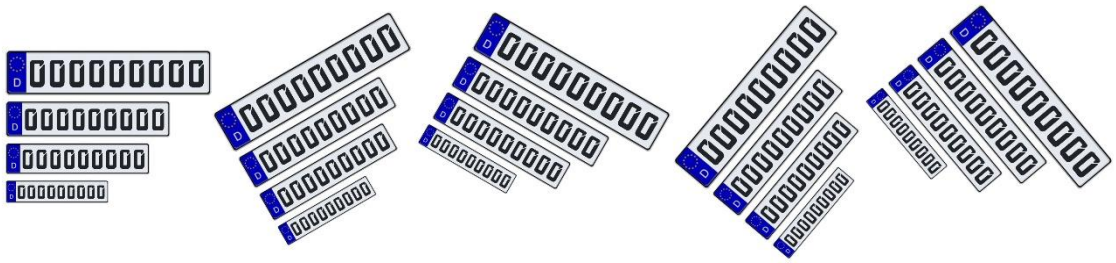


Figura 3-3. Ejemplo de tipo de dato.

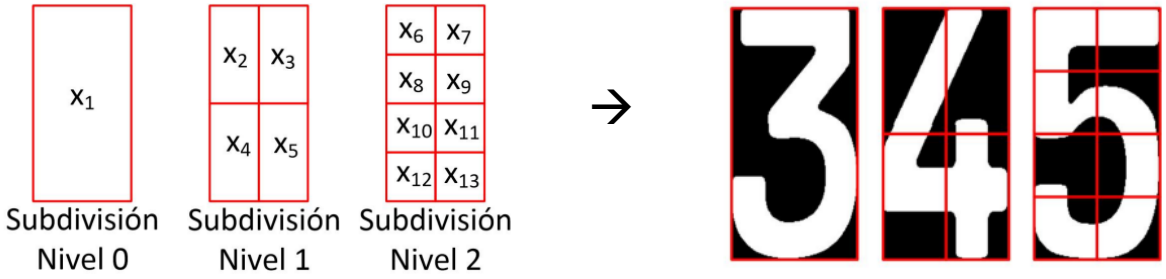


Figura 3-4. Codificación de las características a extraer de cada dígito.

De esta forma, se pueden extraer diferentes vectores de características para cada digito. Estos datos nos permitirán programar un clasificador que se encargue de asociar una clase a cada digito que se quiera identificar.

Si utilizamos técnicas de *aprendizaje automático* debemos crear datos para el entrenamiento de nuestro *algoritmo de aprendizaje automático.* Partiendo de imágenes similares a las de la **figura 3-5.** deberemos *etiquetar* los objetos que buscamos que nuestro sistema detecte, creando diez clases una para cada tipo de carácter. Tras esto tendremos un grupo de datos listos para entrenar nuestro algoritmo, como hemos señalado con anterioridad, estos contienen las entradas (las imágenes en las que detectar) que recibirá nuestro modelo de machine learning, así como las salidas (posición del carácter en la imagen, así como su clase asociada). Es decir, tendremos a la salida un tipo de dato que contendrá las coordenadas de la caja que contiene al digito, así como un identificador asociado a una clase.

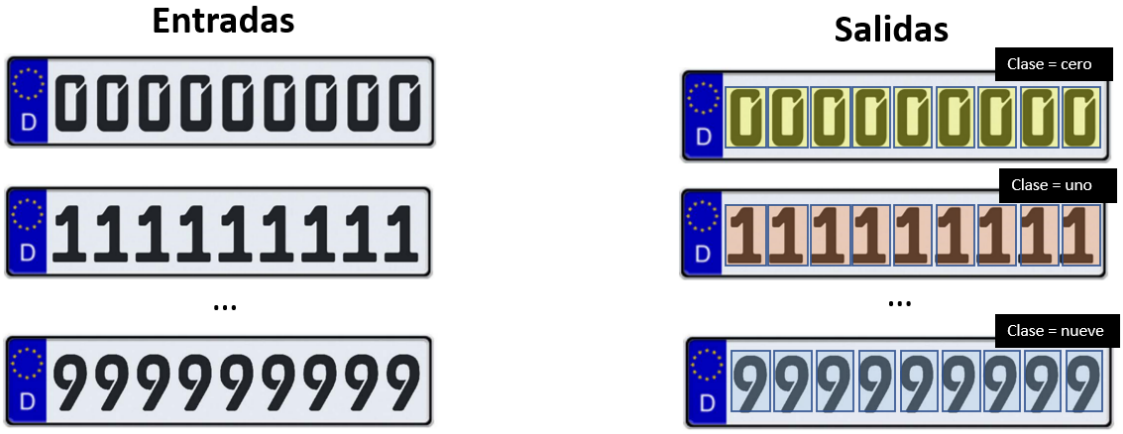


Figura 3-5. Codificación de las características a extraer de cada dígito.

En resumen, la programación tradicional se basa en automatizar una tarea mediante la escritura de instrucciones que un ordenador debe ejecutar. Por otro lado, el *machine* learning consiste en automatizar la tarea de escribir las instrucciones. En lugar de programar explícitamente un conjunto de reglas, el *machine learning* utiliza modelos matemáticos y estadísticos para encontrar patrones en los datos y realizar tareas automatizadas de forma eficiente y precisa.

Podemos concluir en que tanto la *programación tradicional* como el *aprendizaje automático* son dos disciplinas que, aunque cuentan con puntos en común, es importante saber cuál de las dos disciplinas utilizar según la aplicación específica en cuestión. Siguiendo con el ejemplo anterior; si queremos detectar objetos en una imagen, los métodos de *machine learning* son más adecuados, ya que su implementación con los métodos tradicionales puede que no sea evidente (como puede ser el problema de detectar linces en una imagen). Por lo tanto, los métodos de *aprendizaje automático* son más acertados para aplicaciones complejas hasta tal punto que su implementación tradicional sea imposible o demasiado compleja.

Después de adquirir los conceptos [35] fundamentales relacionados con el campo de la inteligencia artificial, exploraremos los modelos matemáticos que permiten a un ordenador aprender, así como los algoritmos de aprendizaje automático.

3.2. Redes neuronales

Conocer y entender la función que realizan las redes neuronales en el proceso de aprendizaje automático es esencial, ya que estos modelos serán los encargados de generar el programa que usaremos en nuestra aplicación.

Las redes neuronales, propiamente llamadas *redes neuronales artificiales* muchas veces simplificadas como NNs (*neuronal networks*); son un modelo matemático que busca imitar la forma de procesar la información de los sistemas nerviosos biológicos. Ya que la forma en la que funciona el cerebro humano es completamente distinta a la que lo hace un computador digital. El cerebro humano es un sistema altamente complejo, no lineal y paralelo, a diferencia que los computadores que son sistemas secuénciales, solo realizan una tarea a la vez [41]. Sin embargo, podemos encontrarnos con diferentes definiciones de lo que es una red neuronal. Sintetizando todas las ideas que rodean al concepto para dar una definición que sea lo más precisa posible, podemos afirmar que, una red neuronal es un sistema de computación basado en modelos matemáticos, compuesto por un gran número de elementos simples muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas [41].

Existen muchos tipos de *rede neuronales,* las cuales pueden clasificarse por sus diferentes características. Si las diferenciamos por su topología, podemos encontrar cinco grupos principales [38].

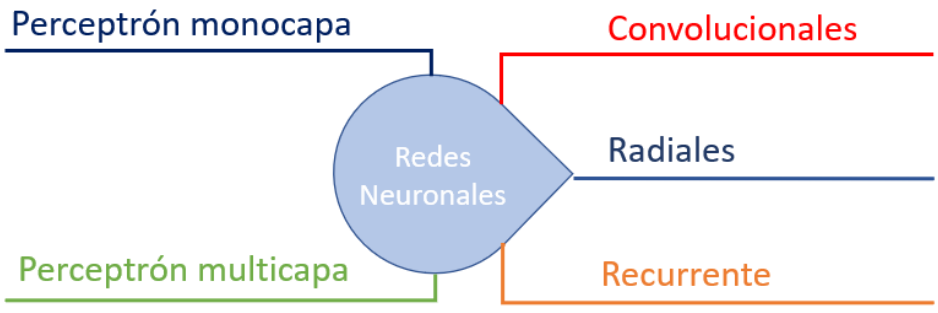


Figura 3-6. Tipos de arquitecturas de redes neuronales.

Nos centraremos en las redes convolucionales más a delante ya que son las usadas en las aplicaciones de visión artificial. Por ahora, para simplificar no centraremos en entender el *perceptrón multicapa,* cada capa forma lo que podemos denominar un vector de neuronas. [39]

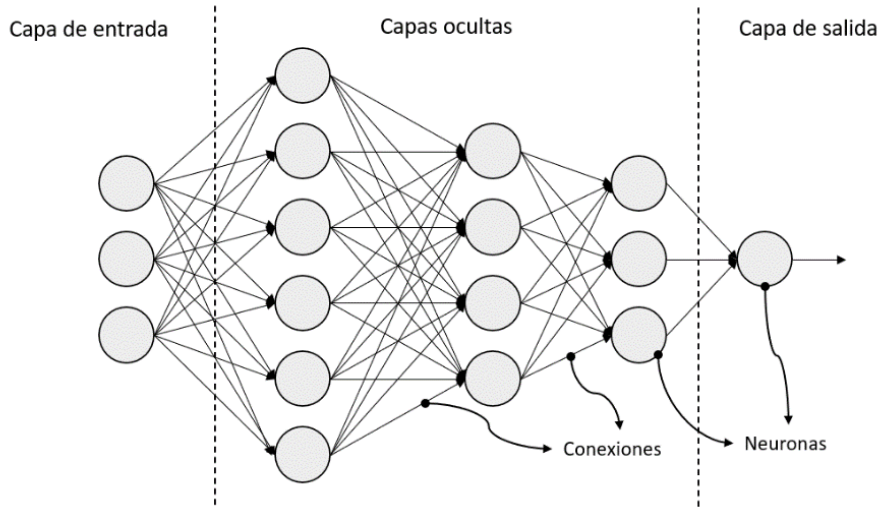


Figura 3-7. Esquema de un perceptrón multicapa.

El *perceptrón multicapa*, como su nombre indica, está formado por capas de perceptrones, los cuales son un modelo de neurona artificial en el que profundizaremos más adelante. Esta red consta de tres tipos de capas: una capa de entrada, una capa de salida y por último una capa oculta, estas capas tomarán los datos de las capas de entrada e irán variando los valores de estos hasta entregarlos a la capa de salida, puede haber varias capas ocultas o ninguna, dependerá del tipo de red.

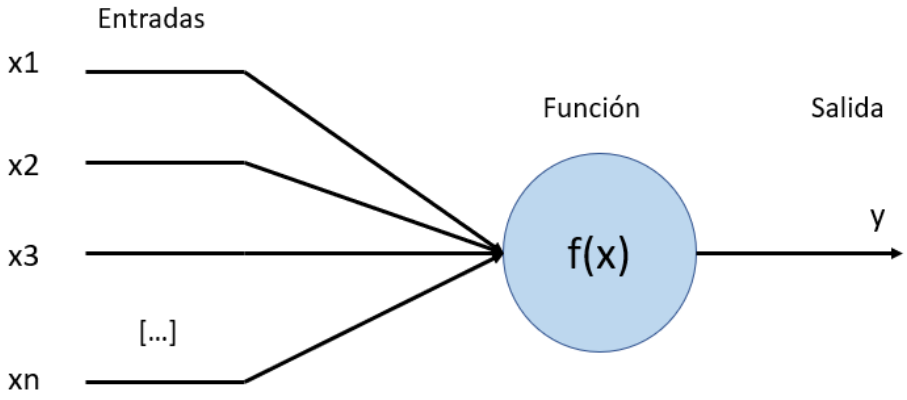


Figura 3-8. Esquema general de una neurona artificial.

Podemos ver un modelo simplificado y genérico de la neurona en la **figura 3-8,** estas cuentan con unas entradas gracias a las cuales y a una función matemática, se podrá calcular una salida [39]. Esa salida a su vez se convertirá en la entrada de otras neuronas. La clave del aprendizaje de una red neuronal se encuentra en los parámetros que forman la función de las neuronas. Estos parámetros se configurarán en el proceso de *entrenamiento* de la red.

A continuación, profundizaremos en el funcionamiento de las neuronas, para ello las compararemos con el sistema biológico nervioso que buscan imitar. Profundizaremos en el modelo matemático que las describe y entenderemos como consiguen aprender.

3.2.1. Analogía biológica

Como ya hemos mencionado previamente, las redes neuronales artificiales buscan replicar la forma en que los sistemas nerviosos biológicos procesan información. En estas redes, la neurona es la unidad fundamental de procesamiento, tal como ocurre en el cerebro humano. Las neuronas son células especializadas en recibir, procesar y transmitir información en forma de señales eléctricas.

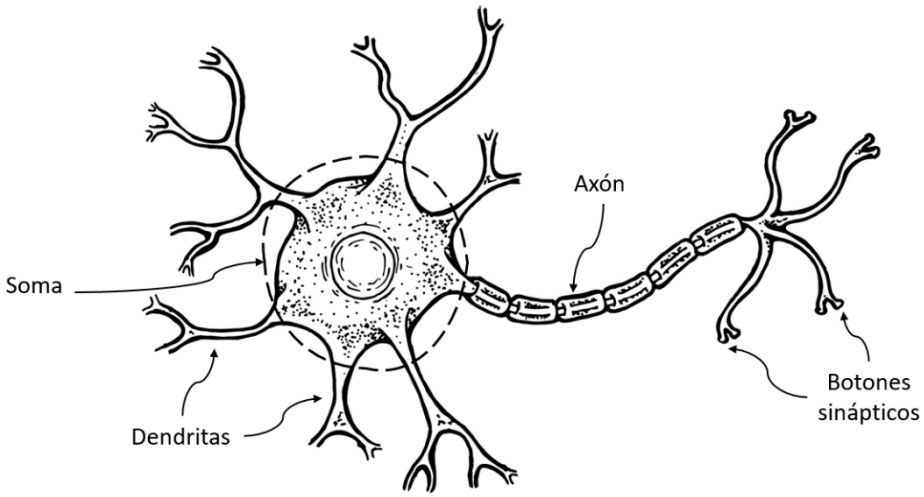


Figura 3-9. Partes de una neurona biológica.

Como podemos observar en la **figura 3-9** (disponible en enlace), podemos distinguir cuatro partes claramente identificables de las neuronas. Las dendritas reciben los neurotransmisores, mientras que el soma procesa la información y alberga el núcleo de la célula, donde se encuentra el ADN que contiene la información genética de la célula. Por su parte, el axón es responsable de transmitir los impulsos eléctricos a través de la neurona, mientras que los botones sinápticos se encargan de liberar neurotransmisores que permiten la comunicación con otras neuronas. [40]

El proceso de comunicación entre dos neuronas se conoce como sinapsis.

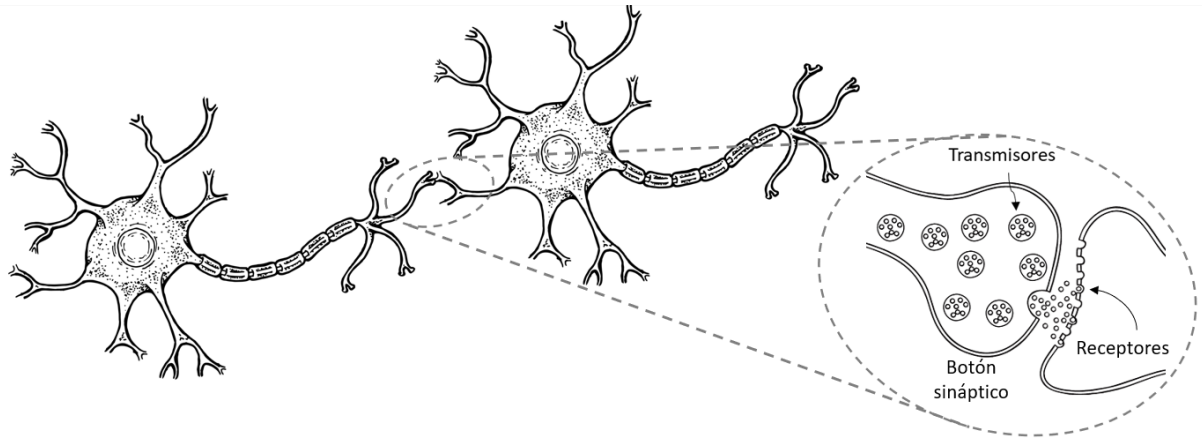


Figura 3-10. Descripción del proceso de sinapsis.

Durante la sinapsis, el impulso eléctrico que viaja a lo largo del axón de la neurona presináptica, libera neurotransmisores que se difunden a través del espacio sináptico hasta llegar a los receptores ubicados en las dendritas de la neurona postsináptica. Existen dos comportamientos a destacar durante el proceso de sinapsis.

El impulso eléctrico generado en una neurona y el que se propaga en la siguiente no siempre son idénticos. De hecho, la magnitud del impulso que se origina en la neurona postsináptica depende en gran medida de la cantidad de neurotransmisores liberados por la neurona presináptica. Esta cantidad de neurotransmisores puede

ser modificada por procesos de aprendizaje, lo que puede dar lugar a la amplificación o atenuación de la señal transmitida en la sinapsis.

Además, en el soma se integran los impulsos eléctricos que se reciben a través de las dendritas. Si la suma de estos impulsos supera un determinado umbral, se generará un impulso eléctrico que se propagará a lo largo del axón de la neurona. [41]

Conociendo cómo funcionan las neuronas biológicas podemos entender mejor su implementación matemática. Las dendritas de las neuronas biológicas y el axón serian equivalentes a las conexiones de entrada y salida de nuestras neuronas artificiales. Por otro lado, las modificaciones que sufre el pulso eléctrico debido a la cantidad de neurotransmisores, así como la activación o desactivación de la neurona se asemejan a las operaciones que lleva a cabo la función de su modelo matemático.

Tras entender en términos generales el funcionamiento de las neuronas tanto naturales como artificiales y su importancia, profundizaremos en su implementación matemática, pudiendo entender así mejor sus similitudes con las neuronas humanas.

3.2.2. Neuronas artificiales

Tras conocer ciertos conceptos bases sobre las redes neuronales y las neuronas biológicas podemos profundizar más en la implementación matemática de las neuronas, para así entender con mayor profundidad el funcionamiento de estas, así como sus similitudes con las neuronas naturales.

Uno de los primeros modelos neuronales que surgieron fue la neurona de *McCulloch-Pitts*, 1943. Esta solo podía tomar dos estados (activo o desactivado), similar a una puerta lógica.

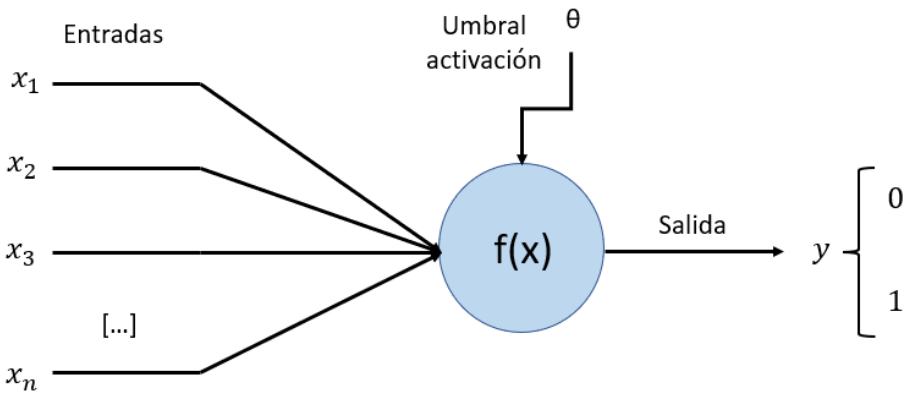


Figura 3-11. Modelo de la neurona de McCulloch-Pitts.

Esta neurona suma todas sus entradas, si esta suma supera un cierto umbral, el valor de la salida será uno en caso contrario cero. Este modelo de neurona forma la base de las redes neuronales artificiales más sencillas, conocidas como redes neuronales binarias. [42] [43]

Estudiaremos el perceptrón, este modelo de neurona fue introducido por *Frank Rosenblantt* en 1958. Esta neurona a diferencia de la de *McCulloch-Pitts,* tiene la capacidad de tomar valores reales.

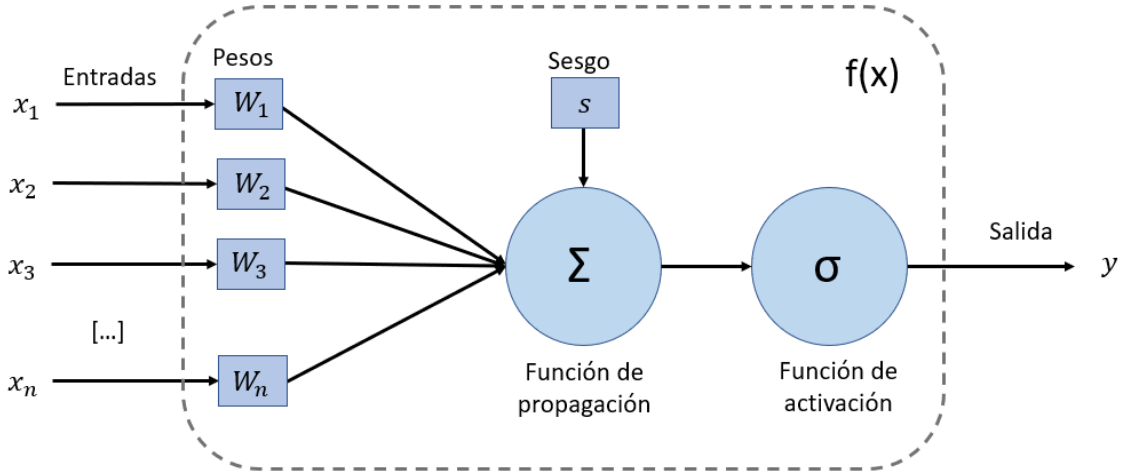


Figura 3-12. Modelo del perceptrón simple.

Podemos ver en la **figura 3-12.** con mayor detalle que en la **figura 3-8** los elementos que forman la función de la neurona**.** A continuación, analizaremos cada uno de sus elementos, los cuales no son únicos de este modelo, forman parte de muchos modelos de neuronas artificiales.

- **Pesos:** Son multiplicadores que decidirán la fuerza y dirección de conexión sináptica entre una neurona y las de la capa anterior. Es decir, cada neurona tiene un conjunto de parámetros que determinan la influencia relativa que tiene cada entrada en la salida de la neurona. Son de gran importancia ya que determinan el comportamiento de la neurona, sus valores se determinan en el proceso de aprendizaje.
  Como se ha comentado anteriormente las neuronas biológicas tienen un mecanismo similar. En su proceso sináptico se da más peso a ciertas entradas dependiendo de la cantidad de neurotransmisores que la neurona postsináptica libere. [44]

- **Función de propagación:** Es la operación matemática que se realiza a la entrada de la neurona. Generalmente esta será la suma de las entradas de la neurona ponderada por los pesos, cuenta además un con un sesgo, que dará más peso a algunas neuronas.
  En la siguiente ecuación vemos la representación matemática.

$$y = s + \sum_{i=1}^{n} w_i x_i \quad (3\text{-}1)$$

  La función de propagación cumple la función del soma, en la neurona humana. La salida de esta será procesada por la función de activación.

- **Función de activación:** Es una función matemática que se aplica tras la operación de la función de propagación. La función de activación determina si la neurona se activa o no, es decir produce una salida nula o no (al igual que ocurre en las neuronas biológicas). Además, añade una componente no lineal a la neurona, añadiéndole cierta complejidad que le permite resolver problemas no lineales. Si la red estuviese formada solo por operaciones lineales, estas se concatenarían siendo el resultado equivalente a haber hecho una única operación lineal. La elección de esta función determinará en gran medida el comportamiento de nuestra red neuronal. [41]
  Podemos ver a continuación algunos ejemplos de funciones de activación.

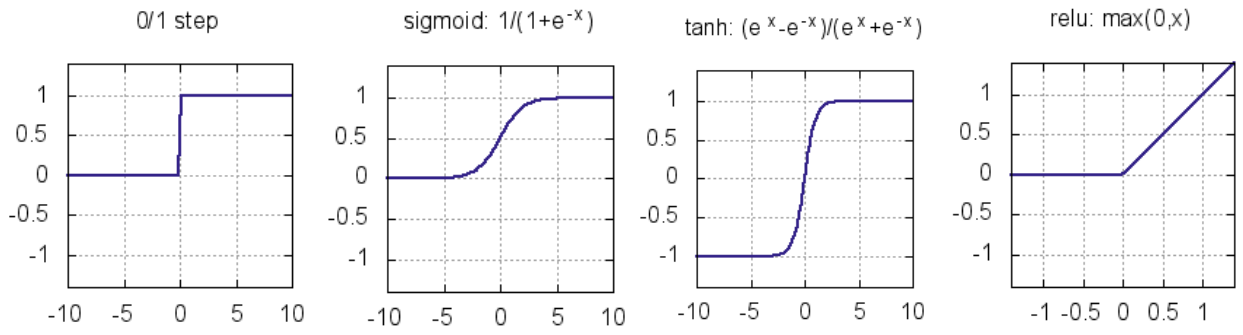


Figura 3-13. Diferentes tipos de funciones de activación [45].

De una forma similar a como se hace con las series de Fourier, la concatenación lineal de las funciones no lineales, seno y coseno, permite aproximarse a cualquier función de manera teórica, aunque en la práctica las redes neuronales son capaces de modelar funciones aún más complejas.

Observado la figura nos encontramos, la función escalón la cual ofrece una salida binaria, activa o desactiva la neurona por lo que es muy útil para aplicaciones de clasificación binaria, por otro lado, la sigmoide ofrece un rango continuo de cero a uno, permitiendo así la representación de probabilidades, así como su aplicación en casos más complejos y con mayor casuística.

Sin embargo, la función sigmoide puede complicar el problema al calcular su derivada para aplicar ciertos métodos de aprendizaje, como la retropropagación (*backpropagation*). Debido a esta limitación, surgió una nueva función de activación que ha ganado gran popularidad: *The Rectified Linear Unit* (ReLU). La ventaja de **ReLU** es que tiene una derivada fácil de calcular y es computacionalmente eficiente, lo que la hace ideal para su uso en redes neuronales convolucionales, por los cual su aplicación en visión artificial será clave. Además, ReLU es una función que mantiene la no linealidad permitiendo a la red neuronal modelar funciones más complejas que las que se pueden lograr con una función lineal. [45]

Tras conocer cómo funcionan las neuronas artificiales, así como su implementación matemática. Continuaremos explorando como se configuran las neuronas en el proceso de aprendizaje para conseguir que nuestra red aproxime la función deseada.

3.2.3. Aprencizaje

Como ya se comentó, aprendizaje de una red se lleva a cabo durante el proceso de entrenamiento. La red partirá con unos valores predefinidos de sus parámetros, será entrenada con un conjunto de ejemplos de las salidas y entradas (*samples*) del problema a resolver. Los datos de entrada se irán propagando por toda la red generando unas salidas, este proceso es conocido como *propagación hacia delante* (*forward propagation)*.

Para evaluar la calidad de las salidas generadas, se utiliza una función de error, llamada *función de costes,* que mide la diferencia entre la salida real y la salida esperada. Luego, se utiliza la retropropagación (*backpropagation)* para ajustar los parámetros de las neuronas de la red y minimizar el error de salida. Este proceso de ajuste continuo de los parámetros de la red, basado en la retroalimentación del error de salida, se repite durante el entrenamiento hasta que se logra una precisión satisfactoria.

Podemos describir el proceso de aprendizaje en los siguientes pasos [46] [39]:

1. Se alimenta la red con la primera muestra, esta se propagará por toda la red (*forward propagation*) hasta llegar a la última capa, donde se comparará el resultado obtenido con el esperado para dicha entrada, haciendo uso de una *función de coste,* gracias a la cual obtendremos un *error de entrenamiento* (*training loss*), la forma de medir el error puede variar, pero podemos representarla de forma genérica en la **ecuación (3-2)**.

$$E_p = \mathfrak{F}\left(y_p - t_p \right) \quad (3\text{-}2)$$

   Podemos ver el error para una muestra (*sample*) p como una función de la salida obtenida (y_p) menos la salida esperada para dicha muestra (t_p).

2. El error calculado se retro propagará por toda la red, se calculará una matriz de variaciones para los pesos de las neuronas de la red. Esta matriz se obtiene como salida de una función que toma como entrada el error.

$$\Delta w_p = \mathfrak{G}(E_p) \quad (3\text{-}3)$$

$$w = w + \Delta w_p \quad (3\text{-}4)$$

   Obtenemos así la variación necesaria de los pesos de la red para ajustar el error cometido. Siendo w la matriz de pesos de la red.

3. Se repetirán los pasos anteriores hasta llegar a un error que cumple nuestras especificaciones de tolerancia.

La ejecución de todos estos pasos para una muestra o *sample* se denominará un bucle o *loop*. En el algoritmo de entrenamiento, es importante tener en cuenta dos hiperparámetros, que son parámetros que no dependen de la red neuronal sino del algoritmo de aprendizaje. El primero es el tamaño de un *batch,* define el número de samples que deben pasar por la red para actualizar los parámetros internos. Es decir, hasta que el ultimo *sample* del *batch* no ha generado una salida, no se pasará al siguiente paso del algoritmo. El segundo hiperparámetro es el número de *epochs*, que define la cantidad de veces que todo el conjunto de datos pasará por la red. [47]

El algoritmo que se ha definido muestra una descripción muy esquemática y teórica. Existen diferentes algoritmos de aprendizaje automáticos que dependen de la red, como de la aplicación en la que se quiere implementar. A continuación, profundizaremos en un método de aprendizaje conocido como *algoritmo de gradiente descendente* ya que además de ser uno de los más extendido será el que usemos para entrenar nuestro sistema de detección.

Particularizaremos el algoritmo para una neurona, gracias a esta simplificación no tendremos en cuenta la *retropropagación* simplificando así en gran medida las operaciones a realizar.

La función por minimizar seria, como ya se ha comentado anteriormente, la *función de costes* esta depende de la salida esperada (t_j), así como de la salida obtenida (y_j) por la neurona, para todo el conjunto de *samples* de un *batch*.

$$E(\vec{w}) = \frac{1}{2}\sum_{j \in N}\left(y_j(\vec{w}) - t_j\right)^2 \quad (3\text{-}5)$$

En la **ecuación (3-5)** podemos notar que el error está expresado en función del vector de pesos de la neurona. Esto se debe a que el vector de pesos es la variable que, para una misma salida deseada, minimizará el error. De hecho, la salida obtenida por la neurona está determinada por este vector, lo que explica su fuerte influencia en el cálculo del error.

Con el objetivo de minimizar el error, es necesario encontrar la combinación óptima de pesos que nos permita generar una salida lo más parecida posible a la deseada. Para lograr esto, es fundamental calcular el gradiente de la función de error para conocer la dirección de mayor descenso. De esta manera, podemos ajustar gradualmente los pesos en la dirección correcta para obtener una salida cada vez más precisa y reducir el error de manera efectiva. [39]

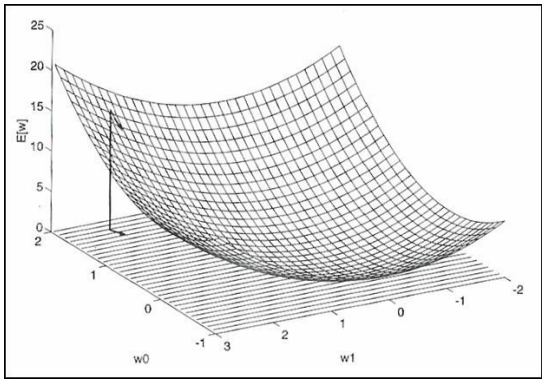


Figura 3-14. Representación del gradiente para dos pesos.

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right] \quad (3\text{-}6)$$

En la **figura 3-14.** (obtenida de la referencia [39]) podemos ver una representación gráfica de la función de costes para una neurona con dos únicas entradas, podemos ver en esta la representación del gradiente.

Podemos simplificar esta grafica aún más para una *función de costes* con una sola dimensión (es decir un solo peso que optimizar), obtendríamos una parábola como se puede ver en la **figura 3-15** (obtenida de la referencia [48]).

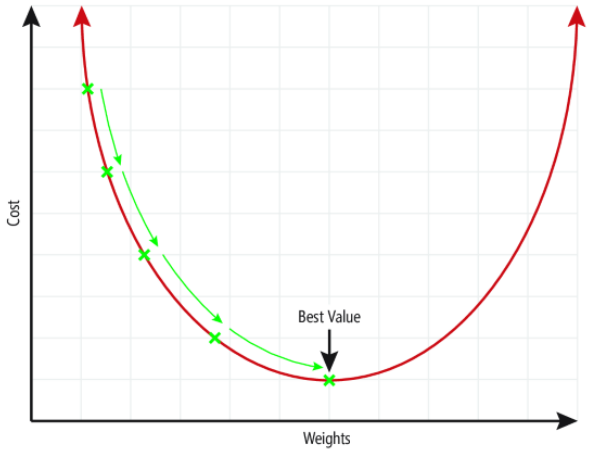


Figura 3-15. Representación del gradiente para un solo peso.

Partiendo de un valor inicial la red se irá ajustando a un valor cada vez más optimo según van pasando los conjuntos de muestras.

Sabemos cómo se encuentra la dirección a seguir, pero aún no conocemos que cantidad se acerca al punto mínimo. El parámetro encargado de decidir como de grande son los pasos que se darán es el llamado *ratio de aprendizaje η,* este tomará valores entre 0 y 1, es interesante que sea lo mayor posible ya que así el aprendizaje de la red será más rápido, pero corremos el riesgo de que la función a optimizar diverja y nos pasemos del mínimo. Por otro lado, una *ratio de aprendizaje* demasiado pequeño puede llevar a un aprendizaje lento. [49]

Por lo cual el incremento a aplicar a los pesos de la neurona se calcularía tal y como se muestra en la **ecuación (3-7)**, tras esto se aplicará a los pesos de la neurona y se repetirá el proceso.

$$\Delta w = -\eta \nabla E_p \quad (3\text{-}7)$$

$$\vec{w} = \vec{w} + \Delta w \quad (3\text{-}8)$$

Si contemplamos redes neuronales multicapa, es esencial aplicar la *retropropagación* [46]. Ya que deberemos trasmitir el error hacia atrás a través de las capas de la red, calculando el error en cada capa en función del error en la capa anterior. De esta forma, podremos aplicar un algoritmo similar al descrito para ajustar gradualmente los pesos de la red.

Antes de dar por concluido este apartado debemos mencionar dos errores que pueden darse durante el entrenamiento estos son el *underfitting* y el *overfitting*. Podemos observar en la **figura 3-16** un ejemplo basado en un ajuste de una nube de puntos.

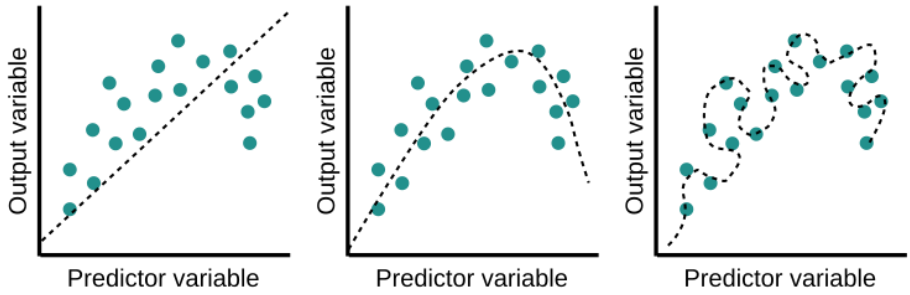


Figura 3-16. Ejemplificación del underfitting y el overfitting en un ajuste de curvas.

En las redes neuronales ocurre de igual forma, puede que el error durante el entrenamiento se haga muy pequeño, pero esto no nos asegura que el modelo se esté ajustando correctamente a nuestro problema. Para asegurarnos de esto necesitamos un grupo de muestras distintas a las usadas en el entrenamiento para comprobar que el que llamaremos *error de validación* tiene tendencia descendente.

En la siguiente **figura 3-17**. (obtenida de la [50]) podemos observar la evolución de estos errores según avanza el entrenamiento.

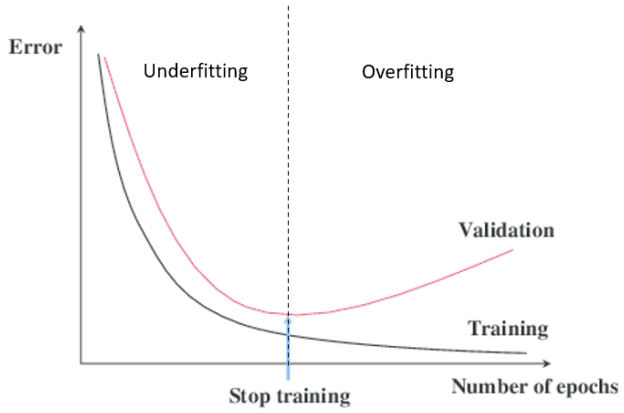


Figura 3-17. Evolución de los errores de validación y entrenamiento.

Si ambos errores tienen un valor alto estaremos en una situación de *underfitting*, sin embargo, si el *error de entrenamiento* es bajo, pero el *error de validación* es alto estaremos en una situación de *overfitting*. Se busca obtener el mínimo posible de ambos errores.

Después de haber comprendido los conceptos generales de la inteligencia artificial, es necesario profundizar en conceptos más específicos que se relacionan con el uso que se le quiere dar a la inteligencia artificial en este proyecto; la detección de objetos. Lo cual implica el uso de técnicas avanzadas de procesamiento de imágenes

y redes neuronales convolucionales para identificar y localizar objetos en una imagen. En los siguientes apartados, nos enfocaremos en estos conceptos específicos para brindar una base completa y detallada de los métodos utilizados.

3.3. Visión por computadora

El sentido de la vista además de complejo es fundamental para resolver muchos de los desafíos que enfrentamos en nuestra vida diaria. Así que uno de los grandes retos del hombre ha sido comprender su funcionamiento, para así poder imitarlo a través de las tecnologías existentes, buscando así automatizar problemas que requerirían de la intervención humana. [51]

De esta forma, nace la *visión por computadora,* inspirada en la visión humana. Su objetivo principal es modelar y automatizar el proceso de reconocimiento visual, como puede ser la detección de objetos mediante el uso de *inteligencia artificial*. Pero también busca otra extraer información de las imágenes como pueden ser, la detección de puntos 3D mediante modelos geométricos o buscar la posición de la cámara. Así como otras aplicaciones de procesado de imágenes como, deformación de imágenes, eliminación de ruidos o incluso realidad aumentada. [52]

Como se puede ver la *visión por computadora*, es una disciplina muy amplia, que incluye métodos para adquirir información de imágenes, así como procesarla y analizarla.

Podemos ver en **figura 3-18** un esquema general de las etapas que se encuentran en la mayoría de las aplicaciones.

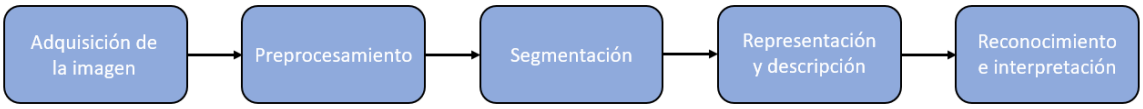


Figura 3-18. Etapas de un sistema de visión por computadora.

Los sistemas de *visión por computadora* pueden tener múltiples aplicaciones, como el seguimiento de objetos a tiempo real, detección de caras, construcción de modelos 3D, vigilancia, inspección robótica, como muchas otras.

Como ya se ha comentado anteriormente, el uso de *inteligencia artificial* en sistemas de *visión por computadora* puede ser clave para la resolución del problema. La corteza visual del cerebro, que es responsable de la funcionalidad de la visión, ha sido ampliamente estudiada y se ha logrado modelar su funcionamiento. De esta manera, surgieron las *redes neuronales convolucionales*, que son una herramienta poderosa para el procesamiento de imágenes. A continuación, profundizaremos en este tema [52].

3.3.1. Redes neuronales convolucionales

Las *redes neuronales convolucionales* o *CNNs* (*convolutional neural networks*) representaron un gran avance en las aplicaciones de *visión por computadora*, ya que están diseñadas específicamente para recibir imágenes como entradas. A diferencia de otras *redes neuronales* que requieren que sus entradas sean vectores, las *redes convolucionales* pueden procesar imágenes directamente, lo que las hace mucho más eficientes y precisas en el análisis de imágenes, dando solución a muchos de los problemas que plantea la *visión por computadora (*en la **figura 3-19.** podemos ver las etapas que resolvería la red que usaremos en el proyecto). Esta capacidad de procesamiento de imágenes ha permitido que las redes neuronales convolucionales sean ampliamente utilizadas en diversas aplicaciones de *visión por computadora.*

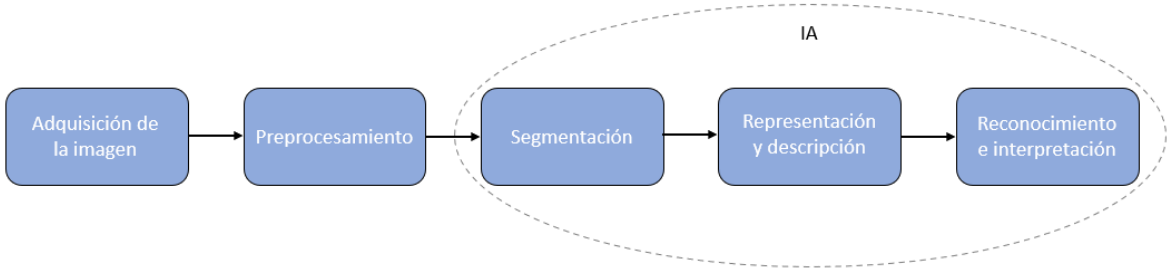


Figura 3-19. Etapas cubiertas por el modelo a usar en este proyecto.

Desde un punto de vista computacional, una imagen se puede representar como una matriz de píxeles, donde cada píxel tiene un valor de intensidad que va desde 0 hasta 255. Para procesar una imagen con una buena resolución de 1600x900 utilizando una red neuronal convencional, necesitaríamos una capa de entrada de 1.440.000 neuronas, una para cada píxel en la matriz.

El problema se complica aún más debido a que cada neurona en la capa siguiente está conectada a todas las neuronas en la capa anterior. Por lo tanto, cada neurona en la segunda capa tendrá un total de 1.440.000 entradas y, por lo tanto, el mismo número de pesos. En total, la segunda capa de la red tendrá un total de 1.440.000xN (siendo N el número de neuronas en la capa), lo que aumenta enormemente el número de parámetros a ajustar durante el entrenamiento, lo que puede producir sobreajustes, además de aumentar el costo computacional asociado.

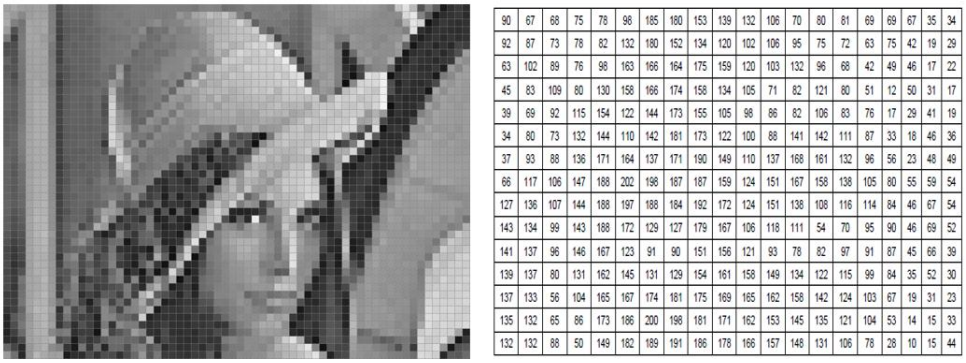


Figura 3-20. Codificación de una imagen en escala de grises.

El problema se vuelve aún más complejo si queremos procesar imágenes a color, ya que para representar los colores necesitamos tener en cuenta todo el espectro visual. Para esto, podemos utilizar la codificación RGB del color, lo que significa que necesitaremos tres matrices numéricas para cada uno de los canales de color; rojo, verde y azul. Esto aumenta el costo computacional en un factor de tres, ya que ahora tenemos que procesar tres matrices de píxeles en lugar de una sola.

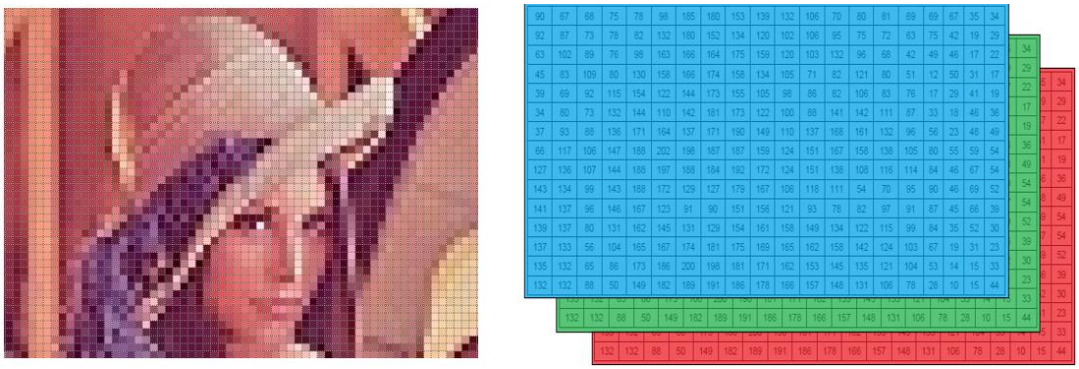


Figura 3-21. Codificación de una imagen a color en RGB.

De esta forma las CNNs modifican dos principales aspectos de las redes neuronales convencionales [53].

- Si nuestra red busca procesar imágenes a color, esta recibirá matrices tridimensionales como entradas. Para que la red sea capaz de procesar este tipo de datos, es necesario que la estructura de sus neuronas sea también tridimensional.
- Para reducir el coste computacional, las capas posteriores se conectan solo a una pequeña región de las capas anteriores. De esta manera, se logra que ciertos subgrupos de neuronas se especialicen en características específicas de la imagen.

Las capas de la red funcionan mediante la concatenación secuencial de capas que procesan información a diferentes niveles de abstracción. En las capas iniciales, se buscan características básicas de la imagen, como bordes y patrones simples, mientras que en las capas posteriores se combinan estas características en formas más complejas. Las capas finales utilizan esta información para hacer una predicción, comparando la entrada con los patrones obtenidos durante el proceso. Estas capas finales forman una red de perceptrones multicapa totalmente conectada, gracias a ella se obtiene una respuesta final que representa la probabilidad de que la entrada pertenezca a cada categoría posible [54].

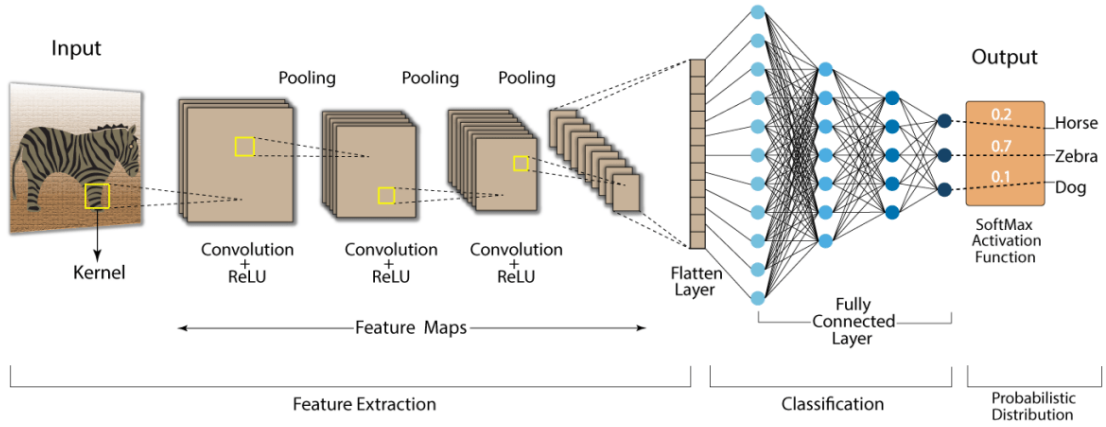


Figura 3-22. Esquema general de una red neuronal convolucional.

A continuación, detallaremos las diferentes capas que forman la red; estas son la capa convolucional, la de reducción o pooling y por último la capa clasificadora.

3.3.1.1. Capa convolucional

La *capa convolucional* es una de las características distintivas de las *redes neuronales convolucionales* (CNNs), ya que utiliza la operación de convolución en lugar del producto vectorial utilizado en otras redes neuronales. La operación de convolución toma una imagen como entrada y aplica un filtro o **kernel** para generar un mapa de características. Esta capa es esencial para el procesamiento de imágenes y es responsable de extraer características relevantes de la imagen de entrada. Además, la capa convolucional es a menudo la primera capa en una CNN y da nombre a la red.

Podemos ver a continuación en la **ecuación (3-9)** como se calcula en el caso de imágenes a color, es decir en matrices tridimensionales.

$$g(i,j) = \sum_{q=-c}^{c} \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t,q) \cdot f(i+s, j+t, k+q) \quad (3\text{-}9)$$

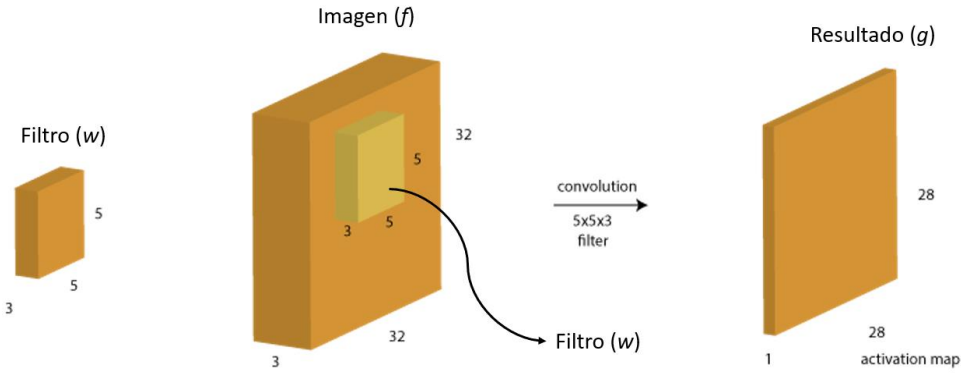


Figura 3-23. Esquema de la operación de convolución sobre una imagen a color.

El filtro en una capa convolucional es una matriz pequeña que se desliza a lo largo de los términos de la matriz de la imagen de entrada. Durante esta operación, la convolución se calcula entre el filtro y los valores de píxeles correspondientes de la imagen. Esto genera un nuevo valor para el píxel de la imagen que coincide con el termino central del filtro. Este proceso se repite para cada píxel en la imagen de entrada, lo que produce una nueva imagen bidimensional de salida. En la **figura 3-23,** podemos observar un ejemplo para poder entender mejor como se realiza esta operación.

Como se puede observar en la **figura 3-24**, el resultado tiene unas dimensiones menores a la de la imagen de entrada, esto se debe a dos factores. Primero, el resultado de una convolución siempre es una imagen en dos dimensiones. En segundo lugar, se produce un **efecto borde**, el cual consiste en que no se puede aplicar la máscara en pixeles bordes de la imagen, ya que parte de la mascará se quedaría fuera de la imagen. Aunque existen soluciones para este problema, como el relleno de ceros (zero padding) o la aplicación parcial de la máscara, no se considerarán en este apartado.

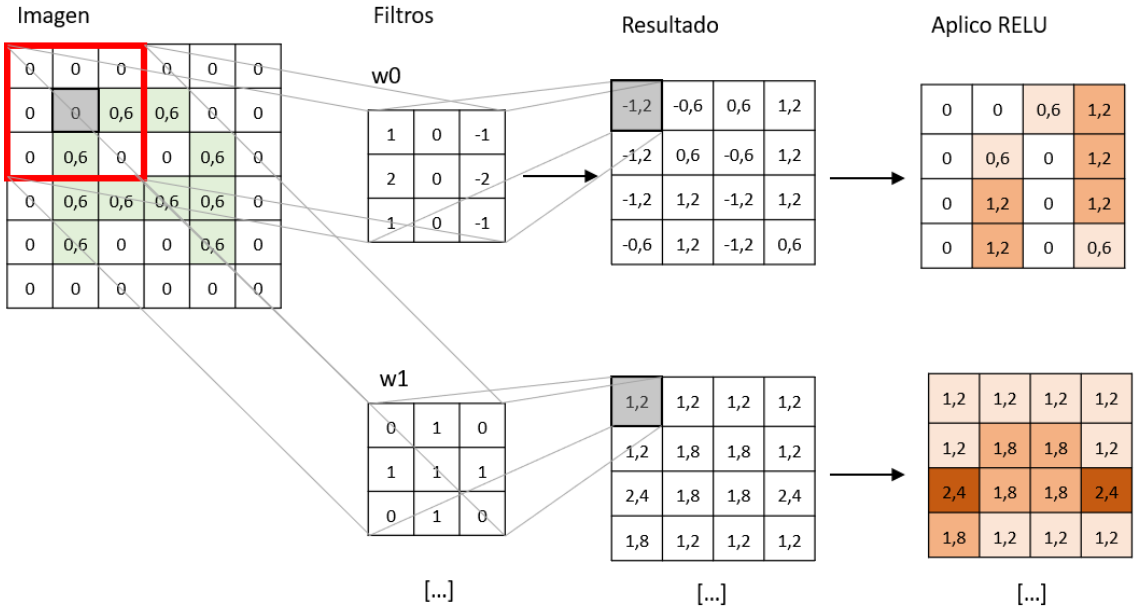


Figura 3-24. Ejemplo de la operación de una capa convolucional sobre una imagen de entrada.

La última, operación que realizará en esta capa será aplicar la función de activación **RELU,** la cual se ha detallado con anterioridad. Aplicar esta función eliminará todos los términos negativos, generándose así un *mapa de características*.

De esta forma en la **figura 3-24**, podemos ver la estructura de las operaciones que realizaría una capa convolucional. Los pesos de esta coinciden con los valores de los términos de las máscaras, gracias a la cual se aplicará la convolución; en las redes neuronales estándares se aplicaría un producto escalar. Finalmente, se aplica una función de activación, en este caso la *Rectified Linear Unit* (ReLU) vista en el **apartado 3.2.2.** [55]

Se aplicarán varios filtros para extraer diferentes mapas de características, dando como salida una matriz tridimensional de mapas. Los filtros tendrán siempre la profundidad de el volumen de entrada, es decir si la primera capa de convolución de la red recibe una imagen de 1600x900x3, los filtros a aplicar deberán ser del tipo WxHx3 (con W el ancho y H la altura del filtro). De esta forma si aplicamos 32 filtros se generará un volumen de salida que tendrá una profundidad de 32, por lo cual los filtros de la siguiente capa deberán tener esta profundidad. Los filtros de una capa convolucional tendrán la profundidad del volumen de entrada. [56]

Tras una capa de convolución, se aplicará a los mapas de características un proceso de reducción para simplificar la red y permitir la aplicación de más convoluciones en capas posteriores.

3.3.1.2. Capa de reducción o pooling

Si aplicamos una capa de convolución con 32 **kernels** a una imagen, obtendremos un volumen de 32 mapas de características como salida. La siguiente capa requerirá filtros con una profundidad de 32 para procesar toda la información de entrada, lo que supone un aumento significativo del número de pesos por neurona. Si aplicamos otra capa de convolución tras esta, el número de parámetros necesarios aumentará aún más, así como el número de operaciones, lo que aumentará la posibilidad de sobreajuste y también aumentará el coste computacional necesario para procesar la imagen. Todo esto puede resultar en una disminución del rendimiento y la eficiencia de la red neuronal. [56].

Para evitar esto, se utilizan técnicas como la reducción de tamaño de los mapas de características a través de capas de *pooling*. Esta técnica busca reducir el tamaño de del mapa (anchura y altura) sin afectar en su profundidad y preservando las características más importantes de estos, reduciendo así el número de operaciones necesarias para procesar una imagen y por tanto el coste computacional.

Por último, tras entender la necesidad de aplicar *reducción,* detallaremos la operación a realizar. Se busca simplificar sin perder información relevante, por lo que se recorrerán los mapas generados por áreas de 2x2, realizando la media (*Average Pooling*) o tomando el valor máximo de los cuatro (*Max Pooling*), este último método es el más usado ya que además hace de filtro de ruido [53]. En la **figura 3-25** podemos ver un ejemplo.

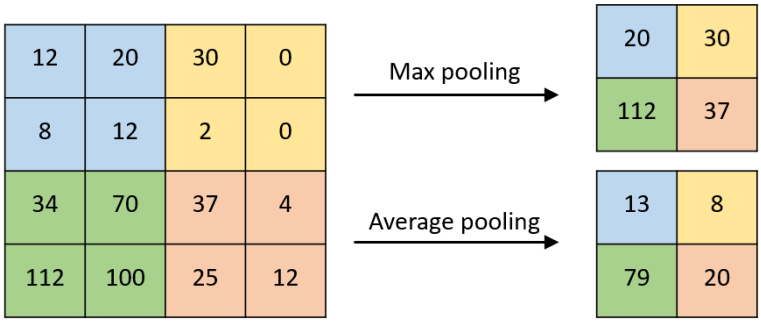


Figura 3-25. Aplicación de una reducción a una imagen de entrada.

Por lo tanto, aplicar una reducción después de una convolución es un paso importante para disminuir la complejidad de la red neuronal y permitir una mayor eficiencia en el procesamiento de la información. Al reducir el tamaño de los mapas de características, se pueden aplicar más convoluciones en capas posteriores de la red, lo que permite extraer patrones más complejos y mejorar el rendimiento de la red. De esta forma tras una convolución siempre se aplicará una reducción. [54]

3.3.1.3. Capa clasificadora

Es la última sección de la red, está formada por unas capas de neuronas totalmente conectadas como las vistas en el **apartado 3.2**, donde se introdujeron las redes neuronales.

La parte de la red en cuestión recibirá un vector con todas las características extraídas de la imagen ya que cada neurona procesará una característica. Por lo cual se deberá aplicar una transformación para de una matriz tridimensional (conjunto de todos los mapas generados), se obtenga un vector. Este proceso de conocer como *aplanado* o *flattening* una forma de llevarlo a cabo es concatenar todos los mapas de características en un único vector como se muestra en la **figura 3-26**. [57]

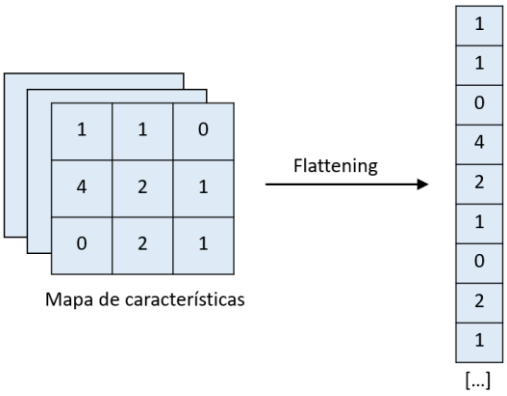


Figura 3-26. Ejemplo de aplicación de la operación de flattening a un mapa de características.

Gracias a la transformación previa, esta sección de la red neuronal, la cual funciona como un clasificador, será capaz de asignar una probabilidad a cada una de las clases posibles. Para lograr esto, se utiliza la función de activación *softmax* (similar a la *sigmoide* vista en el **apartado 3.2.2**, más información [58]), la cual permite convertir las salidas de la red en una distribución de probabilidad sobre las clases. De esta manera, se puede determinar la probabilidad de que la imagen pertenezca a cada una de las clases posibles. La red tendrá una neurona de salida para cada clase posible como se puede ver en la **figura 3-22**.

Podemos resumir esta sección destacando que las *redes neuronales convolucionales* (CNNs) son un tipo de arquitectura de red neuronal especialmente diseñada para el procesamiento de imágenes. Las CNNs utilizan capas de convolución para extraer características de las imágenes y reducir su complejidad, seguido de capas de *pooling* que simplifican aún más la información. Estas características se combinan en capas posteriores de la red para obtener patrones más complejos y finalmente se utiliza una capa de clasificación para asignar una etiqueta o clase a la imagen.

3.3.2. Detección de objetos

La detección de objetos es una disciplina clave en la visión por computadora y será la disciplina en la que centraremos este proyecto. La detección de objetos consiste en reconocer y localizar un objeto perteneciente a una clase en una imagen. Podemos ver en la **figura 3-27** un ejemplo.

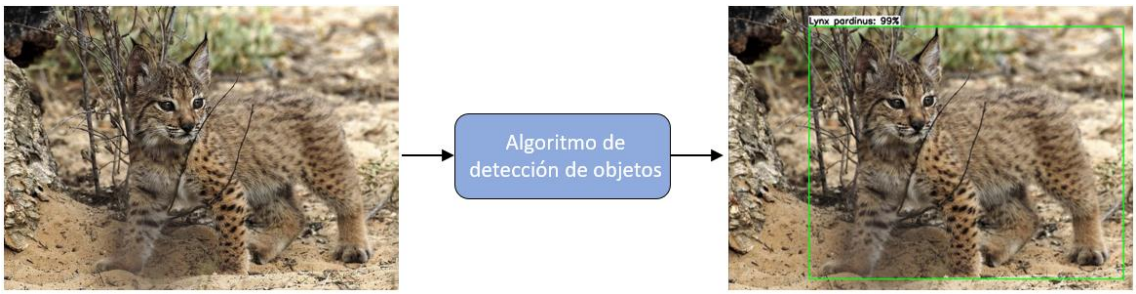


Figura 3-27. Concepto de detección de objetos.

A continuación, listaremos algunos de los métodos principales de detección de objetos para entender el problema y su relación con las redes neuronales convolucionales. Posteriormente nos centraremos en SSD para poder profundizar en este y entender el modelo que se usará en este proyecto.

3.3.2.1. R-CNN: Regions with CNN features

R-CNN método divide la imagen de entrada en diferentes regiones, para aplicar en cada una de ellas un algoritmo de reconocimiento visual, para poder así identificar los diferentes objetos presentes en la imagen. Cuenta con tres partes principales, las podemos observar en la **figura 3-28**.

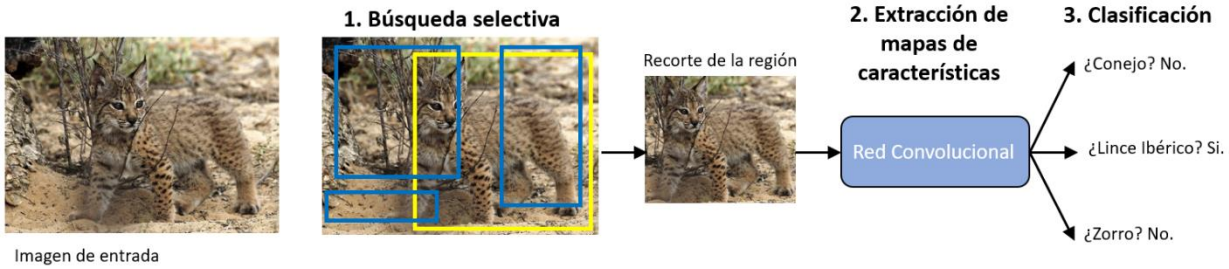


Figura 3-28. Arquitectura de R-CNN.

El *algoritmo de búsqueda selectiva* toma 2000 regiones candidatas (desarrollado en [59]), estas regiones son procesadas por la CNN, la cual funciona como un extractor de características. Estas características son clasificadas por una *máquina de vectores de soporte* (*Support Vector Machine, SVM* [60]), determinando la presencia de un objeto en la región propuesta. SVM es una alternativa más compleja a la explicada en el apartado anterior, la cual hacia uso de la función de activación softmax. Obtendremos como salida un vector que contendrá, la clase del objeto reconocido en cada región, así como su posición (determinada por cuatro puntos que forman *bounding box* es decir una caja que contiene el objeto).

Este método a pesar de ser funcional cuenta con dos grandes problemas que no lo hacen una buena solución. En primer lugar, clasificar 2000 regiones no es óptimo (tarda 47 segundos en procesar una imagen) tanto para su uso en aplicaciones que necesiten una respuesta rápida, así como para su entrenamiento. En segundo lugar, continuando con problemas relacionados con el entrenamiento, el *algoritmo de búsqueda selectiva* es un algoritmo fijo, es decir no tiene la capacidad de aprender, por lo cual no se puede entrenar ya que podría llevar a la generación de regiones erróneas como candidatos, influyendo en un mal aprendizaje de la parte del modelo que si usa inteligencia artificial.

3.3.2.2. Fast R-CNN

Este algoritmo es una versión mejorada de R-CNN, que busca solucionar las desventajas con la que este contaba.

Fast R-CNN [61] aplica directamente una capa convolucional para obtener un mapa de características. A partir de este mapa, se utiliza un *algoritmo de búsqueda selectiva* para proponer regiones de interés. Estas regiones se normalizan a un tamaño fijo utilizando la capa *RoI pooling* (*Region of Interest pooling layer*), aplicando un *Max Pooling* para cada región propuesta. Luego, se aplanan para ser introducidas en una capa clasificadora completamente conectada, que utiliza la función de activación softmax para asignar una probabilidad de pertenecer a una clase a cada región propuesta

Esta versión es mucho más rápida ya que no necesita analizar 2000 regiones propuesta de la imagen de entrada, aplicando una sola vez la capa de convolución para generar un mapa de características, que le servirá para aplicar un *algoritmo de búsqueda selectiva.* Por lo tanto, mejora el problema del tiempo, pero no logra solucionar los problemas que presentaba en el entrenamiento ya que sigue usando un *algoritmo de búsqueda selectiva* que no puede aprender.

3.3.2.3. Fast R-CNN

Con el fin de solventar el problema que generaba el uso de un *algoritmo de búsqueda selectiva*, se diseñó faster R-CNN [62].

Este modelo se basa en la estructura de Fast R-CNN, que utiliza una capa convolucional para extraer características de la imagen y una capa totalmente conectada para la clasificación. Sin embargo, en lugar de utilizar métodos tradicionales de selección de regiones de interés, Faster R-CNN utiliza una red neuronal llamada "*Region Proposal Network*" para generar propuestas de regiones de interés de manera más eficiente y precisa. Una vez generadas las propuestas, se aplica una capa de agrupación RoI para extraer las características de cada región y, posteriormente, se aplana para alimentar una etapa clasificadora totalmente conectada. Esta forma se resuelven los problemas que presentaban las versiones anteriores.

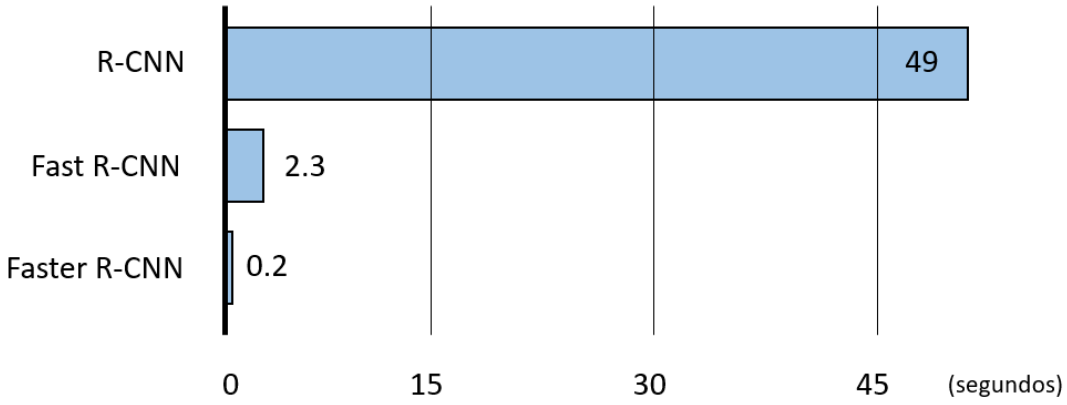


Figura 3-29. Comparación entre las diferentes versiones de R-CNN

3.3.2.4. YOLO (You Only Look Once)

Como hemos podido ver las diferentes versiones de R-CNN se basan en una arquitectura de dos etapas, donde la primera se encarga de generar propuestas de regiones de interés (busca los posibles bounding box) mientras que la segunda las clasifica.

A diferencia de R-CNN, YOLO procesa la imagen completa de una sola vez y es capaz de predecir simultáneamente las clases y los bounding boxes de todos los objetos en la imagen, sin necesidad de trabajar con regiones de interés. Para lograrlo, YOLO divide la imagen en una cuadrícula regular de **NxN** celdas, cada una dedicada a la predicción de un objeto único. Si el centro de un objeto se encuentra dentro de una celda, se considera que el objeto está en esa celda, aunque pueda ocupar más de una. Sin embargo, esta estrategia limita la capacidad de detección de YOLO, ya que solo puede detectar hasta N*N objetos. En caso de que varios objetos tengan su centro en la misma celda, solo se podrá detectar uno de ellos, lo que representa un problema.

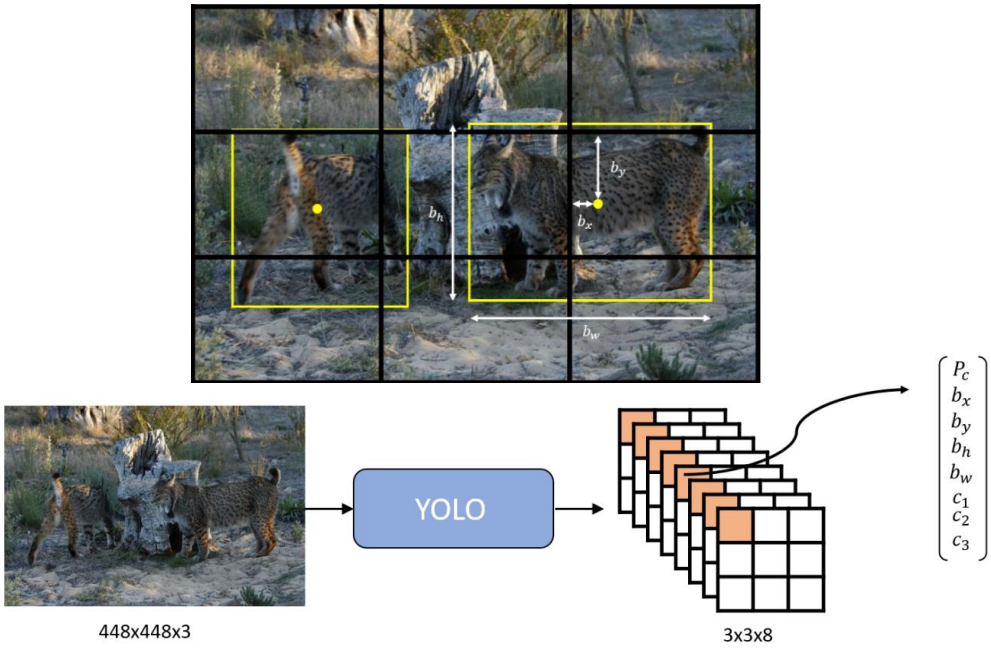


Figura 3-30. Esquema de funcionamiento de YOLO.

YOLO tomará una imagen como entrada, la cual es dividida en una cuadrícula de celdas. Posteriormente, aplicará una red convolucional buscando detectar un objeto como máximo por celda. Como salida tendremos un volumen de NxNx8, es decir el tamaño de la cuadricula aplicada y una profundidad de 8. De esta forma cada columna de esta matriz tridimensional representa un vector en el que estará codificada la información de detección de cada una de las celdas. De esta forma, en cada vector encontramos la probabilidad de que haya un objeto en una celda (P_c), las coordenadas del centro del bounding box (b_x, b_y), así como su ancho y alto (b_h, b_w), por último, tenemos las probabilidades de pertenecer a una clase concreta (c_1, c_2, c_3). [63]

La profundidad del volumen de salida puede variar dependiendo del número de objetos que queramos clasificar (C), así como el número de bounding box que queramos predecir (B).

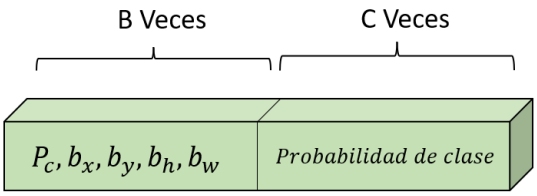


Figura 3-31. Estructura del tipo de dato que YOLO da como salida.

La capacidad de YOLO para procesar una imagen completa de una sola vez le permite detectar objetos en tiempo real de manera rápida y eficiente. Al hacerlo, YOLO puede procesar la imagen "en una sola mirada", lo que lo convierte en un algoritmo ideal para aplicaciones en tiempo real.

YOLO a pesar de sus ventajas cuenta con algún inconveniente, entre ellos se encuentra la dificultad de detectar objetos pequeños, ya que estos pueden situarse en la misma celda que otro objeto, no pudiendo así ser detectados por la red. Una solución puede ser aumentar el número de celdas, pero esta modificación tiene sus limitaciones.

Actualmente existen diferentes versiones de YOLO que mejoran su rendimiento la más reciente es YOLO v8 podemos encontrar más información en [64].

3.3.2.5. SSD (Single Shot Multibox Detector)

SSD [65] al igual que YOLO es un algoritmo de una única etapa, lo que significa que no necesita de una etapa adicional de generación de propuestas de región de interés (RoI) como hacen otros algoritmos de detección que ya vistos, como R-CNN. Esto hace que SSD sea más simple y rápido que estos otros algoritmos. Siendo al igual que YOLO útil para aplicaciones en tiempo real.

SSD es una red convolucional, cuenta con tres partes especializadas en cada uno de los procesos de detección.

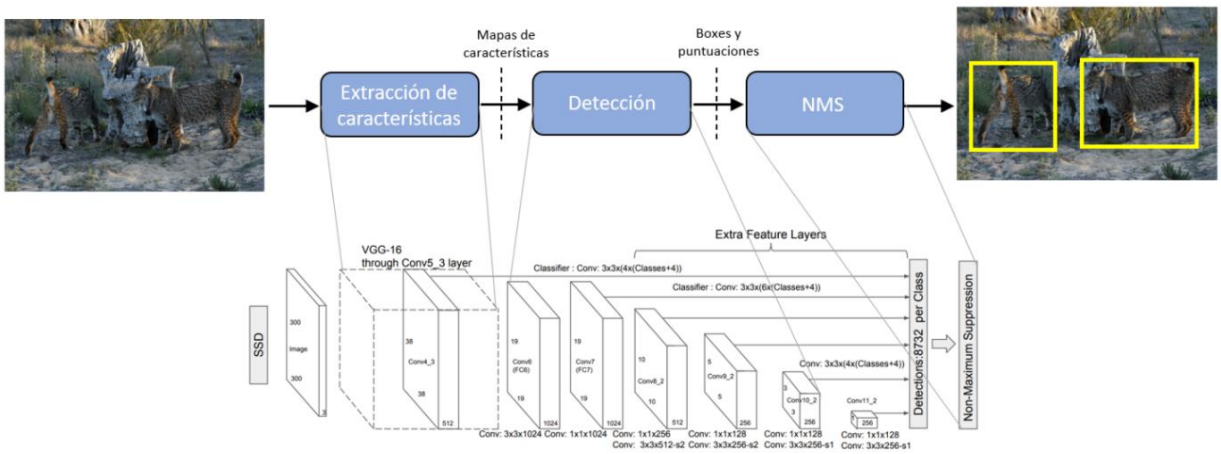


Figura 3-32. Esquema de la arquitectura de SSD.

La SSD usa una red convolucional, como es VGG, para **extraer características,** y así poder generar diferentes mapas de características de diferentes tamaños que serán usados en las siguientes etapas.

De esta forma, el extractor de características, basado en VGG, podrá extraer diferentes mapas de características de distintos tamaños gracias a las diversas capas utilizadas, generando como salida una pirámide de mapas donde los situados en la parte más alta serán los que contengan la información más general. Esto permitirá obtener una variedad de características a diferentes escalas, lo que permitirá a SSD detectar objetos de distintos tamaños con mayor precisión, algo en lo que YOLO solía tener dificultades.

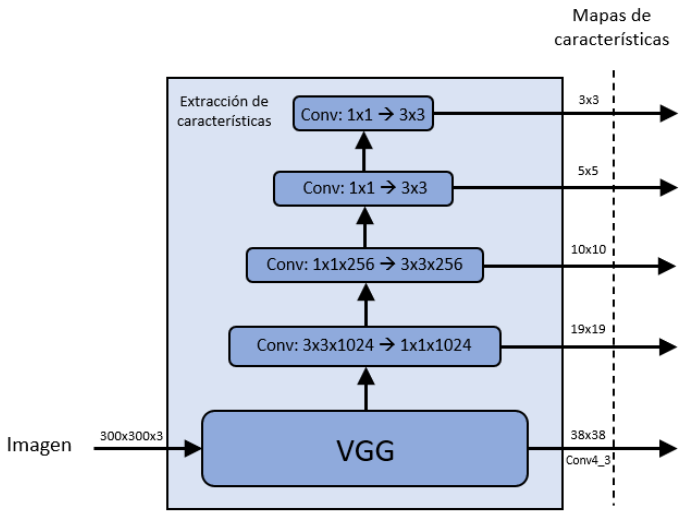


Figura 3-31. Esquema detallado de la etapa de extracción de características.

Para *localizar* y *reconocer* los diferentes objetos SSD usa los mapas de características generados para generar un conjunto de bounding boxes. Estas cajas de anclaje (anchor boxes) o cajas delimitadoras (bounding boxes), son de diferentes tamaños y formas se superpondrán en la imagen para cubrir todas las posibles ubicaciones y tamaños de los objetos.

Esta etapa dará como salida una probabilidad de que el objeto que la caja delimitadora está encapsulando pertenezca a una clase dada, además unos valores $(\beta_x, \beta_y, \beta_w, \beta_h)$, que servirán para ajustar la caja y así obtener una localización más precisa.

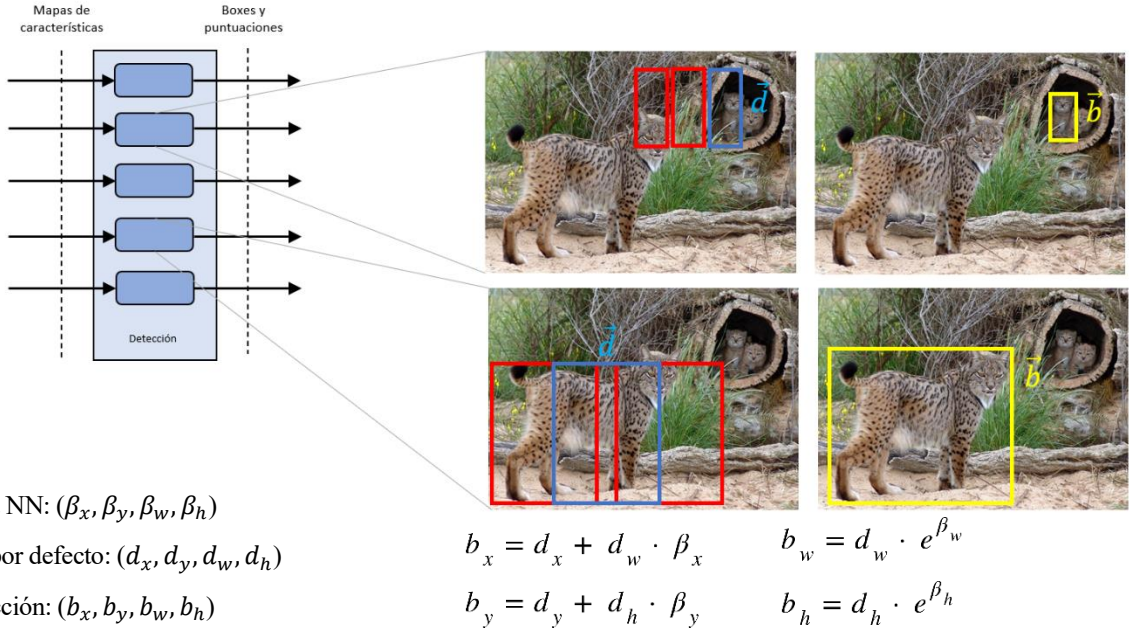


Figura 3-33. Esquema detallado de la etapa de detección.

Se seleccionarán las cajas que tengan una puntuación alta para alguna de las clases disponibles y se le aplicará la corrección indicada en la **figura 3-33** para obtener una predicción más precisa.

Por último, puede que para un objeto se obtengan más de una predicción, generándose varias cajas superpuestas. Para solucionar este problema se aplicará un algoritmo llamado *Non Max Suppression (NMS),* para seleccionar la mejor predicción.

Para entender la NMS debemos conocer un parámetro esencial para la medida de la relación entre diferentes bounding boxes, este es el *Intersection over Union (IoU),* consiste en el cociente del área que forma la intersección de dos cajas entre la unión de estas.

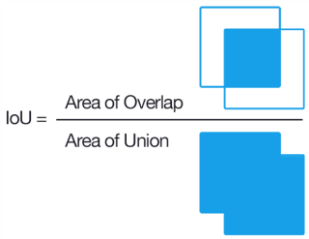


Figura 3-34. Operación, Intersection over Union (IoU).

Esta ratio también se usa para medir la precisión de un modelo durante el proceso de inferencia. Se compara la superposición entre las cajas generadas por el modelo y las cajas de referencia, en un conjunto de datos de validación. Lo usaremos para medir la precisión de nuestro modelo en capitulo posteriores.

Pero en el algoritmo de *NMS* se usará para determinar si ciertos bounding boxes superpuestos predicen la posición de un mismo objeto. Como se puede ver en la **figura 3-35** de todas las cajas con un IoU alto se tomará como valida aquella con una mejor puntuación, obteniendo así la predicción más exacta para cada objeto.

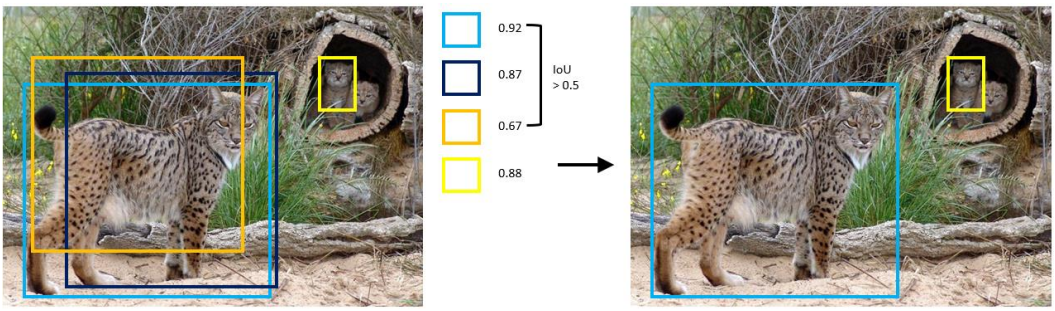


Figura 3-35. Aplicación del algoritmo de NMS.

Se pueden introducir ciertas mejoras para obtener un mejor rendimiento aumentando la velocidad y precisión del modelo. La mejora más inminente es usar un mejor extractor de características, ya que *VGG* era uno de los más potentes cuando se publicó SSD, allá por 2015, pero actualmente existen alternativas más optimizadas como ResNet [66], o Mobilnet [67], pensada para dispositivos móviles. Además, se puede modificar la estructura del extractor de características implementando un *Feature Pyramid Network (FPN)* que tome la información más general de las capas superiores de la pirámide y la transmita a las capas más inferiores mejorando así la calidad de la información de los mapas, lo cual se traduce en una mayor precisión del modelo.

Tabla 3-1. Comparación de velocidad y precisión de detectores entrenado en PASCAL VOC 2007 [63][65]

Detectores	mAP	FPS
Fast R-CNN	70.0	0.5
Faster R-CNN VGG-16	73.2	7
Faster R-CNN ZF	62.1	18
YOLO	63.4	45
SSD	74.3	59

En la **tabla 3-1,** podemos ver una comparación en velocidad, tomando la media de fotogramas por segundo y precisión, usando la métrica mAP (mean Average Precision), de los diferentes modelos de detección vistos.

Para este proyecto usaremos una versión de SSD, la cual implementa MobileNet v2 [68], así como una versión ligera del Feature Pyramid Network, FPNlite [69]. Podemos ver a continuación en la **figura 3-36** un esquema de la estructura del modelo.

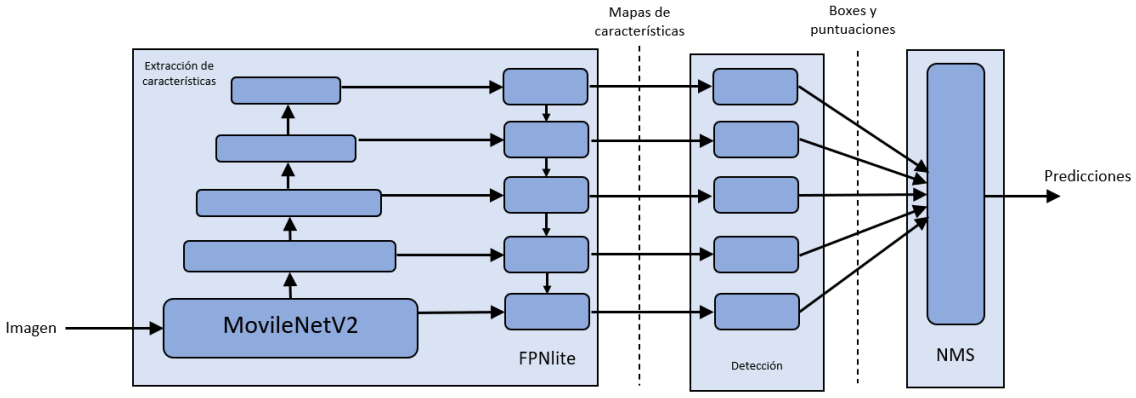


Figura 3-36. Esquema de SSD Mobilenet V2 FPNLite.

Además de los nombrados existen gran variedad de modelos para la detección de objetos, este es un campo amplio, que se encuentra en constante desarrollo y crecimiento.

Una de las últimas novedades incorporadas son los Transformers [70], estos modelos, presentados en 2017, se han vuelto muy populares en el procesamiento del lenguaje natural, así como en la visión por computadora. En relación con la detección de objetos los Transformers se utilizan para mejorar la eficiencia, basándose en el concepto de atención, permiten a los modelos enfocarse en partes de la entrada durante el entrenamiento y predicción. Existen diferentes modelos como DETR (DEtection TRansformer) [71], el cual usa un Transformer para la detección directa, YOLOv4 [72], utiliza una combinación de CNN y Transformer.

Con esta revisión de los diferentes modelos para la detección de objetos, hemos sentado las bases teóricas necesarias para contextualizar nuestro proyecto y brindar una mejor comprensión de este. En los capítulos posteriores, nos enfocaremos en la implementación del sistema, así como en la validación de este.

4 IMPLEMENTACIÓN DEL SISTEMA

En este capítulo abordaremos implementación del sistema que se busca desarrollar. En particular, nos enfocaremos en el hardware y el software del dispositivo, así como en los procedimientos utilizados para recolectar datos, entrenar un modelo e implementarlo en una Raspberry.

El objetivo principal de este proyecto es añadir una capa adicional a los dispositivos de fototrampeo para detectar diferentes tipos de animales, entre ellos el lince ibérico, el cual es el objetivo de estudio y así optimizar y automatizar las acciones de seguimiento poblacional de esta especie en peligro de extinción.

El sistema que se desarrollará debe ser capaz de identificar y clasificar las especies animales que se capturan en las fotografías del fototrampeo, utilizando técnicas de procesamiento de imágenes y aprendizaje automático. Con esto, se busca aumentar la eficiencia en el proceso de recopilación de datos y, por ende, mejorar la toma de decisiones en la gestión de la conservación de la fauna.

4.1 Arquitectura del sistema

Nuestro sistema buscará simular un dispositivo de fototrampeo, para ello se utilizará la Raspberry Pi como unidad de procesamiento y se desarrollará un script en Python para controlar la captura de imágenes mediante una pequeña cámara (se detalla en el **apartado 4.1.2.2**.).

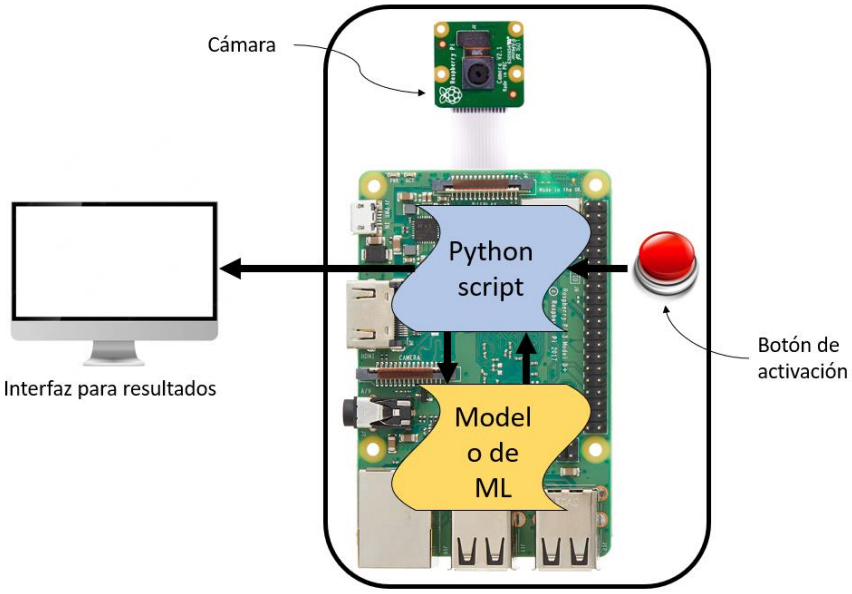


Figura 4-1. Esquema de la arquitectura del sistema de detección.

Como se muestra en la **figura 4-1**, se simulará el sensor de movimiento de las cámaras trampa utilizando un botón que activará la captura de imágenes.

Después de capturar las imágenes, estas serán procesados por el modelo de detección de especies animales que se ha entrenado previamente. Los resultados obtenidos por el modelo serán mostrados a través de una interfaz de usuario, lo que permitirá al usuario ver las especies animales detectadas y clasificadas por el sistema.

A continuación, profundizaremos en los componentes que forman el sistema desde la parte hardware hasta el software que el sistema necesita para realizar sus funciones.

4.1.1. Hardware

El hardware utilizado en el sistema de cámara trampa incluye una cámara, una Raspberry Pi y un botón a modo de sensor de movimiento para activar la captura de imágenes. La cámara puede ser una cámara web de bajo costo o una cámara especializada para fototrampeo. La Raspberry Pi actúa como la unidad central de procesamiento del sistema y está conectada a la cámara y al botón.

4.1.1.1. Raspberry Pi 3B

Las Raspberry Pi son un tipo de ordenador de placa única diseñado originalmente para la enseñanza de informática en las escuelas. Sin embargo, debido a su económico precio y compacto diseño, su uso se ha popularizado en muchos otros ámbitos, incluyendo la robótica y la visión por computadora. Aunque actualmente existen varias versiones de Raspberry Pi, para nuestro trabajo haremos uso de la Raspberry Pi 3 B.

La raspberry pi 3 B es una versión mejorada de la raspberry pi 2, cuenta con una procesador ARM Cortex-A53 de cuatro núcleos a 1,2 GHz, 1 GB de RAM, puerto Ethernet, Wi-Fi incorporado y Bluetooth 4.1.

En el ámbito de la inteligencia artificial y el machine learning, existen diversas opciones de microprocesadores que se han destacado por su capacidad de procesamiento acelerado. Dos de las opciones más populares son el NVIDIA Jetson Nano y Google Coral. Ambas plataformas cuentan con hardware específico para acelerar el proceso de inferencia de modelos de aprendizaje automático, lo que las hace especialmente útiles para proyectos que requieren un alto rendimiento en este ámbito.

Tabla 4-1. Comparación entre diferentes plataformas para aplicaciones de Inteligencia Artificial.

	CPU	GPU	Coprocesador	Especializado en IA	Precio
Raspberry Pi 3B	Quad-core ARM Cortex-A53 de 64 bits	Integrada Broadcom VideoCore IV	No	No	40€
NVIDIA Jetson Nano	Quad-core ARM Cortex-A57 de 64 bits	128-core Nvidia Maxwell	No	Si	200 €
Google Coral	Quad-core ARM Cortex-A53 de 64 bits	integrada, GC7000 Lite Graphics.	Google Edge TPU	Si	150€

Podemos ver en la **tabla 4-1** una comparación entre las diferentes especificaciones de las plataformas comentadas. tanto NVIDIA Jetson Nano como Google Coral son dos opciones populares en el mundo de la inteligencia artificial y el machine learning, ya que cuentan con procesadores específicos que les permiten acelerar el proceso de inferencia de modelos de aprendizaje automático, como la GPU Maxwell en la NVIDIA Jetson Nano o el coprocesador Edge TPU en el Google Coral, diseñado específicamente para la aceleración de inferencia de modelos de machine learning. Estos dispositivos son una buena opción si se busca un mejor rendimiento en la aplicación que se desea implementar.

Sin embargo, para nuestro proyecto hemos optado por la Raspberry Pi 3B debido a su versatilidad y bajo costo. A pesar de que no está diseñada específicamente para ser usada en aplicaciones de machine learning, ofrece un rendimiento aceptable para nuestro propósito. Además, cuenta con una gran comunidad de desarrolladores y una amplia variedad de recursos y documentación disponible en línea, lo cual resulta de gran ayuda en el proceso de desarrollo y solución de problemas. Por estas razones, consideramos que la Raspberry Pi 3 B es la mejor opción para nuestro proyecto. Se utilizará con el sistema operativo Raspberry Pi OS, configurado de forma estándar y cuya instalación puede seguirse en el **anexo A**.

4.1.2. Software

El software utilizado en el proyecto incluye tanto el framework utilizado para la implementación del modelo de detección como el código desarrollado para la captura y procesamiento de imágenes.

La combinación del framework y el código desarrollado permite una implementación completa del sistema de detección, desde la captura de imágenes hasta la inferencia y la generación de resultados. Ambos componentes son esenciales para el funcionamiento del sistema y contribuyen al éxito del proyecto.

4.1.2.1. Tensorflow y tensorflow lite

Tensorflow es una librería de código abierto para aprendizaje automático desarrollada por Google, que se utiliza para crear y entrenar modelos de aprendizaje profundo y aplicaciones de inteligencia artificial en una amplia gama de plataformas, desde servidores hasta dispositivos móviles y sistemas integrados. Es una de las bibliotecas de aprendizaje automático más populares y ampliamente utilizadas en la actualidad.

Puede ejecutarse en gran variedad de hardware, desde CPU hasta GPU y TPU (Tensor Processing Units). Lo permite aprovechar al máximo los recursos disponibles en diferentes tipos de sistemas. Además, es compatible con una gran cantidad de lenguajes de programación incluyendo, Python, C++, Java, y JavaScript. Pero su código por temas de eficiencia está programado principalmente en C++ altamente optimizado y en CUDA (lenguaje de programación de las GPUs de Nvidia) [73]

Para este proyecto usaremos una versión de tensorflow diseñada para dispositivos móviles y sistemas integrados, como es tensorflow lite [74]. Está optimizado para ejecutarse en dispositivos con recursos limitados, como procesadores de baja potencia y memoria limitada. Se detalla su Instalación en el **anexo B**.

4.1.2.2. Código principal, captura y procesamiento

A la hora de programar el funcionamiento del sistema de fototrampeo, es fundamental considerar las limitaciones asociadas a la implementación de un modelo de detección de objetos en un sistema con recursos limitados, como la Raspberry Pi. Con este fin, se ha desarrollado un programa que separa la captura y el procesamiento de imágenes en dos hilos concurrentes. De esta manera, los retrasos que puedan surgir en la parte de procesamiento y detección no afectarán la captura de información del sistema, lo que permite mantener un flujo continuo de captura de imágenes. Esta separación de tareas garantiza un rendimiento eficiente y un aprovechamiento óptimo de los recursos disponibles en la Raspberry Pi.

Para comunicar los hilos haremos uso de una cola de mensajes por la cual se enviarán las imágenes tomadas por el hilo de captura.

Finalmente, una vez que la imagen ha sido procesada y se ha extraído toda la información relevante, se procede a cargarla en una carpeta compartida en Dropbox. Desde esta ubicación, el programa de la interfaz puede acceder a los datos extraídos y mostrarlos al usuario de manera intuitiva y conveniente. Esta integración con Dropbox facilita la gestión y el acceso a los resultados del procesamiento de imágenes, permitiendo una experiencia fluida y eficiente para el usuario.

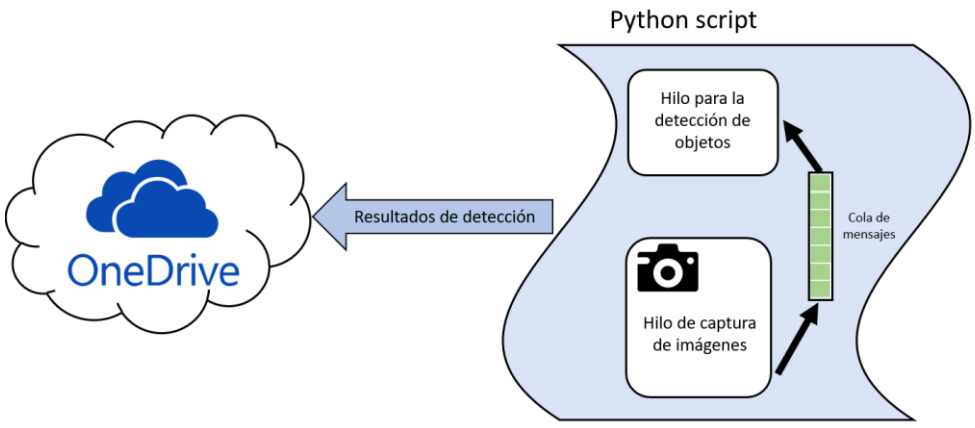


Figura 4-2. Esquema del funcionamiento del programa de captura y procesamiento.

Se detalla más la ejecución del programa a continuación.

1. **INICIO**

   Hilo captura, permanece bloqueado a la espera de recibir una señal del sensor, es decir se pulse la tecla enter para nuestro simulador.

   Hilo de procesamiento, permanece bloqueado a la espera de recibir una imagen por la cola.

2. **PULSACIÓN TECLA ENTER**

   Hilo captura, toma 10 imágenes cada 0.5 segundos.

   Hilo de procesamiento, procesa todas las imágenes que se encuentran en la cola de entrada. Para cada imagen, se realiza la detección de objetos y se marcan en la imagen original indicando su clase y el porcentaje de probabilidad asociado. Además, se genera un archivo de texto específico para cada imagen, siguiendo el siguiente formato: [clase, probabilidad, coordenadas de la caja delimitadora].

   Por último, sube el archivo TXT y JPG a la carpeta de Dropbox.

3. **DEJA DE HABER PULSACIONES**

   En este caso el programa vuelve al estado de inicio y permanece en el hasta que se pulse la tecla.

Código disponible en el **anexo C**.

4.1.2.3. Interfaz de usuario

Con el objetivo de mejorar la experiencia del usuario y facilitar la interpretación de los resultados, se ha implementado una interfaz gráfica utilizando la biblioteca PyQt de Python. Esta interfaz permite al usuario acceder a las imágenes almacenadas en la carpeta compartida, así como visualizar las detecciones realizadas en cada imagen. Además, se muestra un recuento local de las detecciones específicas de la imagen en pantalla, así como un recuento global que muestra el total de detecciones realizadas en todas las imágenes procesadas.

El código se encuentra en el **anexo D**.

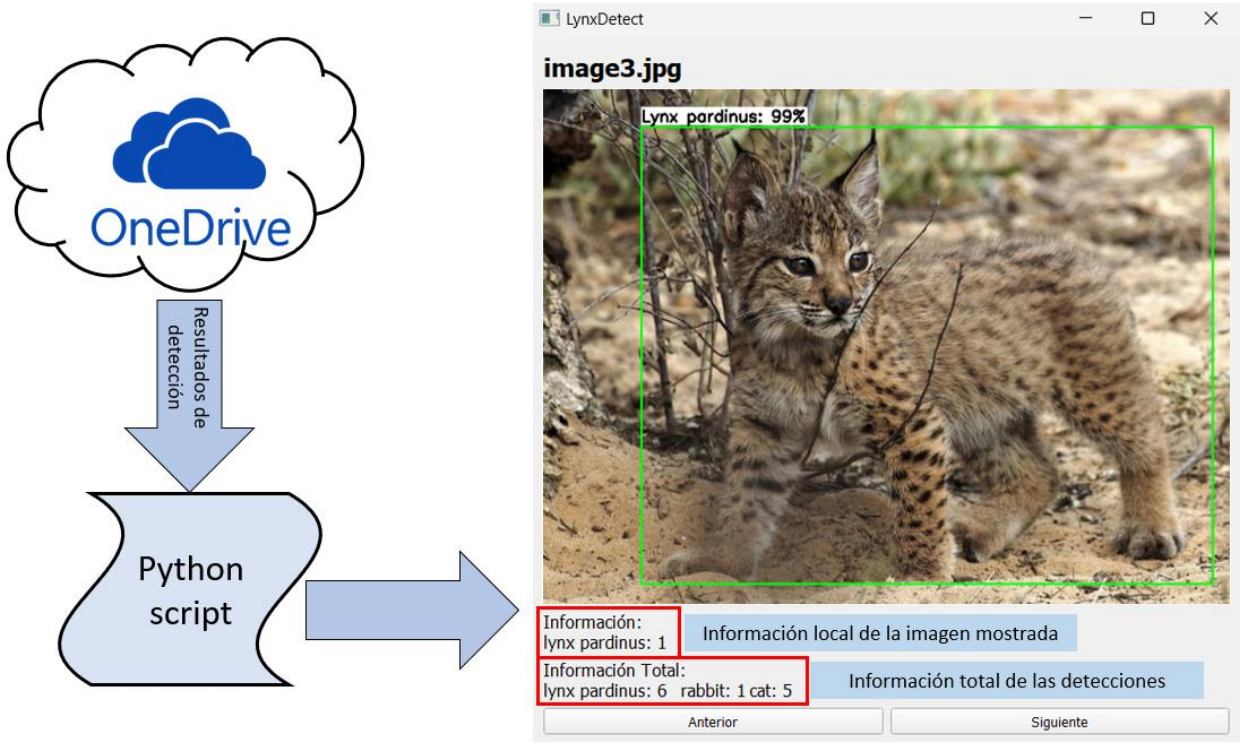


Figura 4-3. Esquema del funcionamiento del programa de la interfaz de usuario.

4.2 Recopilación y etiquetado de datos

La preparación de conjuntos de datos de alta calidad es crucial en aplicaciones de detección de objetos utilizando técnicas de aprendizaje automático. Es fundamental construir un conjunto de datos sólido para garantizar la fiabilidad de nuestro sistema.

En nuestro caso, buscamos desarrollar un sistema capaz de detectar al lince ibérico, así como a otras especies relacionadas, como el conejo (su presa principal) y el zorro (su competidor por alimentos). Esto nos permitirá obtener información valiosa sobre el entorno y las condiciones necesarias para la supervivencia del lince ibérico.

Además, nos interesa poder distinguir entre el gato común y el lince, ya que ambos pertenecen a la familia de los felinos y comparten características similares. También queremos capacitar al sistema para detectar la presencia de humanos, ya que la actividad humana en el hábitat del lince puede representar un peligro para la especie debido a la caza furtiva.

4.2.1 Construcción del conjunto de datos

Para recopilar las imágenes necesarias, hemos utilizado conjuntos de datos previamente preparados para todas las clases, excepto para el lince. Ya que desafortunadamente, no encontramos ningún conjunto de datos que se ajustara a nuestras especificaciones específicas para esta especie. Por lo tanto, hemos llevado a cabo un estudio con el objetivo de recopilar la mayor cantidad posible de datos relevantes.

Durante nuestro estudio, hemos utilizado diversos métodos para obtener imágenes relevantes del lince. Para ello, hemos explorado varias bases de datos científicas, como *gbif.org*, además de recopilar una base genérica de datos a través de Google Fotos y buscar en diversas páginas web. Sin embargo, la mayor parte del conjunto de datos la hemos obtenido a través de videos capturados con cámaras trampa que hemos encontrado en Internet. Utilizando un script de Python llamado **cap_vid.py** (disponible en el repositorio del proyecto), hemos extraído diferentes fotogramas de estos videos para proporcionar un contexto a la clase que deseamos detectar y lograr resultados más precisos.

Este enfoque nos ha permitido recopilar un conjunto de datos amplio y diverso, que abarca diferentes situaciones en las que se puede encontrar el lince. Al utilizar videos capturados con cámaras trampa, hemos logrado obtener imágenes del lince en su hábitat natural y en condiciones reales, lo que enriquece la calidad y representatividad del conjunto de datos.

Figura 4-4. Ejemplo de imágenes obtenidas con el script de captura.

Dado que muchas cámaras trampa operan en condiciones de baja luminosidad, hemos considerado fundamental incluir un porcentaje significativo de imágenes capturadas en estas condiciones en nuestro conjunto de datos.

Cabe destacar que el lince ibérico es un felino principalmente activo durante el crepúsculo y la noche, por lo que es crucial tener imágenes que reflejen su comportamiento en condiciones de poca luz. Para así poder ajustar el comportamiento del modelo a los hábitos de la especie.

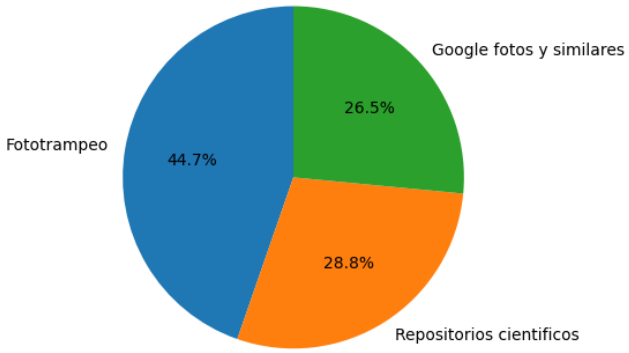


Figura 4-5. Distribución porcentual de la procedencia de las imágenes que forman el dataset del lince.

Después de recopilar todos los datos necesarios para entrenar un modelo, hemos procedido a realizar una partición en tres conjuntos de datos: entrenamiento (train), validación (validation) y prueba (test). Esta división nos permite entrenar la red neuronal y controlar el error de validación durante el proceso de entrenamiento para evitar el sobreajuste del modelo. Además, el conjunto de prueba se utiliza una vez que el modelo ha sido entrenado para evaluar su precisión y obtener medidas de su rendimiento.

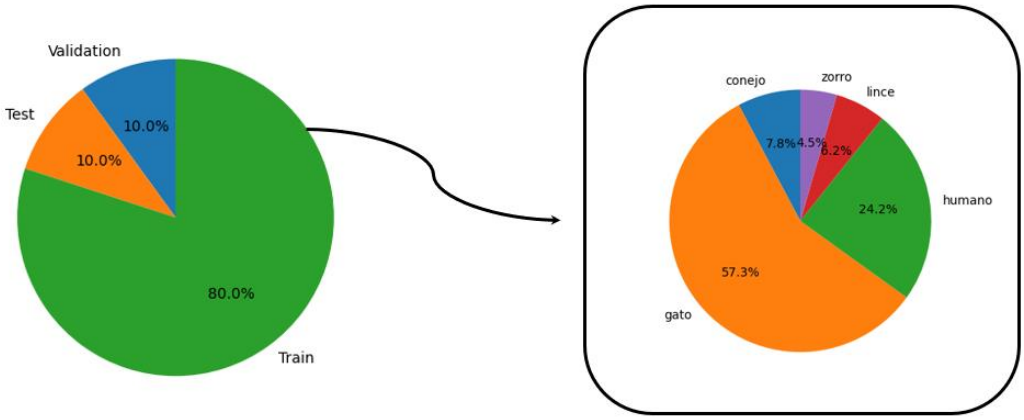


Figura 4-6. Distribución porcentual de las clases que forman el dataset.

En la **figura 4-6** se muestra la distribución de los datos en cada uno de los conjuntos, donde se ha garantizado que cada conjunto tenga una representación porcentual equitativa de datos para cada clase. Esto se ha realizado con el objetivo de obtener un dataset lo más homogéneo posible, asegurando que todas las clases estén representadas de manera similar en los conjuntos de entrenamiento, validación y prueba. Esta estrategia promueve un entrenamiento más equilibrado y confiable del modelo, evitando cualquier sesgo o desequilibrio en la distribución de las clases en los conjuntos de datos.

Para llevar a cabo esta gestión de datos de forma más eficiente hemos hecho uso de diferentes scripts como **crear_subgrupos.py** (disponible en el repositorio del proyecto).

Nota: Cabe destacar que el conjunto de datos de humano cuenta a su vez con dos subclases, siento estas *person* y *person-like* representando cada una el 54.47 % y 45.5 % del conjunto de las etiquetas presentes en el dataset. Esta diferenciación permitirá al modelo diferencias objetos que parecen una persona de una persona real.

4.2.2 Etiquetado de datos

En el proceso de construcción y entrenamiento de modelos de aprendizaje automático, el etiquetado de datos desempeña un papel fundamental. El etiquetado de datos se refiere a la tarea de asignar etiquetas o categorías a los datos de entrenamiento, lo que permite al modelo aprender y generalizar patrones a partir de esos datos.

En este apartado, examinaremos en detalle los procedimientos utilizados durante el etiquetado de nuestro conjunto de datos, así como las herramientas empleadas tanto para etiquetar como para gestionar los archivos generados.

En nuestro proceso de etiquetado, hemos utilizado **LabelImg**, una herramienta de código abierto ampliamente reconocida en la comunidad. LabelImg se destaca por su interfaz gráfica de usuario intuitiva, lo que facilita el proceso de anotación de imágenes de manera interactiva y eficiente.

Una de las ventajas clave de LabelImg es su amplia compatibilidad con varios formatos de archivos, lo que nos brinda flexibilidad en la integración con diferentes frameworks y bibliotecas de aprendizaje automático. Para nuestro proyecto, hemos optado por utilizar el formato **PASCAL VOC**, que es ampliamente utilizado en el campo de la visión por computadora y se adapta a nuestras necesidades específicas.

Gracias a LabelImg, hemos sido capaces de delimitar y etiquetar objetos de interés en nuestras imágenes mediante cuadros delimitadores. Esto nos permite definir las regiones que contienen los objetos y asignarles etiquetas relevantes. Además, hemos aprovechado las funcionalidades adicionales de la herramienta, como los atajos de teclado y la visualización de anotaciones superpuestas, para agilizar el proceso de etiquetado.

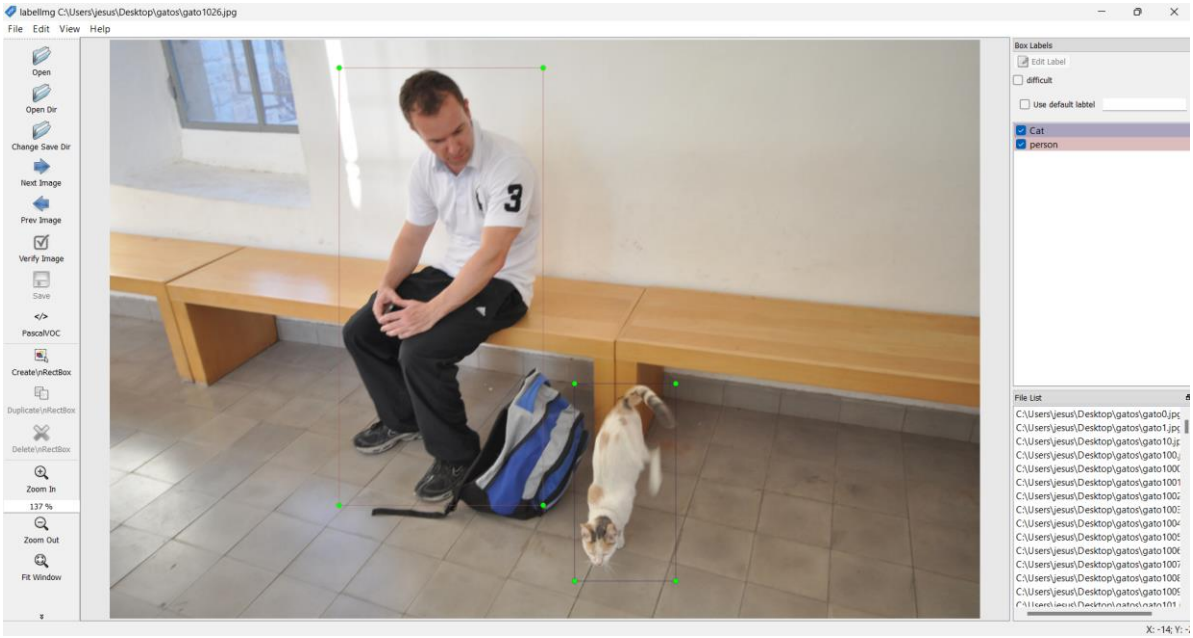


Figura 4-7. Interfaz gráfica de LabelImg.

Nuestro modelo toma como entradas bounding boxes o cajas delimitadoras, como tipo de dato para definir la posición del objeto. Debido a esto, hemos establecido normas y reglas para normalizar la colocación de estas cajas, lo que nos permite obtener un etiquetado más consistente y mejorar la respuesta durante el entrenamiento de nuestro modelo.

1. Dejar cierto margen respecto a los bordes del objeto, ya que las siluetas proporcionan una información valiosa al modelo.
2. Evitar superposiciones innecesarias entre cajas y garantizar que las cajas abarquen adecuadamente el objeto de interés sin incluir áreas irrelevantes.
3. Ignorar objetos que no estén claros o no aporten información relevante.

En general nuestro enfoque se basa en la idea de posicionar las cajas delimitadoras de manera que reflejen la ubicación deseada de los objetos según nuestras expectativas. Al seguir esta lógica, nos aseguramos de etiquetar las imágenes de una manera que represente cómo queremos que el modelo detecte y localice los objetos en el mundo real.

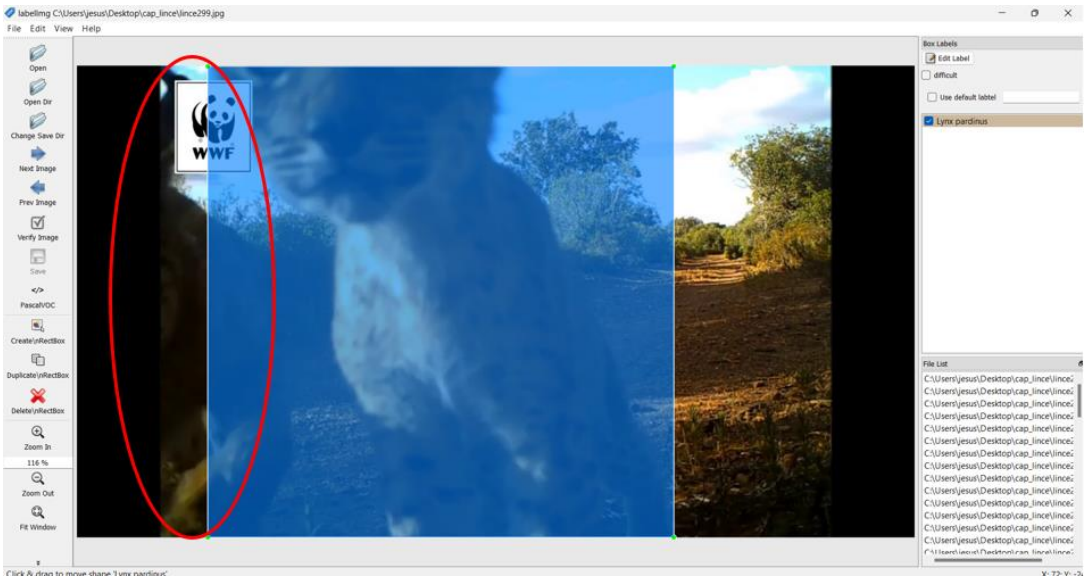


Figura 4-8. Ejemplo de objeto ignorado por no ser claro.

Para los conjuntos de datos de lince, zorro y conejo, hemos utilizado la herramienta mencionada anteriormente para llevar a cabo el etiquetado de los objetos de interés.

En el caso de los conjuntos de datos de gato y humanos, se utilizaron datasets previamente etiquetados, lo que nos permitió ahorrar tiempo en el proceso de etiquetado. Estos conjuntos de datos etiquetados previamente proporcionaron una base confiable para entrenar y desarrollar nuestros modelos de aprendizaje automático. Estos siguen las mismas normas descritas por lo que no supone una contradicción con el resto de los conjuntos.

Después de completar el proceso de etiquetado, hemos llevado a cabo la normalización de los archivos presentes en cada uno de los conjuntos de datos. Estos conjuntos incluyen un archivo de anotación XML correspondiente a cada imagen JPG etiquetada. Con el objetivo de establecer un formato de nombre normalizado, hemos tomado la decisión de generar un nombre consistente para cada objeto en las imágenes.

Para lograr esto, hemos adoptado el siguiente formato de nombre: "*<nombre de la clase><número>.<extensión>*". Además, hemos asegurado que el campo "*<filename>*" en el archivo de anotación XML coincida con el nuevo nombre del archivo JPG correspondiente.

Con el fin de llevar a cabo esta normalización de manera eficiente, hemos desarrollado dos scripts: **cambia_nombre.py** y **modifica_XML.py** (disponibles en el repositorio del proyecto).

El script **cambia_nombre.py** se encarga de renombrar los archivos JPG y XML, aplicando el formato de nombre normalizado a cada objeto. Por ejemplo, si tenemos un objeto de clase "perro" en la imagen "imagen001.jpg", el script cambiará su nombre a "perro001.jpg". Este proceso se realiza de forma automatizada para todas las imágenes y objetos etiquetados en el conjunto de datos.

Por otro lado, el script **modifica_XML.py** se encarga de actualizar el campo "*<filename>*" en los archivos de anotación XML para que coincida con el nuevo nombre del archivo JPG correspondiente. De esta manera, se mantiene la coherencia entre los nombres de archivo y los registros de anotación en los archivos XML.

El formato normalizado de los nombres de archivo ha resultado ser de gran utilidad en diversas tareas, como el recuento de datos, **figura 4-4**, y la división del conjunto de datos en subconjuntos para el entrenamiento.

4.3 Entrenamiento

Tras construir un buen conjunto de datos, procederemos a utilizar Google Colab para entrenar el modelo. Google Colab es una plataforma en línea que nos permite ejecutar código Python, aprovechando los recursos de hardware de Google, como las Unidades de Procesamiento Tensorial (TPUs) y las Unidades de Procesamiento Gráfico (GPUs).

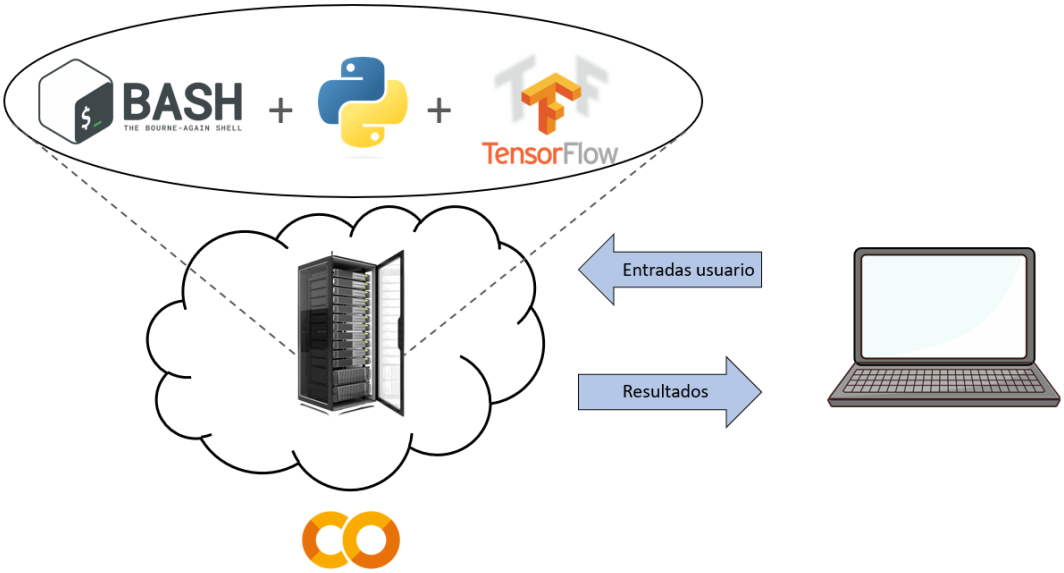


Figura 4-9. Esquema de funcionamiento de Google Colab.

Para realizar el entrenamiento, haremos uso del marco de trabajo TensorFlow estándar en lugar de TensorFlow Lite, ya que la versión Lite no admite el entrenamiento de modelos. TensorFlow Lite está diseñado específicamente para dispositivos móviles y sistemas con recursos limitados, centrándose en la inferencia (es decir en el uso del modelo) de modelos previamente entrenados.

El entrenamiento de un modelo de detección de objetos requiere una capacidad de cálculo intensiva y recursos considerables, como GPU o TPU. Estos recursos generalmente no están disponibles en dispositivos móviles, que están diseñados para ser eficientes en energía y tienen limitaciones de hardware.

Una vez que el modelo está entrenado y se ha alcanzado un nivel satisfactorio de precisión, se puede convertir a un formato compatible con TensorFlow Lite para su implementación en dispositivos móviles. Esto implica una conversión del modelo entrenado a un formato optimizado y compatible con las limitaciones de recursos de los dispositivos móviles, lo que permite una inferencia eficiente.

Para entrenar el modelo se ha hecho uso de una notebook o plantilla de ya preparada para el entrenamiento, la cual ha sido adaptada a nuestras necesidades [75]. En el **anexo E** se detalla el funcionamiento de esta.

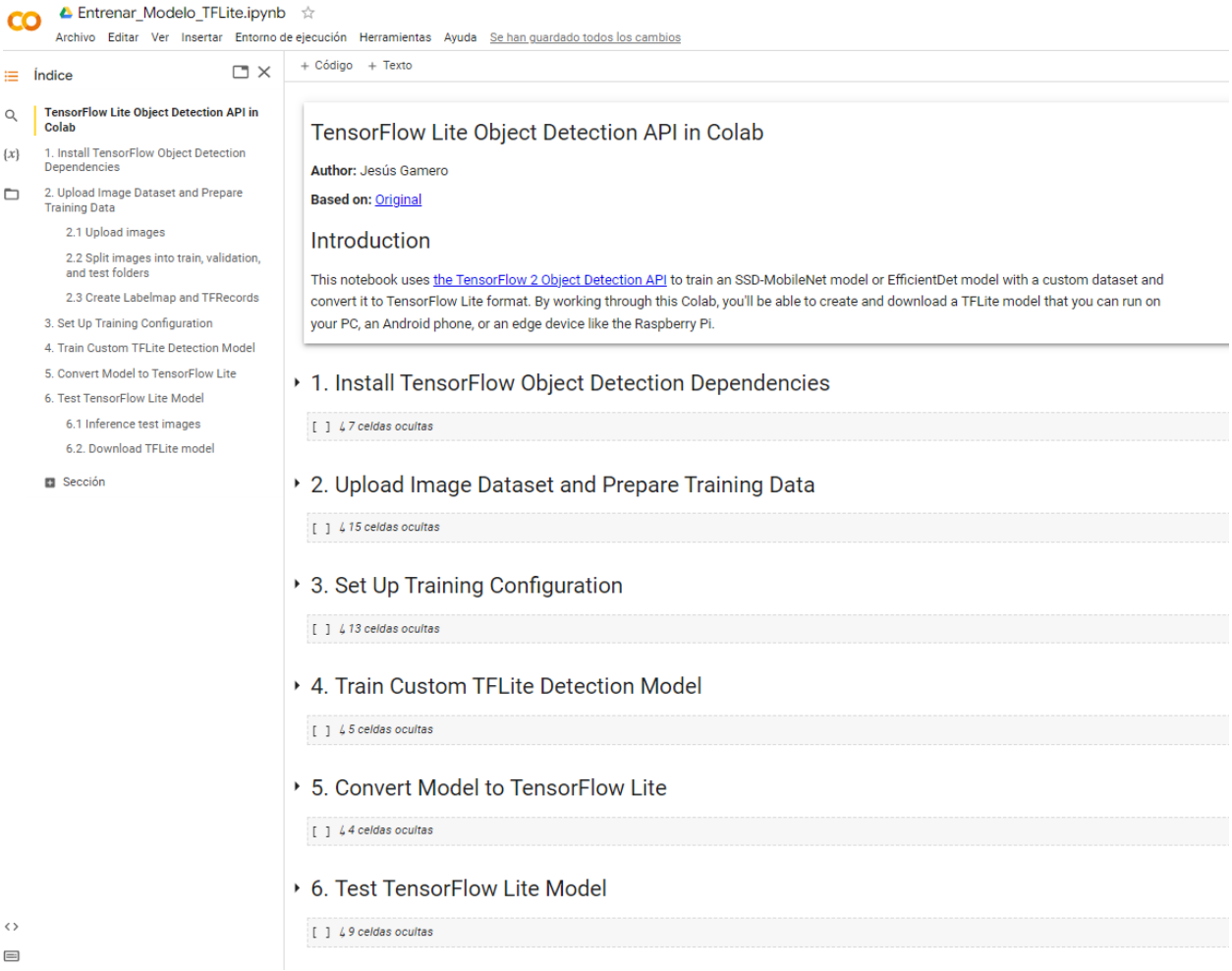


Figura 4-10. Notebook usada para el entrenamiento.

Para lograr un entrenamiento de calidad en el aprendizaje automático, es fundamental contar con un archivo de configuración que aborde los aspectos clave tanto del modelo como del proceso de entrenamiento. En el **anexo F** se encuentra disponible un archivo completo que detalla estos aspectos relevantes. A continuación, analizaremos los puntos más destacados de la configuración del modelo para comprender su importancia y cómo influyen en el éxito del entrenamiento.

Tabla 4-2. Configuración de los componentes del modelo.

Arquitectura		
Extractor de características	El modelo utiliza el extractor de características "ssd_mobilenet_v2_fpn_keras", que se basa en la arquitectura MobileNetV2 con una red de pirámide de características (FPN) agregada en la parte superior. FPN ayuda a capturar características de múltiples escalas.	
Predictor de cajas	El modelo utiliza un predictor de cajas convolucional compartido en pesos con convoluciones separables en profundidad. Consta de varias capas convolucionales con una profundidad de 128 y utiliza convoluciones separables en profundidad para reducir el cálculo.	Inicialización bias: -4.6 Función Activación: RELU_6

Loss Function		
Classification Loss	Se utiliza para medir la discrepancia entre las etiquetas reales y las predicciones. El modelo utiliza la pérdida focal sigmoidal ponderada para la clasificación.	Peso de clasificación; 1.0 Alpha: 0.25 Gamma: 2.0
Pérdida de localización	Se utiliza para medir la discrepancia entre las predicciones del modelo y las ubicaciones reales de los objetos en una imagen. El modelo usa “weighted smooth L1 loss" (pérdida suave L1 ponderada). Esta función de pérdida suaviza la transición entre las predicciones cercanas a las coordenadas reales y penaliza las discrepancias mayores.	Peso de localización: 1.0
Matcher y calculadora de similitud		
Matcher	Es el componente del algoritmo que se encarga de asociar las predicciones de objetos generadas por el modelo con las cajas delimitadoras de objetos reales en la imagen de entrada. El matcher determina qué predicciones se consideran coincidencias positivas, coincidencias negativas o sin coincidencia. Usa una medida de similitud a partir de la cual se toman decisiones sobre las coincidencias entre las predicciones y las cajas delimitadoras reales.	Umbral: 0.5
Calculadora de similitud	Utilizado para medir la similitud entre las cajas delimitadoras predichas por el modelo y las cajas delimitadoras de referencia en la imagen de entrada.	Medida similitud: IoU
Otros ajustes		
Redimensionador de imágenes	El modelo redimensiona las imágenes de entrada a una forma fija.	320x320 píxeles
Normalización por lotes	Consiste en normalizar las activaciones intermedias de la red en cada lote de datos, ajustando su media y varianza. Esto ayuda a reducir el desvanecimiento del gradiente, mejora la generalización del modelo y permite un entrenamiento más rápido y efectivo.	Escala: activo decaimiento: 0.997 épsilon: 0.001
Non-Maximum Suppression (NMS)	Después de que el modelo predice las cajas delimitadoras y sus puntuaciones, se aplica NMS para suprimir las cajas superpuestas y seleccionar las detecciones más confiables.	Umbral IoU: 0.6 Detecciones máximas por clase: 100 Detecciones máximas totales: 100
Train config Contiene la información necesaria para entrenar el modelo.		
Batch size	Establece el tamaño del lote utilizado durante el entrenamiento. Representa el número de muestras de entrenamiento que se utilizan en cada paso de actualización de pesos.	Batch size: 16
Num steps	Especifica el número total de pasos de entrenamiento. Indica la cantidad de veces que el modelo se ajustará a los datos de entrenamiento.	Num steps: 80000
Data augmentation options	Proporciona opciones para el aumento de datos durante el entrenamiento. Esto incluye transformaciones como volteo horizontal y recorte aleatorio de imágenes. El aumento de datos ayuda a mejorar la capacidad del modelo para generalizar.	Random horizontal flip Random crop image

Optimizer	Se encarga de ajustar los pesos y los sesgos del modelo de manera iterativa con el objetivo de minimizar la función de pérdida y mejorar el rendimiento del modelo. Incluye detalles como la tasa de aprendizaje, el tipo de optimizador (como el optimizador de momento) y otros parámetros específicos del optimizador.	Learning rate base: 0.08 Total, steps: 50000 Warmup learning rate: 0.026666 Warmup steps: 1000

Por lo cual ponemos ver que el archivo de configuración es una pieza fundamental para lograr un entrenamiento de calidad. A través de los diferentes ajustes y parámetros presentes en el archivo, se puede personalizar y optimizar el proceso de entrenamiento. La elección adecuada del tamaño del lote y otros aspectos como el aumento de datos y el optimizador permite adaptar el modelo a las necesidades específicas del problema y mejorar su rendimiento. Es crucial comprender y ajustar estos parámetros de acuerdo con las características del conjunto de datos y los objetivos del modelo.

Una vez definidos todos los aspectos del entrenamiento se procederá a llevarlo a cabo, para poder visualizar la evolución de las funciones objetivo iniciaremos una sesión de *tensorboard* donde mostraremos las funciones de error o loss functions.

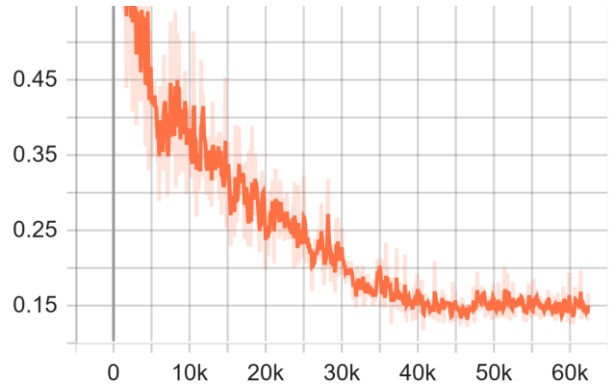


Figura 4-11. Error total.

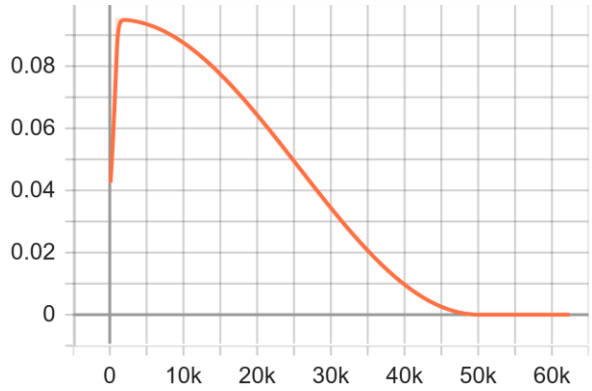


Figura 4-12. Ratio de Aprendizaje.

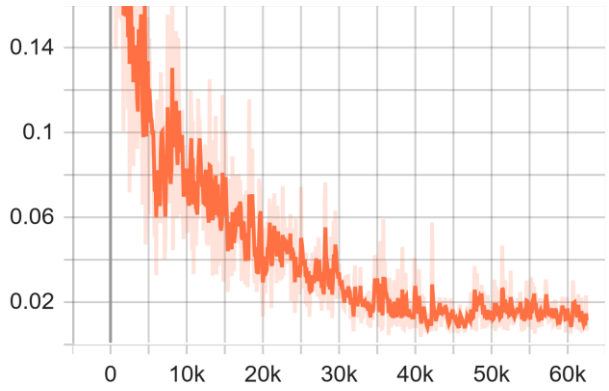


Figura 4-13. Error de localización.

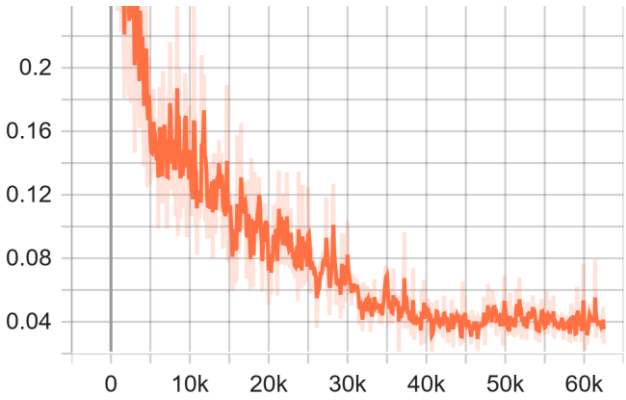


Figura 4-14. Error de clasificación.

Al analizar las figuras, podemos destacar que el proceso de entrenamiento se ha llevado a cabo con éxito. Se ha realizado un total de 60000 pasos, los cuales han sido suficientes para alcanzar valores aceptables en las funciones de pérdida. Esto indica que el modelo ha logrado ajustarse adecuadamente a los datos de entrenamiento y ha aprendido a generalizar para realizar predicciones precisas en nuevos ejemplos. Estos resultados son un indicativo positivo de que el entrenamiento ha sido efectivo y que el modelo tiene el potencial de desempeñarse bien en tareas de detección de objetos. Es importante resaltar que el entrenamiento de un modelo puede requerir diferentes cantidades de pasos dependiendo de la complejidad del problema y del conjunto de datos utilizado. En este caso, los 60000 pasos han sido suficientes para obtener resultados prometedores, pero es posible que en otros casos sea necesario ajustar el número de pasos para lograr un rendimiento óptimo.

5 VALIDACIÓN Y PRUEBAS

En este capítulo vamos a describir y desarrollar todas las pruebas realizadas sobre el modelo entrenado en el capítulo anterior, junto con el código desarrollado para llevar estas acabo. Buscando así cumplir el objetivo de validar el correcto funcionamiento del modelo en la raspberry pi 3B.

Comenzaremos este capítulo describiendo en detalle las técnicas empleadas para contabilizar la precisión del modelo incluyendo métricas y métodos de evaluación pertinentes. Además, se describirán los procedimientos implementados para realizar las pruebas de forma sistemática y reproducible. Luego, nos centraremos en el código desarrollado para llevar a cabo estas pruebas.

Posteriormente, analizaremos en detalle los resultados obtenidos durante las pruebas. Se examinarán las métricas de rendimiento y se compararán. Además, se buscarán posibles áreas de mejora o limitaciones identificadas durante las pruebas.

5.1 Metodología

Para cuantificar el rendimiento del modelo y determinar su calidad, es necesario establecer medidas que representen la cantidad de objetos correctamente clasificados y localizados. Estas medidas nos permitirán evaluar la precisión y la eficacia del modelo en la tarea de detección y clasificación de objetos.

Una vez establecidas estas medidas, procederemos a implementarlas en un script diseñado específicamente para este propósito. Este script se encargará de recorrer un conjunto de datos de prueba previamente seleccionado. Estos datos de prueba son fundamentales para evaluar el rendimiento del modelo.

5.1.1. Métricas de evaluación utilizadas

Para medir la precisión de las detecciones realizadas por el modelo, aplicaremos la inferencia al conjunto de datos de prueba previamente seleccionado. Durante esta etapa, el modelo generará predicciones sobre las imágenes de prueba, indicando la presencia y ubicación de los objetos de interés. Una vez obtenidas las predicciones, utilizaremos una la métrica, F1-score para evaluar la calidad de las detecciones. El F1-score es una medida que combina la precisión y la exhaustividad (recall) de un modelo, ofreciendo una visión general de su rendimiento. Podemos ver las definiciones de sus expresiones matemáticas a continuación.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (5\text{-}1)$$

$$precision = \frac{TP}{TP + FP} \quad (5\text{-}2)$$

$$recall = \frac{TP}{TP + FN} \quad (5\text{-}3)$$

Siendo:

- ***TP (true positive),*** o ***verdadero positivo:*** Un verdadero positivo se da cuando el modelo realiza una detección correcta de un objeto de interés. Es decir, el modelo identifica correctamente la presencia y la ubicación de un objeto que realmente existe en la imagen.
- ***FP (false positive), o falso positivo:*** Un falso positivo ocurre cuando el modelo realiza una detección incorrecta de un objeto que en realidad no existe en la imagen. El modelo identifica erróneamente la presencia y/o la ubicación de un objeto que no forma parte de los objetos de interés.

- ***FN (false negative),*** o ***falso negativo:*** Un falso negativo tiene lugar cuando el modelo no detecta un objeto de interés que realmente existe en la imagen. El modelo no logra identificar la presencia y/o la ubicación de un objeto que debería haber sido detectado.
- ***Precisión:*** Es una ratio que indica la proporción de detecciones correctas realizadas por el modelo en relación con todas las detecciones (verdaderos positivos y falsos positivos). En otras palabras, mide la exactitud de las detecciones realizadas por el modelo.
- ***Recall*** o ***exhaustividad:*** indica la proporción de objetos de interés detectados correctamente por el modelo en relación con todos los objetos de interés presentes en la imagen (verdaderos positivos y falsos negativos). Mide la capacidad del modelo para encontrar todos los objetos relevantes.
- ***F1-Score:*** F1-Score es una medida que combina la precisión y la exhaustividad en una sola métrica. Es útil cuando se desea tener una visión general del rendimiento del modelo que tenga en cuenta tanto la precisión como de la exhaustividad. Se calcula como la media armónica entre la precisión y el recall, proporcionando un balance entre ambas métricas. Un F1-Score alto indica un buen equilibrio entre la precisión y el recall.

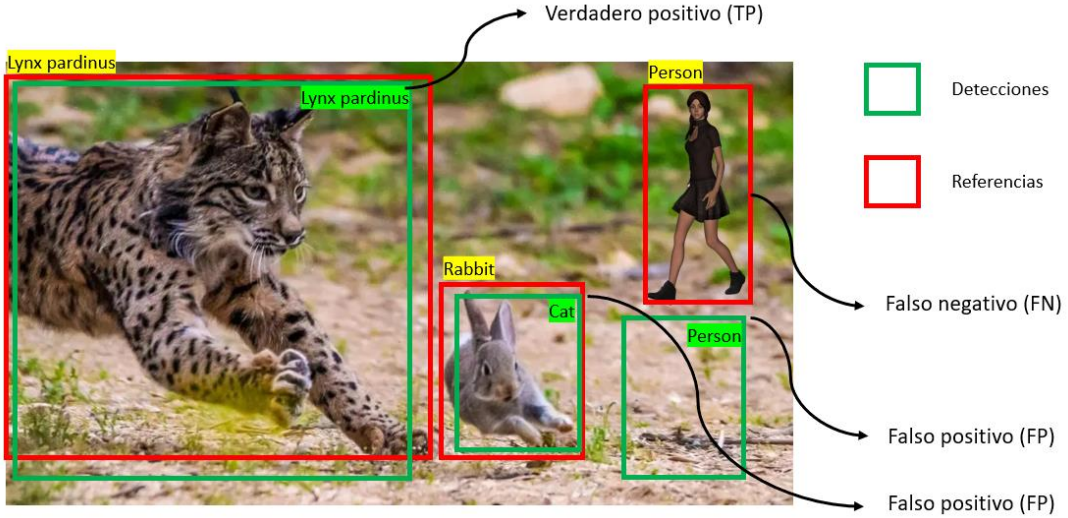


Figura 5-1. Ilustración de los diferentes casos posibles en la detección.

Es importante también definir el criterio por el que decidiremos si una detección es un verdadero o falso positivo, para ello utilizaremos la llamada intersection over union (IoU), (se definió en el **apartado 3.3.2.)** con un umbral de 0.7.

Para obtener una visión más completa de la calidad de los grupos de datos para cada una de las clases que se desean detectar, se aplicaran las métricas específicas a cada clase individualmente.

Al analizar cada clase por separado, podemos obtener información detallada sobre su rendimiento y precisión. Esto nos permite identificar posibles desequilibrios de clase y evaluar cómo se están clasificando correctamente los elementos de cada grupo.

5.1.2. Código para las pruebas de inferencia

Como ya se ha comentado anteriormente para probar la precisión del modelo haremos uso de un conjunto de datos de prueba. Este tiene un tamaño importante por lo que para no sobrecargar la memoria de la Raspberry haremos uso de una carpeta compartida en OneDrive de la que tomaremos los datos necesarios para realizar los experimentos.

Podemos ver a continuación en la **figura 5-2**, un esquema que muestra el funcionamiento del algoritmo implementado para las pruebas de inferencia.

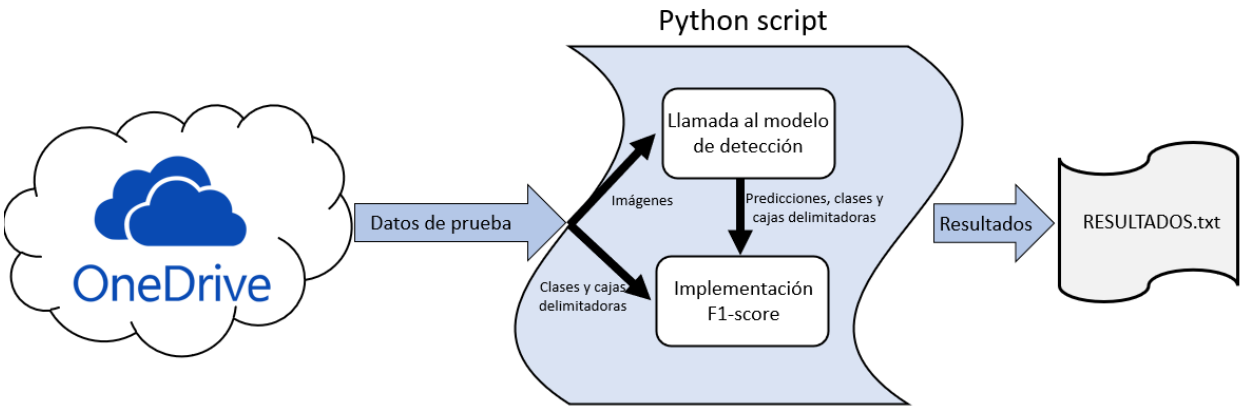


Figura 5-2. Esquema de funcionamiento de programa para las pruebas de inferencia.

Observando la **figura 5-2** podemos explicar la ejecución del algoritmo siguiendo los siguientes pasos.

1) Se descargan de la carpeta de datos de prueba disponible en la nube una unidad de datos, es decir una imagen JPG con su correspondiente archivo de anotación XML.
2) La imagen será procesada por el detector para de esta forma obtener las predicciones de las clases como de la ubicación de los objetos de estudio disponibles en la imagen.
3) Del archivo de anotación XML, se extraerán los datos de todos los objetos que realmente hay en la imagen, es decir su ubicación (como las coordenadas de su caja delimitadores), como su clase.
4) Tras obtener las predicciones como los datos reales de la imagen procederemos a compararlos para así poder determinar como de exactas son las predicciones. Obteniendo así la cantidad de verdaderos positivos, falsos positivos y falsos negativos, para cada una de las clases.
5) Tras recorrer todos los datos de prueba procederemos aplicar las ecuaciones expuestas en el apartado anterior para calcular las métricas precisión, exhaustividad y finalmente el F1-score de cada clase.

Es necesario detallar los puntos 4 y 5 para entender el algoritmo desarrollado para determinar si las detecciones son correctas o incorrectas.

El algoritmo se inicia asignando una puntuación a cada una de las referencias presentes en una imagen. Para lograr esto, se calcula el IoU (Intersection over Union) entre cada referencia y todas las detecciones. Esto genera una matriz de puntuaciones con dimensiones equivalentes al número de detecciones por el número de referencias.

Una vez obtenida esta matriz de puntuaciones, se busca el valor máximo en cada columna que también sea el máximo en su respectiva fila. En otras palabras, se busca el valor máximo por columna, pero solo se consideran aquellos valores que son máximos en su fila correspondiente. De esta manera, se asigna la detección con la puntuación más alta a la referencia correspondiente. Evitando de esta forma asignar la misma detección a diferentes referencias o asignar erróneamente. Podemos ver en la **figura 5-3** más detalladamente el proceso llevado a cabo por el algoritmo.

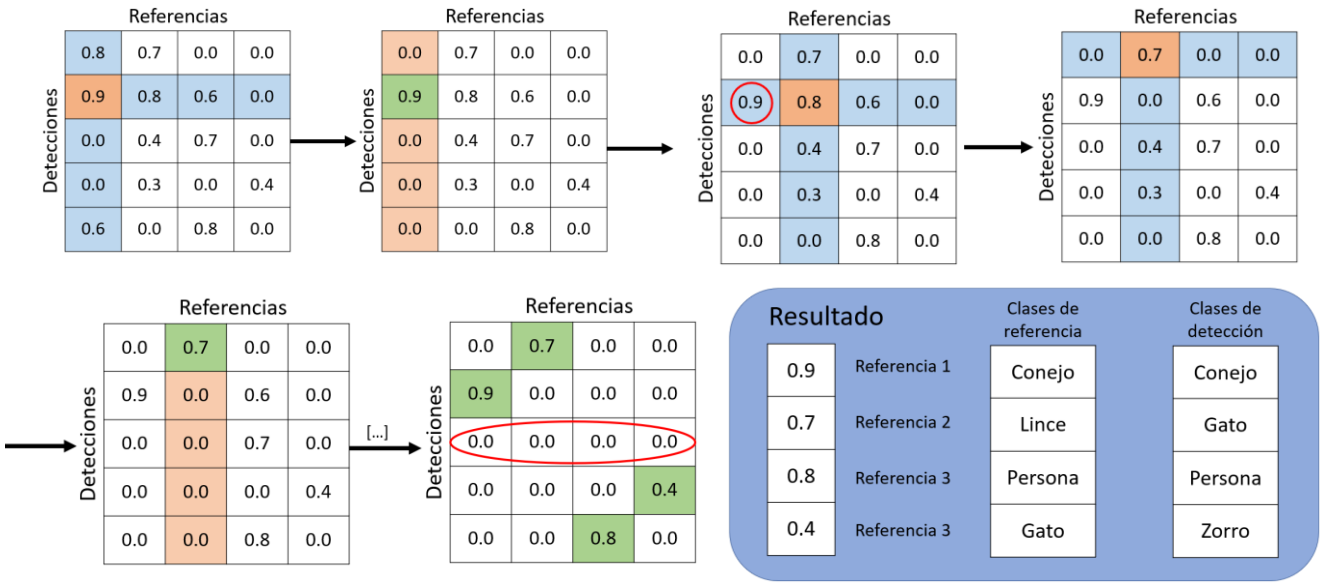


Figura 5-3. Ilustración de la ejecución del algoritmo de búsqueda de máximos.

Como se puede ver en la **figura 5-3** una detección se asignará a la referencia con la que mayor similitud tenga, es decir tenga una puntuación mayor a la hora de calcular el IoU. Por lo cual, tras concluir la ejecución del algoritmo, tendremos acceso a las puntuaciones asignadas a cada una de las asignaciones realizadas entre referencias y detecciones en un único vector, así como las clases asignadas por el modelo. Podremos comparar estas clases con las clases de las referencias y aplicar las condiciones necesarias para determinar si las detecciones son verdaderos positivos, falsos positivos o falsos negativos.

A la hora de aplicar las condiciones es necesario tener en cuenta diferentes casuísticas:

1. **Igual número de referencias que de detecciones (matriz cuadrada)**, por lo que cabe la posibilidad de que todas las detecciones sean verdaderos positivos, por lo que habría que recorrer el vector que se obtiene del algoritmo, para comprobar si las detecciones cumplen la condición de poder ser determinadas como *verdadero positivo*, es decir su puntuación sea mayor de 0.5 y la clase de la detección coincida con la de la referencia. Sin embargo, si la puntuación supera el umbral de 0.5 pero no se ha clasificado bien el objeto se tomará como un *falso positivo* ya que es una localización correcta pero una clasificación errónea. Para el resto de los casos la detección se tomará como un *falso negativo*.
2. **Mayor número de referencias que de detecciones (mayor número de columnas)**, significa que se han producido menos detecciones de las esperadas, lo cual significa que inevitablemente se han producido *falsos negativos*. Para contabilizarlos se seguirá la siguiente metodología. Se rellenará la matriz de puntuaciones con ceros para poder asignar una detección a todas las referencias y así poder obtener un vector con todas las puntuaciones y clases de todas las referencias. Una vez obtenido el vector tanto de puntuaciones como de clases extraeremos las puntuaciones máximas y aplicaremos las condiciones anteriormente descritas, hasta llegar al alcanzar el número de detecciones en este momento contabilizaremos el resto como *falsos negativos* asignándolos a la clase de la referencia.
3. **Mayor número de detecciones que de referencias (mayor número de filas),** se han producido más detecciones de las esperadas, lo cual significa que inevitablemente si han producido *falsos positivos*. Para contabilizarlos al igual que en el caso anterior primero clasificaremos las detecciones que han podido ser asignadas de forma correcta a una referencia. El resto de las detecciones sobrantes serán *falsos positivos* para detectarlos recorreremos la matriz de puntuaciones por filas, dichas filas que sean completamente cero indicarán un *falso positivo* (podemos ver esto en la **figura 5-3**), contabilizándose para la clase de dicha detección.

Tabla 5-1. Diferente casuística representada como matriz de confusión

Comparación salidas del modelo y referencias		Clase referencia = Clase detectada	
		Positivo	Negativo
IoU > 0.5	Positivo	TP	FP
	Negativo	FN	FN

En la **tabla 5-1** se muestra la clasificación de las detecciones asignadas a una referencia, considerando diversos escenarios. Se omite la categoría de **verdaderos negativos** (*TN - True Negative*) ya que el enfoque principal es evaluar las detecciones y determinar si las referencias han sido detectadas correcta o incorrectamente. Esto nos permite analizar la precisión y efectividad del algoritmo de detección sin tener en cuenta las instancias donde no se esperaba una detección.

Nota: Cabe destacar que en la **tabla 5-1** solo se están teniendo en cuenta las detecciones que han sido signadas a una referencia. Las detecciones que no han sido asignadas serán contabilizadas, como ya se ha aclarado como **falsos positivos** y las detecciones que no se han producido como **falsos negativos**.

De esta forma conseguiremos obtener una medida de como de bueno es el modelo detectando cada una de las clases disponibles. Pudiendo así determinar posibles puntos de mejora en los conjuntos de datos, como detectar posibles puntos de mejora en la arquitectura del modelo, o también mejoras en los parámetros del entrenamiento. El código completo lo encontramos en el **anexo G.**

5.2 Resultados y análisis

Después de presentar la metodología a utilizar, se llevarán a cabo una serie de experimentos para evaluar el desempeño de nuestro modelo. Se buscará estudiar las siguientes variables:

1. **Evaluación del modelo:** Se determinará la calidad del modelo mediante el uso de las métricas definidas en la sección anterior. Se analizará su precisión, recall, F1-score.
2. **Rendimiento en Raspberry Pi:** Se analizará el rendimiento del modelo en la plataforma Raspberry Pi, específicamente el tiempo requerido para procesar una imagen. Esto nos permitirá evaluar la eficiencia y velocidad del modelo en un entorno de recursos limitados.
3. **Uso de recursos:** Se analizará el consumo de recursos por parte del sistema de detección en la Raspberry Pi. Esto incluye la memoria RAM utilizada, el uso de la CPU y otros recursos relevantes. El objetivo es comprender el impacto que tiene el sistema en los recursos de la Raspberry Pi y asegurarse de que estén dentro de los límites aceptables.

Una vez establecidos los campos de estudio, procederemos a definir los experimentos que nos permitirán obtener los datos necesarios para realizar el análisis. Los siguientes experimentos se llevarán a cabo

5.2.1. Experimentos a realizar

Para evaluar adecuadamente las diferentes características de nuestro sistema, se llevarán a cabo una serie de experimentos que abordarán aspectos clave. A continuación, se describen los experimentos propuestos:

1. **Experimento de precisión y recall bajo diferentes condiciones de luminosidad:** Utilizaremos un conjunto de datos de prueba que contenga muestras de todas las clases de objetos, como se muestra en la **figura 4-6** del **apartado 4.2.1**. Estos datos estarán anotados con las referencias para evaluar la precisión y el recall del sistema de detección. Compararemos las detecciones generadas por el modelo con las anotaciones de referencia para determinar la calidad de las detecciones.

   El objetivo principal de este experimento es evaluar el desempeño del modelo en diferentes condiciones de luminosidad. Vamos a modificar la luminosidad de las imágenes del conjunto de datos de prueba y obtendremos resultados para diferentes niveles de luminosidad. Es fundamental comprender cómo se comporta el modelo en condiciones de luminosidad extrema, ya que estas situaciones son comunes en sistemas de fototrampeo. Por lo tanto, es crucial garantizar un rendimiento óptimo del modelo bajo estas condiciones.

   Para llevar a cabo este experimento, hemos introducido un parámetro de rango [0, 1], el cual llamaremos ***factor de opacidad***, que multiplicará las intensidades de la imagen, modificando así su valor. Un valor de 0 hará que todos los píxeles de la imagen sean negros, mientras que un valor de 1 no modificará la imagen original.

Figura 5-4. Ejemplo de aplicación de diferentes factores de opacidad.

2. **Experimento de rendimiento:** llevará a cabo un experimento para evaluar la eficiencia del sistema en términos de velocidad de procesamiento. Se medirá el tiempo de ejecución del sistema al procesar un conjunto de imágenes de prueba y se registrarán los tiempos de detección para cada imagen.

   El objetivo principal de este experimento es determinar la capacidad del modelo para procesar fotogramas por segundo (FPS). Dado que el diseño del sistema de fototrampeo ha sido implementado de manera parcialmente independiente de los retrasos generados por la detección, gracias a la implementación de concurrencia en el código principal, resulta crucial obtener una medida precisa de la velocidad de nuestro modelo. Esto nos permitirá evaluar la eficacia de su implementación en una Raspberry Pi u otro dispositivo similar.

3. **Experimento de consumo de recursos:** Se realizarán mediciones del consumo de recursos del sistema durante la ejecución del programa principal. Se registrarán datos como el uso de CPU, temperatura y memoria para evaluar el impacto en los recursos del sistema.

Es importante mencionar que estos experimentos se realizarán utilizando el modelo seleccionado para el sistema, que en este caso es el SSD Mobilenet V2 FPNLite. Además, se llevarán a cabo experimentos comparativos utilizando una versión del modelo sin Feature Pyramid Network (FPN) en la etapa de extracción de características.

El objetivo de realizar estos experimentos comparativos es evaluar la influencia del FPN en el rendimiento del sistema. Compararemos la precisión, el recall, el tiempo de ejecución y el uso de recursos entre ambas versiones del modelo. Esto nos permitirá determinar si la incorporación del módulo FPN aporta mejoras significativas en términos de rendimiento y eficacia.

Al realizar estos experimentos comparativos, obtendremos una visión más completa del desempeño del modelo con y sin FPN, lo que nos ayudará a tomar decisiones informadas sobre la configuración más adecuada para nuestro sistema de detección de fototrampeo.

5.2.2. Resultados

A continuación, analizaremos los resultados obtenidos de los experimentos expuestos en el apartado anterior.

Comenzando por las pruebas realizadas sobre el modelo se ha rellenado la **tabla 5-2**, donde se muestran los resultados globales obtenidos para el modelo, además se puede ver una representación gráfica de estos resultados en la **figura 5-5**. Observando los resultados podemos destacar:

- ➔ Se obtienen resultados destacables en términos de precisión, con un promedio de 0.932. Esto significa que la cantidad de falsos positivos en comparación con los verdaderos positivos se mantiene baja. En otras palabras, la gran mayoría de las detecciones realizadas por nuestro modelo son correctas, incluso en situaciones de iluminación extrema (0.84 para el peor caso de iluminación). Este alto nivel de precisión es un indicativo de la robustez y eficacia de nuestro sistema de detección de objetos.
- ➔ De igual forma, se observan valores destacados de exhaustividad (o recall), aunque ligeramente más bajos, con un promedio de 0.855. Esto implica que el modelo tiene la capacidad de detectar la mayoría de las instancias positivas en el conjunto de datos. Sin embargo, es importante mencionar que, para el caso más extremo, con un factor de opacidad de 0.15, se obtiene un resultado óptimo de 0.629. En contraste, para los valores cercanos a este, como 0.2 y 0.25, se obtiene un resultado más satisfactorio y aceptable 0.743 y 0.798 respectivamente, con menos de tres de cada diez objetos sin ser detectados.

Tabla 5-2. Resultados globales del modelo para los datos de prueba, con diferente factor de opacidad.

Coeficiente de brillo	TP	FP	FN	Precisión	Recall	F1-Score
1	510	24	58	0.955	0.897	0.925
0.95	508	24	61	0.954	0.982	0.922
0.9	506	23	63	0.956	0.889	0.921
0.85	505	23	64	0.956	0.887	0.920
0.8	506	23	63	0.956	0.889	0.921
0.75	505	24	63	0.954	0.889	0.920
0.7	505	25	62	0.952	0.890	0.920
0.65	506	34	61	0.937	0.892	0.914
0.6	504	32	65	0.940	0.885	0.912
0.55	504	31	67	0.942	0.882	0.911
0.5	504	32	67	0.940	0.882	0.910
0.45	500	35	69	0.934	0.878	0.905
0.4	496	43	73	0.920	0.871	0.895
0.35	485	40	80	0.923	0.858	0.889
0.3	469	44	91	0.914	0.837	0.874
0.25	444	44	112	0.909	0.798	0.850
0.2	408	52	141	0.886	0.743	0.808
0.15	336	64	198	0.84	0.629	0.719

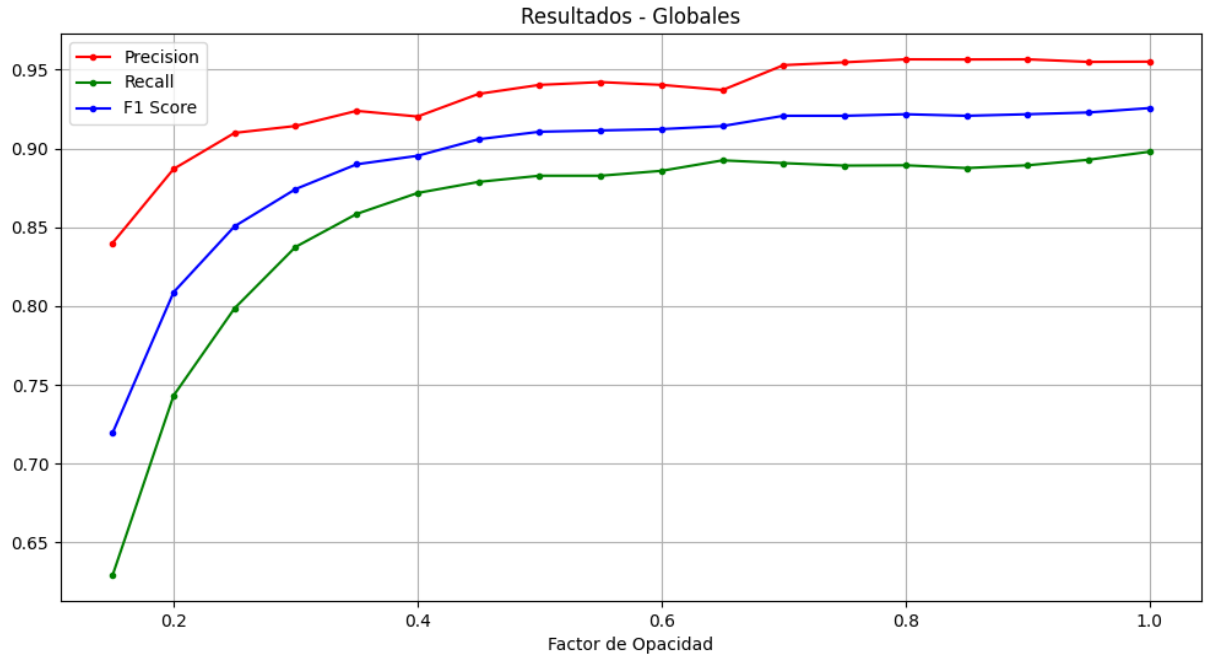


Figura 5-5. Resultados globales SSD Mobilenet V2 FPNLite para diferentes factores de opacidad.

En la **figura 5-5** se muestra de manera más clara la evolución de la precisión y la exhaustividad a medida que varía el factor de opacidad. Es notable que estos valores no experimentan cambios significativos hasta llegar a valores inferiores a 0.35, lo que demuestra la capacidad del modelo para adaptarse a condiciones de baja iluminación. Además, se observa que el F1-score mantiene valores aceptables hasta un factor de opacidad de 0.2, lo cual sugiere que el modelo sigue siendo efectivo en la detección de objetos incluso en situaciones de baja visibilidad.

En gran parte este buen rendimiento se debe a la incorporación de Feature Pyramid Networks (FPN) a él extractor de características ya que combina características de diferentes niveles de la pirámide que MovilnetV2 extrae. Podemos ver en la **figura 5-6** los resultados globales obtenidos para el modelo SSD Mobilenet V2, sin FPN, se puede observar que la tolerancia a la ausencia de luz es menor.

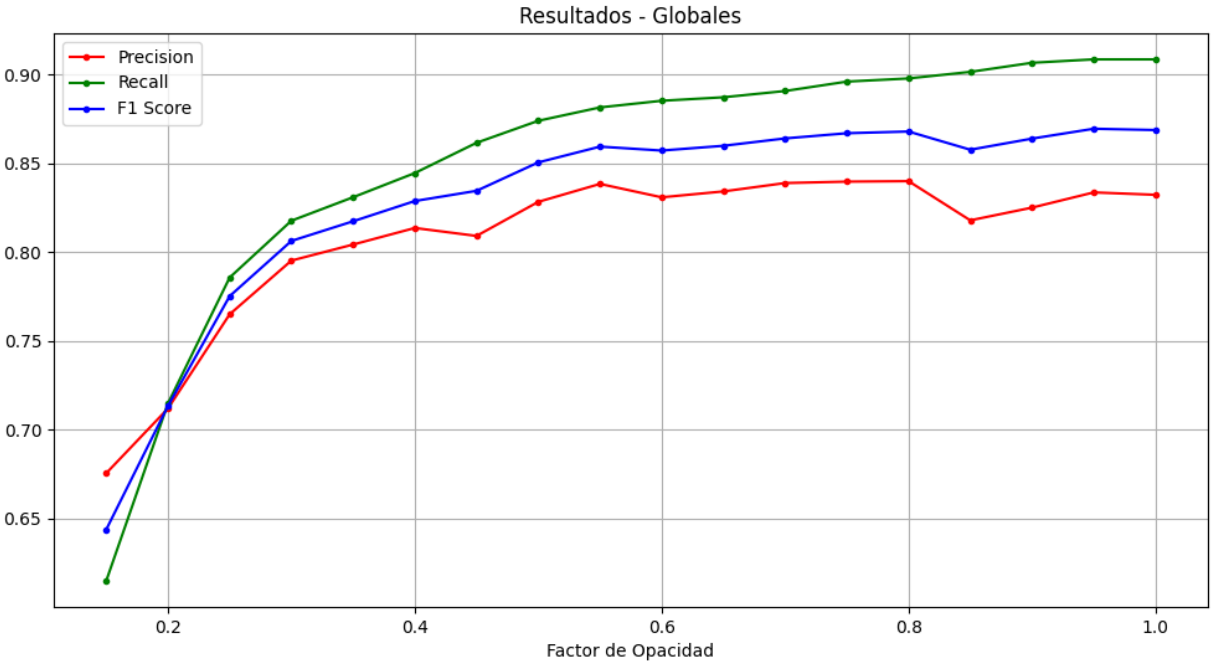


Figura 5-6. Resultados globales SSD Mobilenet V2 para diferentes factores de opacidad.

A continuación, estudiaremos estos resultados obtenidos para el modelo SSD Mobilenet V2 FPNLite para cada clase por separado para así poder realizar un estudio más detallado y determinar posibles puntos a mejorar en el entrenamiento o en el conjunto de datos usado para este.

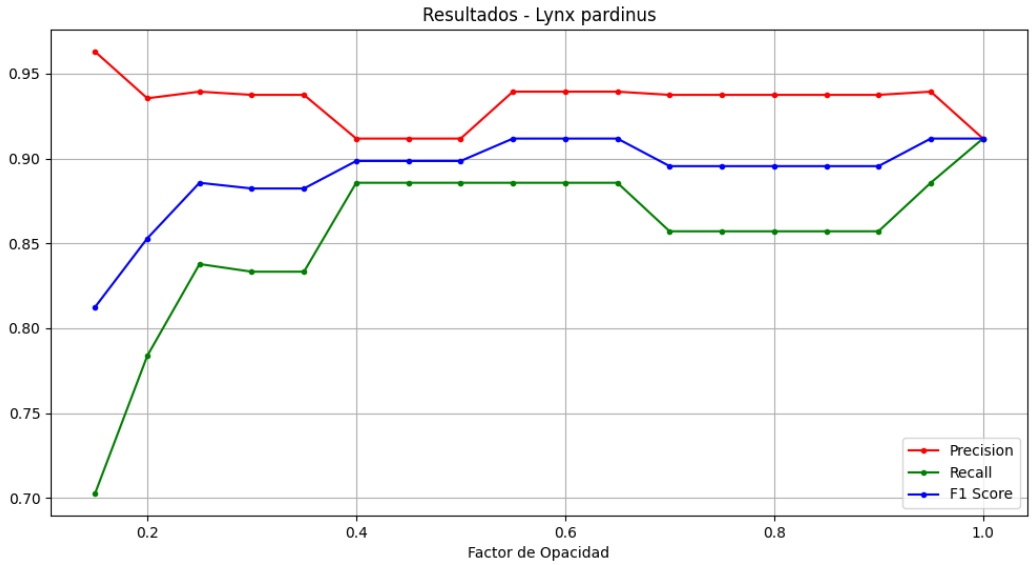


Figura 5-7. Grafica de los para la clase *Lynx Pardinus* para diferentes factores de opacidad.

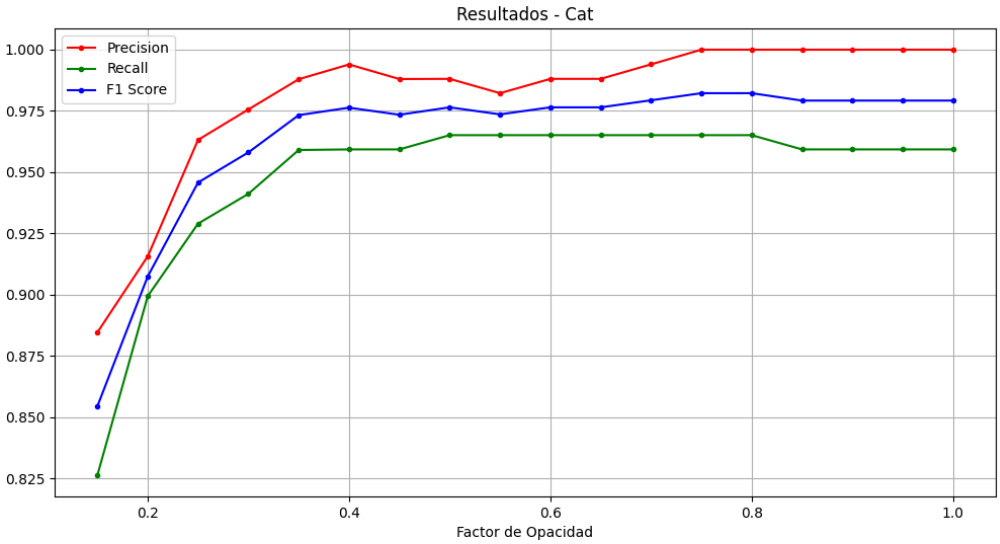


Figura 5-8. Grafica de los para la clase *Cat* para diferentes factores de opacidad.

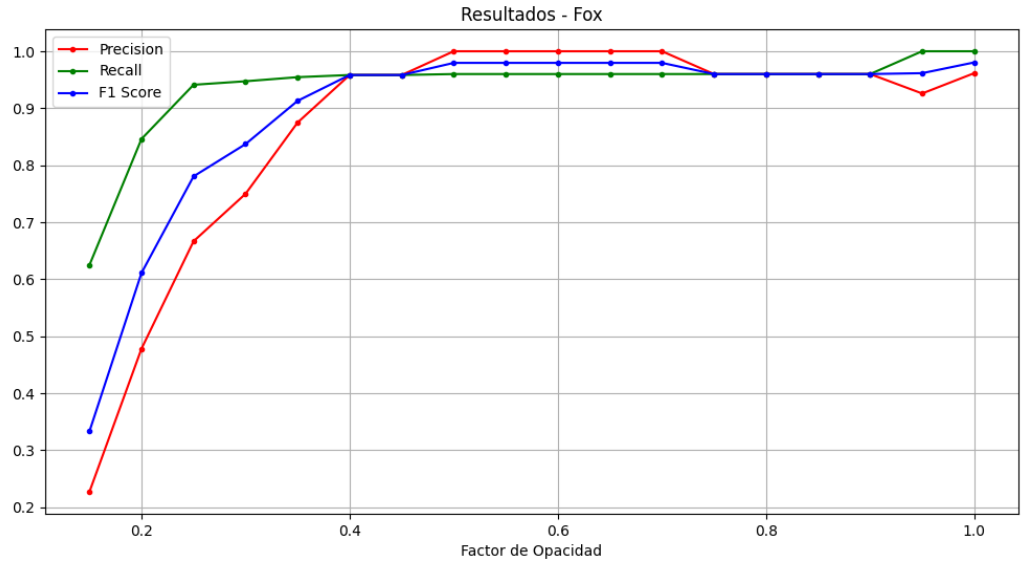


Figura 5-9. Grafica de los para la clase *Fox* para diferentes factores de opacidad.

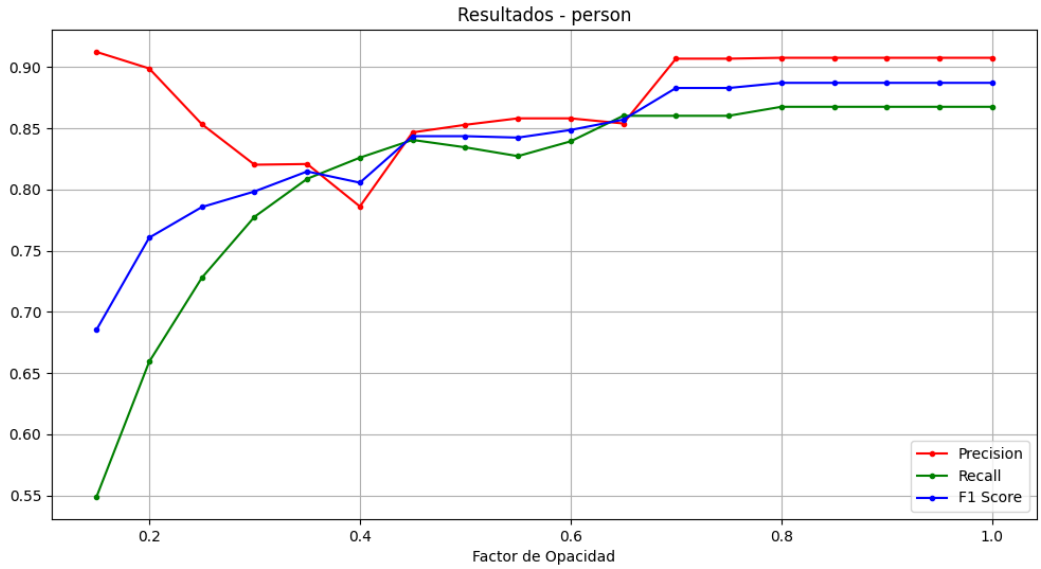


Figura 5-10. Grafica de los para la clase *Person* para diferentes factores de opacidad.

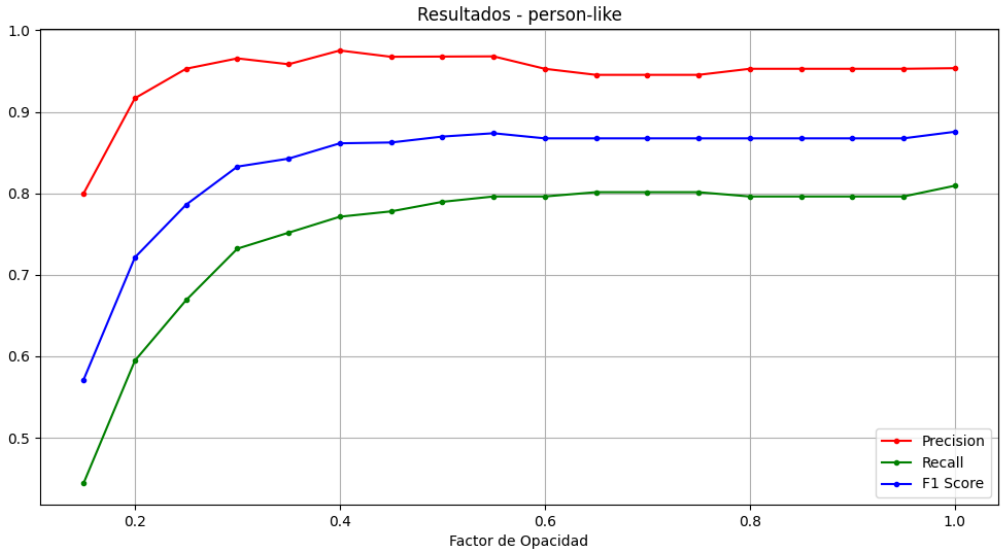


Figura 5-11. Grafica de los para la clase *Person-like* para diferentes factores de opacidad.

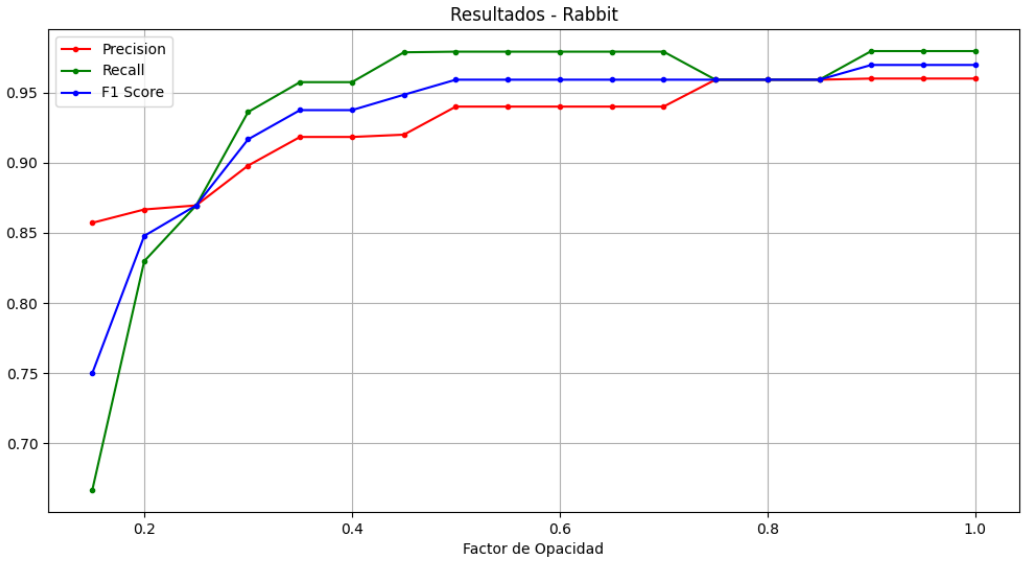


Figura 5-12. Grafica de los para la clase *Rabbit* para diferentes factores de opacidad.

Observando las gráficas expuestas podemos destacar dos puntos positivos.

- En los conjuntos de datos más reducidos, como *Lynx pardinus* y *rabbit*, se obtuvieron resultados satisfactorios a pesar de representar un porcentaje menor del conjunto total (6.2 % y 7.8 % respectivamente). Esto indica que la elección de las imágenes en situaciones límite obtenidas de estudios de fototrampeo, y la metodología empleada para el etiquetado de estas, han sido factores clave en el entrenamiento exitoso del modelo.
- También es importante destacar los excelentes resultados obtenidos para la clase *cat*. En la **figura 5-7** se puede observar que tanto la precisión como la exhaustividad se mantienen por encima de 0.825, incluso en las situaciones más extremas con un factor de opacidad de 0.15. Estos resultados positivos se deben en gran medida a que el subconjunto de datos de la clase *cat* representa el 57.3% del conjunto total. Esto demuestra que la abundancia de datos disponibles para esta clase ha sido fundamental para que el modelo pueda generalizar de manera efectiva y lograr un rendimiento destacado.

Por otro lado, cabe destacar algunos puntos negativos.

- Se puede observar que el rendimiento para la clase *fox* no es del todo deseable para los coeficientes de opacidad más bajos, el modelo consigue mantener unos buenos resultados para factores mayores a 0.4 pero para valores menores la degradación de la precisión como de la exhaustividad son notables llegando a situarse la precisión en valores cercanos a 0.2 lo cual significa que ocho de cada diez detecciones serán erróneas.
- En relación con las clases *person* y *person-like* cabe destacar la degradación de la exhaustividad, se puede apreciar como para valores menores 0.3 esta disminuye en gran medida, dejando ver la dificultad por parte del modelo de detectar objetos de interés pertenecientes a estas clases en condiciones límites de luminosidad.

Tabla 5-3. Análisis las medias para los cuatro factores de opacidad más desfavorables.

Clase	Precisión	Recall	F1	Resultado	Observaciones
Lynx Pardinus	0.943	0.789	0.858	Satisfactorio	La capacidad del modelo para detectar la clase *lynx pardinus* se puede observar que es bueno con una puntuación mayor de 0.8 para el F1-score. Cuenta con cierto margen de mejora, pero teniendo en cuenta el tamaño del conjunto de datos es un resultado más que correcto.
Cat	0.934	0.898	0.916	Satisfactorio	La capacidad del modelo para detectar la clase *cat* se puede observar que es bueno con una puntuación mayor de 0.9 para el F1-score.
Fox	0.530	0.839	0.640	Insatisfactorio	La capacidad del modelo para detectar la clase *fox* se puede observar que es insuficiente con una puntuación inferior de 0.65 para el F1-score. El modelo encuentra dificultades para realizar detecciones correctas en condiciones de baja luminosidad
Person	0.871	0.678	0.757	Insatisfactorio	La capacidad del modelo para detectar la clase *person* se puede observar que es insuficiente con una puntuación inferior de 0.8 para el F1-score. Debido a una baja exhaustividad el modelo no detectará más de tres objetos de interés que se encuentren en una imagen bajo condiciones de baja luminosidad.
Person-like	0.908	0.609	0.727	Insatisfactorio	La capacidad del modelo para detectar la clase *person-like* se puede observar que es insuficiente con una puntuación inferior de 0.8 para el F1-score. Debido a una baja exhaustividad el modelo no detectará más de tres objetos de interés que se encuentren en una imagen bajo condiciones de baja luminosidad.
Rabbit	0.872	0.825	0.846	Satisfactorio	La capacidad del modelo para detectar la clase *rabbit* se puede observar que es bueno con una puntuación mayor de 0.8 para el F1-score. Cuenta con cierto margen de mejora, pero teniendo en cuenta el tamaño del conjunto de datos es un resultado más que correcto.
Global	0.887	0.752	0.813	Parcialmente Satisfactorio	El comportamiento global del modelo para las peores condiciones de luminosidad puede observarse que es aceptable consiguiéndose un F1-score mayor a 0.8. Existe cierto margen de mejora si se trabaja en construir mejores conjuntos de datos para las clases *fox, person* y *person-like.*

Para tener una mejor y más completa comparación entre el modelo que hace uso de FPN y del que no, se han representado en la **tabla 5-4** diferentes valores de precisión y exhaustividad obtenidos para ambos modelos en diferentes situaciones.

Se aprecia que, para el factor de opacidad máximo, es decir no se modifican las imágenes, se obtiene un rendimiento bueno para ambos modelos siendo ligeramente superior SSD Mobilenet V2 FPNLite. Sin embargo, para el valor mínimo SSD Mobilenet V2, no consigue obtener los mismos valores de precisión lo que afecta a si F1-score.

Este peor rendimiento de SSD Mobilenet V2, también se ve reflejado en la media obtenida de los cuatro resultados más desfavorables, por lo que podemos concluir que FPN es una adición importante para que el modelo funcione de la forma más exacta posible en condiciones de luminosidad baja.

Tabla 5-4. Comparativa de resultados entre versión del modelo con y sin FPN.

Condiciones de los resultados	Modelo	Precisión	Recall	F1-Score
Mínimo factor de opacidad	**SSD Mobilenet V2 FPNLite**	0.84	0.629	0.719
	SSD Mobilenet V2	0.675	0.615	0.643
Máximo factor de opacidad	**SSD Mobilenet V2 FPNLite**	0.955	0.897	0.925
	SSD Mobilenet V2	0.832	0.908	0.868
Media para los 4 factores más desfavorables	**SSD Mobilenet V2 FPNLite**	0.887	0.752	0.813
	SSD Mobilenet V2	0.736	0.733	0.734
Media para todos los factores	**SSD Mobilenet V2 FPNLite**	0.932	0.855	0.891
	SSD Mobilenet V2	0.807	0.850	0.828

Por lo cual, para las peores condiciones de trabajo el modelo elegido para ser implementado en el sistema SSD Mobilenet V2 FPN, consigue tener un ***f1-score*** de 0.813, lo cual es un buen resultado comparado con otros modelos de detección de objetos [76].

Tabla 5-5. Comparativa de diferentes modelos con SSD Mobilenet V2 FPNLite.

Modelo	Precisión	Recall	F1-Score
SSD	0.99	0.677	0.804
Faster R-CNN	0.817	0.945	0.872
YOLOV3	0.979	0.912	0.943
SSD Mobilenet V2 FPNLite (media de los 4 peores escenarios)	0.887	0.752	0.813
SSD Mobilenet V2 FPNLite (media de todos los escenarios)	0.932	0.855	0.891

Para los experimentos de rendimiento del modelo obtenemos los siguientes resultados de tiempo para ambas versiones.

Tabla 5-6. Comparación del rendimiento temporal del modelo con y sin FPN.

Modelo	**Duración total** (segundos)	**Tiempo medio por imagen** (segundos)	**Fotogramas por segundo medio**
SSD Mobilenet V2 FPNLite	254.917	0.662	1.511
SSD Mobilenet V2	192.993	0.501	1.999

Se puede observar que SSD Mobilenet V2 tiene un tiempo de procesamiento promedio 0.161 segundos más rápido que SSD Mobilenet V2 FPNLite. Esto le permite procesar medio fotograma por segundo adicional, lo que resulta en un mayor rendimiento en aplicaciones en tiempo real. Sin embargo, esta ventaja en velocidad no es suficiente para compensar el peor rendimiento en la detección de objetos de SSD Mobilenet V2. Además, es importante destacar la capacidad del sistema de fototrampeo diseñado para separar la captura de imágenes y el procesamiento, lo cual brinda una ventaja adicional en términos de eficiencia y uso de recursos. Por lo cual, aunque SSD Mobilenet V2 presenta una pequeña ventaja en velocidad, no supera los beneficios generales de SSD Mobilenet V2 FPNLite en términos de precisión y eficacia en la detección de objetos.

Tabla 5-7. Comparación de los resultados finales de las pruebas de inferencia para ambos modelos.

Modelo	**SSD Mobilenet V2 FPN**	**SSD Mobilenet V2**
Precisión (media todos los factores)	0.932	0.807
Exhaustividad (media todos los factores)	0.855	0.850
F1-score (media todos los factores)	0.891	0.828
Tiempo medio por imagen	0.662	0.501
Fotogramas por segundo medio	1.511	1.999

A continuación, se analizarán los experimentos realizados sobre el uso de los recursos del sistema.

En la **figura 5-13** se puede ver la evolución del uso de los recursos de la CPU durante la ejecución del simulador de fototrampeo o código principal. Podemos ver como al iniciarse la ejecución del programa se produce un pequeño aumento de la actividad. Posteriormente se producen dos detecciones en el segundo 30 y en el 170, podemos ver como la actividad aumenta considerablemente, debido a la toma y procesamiento de los datos. Se observa que la actividad aumenta cerca de un 30 %.

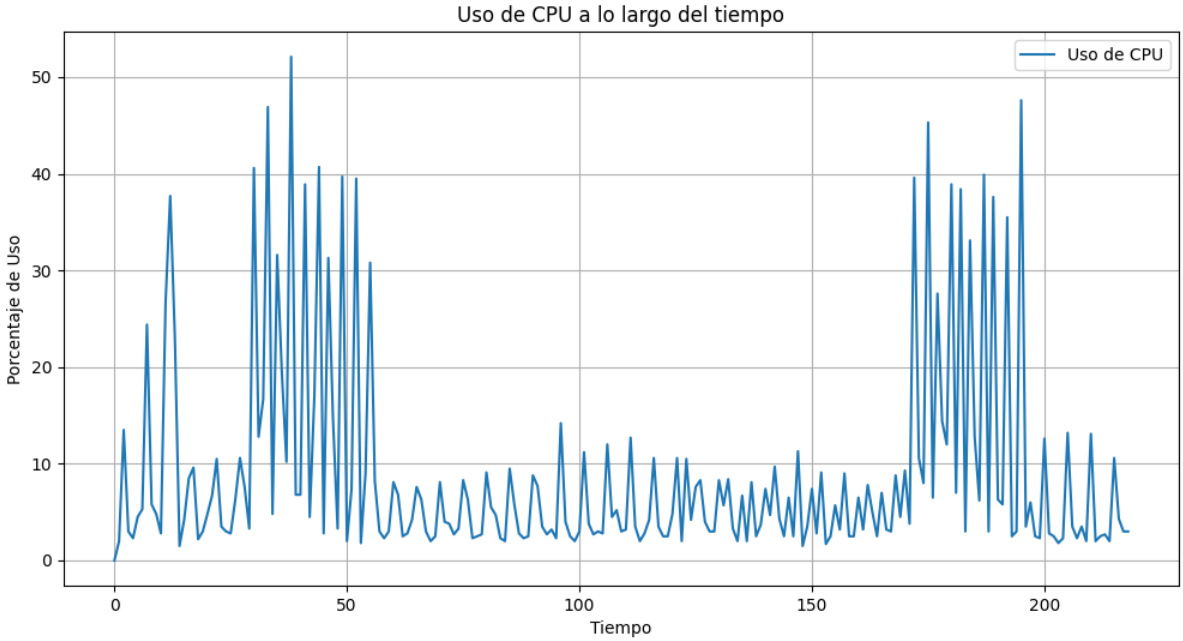


Figura 5-13. Grafica para el porcentaje de uso de la CPU durante la ejecución del código principal.

De igual forma en la **figura 5-14** se puede comprobar que al iniciarse el programa el uso de la memoria aumenta alrededor de un 4 %. Posteriormente en el segundo 30 se produce la primera activación de la cámara trampa y por lo tanto se activa procesamiento de las imágenes por lo que podemos observar que se produce un aumento del 3 % en uso de la memoria que se mantiene con pequeñas oscilaciones en las llamadas a la captura.

Por lo cual, podemos ver que el conjunto del simulador de cámara trampa ocupa un total del 7 % de la memoria.

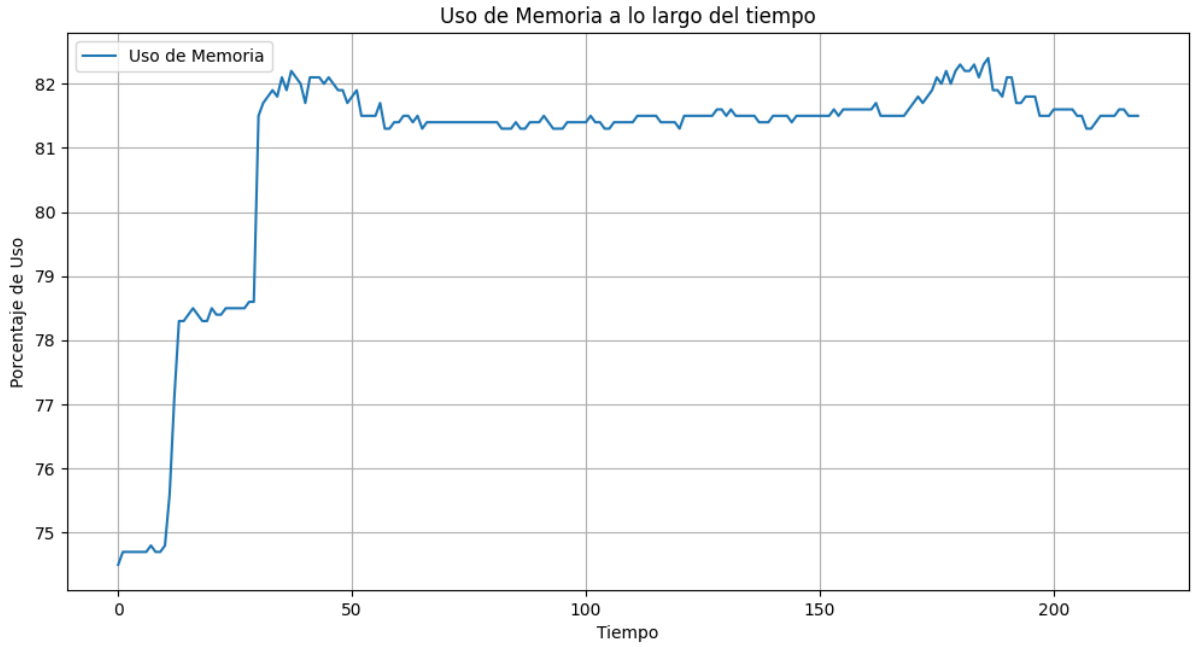


Figura 5-14. Grafica para el porcentaje de uso de la memoria durante la ejecución del código principal.

En la **figura 5-15** se puede ver que al igual que ocurre para las **figuras 5-13** y **5-14**, se produce un aumento de la magnitud medida al momento de capturar y procesar las imágenes. Podemos ver que se produce un incremento de un grado centígrado.

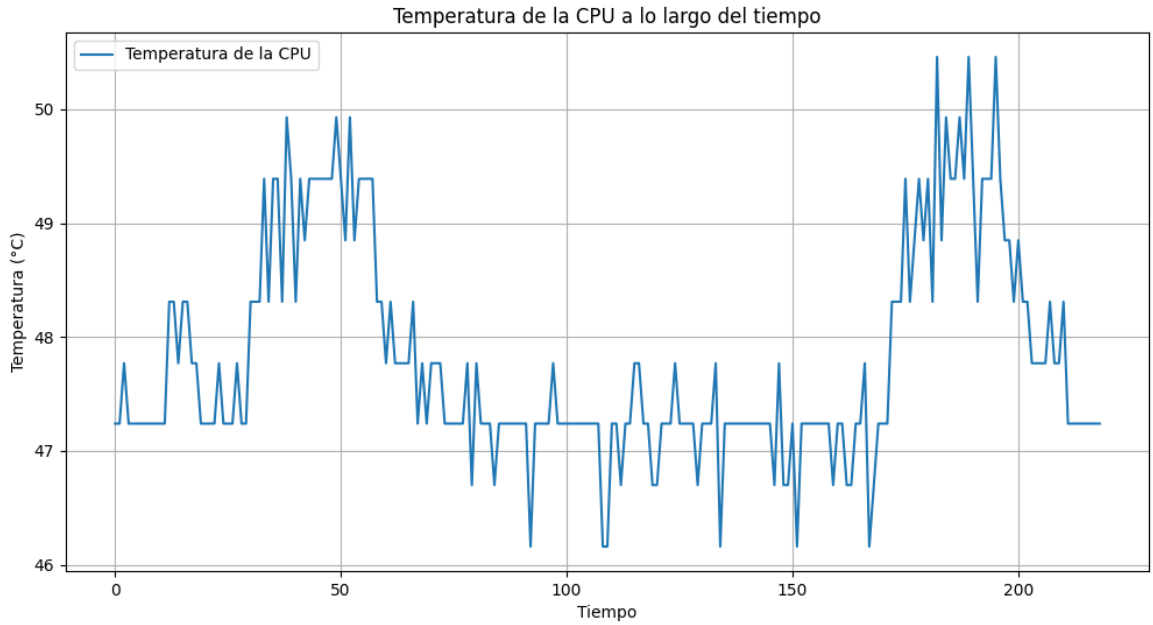


Figura 5-15. Grafica para la temperatura de la CPU durante la ejecución del código principal.

Tabla 5-8. Resumen de las pruebas realizadas.

Objeto de estudio	Resultado	Observaciones
Calidad de las predicciones (bajo diferentes condiciones de luminosidad)	Parcialmente satisfactorio	Las pruebas realizadas sobre el modelo demuestran que se pueden obtener buenas predicciones, incluso en situaciones de baja luminosidad. Sin embargo, es importante destacar que el modelo no logra detectar todas las clases de manera satisfactoria en las peores condiciones. Aunque el resultado general es bueno, existe margen para mejorar en este aspecto y lograr una detección más precisa en todas las clases en las condiciones más desfavorables.
Rendimiento	Satisfactorio	El rendimiento del modelo en cuanto al tiempo de procesamiento de imágenes y los FPS es más que satisfactorio para la aplicación desarrollada en el proyecto. El tiempo de procesado de imágenes se mantiene en niveles aceptables, lo que permite un procesamiento rápido de las imágenes capturadas. En general, el modelo demuestra un buen rendimiento en términos de velocidad y capacidad de respuesta.
Consumo de recursos	Satisfactorio	El consumo de recursos por parte del sistema de simulación de cámara trampa se encuentra dentro de los límites normales, lo que indica que el sistema está optimizado y no sobrecarga los recursos del equipo.

6 CONCLUSIONES

En este capítulo se trataran las conclusiones generales obtenidas del proyecto, se resumirán los resultados obtenidos en los capítulos anteriores, para poder concluir en la viabilidad del proyecto dentro del ámbito de protección de especies actual. Además, se comentarán las posibles mejoras, así como las posibles líneas de investigación futuras.

Como se pudo apreciar en el **capítulo 2**, el uso de algoritmos de aprendizaje automático se ha convertido en una herramienta fundamental en el ámbito de la protección del medio ambiente. Por lo tanto, el estudio y desarrollo de estas herramientas son de suma importancia para abordar los desafíos ambientales actuales y futuros de manera más efectiva. Para de esta forma conseguir automatizar tareas que posteriormente necesitaban de la capacidad humana para ser realizadas.

El uso de técnicas ampliamente utilizadas, como el **fototrampeo**, para rastrear especies esquivas y amenazadas, como el lince ibérico, cuyas poblaciones han sufrido un declive en las décadas pasadas. Se pueden combinar con algoritmos de aprendizaje automático, que maximizan la eficacia de estos dispositivos al actuar como filtros y clasificadores de los datos capturados. Así de esta forma se puede ver que el uso de algoritmos para la detección de objetos puede representar una herramienta clave a la hora de construir una primera capa de interpretación de los datos obtenidos por los sistemas de captura.

Por tanto, la incorporación de algoritmos de detección de objetos en estos sistemas puede aumentar significativamente su eficiencia y versatilidad. Con este objetivo en mente, se ha buscado implementar un modelo preciso y eficiente en un sistema móvil como la Raspberry Pi 3 B, debido a su bajo costo y eficacia. Para garantizar un rendimiento óptimo y por lo tanto la viabilidad de la aplicación, se ha utilizado TFlite, un marco de trabajo para algoritmos de aprendizaje automático optimizado para sistemas con recursos limitados. De esta forma la búsqueda de un modelo que pudiera recoger estas características ha desembocado en **SSD MobileNet V2 FPNLite**, el cual ha demostrado ofrecer las prestaciones necesarias.

- ➔ El modelo ha demostrado buenos resultados a la hora de aplicar la inferencia sobre los grupos de datos de prueba. Mostrando gran capacidad de adaptación frente a situaciones adversas de luminosidad. Por lo cual su uso en aplicaciones de fototrampeo es recomendable ya que estas condiciones adversas son comunes.
- ➔ El modelo mantiene un buen rendimiento a la hora de aplicar la inferencia, se obtiene una media de 1.5 fotogramas por segundo. Lo que puede llegar a hacer pensar que su uso en aplicaciones en tiempo real puede llegar a ser viable, más si se hace uso de versiones más actualizadas del dispositivo como la Raspberry pi 4 o se hace uso de aceleradores como el Coral USB Accelerator el cual añade un coprocesador TPU al sistema.

Tabla 6-1. Resumen de las pruebas realizadas.

Objeto de proyecto	Resultado	Observaciones
Instalación de TFlite y correcto funcionamiento del modelo en una Raspberry pi 3	Satisfactorio	La instalación de TFlite se ha llevado a cabo sin problemas, se ha podido probar con el modelo del cual se han obtenido unos resultados generales aceptables de rendimiento para el procesamiento de las imágenes.
Entrenar el modelo para conseguir detectar y diferenciar especies distintas bajo diferentes condiciones de luminosidad	Parcialmente satisfactorio	Se ha podido demostrar que gracias al dataset construido se consiguen unos buenos resultados generales al aplicar la inferencia. Sin embargo, para algunas clases no se consigue obtener un resultado satisfactorio para las peores condiciones de luminosidad.

Automatizar el proceso de censado de especies	Satisfactorio	El diseño del sistema propuesto permite crear una primera capa en el procesamiento de datos permitiendo clasificar y detectar diferentes especies objetivo. Permitiendo así detectar a tiempo real la presencia de la especie en el área de estudio.

6.1 Líneas futuras

Una vez resumidos los objetivos cumplidos por el proyecto, así como analizados lo resultados se procederá a plantear posibles futuras líneas sobre las que se podría trabajar.

- Construir un conjunto de datos de mayor tamaño y especializado en datos de cámaras trampa para todas las clases, para de esta forma buscar una mayor eficacia en la detección realizadas por el sistema.
- Hacer uso de datos más complejos como objeto de estudio. Es decir, sustituir las secuencias de imágenes tomadas cuando se detecta una presencia, por pequeños videos cuya toma aplique la siguiente lógica; durante el tiempo que el sensor de presencia este activo se grabará un video, si este deja de estar activo, se continuará grabando durante un pequeño periodo de tiempo, por si en dicho periodo se vuelve a activar, evitando así tomar videos separados las mismas detecciones.

  Esta sustitución de imágenes por videos puede abrir las puertas a numerosas aplicaciones, las más destacada puede ser el seguimiento de las detecciones en la imagen, para de esa forma poder obtener información sobre las trayectorias que las diferentes especies siguen.

  Este cambio supone un mayor consumo de recursos, por lo que es necesario mejorar las prestaciones del dispositivo.
- Mejoras en el hardware para obtener un mejor rendimiento y poder trabajar con datos más complejos que permitan obtener una mayor cantidad de información. Para ello se puede hacer uso de la versión más actualizada de Raspberry pi, la 4 modelo B. La cual cuenta con mejores prestaciones como son el procesador de cuatro núcleos ARM Cortex-A72, así como una memoria SDRAM de 4 GB e incluso de 8 GB en comparación a 1 GB que tiene el modelo anterior. Así como la adición de un coprocesador pensado para aplicaciones de aprendizaje automático como es el Coral USB Accelerator.
- Utilizar modelos más sofisticados que realicen un procesamiento posterior de los datos obtenidos por el sistema de seguimiento de objetos. Se propone desarrollar un sistema integral y complejo que automatice todos los aspectos del seguimiento de especies, desde la detección y clasificación por especies realizada por dispositivos móviles ubicados en las áreas de estudio, hasta la identificación de especímenes específicos llevada a cabo por un sistema de procesamiento centralizado, haciendo uso de herramientas como WildBook.

REFERENCIAS

[1] Silogismo, «WIkipedia.org,» 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Silogismo.

[2] G. Boole, «wikipedia.org,» 2023. [En línea]. Available: https://es.wikipedia.org/wiki/George_Boole.

[3] G. Frege, «wikipedia.org,» 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Gottlob_Frege.

[4] A. Turing, «wikipedia.org,» 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Alan_Turing.

[5] W. S. M. a. W. H. Pitts, ««A logical calculus of the ideas immanent in nervous activity,»,» 1943. [En línea]. Available: https://jontalle.web.engr.illinois.edu/uploads/410-NS.F22/McCulloch-Pitts-1943-neural-networks-ocr.pdf.

[6] J. McCarthy, ««A proposal for the Dartmouth summer research project on artificial intelligence,»,» 1955. [En línea]. Available: http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf.

[7] «AI winter; WIkipedia.org,» 2023. [En línea]. Available: https://en.wikipedia.org/wiki/AI_winter.

[8] «Gobierno de España y Gobierno de Portugal «Censo Lince»,» 2021. [En línea]. Available: https://www.miteco.gob.es/es/biodiversidad/temas/inventarios-nacionales/censodelinceiberico2021_tcm30-541552.pdf.

[9] L. D. Mech, ««A Handbook of Animal Radio-Tracking,»,» 1983. [En línea]. Available: https://pubs.er.usgs.gov/publication/5200060.

[10] Amador Huerta Vela y Alejandro Centeno-Cuadros, «Aplicaciones de las técnicas de ADN ambiental al estudio y conservación de los recursos naturales,» [En línea].

[11] E. Aspillaga, R. Arlinghaus y M. Martorell-Barceló, «Performance of a novel system for high-resolution tracking of marine fish societies. Anim Biotelemetry,,» 2021. [En línea]. Available: https://doi.org/10.1186/s40317-020-00224-w.

[12] M. A. Orts., ««Cómo animales y humanos aprovechamos ondas de frecuencias más allá de nuestro rango auditivo,»,» 2022. [En línea]. Available: https://fisiquimicamente.com/blog/2022/08/11/ultrasonidos/.

[13] Instituto, Geográfico y Nacional, «Teledetección,,» 2022. [En línea]. Available: https://www.ign.es/web/resources/docs/IGNCnig/OBS-Teledeteccion.pdf.

[14] M. McMahon., « «What is a Camera Trap?,»,» 2023. [En línea]. Available: https://www.wise-geek.com/what-is-a-camera-trap.htm.

[15] «Fototrampeo.es,» 2023. [En línea]. Available: https://fototrampeo.es/tienda/.

[16] IA for Earth, microsoft, «Multi species biocoustic classification,,» [En línea]. Available: https://github.com/microsoft/Multi_Species_Bioacoustic_Classification.

[17] L. Roberts, «Understanding the Mel Spectrogram,,» 2020. [En línea]. Available: https://acortar.link/hfCsyz.

[18] EarthSpecies, «Automatic Bioacoustic Source Separation with Deep Neural Networks,,» [En línea]. Available: https://github.com/earthspecies/cocktail-party-problem.

[19] IA for Earth, microsoft, «belugasounds,» [En línea]. Available: https://github.com/Microsoft/belugasounds.

[20] Ecologize, «SpeciesClassification,» [En línea]. Available: https://github.com/ecologize/SpeciesClassification.

[21] Beery S, Morris D, Yang S, Simon M, Norouzzadeh A,, «Megadetector,» 2019. [En línea]. Available: https://github.com/ecologize/CameraTraps/blob/main/megadetector.md.

[22] Beery S, Morris D, Yang S, Simon M, Norouzzadeh A,, «CameraTrap,» 2019. [En línea]. Available: ttps://github.com/ecologize/CameraTraps.

[23] IA for Earth, microsoft, «ArticSeals,» 2022. [En línea]. Available: https://github.com/microsoft/arcticseals.

[24] Tanya Y. Berger-Wolf, Daniel I. Rubenstein, Charle, «Wildbook: Crowdsourcing, computer vision, and data science for conservation.,» [En línea]. Available: https://arxiv.org/abs/1710.08880.

[25] Tanya Y. Berger-Wolf, Daniel I. Rubenstein, Charle, «Wildbook-ia,» 2017. [En línea]. Available: https://github.com/WildMeOrg/wildbook-ia.

[26] «sharkbook.ia,» [En línea]. Available: https://www.sharkbook.ai.

[27] «whiskerbook,» [En línea]. Available: https://www.whiskerbook.org.

[28] «lynx.wildbook,» [En línea]. Available: https://lynx.wildbook.org.

[29] «BearID Project,» [En línea]. Available: https://bearresearch.org.

[30] López-Parra et al., «Change in demographic patterns of the Doñana Iberian lynx Lynx pardinus: Management implications and conservation perspectives,,» 2012. [En línea]. Available: https://www.cambridge.org/core/journals/oryx/article/change-in-demographic-patterns-of-the-donana-iberian-lynx-lynx-pardinus-management-implications-and-conservation-perspectives/6CD1752F6102C025B98DDCAB2D004843.

[31] Gil Sánchez et al, «The use of camera trapping for estimating Iberian lynx (Lynx pardinus) home ranges,» 2012. [En línea]. Available: https://acortar.link/jTFkOd.

[32] L. Rouhiainen, «Inteligencia artificial 101 cosas que debes saber hoy sobre nuestro futuro,» 2018. [En línea].

[33] J. A. C. Garofalo y C. A. S. Guevara, «Artificial intelligence, smart systems, smart agents,» 2020. [En línea]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=7591558.

[34] P. Ongsulee,, «Artificial intelligence, machine learning and deep learning,,» 2017. [En línea]. Available: https://ieeexplore.ieee.org/document/8259629.

[35] R. Kohavi y F. Provost, «Machine Learning, Glossary of Terms,» [En línea]. Available: http://ai.stanford.edu/~ronnyk/glossary.pdf.

[36] «https://aprendeia.com,» [En línea]. Available: https://aprendeia.com/diferencia-entre-machine-learning-y-la-programacion-tradicional/.

[37] B. Mahesh, «Machine Learning Algorithms -A Review,» 2019. [En línea]. Available: https://www.researchgate.net/publication/344717762_Machine_Learning_Algorithms_-A_Review.

[38] «diegocalvo.es - Clasificación de redes neuronales artificiales,» [En línea]. Available: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/.

[39] «grupo.us.es - CONCEPTOS BÁSICOS SOBRE REDES NEURONALES,» [En línea]. Available: http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm.

[40] «CogniFit research,» [En línea]. Available: https://www.cognifit.com/es/neuronas.

[41] Izaurieta, F., & Saavedra, C., «Redes neuronales artificiales.,» Departamento de Física, Universidad de Concepción Chile., 2000. [En línea]. Available: https://www.academia.edu/download/36957207/Redes_neuronales.pdf.

[42] «https://repositoriodigital.ipn.mx,» [En línea]. Available: https://repositoriodigital.ipn.mx/bitstream/123456789/8640/2/Verde123_original.pdf.

[43] Representación de la neurona de McCulloch-Pitts, «https://interactivechaos.com,» [En línea]. Available: https://interactivechaos.com/es/manual/tutorial-de-deep-learning/representacion-de-la-neurona-de-mcculloch-pitts.

[44] M. I. O. ESPINOZA, «DETECCIÓN DE GRIETAS MEDIANTE DEEP LEARNING BASADO EN IMÁGENES,» Universidad de Chile, 2019. [En línea]. Available: https://repositorio.uchile.cl/bitstream/handle/2250/174436/cf-orellana_me.pdf?sequence=1&isAllowed=y.

[45] S. BAGCHI, «Artificial Neural Networks & their Activation Functions,» [En línea]. Available: https://medium.datadriveninvestor.com/activation-function-is-any-non-linear-function-applied-to-the-weighted-sum-of-the-inputs-of-a-a4326956cebf.

[46] H. Barrow, «Connectionism and Neural Networks,» [En línea]. Available: https://www.sciencedirect.com/science/article/abs/pii/B9780121619640500078.

[47] J. Brownlee, «What is the Difference Between a Batch and an Epoch in a Neural Network?,» 2018. [En línea]. Available: https://deeplearning.lipingyang.org/wp-content/uploads/2018/07/What-is-the-Difference-Between-a-Batch-and-an-Epoch-in-a-Neural-Network_.pdf.

[48] Frank La La, «¿Cómo aprenden las redes neuronales?,» [En línea]. Available: https://learn.microsoft.com/es-es/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn.

[49] S. H. Haji y A. M. Abdulazeez, «COMPARISON OF OPTIMIZATION TECHNIQUES BASED ON GRADIENT DESCENT ALGORITHM,» 2021. [En línea]. Available: https://archives.palarch.nl/index.php/jae/article/view/6705.

[50] N. B. Subramanian, «6 Regularization Techniques for Deep Learning,» [En línea]. Available: https://aiaspirant.com/6-regularization-techniques-for-deep-learning/.

[51] Alcides David Cantero Alonso y Eustaquio Alcides, «Visión por computadora: identificación, clasificación y seguimiento de objetos.,» [En línea].

[52] Visión por Computadora, «iaarbook.github.io,» [En línea]. Available: https://iaarbook.github.io/vision-por-computadora/.

[53] S. Saha, «A Comprehensive Guide to Convolutional Neural Networks,» 2018. [En línea]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[54] Developers Breach, «convolution neural network deep learning,» [En línea]. Available: https://developersbreach.com/convolution-neural-network-deep-learning/.

[55] guru99.es, «Clasificación de imágenes de TensorFlow: CNN (Red Neural Convolucional),» 2023. [En línea]. Available: https://guru99.es/convnet-tensorflow-image-classification/.

[56] J. Barrios, «Redes Neuronales Convolucionales - Consultores estratégicos en Ciencia de Datos,» [En línea]. Available: https://www.juanbarrios.com/redes-neurales-convolucionales/.

[57] M. S. Ali, «Flattening CNN layers for Neural Network and basic concepts,» [En línea]. Available: https://medium.com/@muhammadshoaibali/flattening-cnn-layers-for-neural-network-694a232eda6a.

[58] Naveen, «Difference between Sigmoid and Softmax activation function?,» [En línea]. Available: https://www.nomidl.com/deep-learning/what-is-the-difference-between-sigmoid-and-softmax-activation-function/.

[59] J.R.R. Uijlings, et al, «Selective Search for Object Recognition,» University of Trento, Italy. University of Amsterdam, the Netherlands, [En línea]. Available: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf.

[60] R. Gandhi, «Support Vector Machine — Introduction to Machine Learning Algorithms,» [En línea]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[61] R. Girshick, «Fast R-CNN,» [En línea]. Available: https://arxiv.org/abs/1504.08083.

[62] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,» [En línea]. Available: https://arxiv.org/abs/1506.01497.

[63] Joseph Redmon, et al, «You Only Look Once: Unified, Real-Time Object Detection,» [En línea]. Available: https://arxiv.org/abs/1506.02640.

[64] Ultralytics YOLOv8 Docs, «ultralytics.com,» [En línea]. Available: https://docs.ultralytics.com/#where-to-start.

[65] Wei Liu, et al., «SSD: Single Shot MultiBox Detector,» [En línea]. Available: https://arxiv.org/abs/1512.02325.

[66] Kaiming He, et al., «Deep Residual Learning for Image Recognition,» [En línea]. Available: https://arxiv.org/abs/1512.03385.

[67] Andrew G., et al, «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,» [En línea]. Available: https://arxiv.org/abs/1704.04861.

[68] Mark Sandler, et al, «MobileNetV2: Inverted Residuals and Linear Bottlenecks,» [En línea]. Available: https://arxiv.org/abs/1801.04381.

[69] Lei Yang, et al, «Lite-FPN for Keypoint-based Monocular 3D Object Detection,» [En línea]. Available: https://arxiv.org/abs/2105.00268.

[70] e. a. Ashish Vaswani, «Attention Is All You Need,» [En línea]. Available: https://arxiv.org/abs/1706.03762.

[71] e. a. Nicolas Carion, «End-to-End Object Detection with Transformers,» [En línea]. Available: https://arxiv.org/abs/2005.12872.

[72] e. a. Alexey Bochkovskiy, «YOLOv4: Optimal Speed and Accuracy of Object Detection,» [En línea]. Available: https://arxiv.org/abs/2004.10934.

[73] aprendeia.com, «¿Qué es TensorFlow? ¿Cómo funciona?,» 2021. [En línea]. Available: https://aprendeia.com/que-es-tensorflow-como-funciona/.

[74] TensorFlow, «Implementa modelos de aprendizaje automático en dispositivos móviles y perimetrales,» [En línea]. Available: https://www.tensorflow.org/lite?hl=es-419.

[75] Evan Juras, «TensorFlow Lite Object Detection API in Colab,» [En línea]. Available: https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb.

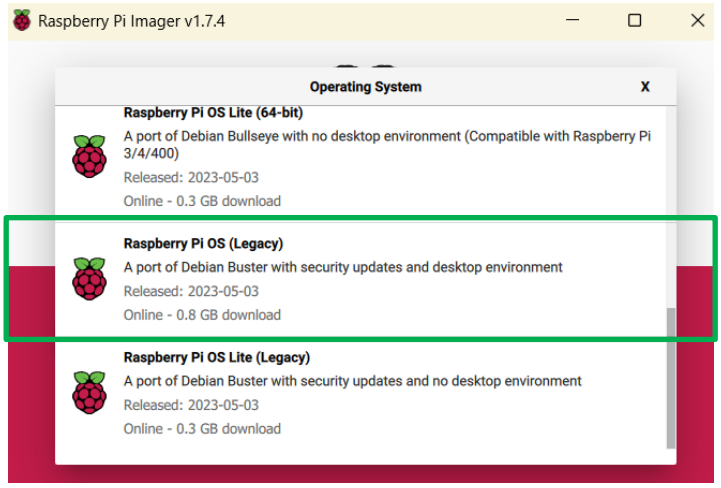
[76] Minji Park, et al, «Two-Step Real-Time Night-Time Fire Detection in an Urban Environment Using Static ELASTIC-YOLOv3 and Temporal Fire-Tube,» [En línea]. Available: https://www.researchgate.net/publication/340612158_Two-Step_Real-Time_Night-

Time_Fire_Detection_in_an_Urban_Environment_Using_Static_ELASTIC-YOLOv3_and_Temporal_Fire-Tube.

ANEXO A: PREPARACIÓN DEL ENTORNO

A continuación, se expondrán los pasos necesarios para preparar el entorno de trabajo sobre el que desarrollaremos el proyecto. Profundizaremos en la instalación del sistema operativo Raspbian, así como en la preparación de la raspberry para poder trabajar sobre ella.

1) Para poder iniciar nuestra instalación de Raspbian deberemos comenzar descargándonos el siguiente software, Raspberry Pi OS – Raspberry Pi el cual nos permite bootear una memoria SD, sobre la que escribiremos la versión del sistema operativo que queramos usar. En nuestro caso haremos uso de Raspberry pi OS (legacy) Buster con entorno de escritorio.



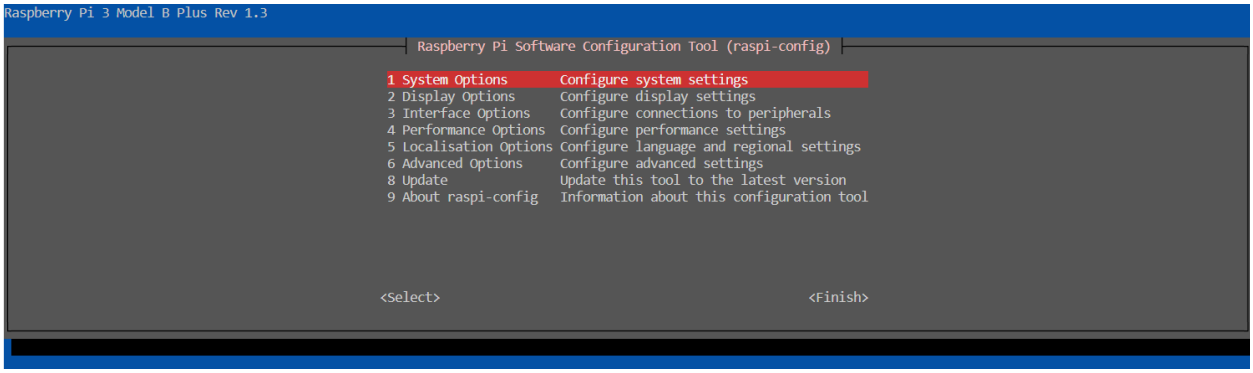
Tras descargar el sistema operativo en la tarjeta SD deberemos introducirla la raspberry, conectarla a la fuente de alimentación y proceder a su configuración.

Tras instalar el sistema operativo procederemos a preparar el entorno de trabajo en el que desarrollaremos nuestro proyecto. Se usará VScode desde nuestro pc y lo comunicaremos con la respberry a través del protocolo SSH.

2) Se debe configurar la raspberry para que siempre que se encienda se active un interfaz SSH por la que podamos conectarnos. Para ello debemos abrir una terminar e introducir el siguiente comando,

```
$> sudo raspi-config
```

Si no abrirá el siguiente menú en la terminal.

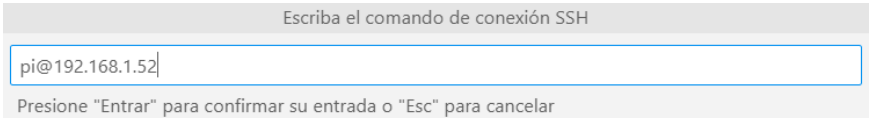


Deberemos navegar hasta Interfacing Options → P2 SSH y pulsamos <Yes>. Tras esto nos pedirá reiniciar la raspberry. Tras completar esto la tendremos configurada para poder conectarnos desde nuestro pc.

3) Finalmente se deberá configurar VScode para poder conectarnos a la raspberry.

   a. Para ello se instalará la extensión ***remote – SSH*** disponible en el centro de extensiones de la aplicación. Tras esto no debe de aparecer el siguiente icono  .

   b. Gracias a la extensión instalada podremos configurar la conexión con el servidor SSH, para ello debemos entrar en el menú de explorador remoto, en este añadiremos una nueva sesión introduciendo el siguiente comando en la ventana emergente que aparece al pulsar el botón +

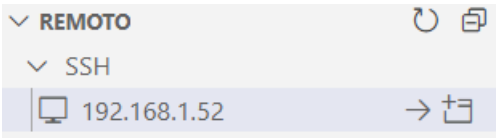


   De esta forma estaremos indicando el nombre de la sesión a la que queremos conectarnos y el servidor, que en este caso es la IP asignada a la raspberry. La aplicación nos mostrará el siguiente archivo de configuración SSH son la siguiente información.

```
Host 192.168.1.52
  HostName 192.168.1.52
  User pi
```

   Nota: El campo Host se puede cambiar al nombre que se desee.

   c. El dispositivo remoto configurado nos aparecerá en el siguiente menú desplegable desde el cual podremos acceder a un directorio de la raspberry.



ANEXO B: INSTALACIÓN DE TFLITE

Todo el código desarrollado durante el proyecto, así como, los modelos entrenados y las dependencias necesarias para usar TFlite se encuentran ele el siguiente repositorio,

https://github.com/jgamero-ibrobotics/TFG-LynxIBDetect.git

1) Clonar el repositorio en la raspberry.

```
$> git clone https://github.com/jgamero-ibrobotics/TFG-LynxIBDetect.git
$> cd TFG-LynxIBDetect
```

2) Instalar *virtualenv*, para poder crear un entorno virtual que nos permita instalar todas las dependencias sin crear incompatibilidades con las versiones ya existentes en nuestro dispositivo.

```
$> sudo pip3 install virtualenv
```

Creamos dicho entorno virtual.

```
$> python3 -m venv tflite1-env
```

3) Activamos el entorno virtual e instalamos todas las dependencias haciendo uso del siguiente script de consola, get_pi_requirements.sh.

```
$> source tflite1-env/bin/actívate
$> bash get_pi_requirements.sh
```

De esta forma ya tendremos instalado OpenCV y Tensorflow lite con todas sus dependencias. Por lo que podremos ejecutar los scripts de detección.

- Para ejecutar el simulador de cámara trampa.

  ```
  $> python3 Camara_trampa.py --modeldir=LynxDetectv2
  ```

  Debemos indicar el modelo que queremos que se ejecute.

  Nota: para que funcione correctamente se debe conectar una cámara a la raspberry, así como introducir correctamente el token de la aplicación que permite acceso a la carpeta compartida.

- Para ejecutar las pruebas de inferencia.

  ```
  $> python3 F1_Score.py --modeldir=LynxDetectv2 --darken_image=1
  --archivo_resultados=resultados.txt
  ```

  Indicando el factor de opacidad a aplicar, así como el nombre del archivo donde se quiere escribir los resultados

ANEXO C: CÓDIGO PRINCIPAL, CAPTURA Y PROCESAMIENTO

Código escrito en Python para simular el comportamiento del sistema de fototrampeo diseñado. Se detalla su funcionamiento en el **apartado 4.1.2.2.**

Disponible en: https://github.com/jgamero-ibrobotics/TFG-LynxIBDetect/blob/main/Camara_trampa.py

```
# -*- coding: utf-8 -*-

# Código para la simulación de una cámara trampa con una Raspberry Pi
# así como para la detección de objetos en las imágenes capturadas

# Creado: 27 Abr 2023
# Última modificación: 02 jun 2023

# @author: Jesús Gamero Borrego

#-------------------------------------------------------------------------------------------

# Import packages
import os
import argparse
import cv2
import time
import numpy as np
import sys
import glob
import importlib.util
import dropbox
import threading
import queue
from dropbox.exceptions import ApiError

# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected
objects',
                    default=0.5)
parser.add_argument('--image', help='Name of the single image to perform detection on. To run
detection on multiple images, use --imagedir',
                    default=None)
parser.add_argument('--imagedir', help='Name of the folder containing images to perform
detection on. Folder must contain only images.',
                    default=None)
parser.add_argument('--save_results', help='Save labeled images and annotation data to a
results folder',
                    action='store_true')
parser.add_argument('--noshow_results', help='Don\'t show result images (only use this if --
save_results is enabled)',
                    action='store_false')

args = parser.parse_args()

# Parse user inputs
MODEL_NAME = args.modeldir
```

```
50. GRAPH_NAME = args.graph
51. LABELMAP_NAME = args.labels
52. 
53. min_conf_threshold = float(args.threshold)
54. 
55. save_results = args.save_results # Defaults to False
56. show_results = args.noshow_results # Defaults to True
57. 
58. IM_NAME = args.image
59. IM_DIR = args.imagedir
60. CWD_PATH = os.getcwd() # Get path to current working directory
61. RESULTS_DIR ='results' # Folder to save results images and data to
62. 
63. # Import TensorFlow libraries
64. # If tflite_runtime is installed, import interpreter from tflite_runtime, else import from
regular tensorflow
65. pkg = importlib.util.find_spec('tflite_runtime')
66. from tflite_runtime.interpreter import Interpreter
67. 
68. # Path to .tflite file, which contains the model that is used for object detection
69. PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
70. # Path to label map file
71. PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
72. # Load the label map
73. with open(PATH_TO_LABELS, 'r') as f:
74.     labels = [line.strip() for line in f.readlines()]
75. 
76. interpreter = Interpreter(model_path=PATH_TO_CKPT) # Cargar el modelo
77. interpreter.allocate_tensors() # Asignar tensores
78. 
79. # Get model details
80. input_details = interpreter.get_input_details()
81. output_details = interpreter.get_output_details()
82. height = input_details[0]['shape'][1]
83. width = input_details[0]['shape'][2]
84. 
85. floating_model = (input_details[0]['dtype'] == np.float32)
86. 
87. input_mean = 127.5
88. input_std = 127.5
89. 
90. # Check output layer name to determine if this model was created with TF2 or TF1,
91. # because outputs are ordered differently for TF2 and TF1 models
92. outname = output_details[0]['name']
93. if ('StatefulPartitionedCall' in outname): # This is a TF2 model
94.     boxes_idx, classes_idx, scores_idx = 1, 3, 0
95. else: # This is a TF1 model
96.     boxes_idx, classes_idx, scores_idx = 0, 1, 2
97. 
98. # Obtén un token de acceso válido para utilizar la API de Dropbox
99. TOKEN = 'sl.Bfg-B5v-2JGRSkmes8jUFZ-lLjYkD3Dm-
dnUET_W33eev1iNwAViMkqsYI2qCWmAuUTnbKw4DJrqn05eW9jcrvGfQ22nnwABy9bU7tDyYZLQ3_OrQp0V_bWkhgkFeHoZOdmQA
qF9zcY'
100. # Crea una instancia del cliente de Dropbox
101. dbx = dropbox.Dropbox(TOKEN)
102. # Ruta al directorio en Dropbox donde se guardarán las imágenes y los resultados
103. remote_directory = '/Detections'
104. #dbx.files_delete(remote_directory)
105. try:
106.     dbx.files_get_metadata(remote_directory)
107.     dbx.files_delete(remote_directory)
108.     # print(f"Directorio {remote_directory} eliminado correctamente")
109. except dropbox.exceptions.ApiError as e:
110.     error = e.error
111.     if isinstance(error, dropbox.files.GetMetadataError) and error.is_path():
112.         if error.get_path().is_not_found():
113.             print(f"El directorio {remote_directory} no existe")
114.         else:
115.             print(f"Error al obtener metadatos del directorio {remote_directory}: {error}")
116.     else:
117.         print(f"Error desconocido al eliminar el directorio {remote_directory}: {e}")
```

```
118.
119. def hilo():
120.     while True:
121.         print(f"\nHILO1: Presiona la tecla 'Enter' para capturar imagenes")
122.         input()  # Esperar a que se presione la tecla Entre
123.         capture_images(10)
124.
125. imagen_queue = queue.Queue(maxsize=0)
126. mutex = threading.Lock()
127. capture_thread = threading.Thread(
128.         target=hilo, args=()
129.     )
130. capture_thread.start()
131.
132. # Función para capturar una imagen de la cámara web
133. def capture_image():
134.     cap = cv2.VideoCapture(0)  # Abrir la cámara web
135.     ret, frame = cap.read()  # Capturar un frame de video
136.     cap.release()  # Liberar la cámara
137.
138.     return frame
139.
140. # Función para guardar la imagen en un archivo
141. def save_image(image, file_name):
142.     cv2.imwrite(file_name, image)
143.
144. # Capturar imágenes después de pulsar una tecla y esperar 10 segundos entre cada captura
145. def capture_images(num_images):
146.     for i in range(num_images):
147.         if i > 0:
148.             print("HILO1: Esperando 0.5 segundos...")
149.             time.sleep(0.5)
150.         image = capture_image()  # Capturar imagen de la cámara web
151.         imagen_queue.put(image,timeout=None)
152.
153. def process_images():
154.     global j
155.     image = imagen_queue.get(block=True, timeout=None)
156.     image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
157.     imH, imW, _ = image.shape
158.     image_resized = cv2.resize(image_rgb, (width, height))
159.     input_data = np.expand_dims(image_resized, axis=0)
160.     # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
161.     if floating_model:
162.         input_data = (np.float32(input_data) - input_mean) / input_std
163.     # Perform the actual detection by running the model with the image as input
164.     interpreter.set_tensor(input_details[0]['index'],input_data)
165.     interpreter.invoke()
166.
167.     # Retrieve detection results
168.     boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding box
coordinates of detected objects
169.     classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class index of
detected objects
170.     scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] # Confidence of
detected objects
171.     detections = []
172.         # Loop over all detections and draw detection box if confidence is above minimum
threshold
173.     for i in range(len(scores)):
174.         if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
175.                 # Get bounding box coordinates and draw box
176.                 # Interpreter can return coordinates that are outside of image dimensions, need
to force them to be within image using max() and min()
177.                 ymin = int(max(1,(boxes[i][0] * imH)))
178.                 xmin = int(max(1,(boxes[i][1] * imW)))
179.                 ymax = int(min(imH,(boxes[i][2] * imH)))
180.                 xmax = int(min(imW,(boxes[i][3] * imW)))
181.                 cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
182.                 # Draw label
```

```
                object_name = labels[int(classes[i])] # Look up object name from "labels" array
using class index
                label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
# Get font size
                label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too
close to top of window
                cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0],
label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
                cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 0), 2) # Draw label text
                detections.append([object_name, scores[i], xmin, ymin, xmax, ymax])

    # Get filenames and paths
    image_fn = os.path.basename("/home/pi/TFG-LynxIBDetect")
    image_savepath = os.path.join(CWD_PATH,RESULTS_DIR,image_fn)

    base_fn, ext = os.path.splitext(image_fn)
    txt_result_fn = base_fn +'.txt'
    txt_savepath = os.path.join(CWD_PATH,RESULTS_DIR,txt_result_fn)
    if not detections==[]: # Si hay detecciones subo. FILTRO
        # Save image with bounding boxes and labels
        file_name = f'/home/pi/TFG-LynxIBDetect/Detections/image{j+1}.jpg'
        save_image(image, file_name)  # Guardar imagen en un archivo
        # Sube la imagen a Dropbox
        with open(file_name, 'rb') as f:
            remote_filename = f'{remote_directory}/image{j+1}.jpg'
            #print(remote_filename)
            dbx.files_upload(f.read(), remote_filename)
        # Elimina el archivo local
        os.remove(file_name)
        # Guarda los resultados de detección en un archivo
        # (asumiendo que los resultados están en la variable 'detections')
        detections_env = []
        detections_env.append('\n'.join(map(str, detections)))
        results = ', '.join(detections_env)
        results_filename = f'/home/pi/TFG-LynxIBDetect/Detections/results_image{j+1}.txt'
        with open(results_filename, 'w') as f:
            f.write(results)
        # Sube el archivo de resultados a Dropbox
        with open(results_filename, 'rb') as f:
            remote_results_filename = f'{remote_directory}/results_image{j+1}.txt'
            dbx.files_upload(f.read(), remote_results_filename)
        # Elimina el archivo local
        os.remove(results_filename)
        j=1+j

# Llamar a la función para capturar una imagen después de pulsar una tecla y esperar 10
segundos entre cada captura
j=0
while(True):
    process_images()

```

ANEXO D: CÓDIGO INTERFAZ USUARIO

Código escrito en Python para la interfaz de usuario. Se detalla su funcionamiento en el **apartado 4.1.2.3.**

Disponible en: https://github.com/jgamero-ibrobotics/TFG-LynxIBDetect/blob/main/Interfaz.py

```
# Código para la interfaz gráfica de la aplicación LynxDetect
# permite visualizar todas las imágenes que has resultado positivas
# en la detección las cuales se encuentran en la carpeta compartida
# de Dropbox. Además, se muestra la información de cada imagen y
# la información total de todas las imágenes detectadas.

# Creado: 10 May 2023
# Última modificación: 02 jun 2023

# @author: Jesús Gamero Borrego

#-----------------------------------------------------------------------------------------

import sys
import os
import dropbox
from dropbox.exceptions import AuthError
from PyQt5.QtWidgets import QApplication, QLabel, QVBoxLayout, QWidget, QPushButton,
QHBoxLayout, QMessageBox,QStyleFactory
from PyQt5.QtGui import QPixmap, QFont
from PyQt5.QtCore import Qt, QTimer

class ImageGallery(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('LynxDetect')
        self.images = []
        self.txt_files = []
        self.current_image_index = 0

        self.layout = QVBoxLayout()
        self.button_layout = QHBoxLayout()

        self.previous_button = QPushButton("Anterior")
        self.next_button = QPushButton("Siguiente")

        self.previous_button.clicked.connect(self.show_previous_image)
        self.next_button.clicked.connect(self.show_next_image)

        self.button_layout.addWidget(self.previous_button)
        self.button_layout.addWidget(self.next_button)

        self.layout.addLayout(self.button_layout)
        self.setLayout(self.layout)

        self.client = dropbox.Dropbox("sl.Bfg-B5v-2JGRSkmes8jUFZ-lLjYkD3Dm-
dnUET_W33eev1iNwAViMkqsYI2qCWmAuUTnbKw4DJrqn05eW9jcrvGfQ22nnwABy9bU7tDyYZLQ3_OrQp0V_bWkhgkFeHoZOdmQA
qF9zcY")

        self.info_total = ""
        self.total_class_counts = {
                "lynx pardinus": 0,
                "fox": 0,
                "rabbit": 0,
                "cat": 0,
                "person": 0,
                "person-like": 0
            }

        self.timer = QTimer()
        self.timer.timeout.connect(self.check_updates)
```

```
59.         self.timer.start(5000)  # Verificar actualizaciones cada 5 segundos
60.
61.         self.load_images()
62.
63.         if len(self.images) > 0:
64.             self.show_current_image()
65.
66.     def load_images(self):
67.         try:
68.             response = self.client.files_list_folder("/Detections")
69.             for entry in response.entries:
70.                 if isinstance(entry, dropbox.files.FileMetadata) and
entry.name.lower().endswith(".jpg"):
71.                     image_path = f"{entry.path_display}"
72.                     image_name = os.path.splitext(entry.name)[0]
73.                     txt_path = f"/Detections/results_{image_name}.txt"
74.                     self.info_total = self.process_txt_file_total(txt_path)
75.
76.                     if image_path not in self.images:  # Evitar duplicados
77.                         self.images.append(image_path)
78.                         self.txt_files.append(txt_path)
79.                 # if isinstance(entry, dropbox.files.FileMetadata) and
entry.name.lower().endswith(".txt"):
80.
81.         except AuthError:
82.             QMessageBox.warning(self, "Error", "Error de autenticación. Por favor, verifica tu
token de acceso.")
83.
84.     def show_current_image(self):
85.         if len(self.images) > 0:
86.             try:
87.                 _, response = self.client.files_download(self.images[self.current_image_index])
88.                 data = response.content
89.                 pixmap = QPixmap()
90.                 pixmap.loadFromData(data)
91.
92.                 label = QLabel()
93.                 label.setPixmap(pixmap)
94.
95.                 filename = os.path.basename(self.images[self.current_image_index])
96.                 filename_label = QLabel(filename)
97.                 filename_label.setStyleSheet("font-size: 23px;  font-weight: bold;")
98.
99.                 info = self.process_txt_file(self.txt_files[self.current_image_index])
100.                info_label = QLabel()
101.                info_label.setText(f"Información:\n{info}")
102.                info_label.setStyleSheet("font-size: 16px;")
103.
104.                info_total_label = QLabel()
105.                info_total_label.setText(f"Información Total:\n{self.info_total}")
106.                info_total_label.setStyleSheet("font-size: 16px;")
107.
108.                # Eliminar cualquier widget existente antes de agregar la imagen, nombre de
archivo e información
109.                while self.layout.count():
110.                    item = self.layout.takeAt(0)
111.                    widget = item.widget()
112.                    if widget:
113.                        widget.deleteLater()
114.
115.                self.layout.addWidget(filename_label)
116.                self.layout.addWidget(label)
117.                self.layout.addWidget(info_label)
118.                self.layout.addWidget(info_total_label)
119.                self.layout.addLayout(self.button_layout)  # Agregar el diseño de los botones
al layout principal
120.
121.            except dropbox.exceptions.HttpError:
122.                QMessageBox.warning(self, "Error", "No se pudo descargar la imagen.")
123.
124.    def process_txt_file(self, txt_file):
```

```
125.         try:
126.             _, response = self.client.files_download(txt_file)
127.             content = response.content.decode("utf-8")
128.             lines = content.split("\n")
129.             class_counts = {
130.                 "lynx pardinus": 0,
131.                 "fox": 0,
132.                 "rabbit": 0,
133.                 "cat": 0,
134.                 "person": 0,
135.                 "person-like": 0
136.             }
137.             for line in lines:
138.                 line = line.strip()  # Ignorar líneas vacías
139.                 if line:
140.                     class_name = line.split(",")[0].strip().strip("[]").replace("'",
"").lower()
141.                     class_counts[class_name] += 1
142.             info = ""
143.             for class_name, count in class_counts.items():
144.                 if count > 0:
145.                     info += f"{class_name}: {count}\t"
146.             return info
147.         except dropbox.exceptions.HttpError:
148.             return ""
149.
150.     def process_txt_file_total(self, txt_file):
151.         try:
152.             _, response = self.client.files_download(txt_file)
153.             content = response.content.decode("utf-8")
154.             lines = content.split("\n")
155.             for line in lines:
156.                 line = line.strip()  # Ignorar líneas vacías
157.                 if line:
158.                     class_name = line.split(",")[0].strip().strip("[]").replace("'",
"").lower()
159.                     self.total_class_counts[class_name] += 1
160.             self.info_total = ""
161.             for class_name, count in self.total_class_counts.items():
162.                 if count > 0:
163.                     self.info_total += f"{class_name}: {self.total_class_counts[class_name]}\t"
164.             return self.info_total
165.         except dropbox.exceptions.HttpError:
166.             return ""
167.
168.     def show_previous_image(self):
169.         if self.current_image_index > 0:
170.             self.current_image_index -= 1
171.             self.show_current_image()
172.
173.     def show_next_image(self):
174.         if self.current_image_index < len(self.images) - 1 and self.current_image_index <
len(self.txt_files) - 1:
175.             self.current_image_index += 1
176.             self.show_current_image()
177.
178.     def check_updates(self):
179.         # Verificar si hay cambios en la carpeta compartida
180.         try:
181.             response = self.client.files_list_folder('/Detections')
182.             updated_images = []
183.             updated_txt_files = []
184.             for entry in response.entries:
185.                 if isinstance(entry, dropbox.files.FileMetadata) and
entry.name.lower().endswith('.jpg'):
186.                     download_path = f'{entry.path_display}'
187.                     if download_path not in self.images:
188.                         updated_images.append(download_path)
189.                 if isinstance(entry, dropbox.files.FileMetadata) and
entry.name.lower().endswith('.txt'):
```

```
                    download_path = f'{entry.path_display}'
                    if download_path not in self.txt_files:
                        self.info_total = self.process_txt_file_total(download_path)
                        updated_txt_files.append(download_path)

            if updated_images:
                self.images.extend(updated_images)
                self.txt_files.extend(updated_txt_files)
                self.show_current_image()

        except dropbox.exceptions.AuthError:
            QMessageBox.warning(self, 'Error', 'Error de autenticación. Por favor, verifica tu token de acceso.')

if __name__ == "__main__":
    app = QApplication(sys.argv)
    gallery = ImageGallery()
    app.setStyle(QStyleFactory.create("Fusion"))
    gallery.show()
    sys.exit(app.exec_())

```

ANEXO E: PASOS PARA EL ENTRENAMIENTO EN LA NUBE CON GOOGLE COLAB

Paso	Código	Explicación
Instalación de las dependencias de TensorFlow		
1	!git clone --depth 1 https://github.com/tensorflow/models	Clonamos el repositorio de modelos de tensorflow
2	%%bash cd models/research/ protoc object_detection/protos/*.proto --python_out=. cp object_detection/packages/tf2/setup.py	El código compila archivos .proto utilizando protoc y genera código Python en el directorio actual.
3	import re with open('/content/models/research/object_detection/packages/tf2/setup.py') as f: s = f.read() with open('/content/models/research/setup.py', 'w') as f: s = re.sub('tf-models-official>=2.5.1', 'tf-models-official==2.8.0', s) f.write(s)	El código realiza modificaciones en el archivo setup.py para instalar la versión 2.8.0 de tf-models-official.
4	!pip install /content/models/research/ !pip install tensorflow==2.8.0	El primero instala la API de Detección de Objetos, mientras el segundo código instala TensorFlow en la versión 2.8.0.
Preparación de los datos para el entrenamiento		
1	from google.colab import drive drive.mount('/content/gdrive') !cp /content/gdrive/MyDrive/train.zip /content !cp /content/gdrive/MyDrive/validation.zip /content !cp /content/gdrive/MyDrive/test.zip /content !mkdir /content/images !unzip -q test.zip -d /content/images !unzip -q train.zip -d /content/images !unzip -q validation.zip -d /content/images	El código monta Google Drive y copia los archivos train.zip, validation.zip y test.zip desde Google Drive a la carpeta /content. Además, los descomprime y los introduce en la carpeta content/images.
2	%%bash cat <<EOF >> /content/labelmap.txt person-like person Lynx pardinus Fox	El código agrega etiquetas al archivo labelmap.txt.

	Rabbit Cat EOF	
3	! wget https://raw.githubusercontent.com/jgamero-ibrobotics/TFG-LynxIBDetect/master/util_scripts/create_csv.py ! wget https://raw.githubusercontent.com/jgamero-ibrobotics/TFG-LynxIBDetect/master/util_scripts/create_tfrecord.py	El código descarga dos scripts de utilidad desde direcciones URL específicas.
4	!python3 create_csv.py !python3 create_tfrecord.py --csv_input=images/train_labels.csv --labelmap=labelmap.txt --image_dir=images/train --output_path=train.tfrecord !python3 create_tfrecord.py --csv_input=images/validation_labels.csv --labelmap=labelmap.txt --image_dir=images/validation --output_path=val.tfrecord	El código ejecuta scripts de Python para crear archivos CSV y TFRecord a partir de imágenes y etiquetas de entrenamiento y validación.
5	train_record_fname = '/content/train.tfrecord' val_record_fname = '/content/val.tfrecord' label_map_pbtxt_fname = '/content/labelmap.pbtxt'	Se asignan rutas de archivo a tres variables: train_record_fname, val_record_fname y label_map_pbtxt_fname.
Establecer la configuración del entrenamiento		

1	chosen_model = 'ssd-mobilenet-v2-fpnlite-320'
	
	MODELS_CONFIG = {
	'ssd-mobilenet-v2': {
	'model_name': 'ssd_mobilenet_v2_320x320_coco17_tpu-8',
	'base_pipeline_file': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.config',
	'pretrained_checkpoint': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz',
	},
	'efficientdet-d0': {
	'model_name': 'efficientdet_d0_coco17_tpu-32',
	'base_pipeline_file': 'ssd_efficientdet_d0_512x512_coco17_tpu-8.config',
	'pretrained_checkpoint': 'efficientdet_d0_coco17_tpu-32.tar.gz',
	},
	'ssd-mobilenet-v2-fpnlite-320': {
	'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
	'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
	'pretrained_checkpoint': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
	},
	'custom-model': {
	'model_name': 'custom_model',
	'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
	'pretrained_checkpoint': 'custom_model.zip',
	},
	}
	
	model_name = MODELS_CONFIG[chosen_model]['model_name']
	pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
	base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']
		Código define un diccionario con configuraciones de diferentes modelos de detección de objetos. Luego, se selecciona un modelo específico utilizando la variable chosen_model y se extraen sus valores correspondientes.

2	# Create "mymodel" folder for holding pre-trained weights and configuration files %mkdir /content/models/mymodel/ %cd /content/models/mymodel/	El código crea un directorio llamado "mymodel" para almacenar los archivos del modelo. Luego, descarga los pesos pre-entrenados del

	# Download pre-trained model weights import tarfile download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' + pretrained_checkpoint !wget {download_tar} tar = tarfile.open(pretrained_checkpoint) tar.extractall() tar.close() # Download training configuration file for model download_config = 'https://raw.githubusercontent.com/tensorflow/models/master/research/object_ detection/configs/tf2/' + base_pipeline_file !wget {download_config}	modelo y extrae los archivos del archivo comprimido. Finalmente, descarga el archivo de configuración de entrenamiento del modelo.

3	num_steps = 60000
	
	if chosen_model == 'efficientdet-d0':
	  batch_size = 4
	else:
	  batch_size = 16
		El código define el número de pasos de entrenamiento (num_steps) y el tamaño de lote (batch_size) según el modelo seleccionado.

4	pipeline_fname = '/content/models/mymodel/' + base_pipeline_file
	fine_tune_checkpoint = '/content/models/mymodel/' + model_name +
	'/checkpoint/ckpt-0'
	
	def get_num_classes(pbtxt_fname):
	  from object_detection.utils import label_map_util
	  label_map = label_map_util.load_labelmap(pbtxt_fname)
	  categories = label_map_util.convert_label_map_to_categories(
	  label_map, max_num_classes=90, use_display_name=True)
	  category_index = label_map_util.create_category_index(categories)
	  return len(category_index.keys())
	num_classes = get_num_classes(label_map_pbtxt_fname)
	print('Total classes:', num_classes)
		El código asigna rutas de archivo a pipeline_fname y fine_tune_checkpoint. A continuación, utiliza una función llamada get_num_classes para obtener el número de clases del archivo de mapa de etiquetas. El número de clases se muestra en la salida.

5	import re
	
	%cd /content/models/mymodel
	print('writing custom configuration file')
	
	with open(pipeline_fname) as f:
	  s = f.read()
	with open('pipeline_file.config', 'w') as f:
	
	  # Set fine_tune_checkpoint path
	  s = re.sub('fine_tune_checkpoint: ".*?"',
	  'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)
	
	  # Set tfrecord files for train and test datasets
	  s = re.sub(
	  '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")',
	'input_path: "{}"'.format(train_record_fname), s)
	  s = re.sub(
	  '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")',
	'input_path: "{}"'.format(val_record_fname), s)
	
	  # Set label_map_path
	  s = re.sub(
	  'label_map_path: ".*?"', 'label_map_path:
	"{}"'.format(label_map_pbtxt_fname), s)
	
	  # Set batch_size
		El código realiza modificaciones en un archivo de configuración llamado pipeline_fname. Utiliza expresiones regulares para buscar y reemplazar ciertos patrones en el contenido del archivo. Las modificaciones incluyen establecer la ruta del punto de control de afinamiento, las rutas de los archivos tfrecord, la ruta del archivo de mapa de etiquetas, el tamaño del lote, el número de pasos de entrenamiento, el número de clases y otros ajustes específicos para ciertos modelos. El archivo de configuración modificado se guarda en un nuevo archivo llamado "pipeline_file.config".

	s = re.sub('batch_size: [0-9]+',
	'batch_size: {}'.format(batch_size), s)
	
	# Set training steps, num_steps
	s = re.sub('num_steps: [0-9]+',
	'num_steps: {}'.format(num_steps), s)
	
	# Set number of classes num_classes
	s = re.sub('num_classes: [0-9]+',
	'num_classes: {}'.format(num_classes), s)
	
	# Change fine-tune checkpoint type from "classification" to "detection"
	s = re.sub(
	'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)
	
	# If using ssd-mobilenet-v2, reduce learning rate (because it's too high in the default config file)
	if chosen_model == 'ssd-mobilenet-v2':
	s = re.sub('learning_rate_base: .8',
	'learning_rate_base: .08', s)
	
	s = re.sub('warmup_learning_rate: 0.13333',
	'warmup_learning_rate: .026666', s)
	
	# If using efficientdet-d0, use fixed_shape_resizer instead of keep_aspect_ratio_resizer (because it isn't supported by TFLite)
	if chosen_model == 'efficientdet-d0':
	s = re.sub('keep_aspect_ratio_resizer', 'fixed_shape_resizer', s)
	s = re.sub('pad_to_max_dimension: true', '', s)
	s = re.sub('min_dimension', 'height', s)
	s = re.sub('max_dimension', 'width', s)
	
	f.write(s)	

6	pipeline_file = '/content/models/mymodel/pipeline_file.config' model_dir = '/content/training/'	pipeline_file contiene la ruta del archivo de configuración modificado, mientras que model_dir almacena la ruta del directorio de salida para el entrenamiento.
Comenzar el entrenamiento		
7	!python /content/models/research/object_detection/model_main_tf2.py \ --pipeline_config_path={pipeline_file} \ --model_dir={model_dir} \ --alsologtostderr \ --num_train_steps={num_steps} \ --sample_1_of_n_eval_examples=1	El código inicia el entrenamiento del modelo de detección de objetos en TensorFlow utilizando el script model_main_tf2.py. Se proporciona el archivo de configuración y el directorio de salida, junto con otros parámetros opcionales.
Convertir el modelo a Tensorflow lite		
1	!mkdir /content/custom_model_lite2 output_directory = '/content/custom_model_lite2' # Path to training directory (the conversion script automatically chooses the highest checkpoint file)	Se crea un directorio de salida y se ejecuta un script para exportar el modelo entrenado a TensorFlow Lite.

	last_model_path = '/content/training' !python /content/models/research/object_detection/export_tflite_graph_tf2.py \ --trained_checkpoint_dir {last_model_path} \ --output_directory {output_directory} \ --pipeline_config_path {pipeline_file}	
2	import tensorflow as tf converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite2/saved_model') tflite_model = converter.convert() with open('/content/custom_model_lite2/detect.tflite', 'wb') as f: f.write(tflite_model)	Se carga el modelo guardado en formato SavedModel y se convierte a formato TensorFlow Lite. El modelo convertido se guarda en un archivo "detect.tflite".
3	!cp /content/labelmap.txt /content/custom_model_lite !cp /content/labelmap.pbtxt /content/custom_model_lite !cp /content/models/mymodel/pipeline_file.config /content/custom_model_lite %cd /content !zip -r custom_model_lite.zip custom_model_lite from google.colab import files files.download('/content/custom_model_lite.zip')	Comprime los archivos del modelo y los descarga.

ANEXO F: ARCHIVO DE CONFIGURACIÓN PARA EL ENTRENAMIENTO

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 6
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      multiscale_anchor_generator {
        min_level: 3
        max_level: 7
        anchor_scale: 4.0
        aspect_ratios: [1.0, 2.0, 0.5]
        scales_per_octave: 2
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    box_predictor {
      weight_shared_convolutional_box_predictor {
        depth: 128
        class_prediction_bias_init: -4.6
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
          initializer {
            random_normal_initializer {
              stddev: 0.01
              mean: 0.0
            }
          }
          batch_norm {
            scale: true,
```

```
            decay: 0.997,
            epsilon: 0.001,
          }
        }
        num_layers_before_predictor: 4
        share_prediction_tower: true
        use_depthwise: true
        kernel_size: 3
      }
    }
    feature_extractor {
      type: 'ssd_mobilenet_v2_fpn_keras'
      use_depthwise: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
      min_depth: 16
      depth_multiplier: 1.0
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.00004
          }
        }
        initializer {
          random_normal_initializer {
            stddev: 0.01
            mean: 0.0
          }
        }
        batch_norm {
          scale: true,
          decay: 0.997,
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    loss {
      classification_loss {
        weighted_sigmoid_focal {
          alpha: 0.25
          gamma: 2.0
        }
      }
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    normalize_loc_loss_by_codesize: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
```

```
133.   fine_tune_checkpoint_version: V2
134.   fine_tune_checkpoint: "/content/models/mymodel/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-
8/checkpoint/ckpt-0"
135.   fine_tune_checkpoint_type: "detection"
136.   batch_size: 16
137.   sync_replicas: true
138.   startup_delay_steps: 0
139.   replicas_to_aggregate: 8
140.   num_steps: 60000
141.   data_augmentation_options {
142.     random_horizontal_flip {
143.     }
144.   }
145.   data_augmentation_options {
146.     random_crop_image {
147.       min_object_covered: 0.0
148.       min_aspect_ratio: 0.75
149.       max_aspect_ratio: 3.0
150.       min_area: 0.75
151.       max_area: 1.0
152.       overlap_thresh: 0.0
153.     }
154.   }
155.   optimizer {
156.     momentum_optimizer: {
157.       learning_rate: {
158.         cosine_decay_learning_rate {
159.           learning_rate_base: .09
160.           total_steps: 50000
161.           warmup_learning_rate: .036666
162.           warmup_steps: 1000
163.         }
164.       }
165.       momentum_optimizer_value: 0.9
166.     }
167.     use_moving_average: false
168.   }
169.   max_number_of_boxes: 100
170.   unpad_groundtruth_tensors: false
171. }
172.
173. train_input_reader: {
174.   label_map_path: "/content/labelmap.pbtxt"
175.   tf_record_input_reader {
176.     input_path: "/content/train.tfrecord"
177.   }
178. }
179.
180. eval_config: {
181.   metrics_set: "coco_detection_metrics"
182.   use_moving_averages: false
183. }
184.
185. eval_input_reader: {
186.   label_map_path: "/content/labelmap.pbtxt"
187.   shuffle: false
188.   num_epochs: 1
189.   tf_record_input_reader {
190.     input_path: "/content/val.tfrecord"
191.   }
192. }
193.
```

ANEXO G: CÓDIGO PRUEBAS INFERENCIA

Código escrito en Python para el cálculo automático del F1-scores para el conjunto de datos de prueba. Se detalla su funcionamiento en el **apartado 5.1.2.**

Disponible en: https://github.com/jgamero-ibrobotics/TFG-LynxIBDetect/blob/main/F1_Score.py

```
# Código para el cálculo automático del F1 Score
# para un modelo corriendo en TFlite, bajo
# diferentes condiciones luz

# Creado: 29 Abr 2023
# Última modificación: 02 jun 2023

# @author: Jesús Gamero Borrego

#--------------------------------------------------------------------------------------------

import os
import dropbox
import importlib.util
import argparse
import cv2
import numpy as np
import xml.etree.ElementTree as ET
import time

def elimina_no_maximos(matriz, col, indice_preservar):

    for i in range(len(matriz[:,col])):
        if i != indice_preservar:
            matriz[i,col] = 0
    return matriz

def darken_image(image, threshold_brightness, darkness_factor):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    brightness = np.mean(gray_image)

    if brightness < threshold_brightness:
        darkened_image = np.copy(image)
        # 1 es más brillante, 0 es más oscuro
        darkened_image = darkened_image.astype(np.float32) * darkness_factor # Multiplicar por un factor de oscuridad
        darkened_image = np.clip(darkened_image, 0, 255).astype(np.uint8) # Limitar los valores entre 0 y 255
        return darkened_image
    else:
        return image

def read_xml_file(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    references = []
    for obj in root.findall('object'):
        xmin = int(obj.find('bndbox/xmin').text)
        ymin = int(obj.find('bndbox/ymin').text)
        xmax = int(obj.find('bndbox/xmax').text)
        ymax = int(obj.find('bndbox/ymax').text)
        references.append([obj.find('name').text,xmin, ymin, xmax, ymax])

    return references

def apply_detection_model(image):
    global fps
    global elapsed_time
    global total_elapsed_time
    # Cargar la imagen con OpenCV
```

```
    # image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    imH, imW, _ = image.shape
    image_resized = cv2.resize(image_rgb, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)
    # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    start_time = time.time()
    # Perform the actual detection by running the model with the image as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()
    end_time = time.time()

    elapsed_time = end_time - start_time
    fps = 1 / elapsed_time
    total_elapsed_time = total_elapsed_time + elapsed_time

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class index of detected objects
    scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] # Confidence of detected objects

    detections = []

    for i in range(len(scores)):
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            object_name = labels[int(classes[i])] # Look up object name from "labels" array using class index

            detections.append([object_name, scores[i], xmin, ymin, xmax, ymax]) # Add detection data to list

    return detections

def calculate_iou(boxA, boxB):
    # Obtener las coordenadas de los rectángulos
    xA = np.maximum(boxA[0], boxB[0])
    yA = np.maximum(boxA[1], boxB[1])
    xB = np.minimum(boxA[2], boxB[2])
    yB = np.minimum(boxA[3], boxB[3])

    # Calcular el área de la intersección
    intersection_area = np.maximum(0, xB - xA + 1) * np.maximum(0, yB - yA + 1)

    # Calcular el área de los rectángulos
    boxA_area = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxB_area = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # Calcular el coeficiente de IoU
    iou = intersection_area / (boxA_area + boxB_area - intersection_area)

    return iou

########################CODIGO PRINCIPAL#########################

# Define and parse input arguments
parser = argparse.ArgumentParser()
```

```
123. parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
124.                     required=True)
125. parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
126.                     default='detect.tflite')
127. parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
128.                     default='labelmap.txt')
129. parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected
objects',
130.                     default=0.5)
131. parser.add_argument('--image', help='Name of the single image to perform detection on. To run
detection on multiple images, use --imagedir',
132.                     default=None)
133. parser.add_argument('--imagedir', help='Name of the folder containing images to perform
detection on. Folder must contain only images.',
134.                     default=None)
135. parser.add_argument('--save_results', help='Save labeled images and annotation data to a
results folder',
136.                     action='store_true')
137. parser.add_argument('--noshow_results', help='Don\'t show result images (only use this if --
save_results is enabled)',
138.                     action='store_false')
139. parser.add_argument('--darken_image', help='Darken image for testing in low light conditions',
140.                     default=1)
141. parser.add_argument('--archivo_resultados', help='Nombre del archivo donde se guardan los
resultados',
142.                     default='resultados.txt')
143.
144. args = parser.parse_args()
145.
146. # Parse user inputs
147. MODEL_NAME = args.modeldir
148. GRAPH_NAME = args.graph
149. LABELMAP_NAME = args.labels
150.
151. FO = float(args.darken_image)
152.
153. archivo_resultados = '/home/pi/TFG-LynxIBDetect/Resultados/' + args.archivo_resultados
154. archivo_resultados_rendimiento = '/home/pi/TFG-LynxIBDetect/Resultados/' + 'Rendimiento.txt'
155.
156. min_conf_threshold = float(args.threshold)
157.
158. save_results = args.save_results # Defaults to False
159. show_results = args.noshow_results # Defaults to True
160.
161. IM_NAME = args.image
162. IM_DIR = args.imagedir
163. CWD_PATH = os.getcwd() # Get path to current working directory
164. RESULTS_DIR ='results' # Folder to save results images and data to
165.
166. # Import TensorFlow libraries
167. # If tflite_runtime is installed, import interpreter from tflite_runtime, else import from
regular tensorflow
168. pkg = importlib.util.find_spec('tflite_runtime')
169. from tflite_runtime.interpreter import Interpreter
170.
171. # Path to .tflite file, which contains the model that is used for object detection
172. PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)
173. # Path to label map file
174. PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
175. # Load the label map
176. with open(PATH_TO_LABELS, 'r') as f:
177.     labels = [line.strip() for line in f.readlines()]
178.
179. interpreter = Interpreter(model_path=PATH_TO_CKPT) # Cargar el modelo
180. interpreter.allocate_tensors() # Asignar tensores
181.
182. # Get model details
183. input_details = interpreter.get_input_details()
184. output_details = interpreter.get_output_details()
```

```
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

# Check output layer name to determine if this model was created with TF2 or TF1,
# because outputs are ordered differently for TF2 and TF1 models
outname = output_details[0]['name']
if ('StatefulPartitionedCall' in outname): # This is a TF2 model
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # This is a TF1 model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Configuración de Dropbox
access_token =
'sl.Bfir1Hjr9yvmr__nbtKzYVdG46EfOdGtnnXPPUvT7WRkIuREdpDrbwhVokdcCiS7UnNuggUAs5_t4peyFVAvknJtcMo9NJgX
OtOoFckOi9agc-qY2AHEps3iD8hUPNAeqnqWKNFM-cE'
dropbox_folder_xml = '/pruebaXML'
dropbox_folder_jpg = '/pruebaJPG'
# dropbox_folder_xml = '/prueba _lince_pocaluzXML'
# dropbox_folder_jpg = '/prueba _lince_pocaluzJPG'

# Directorio local para descargar los archivos
local_folder = '/home/pi/TFG-LynxIBDetect/IoU'

# Crear la carpeta local si no existe
if not os.path.exists(local_folder):
    os.makedirs(local_folder)

# Crear una instancia del cliente de Dropbox
client = dropbox.Dropbox(access_token)

# Obtener la lista de archivos en la carpeta de Dropbox
file_list = client.files_list_folder(dropbox_folder_xml).entries

for nombre_archivo in os.listdir("/home/pi/TFG-LynxIBDetect/IoU"):
    ruta_archivo = os.path.join("/home/pi/TFG-LynxIBDetect/IoU", nombre_archivo)
    if os.path.isfile(ruta_archivo):
        os.remove(ruta_archivo)

j = 0
k = 0
iou_total = 0
total_elapsed_time = 0
resultados = {
    "person-like": {"TP": 0, "FP": 0, "FN": 0},
    "person": {"TP": 0, "FP": 0, "FN": 0},
    "Lynx pardinus": {"TP": 0, "FP": 0, "FN": 0},
    "Fox": {"TP": 0, "FP": 0, "FN": 0},
    "Rabbit": {"TP": 0, "FP": 0, "FN": 0},
    "Cat": {"TP": 0, "FP": 0, "FN": 0},
}
size = len(file_list)
print("Número de archivos:", size)

with open(archivo_resultados_rendimiento, 'w') as file:
    file.write('Imagen\tTiempo de procesamiento (s)\tFPS\n')
    for file_entry in file_list:
        # Obtener el nombre del archivo
        file_name = file_entry.name
        print("Archivo: ", file_name)

        # Verificar si es un archivo XML
        if file_name.endswith('.xml'):
            # Descargar el archivo XML desde Dropbox
            xml_path = os.path.join(local_folder, file_name)
            client.files_download_to_file(xml_path, f'{dropbox_folder_xml}/{file_name}')
```

```
253.
254.             # Leer las coordenadas de boxA desde el archivo XML
255.             references = read_xml_file(xml_path)
256.
257.             # Obtener el nombre del archivo de imagen relacionado
258.             image_file = file_name.replace('.xml', '.jpg')
259.
260.             # Descargar el archivo JPG desde Dropbox
261.             image_path = os.path.join(local_folder, image_file)
262.             client.files_download_to_file(image_path, f'{dropbox_folder_jpg}/{image_file}')
263.             # Aplicar el modelo de detección y obtener las predicciones de boxB
264.             image = cv2.imread(image_path)
265.
266.
267.
268.             image = darken_image(image, 255,
FO)
269.             #cv2.imwrite("/home/pi/TFG-LynxIBDetect/IoU/image_darken.jpg", image)
270.
271.             detections = apply_detection_model(image)
272.
273.             file.write(f'{k + 1}\t{elapsed_time}\t{fps}\n')
274.
275.
276.             # Elimina el archivo local
277.             os.remove(image_path)
278.             os.remove(xml_path)
279.
280.             iou_scores = []
281.             max_iou_scores = []
282.             clases_iou_max = []
283.             clases_referencia = []
284.
285.             num_detections = len(detections)
286.             num_references = len(references)
287.
288.             print("numero de referencias")
289.             print(len(references))
290.             print("numero de detecciones")
291.             print(len(detections))
292.
293.
294.             # asigno a cada box de referencia un coeficiente maximo de IoU
295.             if detections == []: # si no hay detecciones
296.                 for i in range(len(references)):
297.                     detections.append(["NONE", 0, 0, 0, 0, 0])
298.
299.             elif len(detections) < len(references):
300.                 for i in range(len(references)):
301.                     if i < len(detections):
302.                         continue
303.                     else:
304.                         detections.append(["NONE", 0, 0, 0, 0, 0])
305.
306.
307.             matrix_iou = np.zeros((len(detections), len(references)))
308.             col = 0
309.             for reference in references: # recorre cada box en el archivo xml
310.                 boxA = [reference[1],  reference[2],  reference[3],  reference[4]] # [xmin,
ymin, xmax, ymax]
311.                 print('box Referencia')
312.                 print(boxA)
313.                 cv2.rectangle(image, (boxA[0],  boxA[1]), (boxA[2],  boxA[3]), (0, 255, 0), 2)
# ground truth box
314.                 fil = 0
315.                 for detection in detections: # recorre cada box por objeto detectado
316.                     boxB = [detection[2],  detection[3],  detection[4],  detection[5]] # [xmin,
ymin, xmax, ymax]
317.                     cv2.rectangle(image, (boxB[0],  boxB[1]), (boxB[2],  boxB[3]), (0, 0, 255),
2) # detection box
318.                     print('box Detección')
```

```
                    print(boxB)
                    # Calcular el coeficiente de IoU
                    iou_score = calculate_iou(boxA, boxB)
                    matrix_iou[fil][col] = iou_score
                    fil = fil + 1
                col = col + 1

            iou_max_score = []
            for col in range(num_references):
                column = matrix_iou[:,col]
                iou_max_index_col  = np.argmax(column)
                row = matrix_iou[iou_max_index_col,:]
                iou_max_index_row  = np.argmax(row)
                if [iou_max_index_col, iou_max_index_row] == [iou_max_index_col, col]: # si el
maximo IoU de la columna es el mismo que el maximo IoU de la fila
                    iou_max_score.append(column[iou_max_index_col])
                    clases_referencia.append(references[col][0])
                    clases_iou_max.append(detections[iou_max_index_col][0])
                    matrix_iou = elimina_no_maximos(matrix_iou, col, iou_max_index_col)
                else:
                    while [iou_max_index_col, iou_max_index_row] != [iou_max_index_col, col]:
                        column[iou_max_index_col] = 0
                        iou_max_index_col  = np.argmax(column)

                        row = matrix_iou[iou_max_index_col,:]
                        iou_max_index_row  = np.argmax(row)
                        if np.all(column == 0.0): # no se han producido detecciones para esta
referencia
                            break
                    matrix_iou = elimina_no_maximos(matrix_iou, col, iou_max_index_col)
                    iou_max_score.append(column[iou_max_index_col])
                    clases_referencia.append(references[col][0])
                    clases_iou_max.append(detections[iou_max_index_col][0])

            print("El coeficiente de IoU es:", iou_max_score)

            # menos detecciones de las esperadas, hay falsos negativos
            # igual numero de detecciones que de referencias, no hay falsos negativos
            if len(references) >= num_detections:
                for i in range(1,len(references)+1):
                    if i <= num_detections: # solo contabiliza hasta las detecciones que hay
                        # tomamos primero las referencias con mayor iou ya que son las que
tienen
                        # mayor probabilidad de ser correctas el resto seran falsos negativos
                        iou_index  = np.argmax(iou_max_score) # indice del maximo coeficiente
de IoU
                        iou = iou_max_score[iou_index] # maximo coeficiente de IoU
                        clase = clases_iou_max[iou_index] # clase del objeto detectado con el
maximo coeficiente de IoU
                        clase_ref = clases_referencia[iou_index] # clase de la referencia con
el maximo coeficiente de IoU

                        clases_iou_max.pop(iou_index)
                        iou_max_score.pop(iou_index)

                        if iou > 0.5 and clase_ref == clase: #Verdadero Positivo
                            resultados[clase_ref]["TP"] += 1
                        elif clase_ref != clase and iou > 0.5: #Falso Positivo
                            resultados[clase_ref]["FP"] += 1
                        else: #Falso Negativo
                            resultados[clase_ref]["FN"] += 1
                    # no hay mas detecciones, hay falsos negativos
                    else:
                        # tomamos primero las referencias con mayor iou ya que son las que
tienen mayor probabilidad de ser correctas
                        # el resto seran falsos negativos
                        iou_index  = np.argmax(iou_max_score) # indice del maximo coeficiente
de IoU
                        clase_ref = references[iou_index][0] # esta mal

```

```
                        clases_iou_max.pop(iou_index)
                        iou_max_score.pop(iou_index)

                        resultados[clase_ref]["FN"] += 1 # asigna un falso negativo a la clase de la referencia

            # mas detecciones de las esperadas, hay falsos positivos
            else:
                for i in range(1,num_detections+1):
                    if i <= len(references): # solo contabiliza hasta las referencias que hay

                        # tomamos primero las referencias con mayor iou ya que son las que tienen
                        # mayor probabilidad de ser correctas el resto seran falsos positivos
                        iou_index  = np.argmax(iou_max_score) # indice del maximo coeficiente de IoU
                        iou = iou_max_score[iou_index] # maximo coeficiente de IoU
                        clase = clases_iou_max[iou_index] # clase del objeto detectado con el maximo coeficiente de IoU
                        clase_ref = clases_referencia[iou_index]

                        clases_iou_max.pop(iou_index)
                        iou_max_score.pop(iou_index)

                        if iou > 0.5 and clase_ref == clase: #Verdadero Positivo
                            resultados[clase_ref]["TP"] += 1
                        elif clase_ref != clase and iou > 0.5: #Falso Positivo
                            resultados[clase_ref]["FP"] += 1
                        else: #Falso Negativo
                            resultados[clase_ref]["FN"] += 1

                    # no hay mas referencias, hay falsos positivos
                    else:
                        for i in range(num_detections):
                            if all(matrix_iou[i,:]) == all(np.zeros(num_references)):
                                resultados[detections[i][0]]["FP"] += 1 # asigna un falso positivo a la clase de la deteccion

            k = k + 1

            # # file_name = f'/home/pi/TFG-LynxIBDetect/IoU/image_IoU_{j+1}.jpg'
            file_name = f'/home/pi/TFG-LynxIBDetect/IoU/image_IoU.jpg'
            j=j+1
            cv2.imwrite(file_name,image)  # Guardar imagen en un archivo

            #print(f"Archivo XML: {file_name}")
            print("-------------------------------------------")
    file.write(f'Total\t{total_elapsed_time}\t\n')
# Calcular los valores totales de TP, FP, FN
Tp = sum(conteos["TP"] for conteos in resultados.values())
Fp = sum(conteos["FP"] for conteos in resultados.values())
Fn = sum(conteos["FN"] for conteos in resultados.values())

print("   ")
print("-------------RESULTADOS POR CLASE-------------")
for clase, conteos in resultados.items():
    print(clase)
    print("Verdaderos Positivos (TP):", conteos["TP"])
    print("Falsos Positivos (FP):", conteos["FP"])
    print("Falsos Negativos (FN):", conteos["FN"])

    precision = conteos["TP"]/(conteos["TP"]+conteos["FP"])
    recall = conteos["TP"]/(conteos["TP"]+conteos["FN"])
    f1_score = 2*((precision*recall)/(precision+recall))

    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1_score)
    print("-------------------------------------------")

precision = Tp/(Tp+Fp)
```

```
recall = Tp/(Tp+Fn)
f1_score = 2*((precision*recall)/(precision+recall))

print("   ")
print("--------------RESULTADOS GLOBALES--------------")
print("Verdaderos Positivos:", Tp)
print("Falsos Positivos:", Fp)
print("Falsos Negativos:", Fn)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1_score)
print("-----------------------------------------------")

with open(archivo_resultados, "w") as file:
    file.write("\n")
    file.write("--------------RESULTADOS POR CLASE--------------\n")
    for clase, conteos in resultados.items():
        file.write(clase + "\n")
        file.write("Verdaderos Positivos (TP): {}\n".format(conteos["TP"]))
        file.write("Falsos Positivos (FP): {}\n".format(conteos["FP"]))
        file.write("Falsos Negativos (FN): {}\n".format(conteos["FN"]))

        precision = conteos["TP"] / (conteos["TP"] + conteos["FP"])
        recall = conteos["TP"] / (conteos["TP"] + conteos["FN"])
        f1_score = 2 * ((precision * recall) / (precision + recall))

        file.write("Precision: {}\n".format(precision))
        file.write("Recall: {}\n".format(recall))
        file.write("F1 Score: {}\n".format(f1_score))
        file.write("-----------------------------------------------\n")

    file.write("\n")
    file.write("--------------RESULTADOS GLOBALES--------------\n")
    file.write("Verdaderos Positivos: {}\n".format(Tp))
    file.write("Falsos Positivos: {}\n".format(Fp))
    file.write("Falsos Negativos: {}\n".format(Fn))

    precision = Tp / (Tp + Fp)
    recall = Tp / (Tp + Fn)
    f1_score = 2 * ((precision * recall) / (precision + recall))

    file.write("Precision: {}\n".format(precision))
    file.write("Recall: {}\n".format(recall))
    file.write("F1 Score: {}\n".format(f1_score))
    file.write("-----------------------------------------------\n")

```