universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

VISIÓN POR COMPUTADOR PARA LA MONITORIZACIÓN DE REBAÑOS MEDIANTE IMÁGENES AÉREAS

Autor: Martín Suárez Fernández

Tutora: Lidia Sánchez González

(Junio, 2023)

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

ALUMNO: Martín Suárez Fernández

TUTORA: Lidia Sánchez González

TÍTULO: Visión por computador para la monitorización de rebaños mediante imágenes aéreas

CONVOCATORIA: Junio, 2023

RESUMEN:

El presente Trabajo de Fin de Grado refleja el proceso de desarrollo llevado a cabo para la obtención de un modelo de visión artificial mediante el cual poder llevar a cabo la monitorización en tiempo real de rebaños con la utilización de imágenes aéreas. El trabajo realizado está conformado por tres etapas principales, siendo la primera un estudio exhaustivo de las mejores y más nuevas tecnologías del ámbito de la detección de animales así como de conjuntos de datos relacionados con el problema, la segunda la elección y ajuste de los mejores modelos encontrados durante la fase de estudio, y la última una fase de evaluación en la que se llevan a cabo una serie de experimentos con los modelos YOLOv8 y YOLO-NAS sobre varios conjuntos de datos para posteriormente, realizar una comparación del desempeño de los modelos con el fin de poder determinar qué modelo trabaja mejor sobre qué condiciones.

Palabras clave: Detección de objetos, visión por computador, deep learning, monitorización de animales, YOLO.

ABSTRACT:

This Final Degree Project shows the process taken in order to obtain a computer vision model through which to carry out real time herd monitorization using aerial imagery. The work carried out consists of three main stages, the first one being an exhaustive study of the best and newest technologies in the field of animal detection as well as related datasets, the second one being the selection and fine tuning of the best models found during the study phase, and the last one being an evaluation phase in which a series of experiments are carried out using the YOLOv8 and YOLO-NAS models on several datasets to later compare the performance of the models in order to determine which model works best under which conditions.

Índice general

Índice de figuras

Índice de tablas

Glosario de términos

Arquitectura : Una arquitectura de una red neuronal es el diseño mediante el que se definen las capas de la red y cómo se conectan entre ellas.

Bounding box : Es el término en inglés utilizado para referirse a los rectángulos que determinan la región que ocupa un objeto en una imagen. Estos no tienen porque contener únicamente el objeto pero sí han de determinar los límites del mismo.

Dataset : Se trata de un conjunto de datos con una serie de características específicas y generalmente organizado.

Deep learning : El Deep learning es un campo incluido dentro del *Machine learning* basado en el uso de redes neuronales con un gran número de nodos, conexiones y capas.

Entorno (Anaconda) : Un entorno de Anaconda es una carpeta en la que se almacenan diferentes paquetes y dependencias de manera que se pueden gestionar y utilizar por separado sin que interfieran unos con otros.

Entrenamiento : Proceso mediante el cual se utiliza un dataset para modificar los pesos de un modelo con el fin de ajustar la salida de tal manera que se pueda realizar la tarea que se pretende con el mejor desempeño posible.

Etiqueta : La etiqueta, en el ámbito del aprendizaje automático, es aquella variable que se intenta predecir.

FPS : Las siglas significan *Fotogramas Por Segundo*, y se suele utilizar como medida de la frecuencia con la que se muestran imágenes, pero en este proyecto se utiliza también como medida de la frecuencia con la que un modelo procesa imágenes.

Librería : Una librería es un conjunto de funciones de funciones definidas previamente y que facilitan el proceso de desarrollo evitando el tener que implementar dichas funciones.

Métrica : Una métrica es un valor utilizado para medir algún aspecto del rendimiento de un modelo de aprendizaje automático.

Visión por computador : También denominada visión artificial, es una disciplina parte del ámbito de la inteligencia artificial asociado al análisis y procesamiento de imágenes. Se centra en conseguir que el computador sea capaz de realizar el mismo proceso que lleva a cabo un humano al ver y extraer información de una imagen.

Introducción

La presente sección expone las razones por las que se ha seleccionado el problema planteado y como se ha afrontado dicho problema.

Planteamiento del problema

Durante los últimos años, se ha producido un avance considerable en el ámbito de la inteligencia artificial, pues la popularidad que ha ganado ha llevado a una mayor inversión tanto en el apartado de software como en el de hardware. La inversión en hardware ha llevado a la creación de computadores más potentes ya sea por la agrupación de un mayor número de tarjetas gráficas o por el desarrollo de nuevas arquitecturas como se puede observar en el caso de Nvidia, principal compañía desarrolladora de hardware destinado a la inteligencia artificial, quien ha desarrollado las arquitecturas Pascal, Turing, Ampere y Hopper en los últimos siete años.

La evolución del hardware mencionada previamente ha proporcionado a los desarrolladores de modelos de inteligencia artificial la oportunidad de probar nuevos modelos que requieren una capacidad de cómputo mucho mayor comparados con los modelos desarrollados durante la década anterior. Esto, en el ámbito de la visión artificial, la cual requiere de una gran capacidad de cómputo, resulta ser un factor de gran importancia, pues el procesamiento de imágenes se puede llevar a cabo de manera mucho más rápido en la actualidad.

Por otra parte, a la vez que se mejoraban las tecnologías relacionadas a la inteligencia artificial, se producía un aumento considerable en la cantidad de datos recogidos en todos los ámbitos. Esto resulta ser un factor igual o más importante que los avances mencionados anteriormente, pues para el desarrollo de cualquier modelo es necesario tener los datos suficientes y con la calidad adecuada para llevar a cabo un entrenamiento que permita obtener las características adecuadas con el fin de generalizar de la mejor manera posible.

Así pues, las mejoras planteadas desvelan la posibilidad de la creación de aplicaciones capaces de detectar y hacer un seguimiento de determinados objetos en tiempo real gracias a la eficiencia de los modelos y a la capacidad de cómputo de los computadores actuales. Esto, sumado a la recolección de datos de prácticamente cualquier entorno permite la aplicación de este tipo de aplicaciones en un amplio rango de ámbitos.

Teniendo en cuenta todos los puntos anteriores, se ha decidido utilizar métodos de visión artificial para la detección y monitorización de animales mediante el uso de imágenes de cámaras tanto de drones como a nivel de suelo.

La monitorización de animales es una tarea que ha de llevarse a cabo en diversidad de campos como pueden ser la ganadería o la conservación de especies. Para ello, hay métodos como los tradicionales requieren el contacto con el animal en concreto, siendo un ejemplo de esto los métodos de captura y marcaje. Sin embargo, dichos métodos son costosos, invasivos y no presentan una buena escalabilidad, por lo que a lo largo de los últimos años se han desarrollado nuevas tecnologías capaces de compensar las fallas de los métodos tradicionales.

De este modo, los avances en el área de la visión artificial permiten la aplicación de dicha tecnología en este ámbito, pudiendo llevar a cabo un seguimiento preciso de los animales mediante el uso de programas de reconocimiento y detección.

Objetivos

El objetivo inicial y más importante del proyecto, es el desarrollo de un modelo capaz de realizar una detección precisa de los animales presentes en la imagen, indicando la posición del mismo delimitando el área ocupada por cada animal mediante una "bounding box".

En segundo lugar, se pretende que el modelo sea capaz de, además de detectar los animales, identificarlos correctamente, es decir, para cada detección asignar una clase, correspondiente con un animal en concreto. Dicha funcionalidad permite llevar a cabo un seguimiento más preciso en conjuntos de animales de diferentes especies.

Por último, las predicciones del modelo se tienen que poder visualizar como mínimo en formato imagen y en caso de que la fuente sea un video, la predicción se visualizará por ende, como un video, en el cual se puedan identificar los animales detectados y la especie a la que pertenecen.

Para poder cumplir los objetivos planteados se realizarán las siguientes tareas:

1. Estudio de las más nuevas y populares tecnologías relativas al ámbito de la detección de objetos.
2. Selección de algunos de los modelos hallados durante el estudio y prueba de los mismos para comprobar el rendimiento.
3. Búsqueda y selección de datasets para la prueba, entrenamiento y evaluación de los modelos utilizados.
4. Adaptación de los modelos encontrados al problema planteado y evaluación de los mismos con los datasets obtenidos anteriormente.
5. Elaboración de un informe en el que se reflejen el estudio y el trabajo llevado a cabo.

Metodología

Dada la naturaleza evolutiva de las tecnologías del ámbito de la visión artificial, se ha optado por el uso de metodologías ágiles, y más concretamente, se ha decidido utilizar la metodología SCRUM y se definen para los tres roles principales de la metodología los siguientes:

- SCRUM Master: Tutora
- Equipo de desarrollo: Estudiante
- Product Owner: Tutora

Se han planificado Sprints de entre 1 y 4 semanas en función del volumen de las tareas a realizar y se han planificado reuniones cada miércoles para revisar el trabajo realizado y lo que quede por hacer.

Estructura del trabajo

A continuación se detalla la estructura que sigue la memoria del presente Trabajo de Fin de Grado, definiendo los apartados y los contenidos que estos cubren:

- Capítulo 1: Estudio del problema. La primera sección consta de la descripción del contexto en el que se inicia el proyecto, determinando qué problemas surgen y en que situación. Para ello, se lleva a cabo un estudio de las tecnologías utilizadas en dicho contexto como medidas para el problema seleccionado.

- Capítulo 2: Gestión del proyecto software. Se determina el alcance del proyecto, identificando las tareas que han de llevarse a cabo, describiendo dichas tareas y realizando una estimación de estas. Además, por último se define la planificación de las tareas descritas previamente.
- Capítulo 3: Solución. En esta sección se lleva a cabo una descripción detallada de la solución propuesta al problema, determinando la estructura del modelo utilizado y las especificaciones del entorno en el que se ejecuta así como las instrucciones para hacerlo, de manera que sea fácilmente replicable.
- Capítulo 4: Evaluación. Este capítulo consta de las pruebas llevadas a cabo para tomar datos relevantes con los que se puedan medir la eficacia de la solución propuesta.
- Capítulo 5: Conclusión. En último lugar, se exponen una serie de conclusiones obtenidas tras la realización del trabajo, indicando posibles dificultades u obstáculos y posibles tareas que se puedan realizar haciendo uso de la solución planteada.

Capítulo 1

Estudio del problema

A lo largo de este capítulo se presenta una descripción del contexto inicial del proyecto mediante la revisión de las tecnologías, herramientas y trabajos realizados con anterioridad relativos al problema que se presenta en esta memoria.

1.1. El contexto del problema

A lo largo de los últimos años, el control de animales se ha vuelto una tarea de gran importancia debido a diversos factores, entre los que cabe destacar la creciente preocupación por la conservación de especies o el aumento en la densidad de animales en el sector ganadero [24]. Dado el caso, realizar dicho seguimiento mediante el uso de técnicas manuales resulta inviable, y el uso de chips para controlarlos, aunque pueda resultar en un aumento de la información del animal, llega a ser en ocasiones demasiado intrusivo. En la tabla 1.1 se muestra información del número de cebaderos dados de alta en España, para mostrar que este sector podría beneficiarse de la aplicación de las nuevas tecnologías.

Tabla 1.1: Número de cebaderos en estado de alta en España (a fecha 1 de enero) [24]

	2011	**2018**	**2022**	**Var 22/11**	**Var 22/18**
Total Cebaderos	24.563	17.469	17.469	-28,88 %	0,00 %
Total Censo	1.025.049	1.363.000	1.408.862	37,44 %	3,36 %

Por otra parte, un seguimiento visual es mucho menos intrusivo y se puede realizar de diversas maneras, como pueden ser las imágenes aéreas tomadas por drones o imágenes tomadas por cámaras fijas a nivel de suelo o una altura no muy elevada.

1.2. El estado de la cuestión

Como ya se ha mencionado previamente, el ámbito de la visión artificial ha experimentado un crecimiento considerable a lo largo de los últimos años, lo que ha llevado al desarrollo de una amplia gama de modelos destinados a diferentes tareas, entre las que cabe destacar la detección y la clasificación, pues son las tareas a realizar para este problema.

Los modelos mencionados en el párrafo anterior se basan mayoritariamente en el uso de las denominadas redes neuronales profundas, las cuales son redes que constan de varias capas con el fin de obtener patrones y características en los datos de entrada. Están organizadas de tal manera que las primeras capas descubren características más generales y las obtenidas por las capas más profundas de la red son mucho más complejas, de manera que a medida que se avanza, aumenta el nivel de detalle. Además, una de las características de dichas redes por la que se priorizan estas a modelos más antiguos es el hecho de que no necesitan preprocesamiento de datos, son las propias redes las que extraen las características necesarias para poder llevar a cabo la tarea que se pretende realizar [47].

Dentro de las redes neuronales profundas, cabe destacar las redes neuronales convolucionales (CNN), como la que se muestra en la figura 1.1 es una de las más populares en la actualidad; estas redes funcionan especialmente bien a la hora de reconocer patrones en imágenes las cuales hacen uso de múltiples filtros para la obtención de características, cada filtro consta de una matriz de pesos que se aplica a los datos de entrada [38].

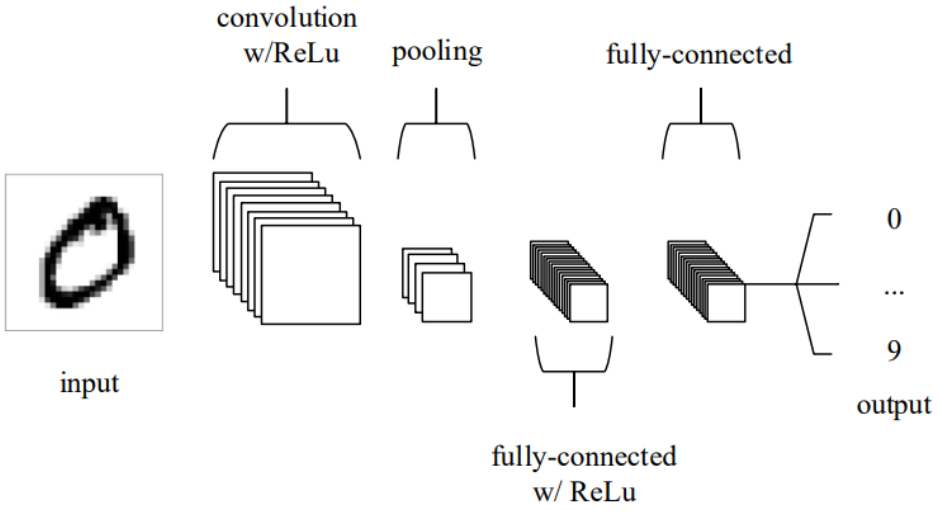

Figura 1.1: Esquema de una red convolucional [38]

El entrenamiento de estas redes se basa en el ajuste de dichos pesos, por lo que el entrenamiento de una red compleja puede requerir de un gran tiempo debido a la cantidad de operaciones que se llevan a cabo. Así pues, se plantea una nueva estrategia denominada *Transfer Learning* con la que se disminuyen los recursos utilizados a la vez que se mantiene una precisión adecuada.

El *Transfer Learning* es una técnica utilizada por primera vez en 1976 en un informe escrito por Bozinovski y Fulgosi [23] que se basa en el uso de otros modelos ya existentes para la realización de una tarea distinta de la original. La idea principal de esta técnica es la reutilización de las capacidades desarrolladas por un modelo entrenado con un gran conjunto de datos en un problema del que no se dispone de un conjunto de datos muy grande [25].

Esta técnica funciona gracias al funcionamiento de las redes neuronales mencionadas previamente, es decir, en el aumento de la complejidad de las características aprendidas conforme se profundiza en la red. De manera generalizada se podría decir que en las zonas inicial e intermedia de la red, esta aprende a reconocer formas mientras que en las últimas capas, se aprenden características propias de la tarea para la que se entrena dicha red. Así pues, dado que el conocimiento adquirido por la red en las capas iniciales e intermedias sirven para el reconocimiento de objetos, estas se pueden reutilizar en otro tipo de tareas cambiando únicamente las últimas capas, encargadas de detectar las características propias del contexto. En la figura 1.2 se muestra un esquema de la diferencia entre aprendizaje desde cero y usando *transfer learning*.

Así pues, se obtienen modelos precisos en menos tiempo de lo que llevaría construir y entrenar uno desde cero y en muchas ocasiones, los modelos obtenidos mediante el uso de *Transfer Learning* tienden a superar a los ya existentes.

1.2.1. Detección de objetos

A diferencia de los clasificadores de imágenes, que asignan a una imagen entera una etiqueta, los detectores encuentran el objeto en la imagen y le asignan la etiqueta correspondiente (ver Figura 1.3). Esto permite llevar a cabo un análisis más detallado de la imagen, ya que además de asignar la etiqueta y determinar la localización, los detectores de objetos pueden identificar varios objetos dentro de una misma imagen, incluso si estos no pertenecen a la misma categoría.

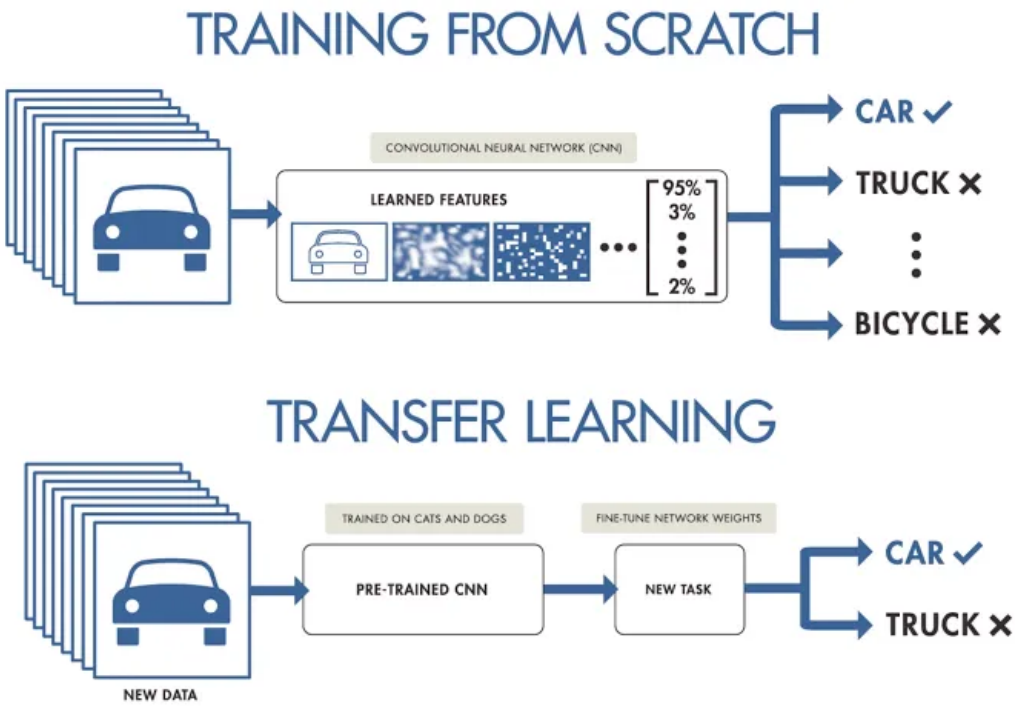

Figura 1.2: Entrenamiento desde cero vs *Transfer Learning* [19]

Funcionamiento

En primer lugar, se debe hacer una distinción entre los distintos tipos de detectores de objetos, pues estos funcionan de manera distinta [46]. Se pueden encontrar dos tipos:

- **Detectores de objetos de dos fases:** Esta clase de detectores hacen una separación en las tareas a realizar para llevar a cabo la detección [39].
  - En primer lugar, buscan las denominadas RoI (*Regions of Interest*), que como su propio nombre indica, son regiones que contienen alguna característica relevante por la cual merecen ser examinadas.
  - En segundo lugar, se clasifican dichas regiones y se define de una manera más precisa la región que comprende el objeto.

  Existen una gran variedad de detectores que hacen uso de esta metodología entre los que cabe destacar modelos como R-CNN, Fast R-CNN, Faster R-CNN o Mask R-CNN.

- **Detectores de objetos de una sola fase:** Como se puede deducir, los detectores de una sola fase, no hacen la separación propia de los de dos fases, sino que hacen la clasificación directamente, sin la extracción de las RoI, lo que

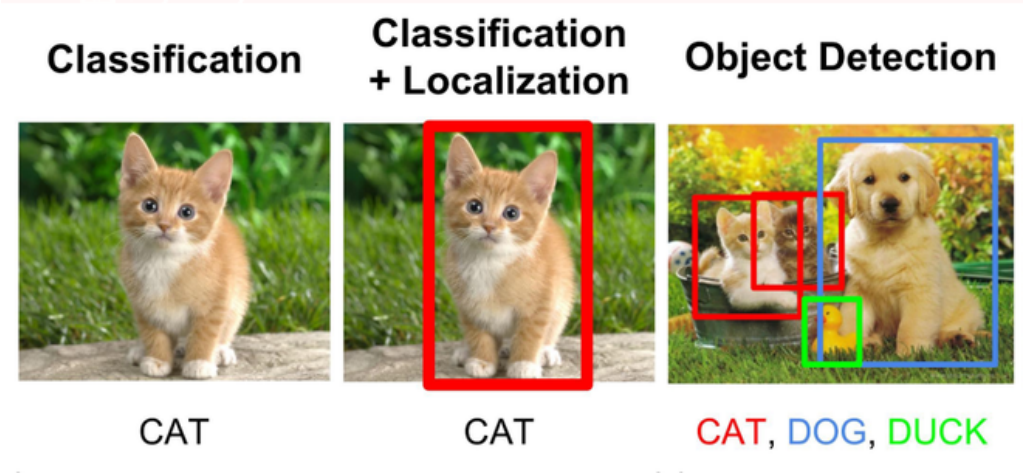

Figura 1.3: Detección de objetos [36]

supone una reducción en el tiempo de procesamiento que permite la utilización de dichos modelos para el procesamiento de imágenes en tiempo real.

Algunos de los modelos que hacen uso de este método más populares son la familia de YOLO (YOLOv3, YOLOv5, etc), RetinaNet y SSD (Single Shot Detector).

Dado que el objetivo del proyecto es el análisis de imágenes en tiempo real, los detectores de una sola fase parecen una clara elección dada la rapidez que presentan a la hora de procesar imágenes. Los datos obtenidos en la comparación [29] realizada en 2018, demuestran que para el análisis de imágenes en tiempo real los detectores de una sola fase tienen un mejor rendimiento. En las figuras 1.4 y 1.5 se puede ver una comparación de la precisión y los FPS (Fotogramas Por Segundo) para distintos modelos previamente mencionados en el conjunto de datos de test de PASCAL VOC 2012 [11], conjunto de datos utilizado con gran frecuencia para probar la eficiencia de modelos de detección de objetos.

1.2.2. Detección de animales

Una vez se ha determinado qué clases de detectores de objetos son los más convenientes para el proyecto, se ha de realizar un estudio de los principales modelos que se han desarrollado en el ámbito de la detección de animales.

Dentro de este ámbito se pueden identificar dos secciones distintas en función del punto desde el que se realiza la toma de imágenes. Existen los modelos destinados a la detección de animales utilizando imágenes de cámaras trampa, las cuales suelen estar situadas a la altura del animal, y por otra parte están los modelos destinados a

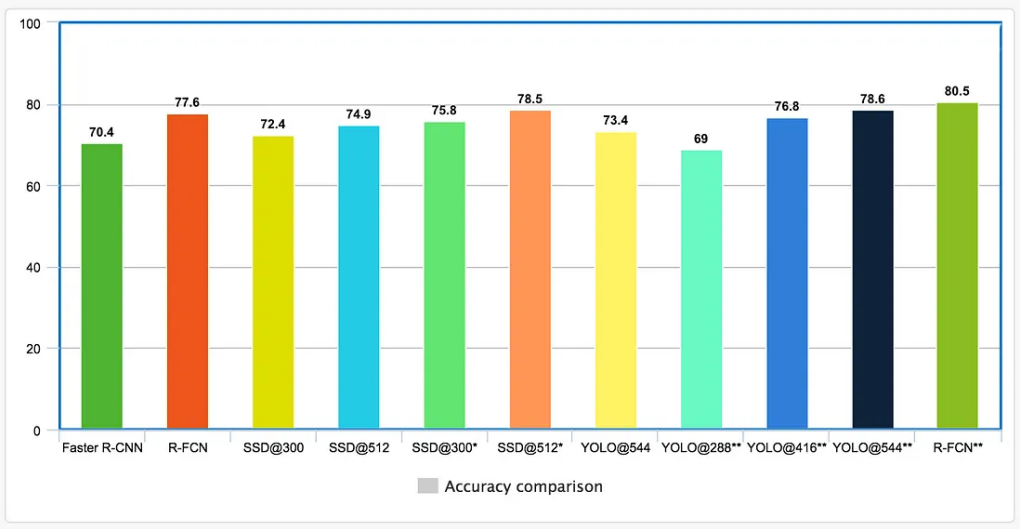

Figura 1.4: Comparación de la precisión [29]

la detección de animales con imágenes aéreas, las cuales normalmente son tomadas por drones.

En el primer caso se pueden encontrar modelos como los propuestos en [37,49,53], los cuales llevan a cabo la construcción de modelos dedicados a la identificación de animales con las cámaras previamente mencionadas tanto en entornos salvajes como en entornos creados por humanos. Sin embargo, los modelos que utilizan este tipo de imágenes se centran más en la correcta identificación del animal más que en la detección del mismo, por lo que las métricas que presentan como resultado de la evaluación son referentes a dicha identificación, obteniendo valores ligeramente superiores al 90 % de accuracy. Por otra parte, dichos modelos hacen uso de la técnica *Transfer Learning* previamente mencionada, lo que indica que la mejor opción en un principio es la utilización de esta.

En el segundo caso, por otra parte, se han construido ya modelos dedicados concretamente a la detección de ovejas mediante el uso de imágenes aéreas. Así pues, se pueden encontrar estudios como el que se presenta en [48] que lleva a cabo una evaluación de 11 modelos distintos obteniendo precisiones medias desde un 87 % hasta un 99 % con entrenamientos de 200 épocas o el realizado en [43] que utiliza los modelos de YOLO y SSD obteniendo hasta una precisión de un 86 % con la versión 4 de YOLO.

Por otra parte, cabe destacar la existencia de empresas que se dedican a la detección de animales haciendo uso de drones como dronintegra [4] pero que llevan

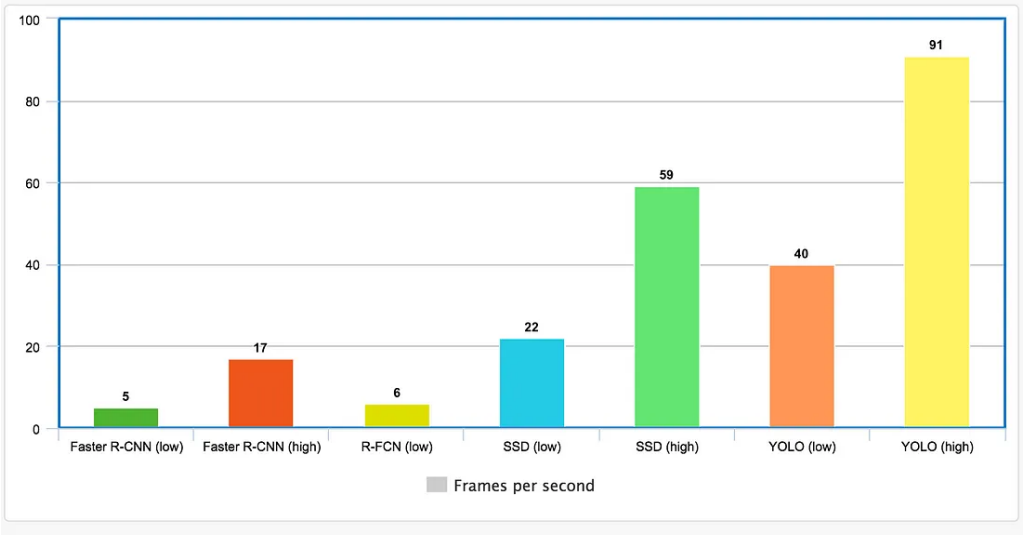

Figura 1.5: Comparación de FPS [29]

a cabo dicha tarea de manera manual, es decir, el análisis de las imágenes no está automatizado, sino que lo lleva a cabo un humano.

1.3. La definición del problema

Dada la necesidad de automatizar un proceso como es el de la monitorización de animales se ha optado por el uso de las redes neuronales profundas propias de la visión artificial dado su excelente rendimiento a la hora de procesar imágenes.

Así pues, la idea principal del proyecto es el desarrollo de un modelo de visión por computador capaz de detectar animales mediante el análisis de imágenes para que se pueda llevar a cabo una monitorización de los mismos de manera automática.

Concretamente, se pretende que sea un modelo fácilmente implementable, adaptable y que sea comprensible por la mayoría de las personas. Para ello se debe desarrollar un modelo no muy complejo, fácilmente reentrenable con otros datos y lo suficientemente eficiente como para poder llevar a cabo la tarea planteada con un margen de error mínimo.

Capítulo 2

Gestión de proyecto software

En esta sección se define el alcance del proyecto, llevando a cabo la identificación, descripción y estimación de las tareas que comprenden el mismo. Además, se realizará un estudio de los riesgos que puedan afectar al proyecto y los recursos que este requiera.

2.1. Alcance del proyecto

2.1.1. Definición del proyecto

Dado que el objetivo principal del proyecto es el desarrollo de un modelo capaz de detectar animales en imágenes, el alcance del proyecto se puede dividir en tres fases principales:

- Desarrollo de un modelo de visión artificial capaz de detectar animales
- Evaluación del modelo diseñado con diferentes conjuntos de datos para comprobar la eficacia de este
- Generación de una guía para la ejecución del modelo con un conjunto de datos propio

Sin embargo, dado que el desarrollo de un modelo eficiente desde cero requiere una cantidad de recursos considerable, se optará por el uso de *Transfer Learning* de modo que la fase de desarrollo del modelo se centrará en la investigación de los modelos más eficientes existentes en la actualidad y la adaptación de estos al problema propuesto.

2.1.2. Estimación de tareas y recursos

Para las tareas se estima que se dedique alrededor del 50 % de la duración del proyecto (dos meses aproximadamente) al estudio del problema y el análisis de las últimas tecnologías relativas al contexto de este. Posteriormente, del 50 % restante se planea hacer uso de un 30 % (en torno a cinco semanas) para el desarrollo del modelo y el tiempo restante (3 semanas) para la evaluación del modelo y la generación de documentación y de la guía de uso. Una estimación más detallada se presenta en la sección 2.2.2.

Por otra parte, en cuanto a los recursos, la estimación se verá dividida en dos partes:

- **Personal:** Se prevé que para la realización del proyecto se requiera de:
  - Desarrollador (Ingeniero de Inteligencia Artificial)
  - Scrum Master (Jefe de proyectos)
- **Hardware:** Se hará uso de un ordenador con capacidades medias que posea una tarjeta gráfica y de servicios en la nube como Google Collab o los notebooks de Kaggle.

2.1.3. Presupuesto

Para el cálculo del presupuesto se hace una división inicial de los recursos según el tipo, diferenciando entre personal (Figura 2.1) y hardware.

Coste de personal

Tabla 2.1: Presupuesto de personal

Tarea	Perfil	Horas	Euros/Hora	Total
Desarrollo modelo	Ingeniero de IA	240	26	6.240 €
Supervisión del Proyecto	Jefe de Proyecto	20	30	600 €
Total				6.840 €

Coste del hardware

Para la realización de este proyecto se ha requerido el uso del siguiente material:

1. Ordenador con Intel i5:

Placa base: Gigabyte H310M S2H 2.0

Procesador: Intel i5-9400F

RAM: 32 GB - 2666 Mhz

Disco duro: 1TB + 1TB SSD

Tarjeta gráfica: Nvidia GTX1660 Super

Fuente de alimentación: 650 W

Monitor: 23,8"

- Precio (sin IVA): 652,89 €

Se imputa al proyecto un valor de 652,89 correspondiente con una amortización del 25 % anual a 4 meses.

Coste total

En la tabla 2.2 se detalla el coste total de la aplicación desarrollada, que tiene un valor de 8.405,95 euros. Se incluye un 15 % de costes generales que corresponderían con el uso de una oficina, luz, etc.

Tabla 2.2: Presupuesto total

Concepto	Coste (Euros)
Costes de personal	6.840
Amortización de equipos informáticos	54,40
Costes generales	1034,16
Beneficio industrial (11 %)	872.14
Subtotal	8.800,7
IVA (21 %)	1.848,15
Total Proyecto	**10.648,85 €**

2.2. Plan de trabajo

2.2.1. Identificación de tareas

Para el desarrollo del proyecto se ha llevado a cabo una división del mismo en una serie de tareas a realizar:

- **Estudio de las tecnologías para la detección de animales** Ha de llevarse a cabo un estudio de las tecnologías existentes más avanzadas en el ámbito de la detección de animales con el fin de identificar aquellas características por las que un sistema de detección de animales sea eficiente.
- **Búsqueda de conjuntos de datos** Con el fin de adaptar el modelo desarrollado al contexto del problema, se ha de encontrar conjuntos de datos (imágenes y/o videos) con los cuales se pueda llevar a cabo el entrenamiento y la evaluación del modelo de una manera adecuada.
- **Desarrollo del modelo o adaptación de uno ya existente** En función de los conjuntos de datos hallados y de las tecnologías descubiertas, se procederá al desarrollo de un modelo o de la adaptación de uno ya existente al problema planteado si se considera que la adaptación supone un aumento en rendimiento o una reducción de costes significativa.
- **Evaluación del modelo** Una vez desarrollado el modelo, se debe llevar a cabo un proceso de evaluación mediante el cual se compruebe la eficiencia del mismo, es decir, que las detecciones que haga son correctas y que las haga un margen de tiempo determinado.
- **Ajustes del modelo en función de la evaluación** En función de los resultados obtenidos durante el proceso de evaluación, se llevarán a cabo una serie de ajustes al modelo si se considera que los resultados obtenidos son mejorables.
- **Evaluación final** Una vez realizados los ajustes necesarios, se realizará una evaluación del modelo con diferentes conjuntos de datos y se obtendrán una serie de métricas mediante las cuales se reflejará el rendimiento del modelo.

2.2.2. Estimación de tareas

La estimación inicial de las tareas se define en función de la importancia que se le da a cada una. Además, dicha definición viene dada por los *Sprints* que se pretenden realizar y la duración de estos viene dada en semanas:

Primer sprint:

- Tareas:
  - Estudio de modelos de detección de objetos

    - Búsqueda de conjuntos de datos de imágenes a nivel de suelo
- Inicio - 22/02/2023
- Fin - 22/03/2023
- Duración - 4

Segundo sprint:

- Tareas:
    - Estudio de sistemas de detección de animales
    - Búsqueda de conjuntos de datos de imágenes aéreas
- Inicio - 23/03/2023
- Fin - 19/04/2023
- Duración - 4

Tercer sprint:

- Tareas:
    - Pruebas de diferentes modelos encontrados
- Inicio - 20/04/2023
- Fin - 17/05/2023
- Duración - 4

Cuarto sprint:

- Tareas:
    - Reentreno de los mejores modelos encontrados
    - Evaluación de los modelos encontrados
- Inicio - 18/05/2023
- Fin - 07/06/2023
- Duración - 3

Quinto sprint:

- Tareas:
  - Comparación de los modelos
  - Elección de uno de los modelos
- Inicio - 08/06/2023
- Fin - 14/06/2023
- Duración - 1

Sexto sprint:

- Tareas:
  - Evaluación final del modelo con distintos conjuntos de datos
- Inicio - 15/06/2023
- Fin - 21/06/2023
- Duración - 1

2.2.3. Planificación de tareas

Con el fin de representar la planificación de las tareas definidas, se ha optado por la realización de un diagrama de Gantt (Figura 2.1), mediante el cual se puede identificar el orden de las tareas y las dependencias entre estas.

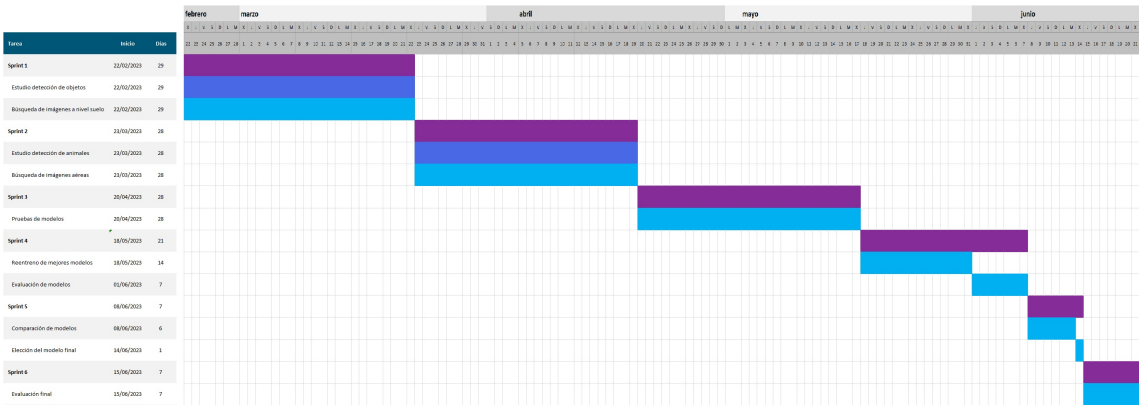

Figura 2.1: Diagrama de Gantt

2.3. Gestión de recursos

2.3.1. Especificación de recursos

Como ya se mencionó en la sección de presupuesto 2.1.3, los recursos necesarios para llevar a cabo el proyecto se pueden dividir en personal y hardware:

- Personal: En cuanto al personal, serán necesarios un ingenierio de inteligencia artificial y un scrum máster, que será el que supervise y mantenga la organización del proyecto.
- Hardware: Dado que se puede hacer uso de servicios en la nube como Google Collab para la ejecución de los modelos, los recursos necesarios para hardware no son demasiados, con un ordenador de sobremesa con las características especificadas en la sección de presupuesto es suficiente.

2.4. Gestión de riesgos

Como en todo proyecto, se deben tener en cuenta una serie de factores que puedan afectar al desarrollo de este de manera negativa, así pues, a continuación se definen una serie de riesgos a los que está sometido el proyecto.

2.4.1. Identificación de riesgos

Riesgo 1 .....

Descripción: Mala planificación del Sprint

Consecuencias:

- Tareas sin finalizar o tiempo perdido al final del Sprint

Impacto: Medio

Origen: Estimación de tareas

Posibles soluciones:

- Ajustar la planificación de los siguientes sprints en función del error cometido

Riesgo 2 .....

Descripción: Falta de personal

Consecuencias:

- Retraso en las tareas planificadas
- Aumento de costes

Impacto: Alto

Origen: Estimación de personal

Posibles soluciones:

- Contratación de más personal

Riesgo 3 .....

Descripción: Hardware poco potente

Consecuencias:

- Retraso en las tareas planificadas
- Aumento de costes

Impacto: Alto

Origen: Estimación de recursos de hardware

Posibles soluciones:

- Adquisición de un equipo con mayor capacidad de cómputo
- Contratación de servicios como Collab Pro

Riesgo 4 .....

Descripción: Personal poco cualificado

Consecuencias:

- Retraso en las tareas planificadas
- Aumento de costes

Impacto: Alto

Origen: Estimación de personal

Posibles soluciones:

- Contratación de personal más cualificado
- Formación del personal actual

2.4.2. Análisis de riesgos

En la tabla 2.3 se define el nivel de impacto de cada riesgo tanto a nivel de coste como de retraso que pueda suponer en el calendario, identificando a su vez la probabilidad de cada riesgo.

Tabla 2.3: Impacto de riesgos

Riesgo	**Coste**	**Calendario**	**Probabilidad**
Riesgo 1	Bajo	Medio	Media
Riesgo 2	Alto	Alto	Baja
Riesgo 3	Alto	Medio	Media
Riesgo 4	Alto	Alto	Baja

2.5. Legislación y normativa

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

Los datasets públicos utilizados en el proyecto cumplen con las pautas definidas en la normativa ISO/IEC 29155-4:2016(en) sobre la recolección y el mantenimiento de datos, de manera que se garantiza la fiabilidad de los resultados obtenidos durante el proceso de evaluación.

Capítulo 3

Solución

3.1. Descripción de la solución

Dado que el objetivo del proyecto es la creación de un modelo capaz de realizar un seguimiento visual de animales preciso, en vez de empezar de cero a desarrollar dicho modelo, se ha procedido en primer lugar a realizar un análisis y estudio de las principales tecnologías de detección y seguimiento de objetos.

Dicho estudio se ha centrado en los sistemas de seguimiento de objetos, en especial, al seguimiento de animales, dado que la adaptación de dichos sistemas a nuestro contexto resultará mucho más sencilla que la adaptación de otra clase de sistemas. Además, los sistemas de seguimiento de objetos más utilizados en la actualidad cumplen con su tarea de manera lo suficientemente eficiente como para poder ser trasladados al problema que se presenta.

Así pues, se opta por trasladar uno de estos modelos al seguimiento de animales y se provee una guía para utilizar dicho modelo en un conjunto de datos personalizado.

3.2. El proceso de desarrollo

Con la finalidad de aumentar la capacidad de respuesta a cambios del desarrollador, se ha optado por un modelo de proceso iterativo e incremental, el cual se basa en la adición de funcionalidad y revisión del producto de manera iterada. Se trata de un modelo que se asocia por defecto a metodologías ágiles y con el que se consigue tras cada iteración una versión del producto más estable, de mayor calidad y con nuevas funcionalidades.

La flexibilidad que aporta dicho modelo de proceso es realmente útil al trabajar en el ámbito de la visión artificial, ya que en caso de que surjan nuevas tecnologías a lo largo del proceso de desarrollo, estas pueden ser adoptadas y utilizadas durante el mismo. Además, dado que el modelo es incremental, también se verá cómo crece el proyecto en cada iteración.

Por otra parte, como en cualquier proceso de desarrollo de software se debe pasar por una serie de etapas, siendo estas:

1. **Análisis:** Durante esta fase se definen los requisitos del software que servirán para validar el producto una vez finalice su desarrollo.
2. **Diseño:** Fase en la cual se define cómo va a ser el sistema que se pretende construir de tal manera que cumpla con los requisitos previamente especificados.
3. **Implementación:** Durante el desarrollo se lleva a cabo la construcción del sistema definido en la fase de diseño.
4. **Evaluación:** Fase en la que se prueba el sistema y se comprueba que se cumplen todos los requisitos establecidos en la fase de análisis.

El correcto desarrollo de cada una de las fases mencionadas asegura que se cree un producto con una funcionalidad bien definida y que además aporte valor, ya que si el análisis se realiza correctamente, realizando un estudio adecuado del estado de la cuestión, se tienen en cuenta las mejores tecnologías utilizadas en el ámbito del problema, que herramientas se utilizan para implementarlas y en que aspectos se pueden encontrar debilidades y fortalezas sobre las que poder trabajar.

Así pues, se pretende que además de cumplir con las funcionalidades que se detallan más adelante, el sistema cumpla con los siguientes estándares:

- Aporte de valor
- Sencillez de uso
- Sencillez a la hora de ser modificado
- Uso de las últimas y mejores tecnologías del ámbito
- Versatilidad

3.2.1. Análisis

Con el fin de aclarar las tareas que debe realizar el sistema y de asentar unas bases sobre las que poder evaluar el producto una vez finalice el desarrollo se ha procedido a realizar una especificación de requisitos.

Dicha especificación viene dada por dos tipos de requisitos, los funcionales y no funcionales. Los requisitos funcionales son aquellos que definen las tareas que el sistema debe ser capaz de realizar, cómo las debe realizar y cómo debe reaccionar en función de los datos de entrada. Por otra parte, los requisitos no funcionales detallan aspectos que debe cumplir el sistema como pueden ser el rendimiento, la fiabilidad o la usabilidad. La especificación de dichos requisitos ayuda a crear unas bases claras sobre las que construir el producto.

Definición de requisitos

Requisitos funcionales:

- **Requisito funcional 1:** Aceptar como entrada al modelo archivos en formato imagen y video.
- **Requisito funcional 2:** Proporcionar como salida archivos con el mismo formato que los archivos de entrada.
- **Requisito funcional 3:** Predicciones claramente definidas en las imágenes creadas como salida del modelo.
- **Requisito funcional 4:** Se deberá poder entrenar el modelo nuevamente con datos personalizados.
- **Requisito funcional 5:** Se deberá poder evaluar el modelo con datos personalizados.
- **Requisito funcional 6:** Se proporcionará una guía en la que se detalle como utilizar el modelo.
- **Requisito funcional 7:** Se proporcionará una guía para el reentreno y la evaluación.

Requisitos no funcionales:

- **Requisito no funcional 1:** El sistema debe ser capaz de realizar la detección en tiempo real.

- **Requisito no funcional 2:** El sistema debe ser reentrenable de manera sencilla.
- **Requisito no funcional 3:** Los modelos reentrenados deben poder ser utilizados de manera sencilla.
- **Requisito no funcional 4:** El modelo debe ser capaz de manejar conjuntos de datos de gran tamaño.

Especificación de requisitos

- **Especificación RF1:** El sistema deberá ser capaz de tomar como entrada archivos tanto en formato video como imagen. Además, podrá tomar expresiones regulares como ruta de entrada para el procesamiento de varios archivos.
- **Especificación RF2:** El formato de la salida que proporcione el sistema será el mismo que el de los datos de entrada. En caso de tomar imágenes como entrada, la salida serán imágenes y en caso de ser video, la salida también será un video. Además, dicha salida se encontrará en la ruta introducida por el usuario.
- **Especificación RF3:** La salida del sistema deberá ser la imagen o el video de entrada en el que se puedan ver las denominadas *bounding box* definidas sobre los animales correspondientes y además de dichas cajas, se indicará la etiqueta que se ha asignado a dicho objeto.
- **Especificación RF4:** El sistema se podrá reentrenar con un conjunto de datos personalizado de modo que las etiquetas se ajusten a dichos datos siempre que dicho conjunto de datos esté organizado de manera adecuada.
- **Especificación RF5:** Se podrá llevar a cabo una evaluación del sistema con un conjunto de datos personalizado.
- **Especificación RF6:** Se deberá proporcionar una guía para la utilización del sistema tanto con video como con imágenes.
- **Especificación RF7:** Se deberá proporcionar una guía para llevar a cabo el reentreno y la evaluación del modelo con conjuntos de datos personalizados, indicando el formato del conjunto de datos y los pasos a seguir para reentrenar el modelo.

3.2.2. Diseño

A lo largo de esta sección se presentará un análisis detallado de los conjuntos de datos que se han utilizado para el entrenamiento y evaluación de los modelos, las arquitecturas de los modelos utilizados a lo largo del proyecto, cómo han sido implementado estos y las diferentes herramientas utilizadas durante el desarrollo.

a) Datasets

Los conjuntos de datos utilizados para el aprendizaje en la visión por computador no son simplemente imágenes, estos requieren de un largo proceso de etiquetamiento, el cual resulta aún más tedioso pero a la vez importante en el ámbito de la detección de objetos pues se deben fijar las *bounding boxes* de cada objeto en cada imagen además de determinar la clase a la que estos pertenecen. Sin embargo, no todos los modelos usan el mismo formato de entrada, por lo que resulta complicado encontrar un conjunto de datos de un contexto determinado con un formato específico.

El formato utilizado durante el proyecto es el de YOLOv8, del cual se detalla una explicación a continuación:

Dado que los modelos de la familia YOLO hacen uso de dos conjuntos de datos a la hora de realizar el entrenamiento, se deben dividir todos los conjuntos de datos encontrados en tres subconjuntos, los dos que se utilizan durante el entrenamiento y uno final para evaluación:

- **Entrenamiento:** Este subconjunto es el más grande de los tres, ya que como su nombre indica es el que utilizará el modelo para entrenar y por lo general suele comprender en torno al 70 u 80 % del dataset entero.
- **Validación:** Este subconjunto se utiliza para comprobar el rendimiento del modelo época tras época y poder determinar así si se produce *overfitting* o *underfitting*. El modelo no puede hacer uso de este subconjunto para mejorar las métricas ya que en ese caso, este pasaría a ser también parte del subconjunto de entrenamiento.
- **Test:** Este último es el utilizado para comprobar el rendimiento del modelo una vez finaliza el entrenamiento. Este subconjunto se puede omitir si se cuenta con otro conjunto de datos sobre el que poder evaluar el modelo.

Además, cada subconjunto debe tener el mismo formato de manera que la estructura final de todo el conjunto de datos sea la siguiente:

```
Directorio raíz del conjunto de datos
|
|-- train
| |-- images
| | |-- imagen1.jpg
| | |-- imagen2.jpg
| | |-- ...
| |
| |-- labels
| | |-- imagen1.txt
| | |-- imagen2.txt
| | |-- ...
|
|-- valid
| |-- images
| | |-- imagen3.jpg
| | |-- imagen4.jpg
| | |-- ...
| |
| |-- labels
| | |-- imagen3.txt
| | |-- imagen4.txt
| | |-- ...
|
|-- test
| |-- images
| | |-- imagen5.jpg
| | |-- imagen6.jpg
| | |-- ...
| |
| |-- labels
| | |-- imagen5.txt
| | |-- imagen6.txt
| | |-- ...
```

Nótese que los nombres de las imágenes han de ser iguales que los de las etiquetas correspondientes a dichas imágenes.

Sin embargo, no es suficiente con solo tener los datos estructurados de una manera específica, las etiquetas tienen que tener un formato concreto, pues existen diversas maneras de realizar el etiquetamiento de un conjunto de datos para la detección de objetos.

Según la página oficial de Ultralytics [31] (desarrollador de YOLO), el formato de etiquetado de YOLO se realiza mediante archivos con la extensión "txt" en los cuales, para cada objeto que aparezca en la imagen correspondiente se añade una línea, indicando:

- En primer lugar el número correspondiente a la clase a la que pertenece el objeto.
- Tras la clase, se indican las coordenadas del centro de la *bounding box* que contiene al objeto y el ancho y alto de la misma

Así pues, se tiene que el formato de etiquetado de YOLO es `clase centro_x centro_y ancho alto`. Sin embargo, hay que tener en cuenta que los valores relativos a la *bounding box* han de estar en formato xywh normalizado, es decir, los valores han de estar comprendidos entre 0 y 1. Además, para las coordenadas se toma como origen la esquina superior izquierda. Un ejemplo de etiquetado se puede observar en las figuras 3.1 y 3.2.

Figura 3.1: Imagen con *bounding boxes* dibujadas [31]

i. Dataset Propio

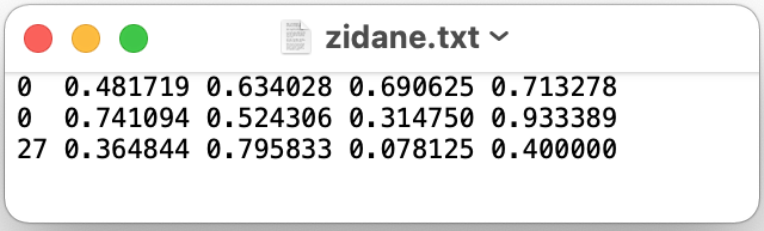

Figura 3.2: Texto definiendo las etiquetas [31]

En primer lugar se tiene un dataset creado a partir de videos proporcionados por la tutora. Dichos videos fueron tomados de un conjunto de ovejas en un solar en el que se prueba un robot. En primer lugar se seleccionó uno de los videos en el que se distinguieran adecuadamente las instancias a reconocer y se tomaron las imágenes correspondientes a cada frame. Una vez tomados los frames, se procedió a realizar el etiquetado de estos mediante la utilización de una herramienta denominada LabelMe.

La herramienta proporciona una interfaz gráfica para etiquetar las imágenes mediante la creación de rectángulos que contengan a los objetos y la posterior clasificación de dicho rectángulo en una de las clases que se definan.

El dataset consiste en un conjunto de 186 imágenes a color para entrenamiento, 18 para validación y 9 para test teniendo cada imagen el archivo de texto correspondiente en el cual se definen las *bounding boxes* de cada instancia de la imagen así como la clase a la que pertenece, siendo esta una de las siguientes: "sheep", "robot dog" o "person". En un principio, el número de imágenes era menor pero gracias a las herramientas que proporciona Roboflow, se realizó un proceso de aumento de datos (Data augmentation) para hacer el dataset un poco más grande y seguido de esto, haciendo uso también de las opciones que proporciona Roboflow, se cambió el tamaño de las imágenes a una resolución de 640x640 píxeles. Sin embargo, aun así sigue siendo un dataset muy pequeño comparado con otros utilizados, lo que ayuda a comprender cómo de bien desempeñan los modelos cuando los datos disponibles no son muchos. En las figuras 3.3 y 3.4 se puede observar un ejemplo de imagen del dataset con y sin etiquetado.

Figura 3.3: Imagen del dataset propio

ii. Aerial Sheep Image Dataset [45] - Roboflow

Este dataset público cuenta con un total de 4133 imágenes aéreas de ovejas repartidas en tres conjuntos: entrenamiento (3609 imágenes), validación (350 imágenes) y test (174 imágenes). Todas las imágenes son a color con una resolución de 600x600 píxeles y están etiquetadas utilizando el formato de YOLO previamente mencionado y al igual que con el dataset propio, este ha pasado por un proceso de aumento de datos.

Este dataset, por otra parte, solo presenta una clase ("sheep"), dado que el único objetivo del mismo es la detección de ovejas. En la figura 3.5 se presenta un ejemplo de imagen de este dataset.

iii. Sheeps Image Dataset [27] - Roboflow

Figura 3.4: Imagen de la figura 3.3 etiquetada

Al igual que el anterior, este dataset, es público y accesible a través de Roboflow y, al igual que el primero, cuenta con un número de imágenes relativamente pequeño. Contiene un total de 189 imágenes divididas en entrenamiento (152 imágenes), validación (21 imágenes) y test (16 imágenes), todas ellas a color y con una resolución de 416x416 píxeles.

Igualmente, solo tiene una clase ("sheep"), dado que el objetivo del dataset es la detección de ovejas, por lo que, además, están etiquetadas y estructuradas según el formato de YOLO. En la figura 3.6 se presenta un ejemplo de imagen de este dataset.

iv. UAV sheep images [35] - Roboflow

Como los dos últimos datasets, se trata de un conjunto de imágenes aéreas de ovejas pero tomadas por drones a una mayor altura que los anteriores. Se trata de

Figura 3.5: Imagen del dataset *Aerial Sheep Image Dataset*

un conjunto de imágenes a color con una resolución de 1333x600 píxeles destinado a proyectos de detección, por lo que las etiquetas son *bounding boxes*. Por otra parte, al contrario que el anterior dataset, este resulta mucho mayor contando con aproximadamente 30.000 imágenes para entrenamiento, 8.500 para validación y 4.300 para test, formando un total de 42.555 imágenes.

Este dataset presenta un total de seis clases, las cuales varían en función de aspectos como el color de la oveja aunque para el entrenamiento se tomarán todas las clases como una sola. Además, las imágenes están etiquetadas y estructuradas según el formato de YOLO. En la figuras 3.7 y 3.8 se presenta dos ejemplos de imágenes de este dataset.

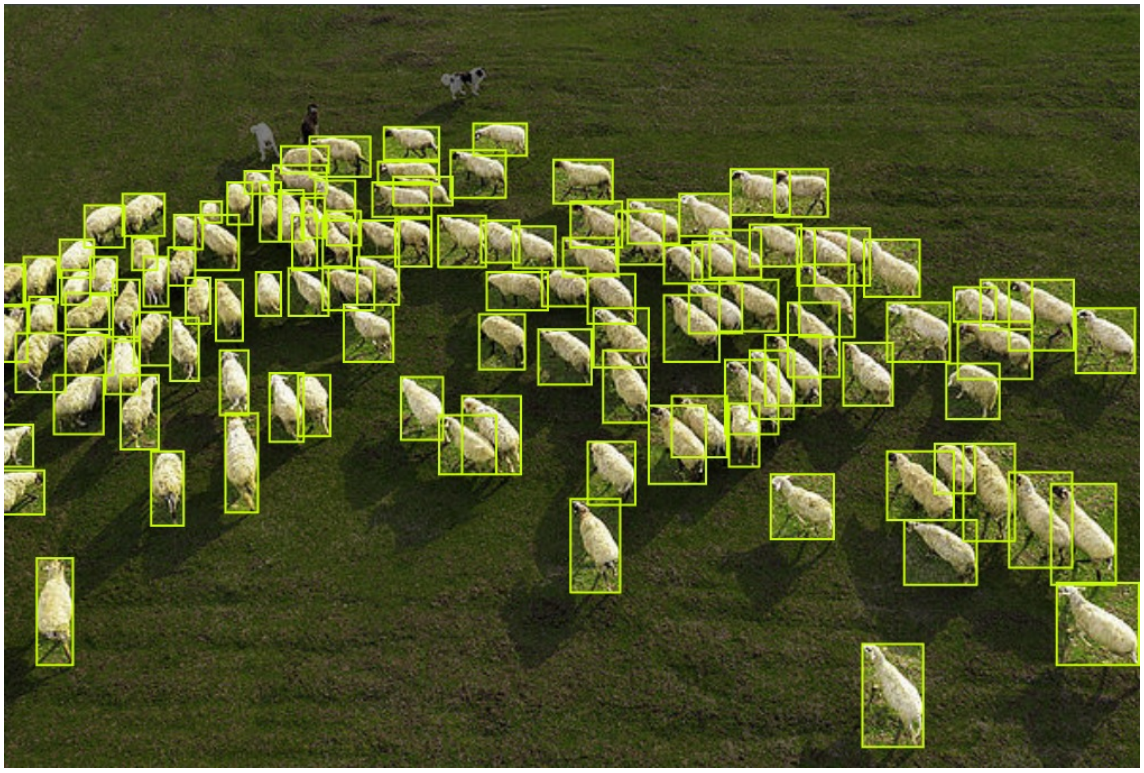

Figura 3.6: Imagen del dataset *Sheeps Image Dataset*

Cabe destacar que este dataset contiene imágenes vacías, es decir, imágenes que no contienen ninguna oveja, lo que favorece la robustez del modelo.

b) Arquitecturas o modelos

A lo largo de esta sección se presentarán los modelos que se han tenido en cuenta para la realización del proyecto, indicando entre otros datos, la arquitectura de estos y en qué se especializan.

i. Capas

Para proveer algo de contexto a las arquitecturas que se explican más adelante, primero se explican los tipos de capas que se pueden encontrar habitualmente en una red neuronal dedicada al ámbito de la visión por computador.

En primer lugar se ha de hacer una división de tres zonas en la red como se puede observar en la figura 3.9, de manera que se tienen tres tipos de capas:

1. **Capa de entrada:** Como primera capa en la red, es la encargada de recibir los datos correspondientes a las imágenes que se pretenden procesar. Se debe indicar en dicha capa las dimensiones de las imágenes (ancho, alto y número de canales).

Figura 3.7: Imagen del dataset *UAV sheep images*

Figura 3.8: Imagen del dataset *UAV sheep images*

2. **Capa oculta:** Conecta la de entrada con la de salida y supone el conjunto de todas las capas que procesan las imágenes y a diferencia de la capa de entrada y la capa de salida, la capa oculta puede y suele estar compuesta por más de una capa. El tamaño y formato de esta capa es variable, pues se pueden añadir diferente número de capas y dichas capas pueden ser de distintos tipos. Sin embargo, independientemente del número y tipo, todas las capas presentan una serie de pesos que se corresponden con las conexiones con las siguientes capas y que son los valores que se modifican durante el entrenamiento.

3. **Capa de salida:** Capa final de la red y encargada de proporcionar el resultado final haciendo uso de todos los datos procesados en la capa oculta. Esta puede estar compuesta por una sola neurona, de modo que el resultado de la red

sea un único número, o puede estar compuesta por varias, de manera que la salida sean diversos números. Este último caso se utiliza principalmente para clasificación multiclase, pues se toma cada número como la probabilidad de pertenecer a una clase en concreto.

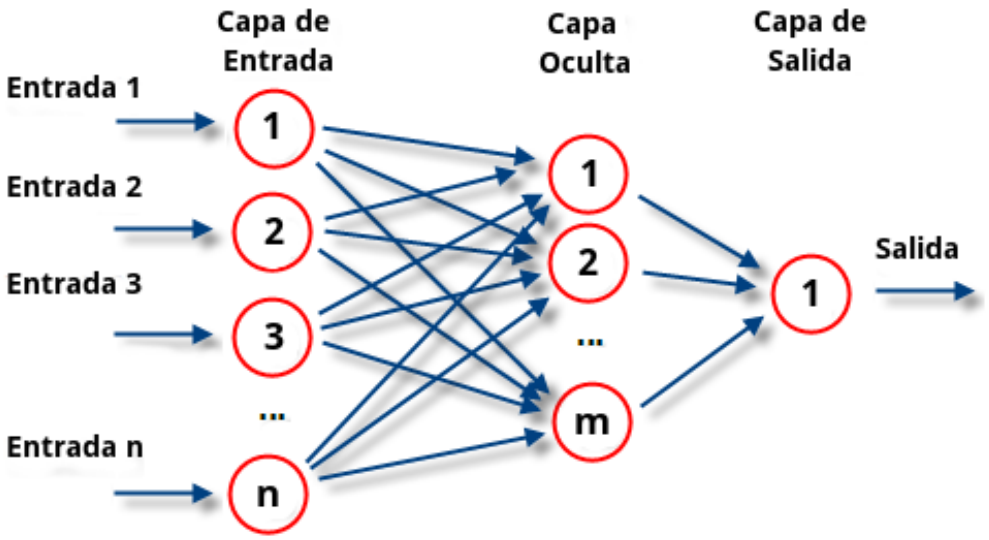

Figura 3.9: Capas de una red neuronal [13]

Tras esta división se procede a explicar los tipos de capas más utilizadas en el ámbito de la visión por computador [20].

- **Capa convolucional:** Son las capas más importantes y utilizadas de las redes neuronales modernas. Se basan en la aplicación de una serie de filtros que se aplican sobre los datos de entrada mediante el desplazamiento del filtro a través de estos.
- **Capa de activación:** Se trata de una capa que aplica una función de activación sobre los datos de entrada. Este tipo de capas introducen la no linealidad a la salida de la neurona, lo que permite que se aprendan relaciones complejas entre los datos.
- **Capa de *Batch Normalization*:** Son capas utilizadas para acelerar y estabilizar el proceso de entrenamiento. El objetivo de la capa es centrar y normalizar los subconjuntos que entran a la red tomando una media y desviación estándar obtenidas de dicho subconjunto.
- **Capa de *Dropout*:** Es una técnica utilizada con el fin de mejorar la capacidad del modelo para generalizar. Para ello, en cada época del entrenamiento, se eliminan los valores de neuronas al azar.

- **Capa de *Pooling*:** Es una capa utilizada para reducir el tamaño de los datos mediante la aplicación de filtros y se pueden encontrar varias técnicas para lograr dicho objetivo. Un ejemplo de este tipo de capa es la de *MaxPooling* que toma el valor máximo de cada grupo de n píxeles.
- **Capa *Flatten*:** Como se deduce de su nombre, esta capa "aplana" los datos de entrada. Es decir, toma todos los datos de entrada y los transforma en un vector unidimensional.
- **Capa *Dense*:** Este tipo de capa realiza una conexión entre cada neurona de la capa y todas las neuronas de la capa anterior. Se suele utilizar al final de las redes neuronales.

ii. YOLO

You-Only-Look-Once (YOLO) es un modelo presentado en 2015 por Joseph Redmon y otros [44] diseñado para la detección de objetos que se diferencia de otros modelos por la manera en la que realiza las predicciones, pues predice las *bounding boxes* de una sola pasada. Dicha característica desemboca en unas predicciones mucho más rápidas que las realizadas por otros modelos, llegando a procesar imágenes a 45 cuadros por segundo en su modelo base.

Sin embargo, como se deduce del último párrafo, existen varias versiones de YOLO, pues este ha sido mejorado a lo largo de los años, llevando a modelos como YOLOv8 o YOLO-NAS, los cuales han sido utilizados para el presente proyecto.

iii. YOLOv8

Como se ha mencionado anteriormente YOLOv8 [30] es una versión mejorada y más nueva de YOLO. Esta version es el resultado de las distintas mejoras que se han ido aplicando a lo largo de los años a las versiones de YOLO, desde la original hasta YOLOv7. YOLOv8 es capaz de realizar varias tareas además de la detección que realizaba la versión original de 2015, entre dichas tareas cabe destacar la clasificación, segmentación, estimación de pose (determinar la postura de una persona por ejemplo), y seguimiento.

Además, como es de esperar de una versión nueva y mejorada, esta presenta mejores resultados que sus predecesores. El rendimiento de las versiones de YOLO se mide mediante el resultado para el dataset COCO [33] de la métrica mAP (*mean Average Precision*) que se explicará más adelante en la sección 3.2.2. En la figura 3.10 se puede observar la comparación del rendimiento de YOLOv8 con las versiones 5, 6 y 7.

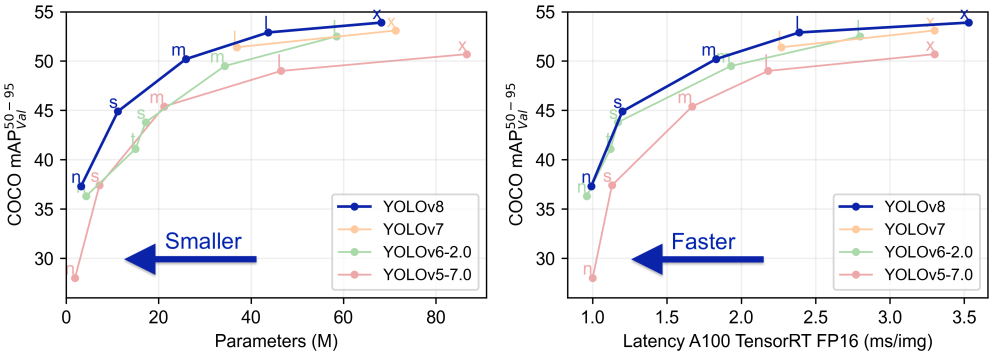

Figura 3.10: Comparación de YOLOv8 con versiones anteriores [30]

Por otra parte, al igual que en versiones anteriores de YOLO, existen distintas variaciones del modelo, teniendo así YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l y YOLOv8x para los cuales se obtuvieron distintos valores para medir su rendimiento 3.1

Tabla 3.1: Métricas versiones YOLOv8

Modelo	**Tamaño (píxeles)**	**mAP**	**Velocidad CPU ONNX (ms)**	**Velocidad A100 TensorRT (ms)**	**Parámetros (M)**	**FLOPs (B)**
YOLOv8n	640	37,3	80,4	0,99	3,2	8,7
YOLOv8s	640	44,9	128,4	1,20	11,2	28,6
YOLOv8m	640	50,2	234,7	1,83	25,9	78,9
YOLOv8l	640	52,9	375,2	2,39	43,7	165,2
YOLOv8x	640	53,9	479,1	3,53	68,2	257,8

Dicha tabla, cuyos valores fueron obtenidos del repositorio de github de ultralytics [30] (desarrolladora de YOLO) indica:

- **Tamaño**: El tamaño de las imágenes procesadas en píxeles mediante un solo valor dado que se trata de imágenes cuadradas.
- **mAP**: Mean Average Precision, la métrica previamente mencionada.
- **Velocidad CPU**: El tiempo que toma procesar cada imagen expresado en milisegundos haciendo uso únicamente de la CPU utilizando el framework ONNX.
- **Velocidad A100**: El tiempo que tomar procesar cada imagen expresado en milisegundos haciendo uso de la GPU A100 utilizando el kit de desarrollo TensorRT de NVIDIA.

- **Parámetros**: El número de parámetros que tiene el modelo expresado en millones.
- **FLOPs**: EL número de operaciones de coma flotante por segundo expresado en billones.

Así pues, se puede observar que modelos más pequeños como el nano o el small (YOLOv8n y YOLOv8s) aunque ofrecen unos valores de mAP más bajos, demuestran unas velocidades muy superiores, mientras que para modelos más grandes como YOLOv8l y YOLOv8x, ocurre lo contrario, se obtienen unos valores más altos de mAP más altos a costa de un tiempo de procesamiento más alto. De este modo, se puede deducir que para tareas que requieran una velocidad de procesamiento muy alta y no requieran una precisión demasiado elevada, se pueden usar modelos más pequeños y viceversa.

En cuanto a la arquitectura del modelo, YOLOv8 utiliza la misma arquitectura que versiones anteriores pero con pequeños cambios para mejorar el rendimiento. Sin embargo, dado que el modelo es muy reciente, no se ha publicado un artículo explicando en detalle la arquitectura y las mejoras que presenta YOLOv8, por lo que no se puede realizar un análisis demasiado detallado. Un esquema de la arquitectura se puede encontrar en la Figura 3.11

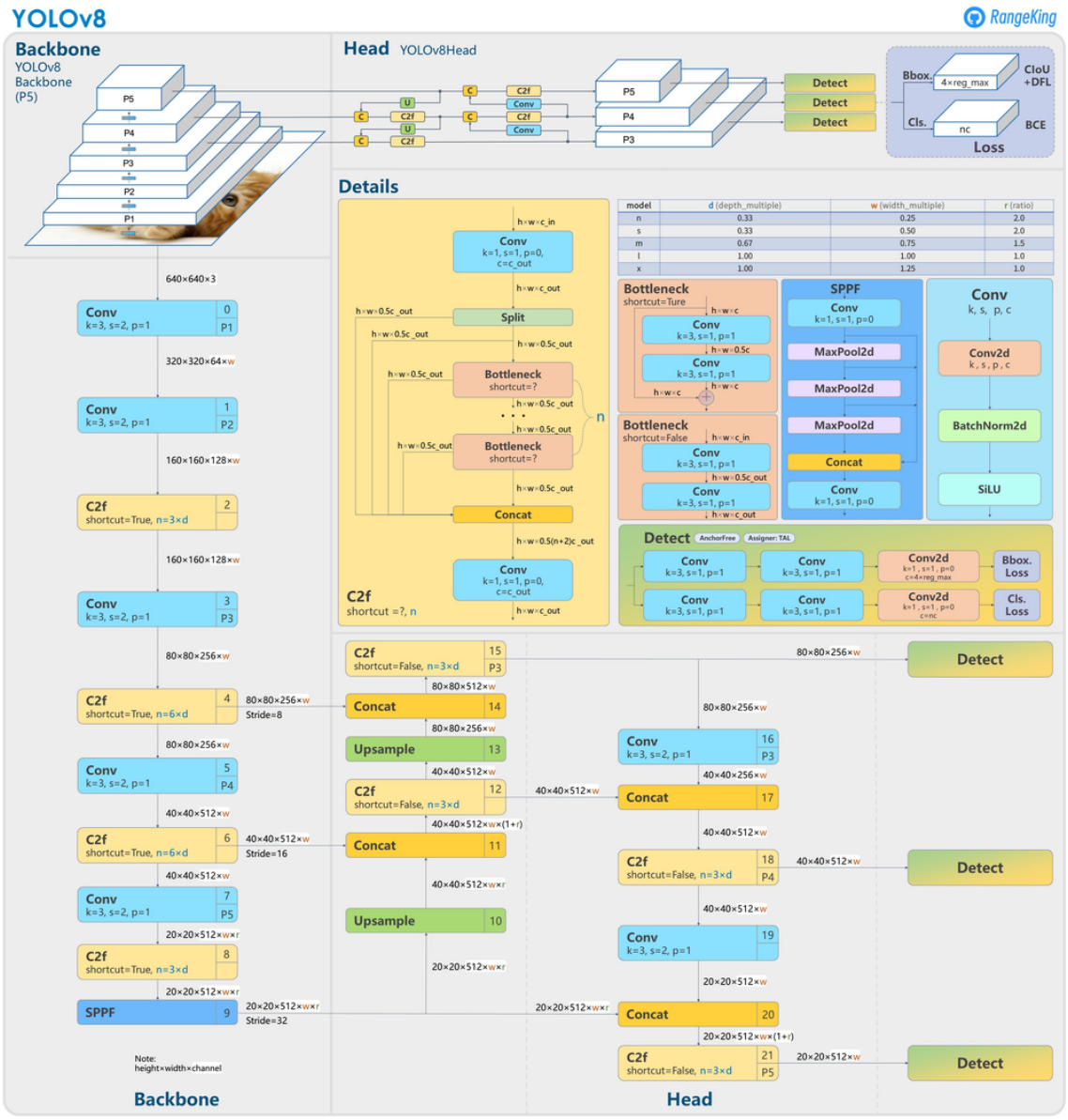

Figura 3.11: Arquitectura de YOLOv8, esquema creado por RangeKing (GitHub) [42]

Como se puede observar, el modelo se divide principalmente en tres partes:

- **Backbone:** es la estructura principal del modelo, y de hecho, es la parte del modelo encargada de extraer las principales características. Como se observa en la Figura 3.11, está compuesto mayormente por capas convolucionales, conocidas por la capacidad que presentan para obtener características de las imágenes. Para formar el Backbone, YOLOv8 hace uso de una versión modificada de la arquitectura CSPDarknet53 [22], la cual cuenta con 53 capas y que hace uso de conexiones parciales entre etapas para mejorar el flujo de

información entre capas [34]. En la figuras 3.12 y 3.13 se puede observar un ejemplo de aplicación de dichas conexiones.

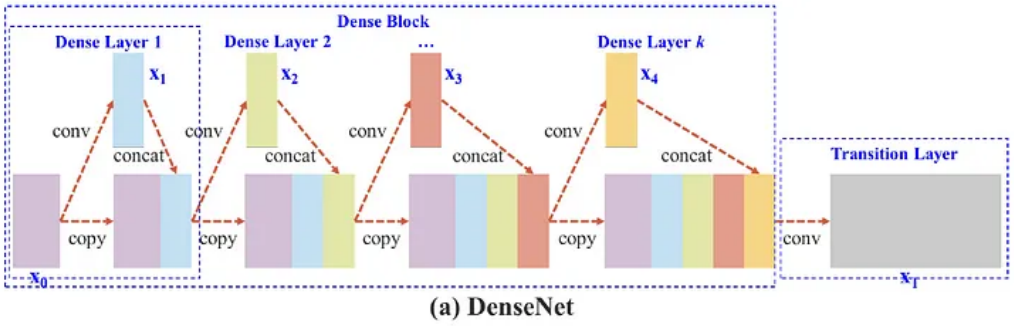

Figura 3.12: Bloque *Dense* de DenseNet sin conexiones entre etapas [51]

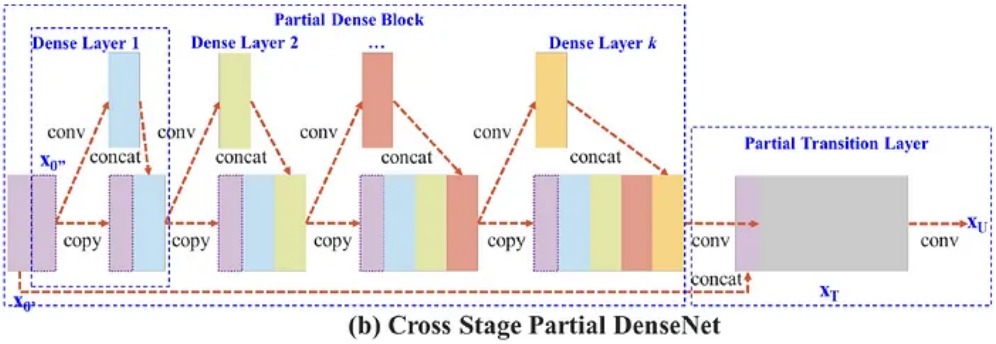

Figura 3.13: Mismo bloque que en la figura 3.12 haciendo uso de conexiones entre etapas [51]

- **Neck:** Es la sección encargada de conectar el cuerpo con la cabeza del modelo y está compuesta principalmente por tres tipos de capas, las capas Bottleneck (Cuello de botella), SPPF y las denominadas CBS (Convolution, Batch Normalization y SiLU):
  - Bottleneck: Son capas cuya principal función es la reducción de la dimensionalidad de los datos que reciben.
  - SPPF: Son capas compuestas por capas de MaxPooling diseñadas para eliminar el requisito de una entrada de tamaño fijo, lo que permite que el modelo acepte entradas de tamaño variable [28]. En la figura 3.14 se muestra una comparación de las estructuras de las capas SPP y SPPF, siendo la primera la utilizada en versiones anteriores a YOLOv5 y la segunda la utilizada desde YOLOv5 en adelante.

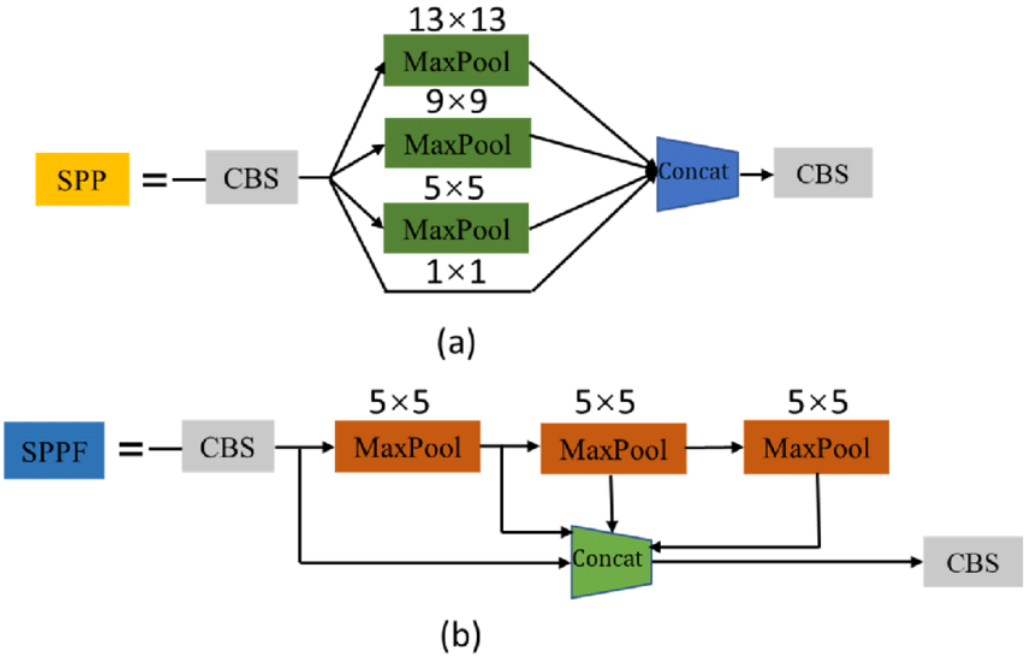

Figura 3.14: Comparación de una capa SPP(a) con una SPPF(b) [40]

- CBS: Son capas compuestas por tres subcapas, con las iniciales de cuyos nombres se forma el nombre CBS. Estas capas son una convolucional, una de Batch Normalization y otra SiLU. Las dos primeras se explicaron previamente, y la SiLU [26] es una capa de activación que se calcula mediante la multiplicación de la entrada por el valor de la función sigmoide para la entrada:

  $$x\sigma(x).$$

  En la figura 3.15 se puede observar una gráfica de la función SiLU en comparación con la función ReLU más comúnmente utilizada.

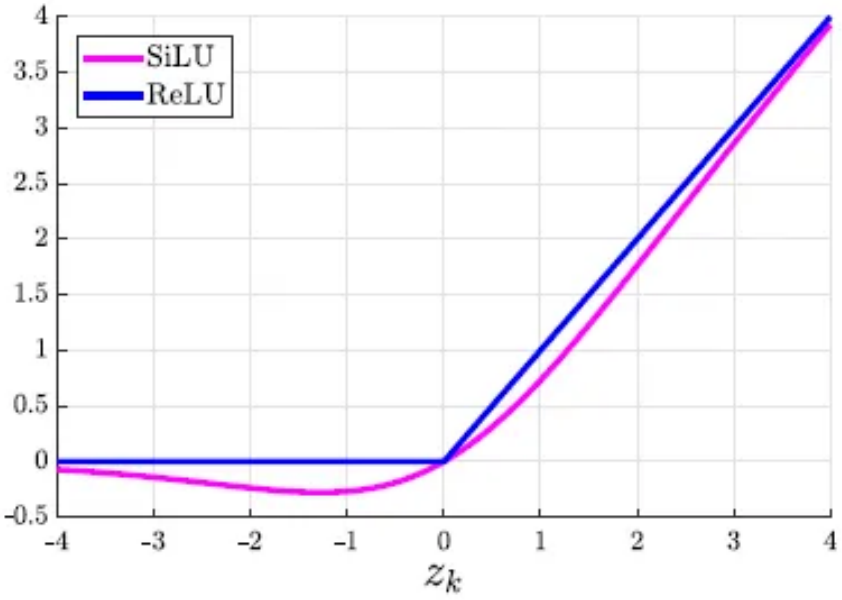

Figura 3.15: Gráfica de la función SiLU comparada con la de ReLU [26]

- **Head:** Es la parte encargada de realizar las detecciones haciendo uso de los datos que recibe de la capa anterior.

iv. YOLO-NAS

Se trata de un modelo desarrollado por la organización Deci AI basado en la arquitectura de YOLO destinado únicamente a la detección de objetos. Al igual que otros modelos de YOLO, también cuenta con varias versiones: YOLO-NAS-s, YOLO-NAS-m y YOLO-NAS-l preentrenados en los datasets COCO, Objects365 y Roboflow 100 [18].

Este modelo fue creado con el fin de avanzar aún más que los modelos ya existentes de YOLO en el ámbito de la detección de objetos, llegando a superar así incluso a las versiones más recientes de YOLO.

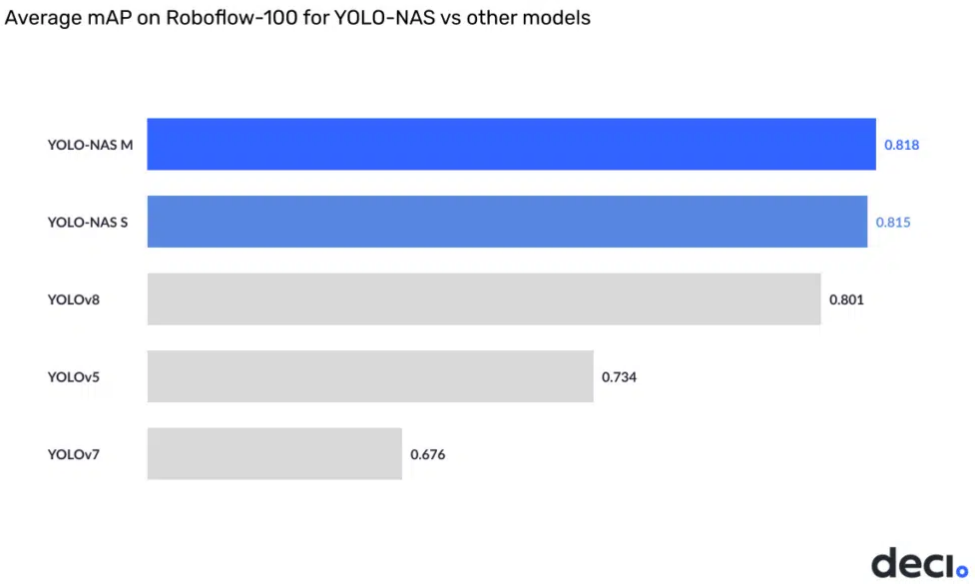

Figura 3.16: Comparación entre YOLO NAS y otros modelos de la mAP sobre el dataset Roboflow100 [50]

La característica principal de este modelo y sobre la que se basa es la búsqueda de arquitectura neuronal (NAS, Neural Architecture Search) que siguiendo una estrategia intenta encontrar la mejor arquitectura para el problema en cuestión. NAS se puede dividir en tres bloques:

- Espacio de búsqueda: En este bloque se definen las operaciones que se utilizarán para diseñar las redes neuronales profundas.
- Estrategia de búsqueda: En este bloque se optimizan las métricas en función del enfoque utilizado para explorar el espacio de búsqueda.
- Estrategia de evaluación: Evaluación de la red neuronal profunda construida después de haber sido entrenada.

Así pues, se podría afirmar que YOLO NAS no tiene una arquitectura fija, sino que esta se redefine en función del problema que afronte.

c) Métricas

A lo largo de esta sección se identifican y explican las principales métricas utilizadas en el ámbito de la detección de objetos [21, 41, 54, 55] y se indica cuáles de estas son utilizadas durante el desarrollo del proyecto.

- **TP (*True Positives*):** Los verdaderos positivos son aquellas instancias cuyo valor de etiqueta es verdadero y el valor predicho para las mismas también es verdadero.
- **FP (*False Positives*):** Los falsos positivos son aquellas instancias cuyo valor de etiqueta es falso y el valor predicho para las mismas es verdadero.
- **TN (*True Negatives*):** Los verdaderos negativos son aquellas instancias cuyo valor de etiqueta es falso y el valor predicho para las mismas también es falso.
- **FN (*False Negatives*):** Los falsos negativos son aquellas instancias cuyo valor de etiqueta es verdadero y el valor predicho para las mismas es falso.

En la figura 3.17 se puede observar una tabla definiendo las métricas previas de manera visual. A dicha representación se le denomina matriz de confusión, la cual se explicará más adelante.

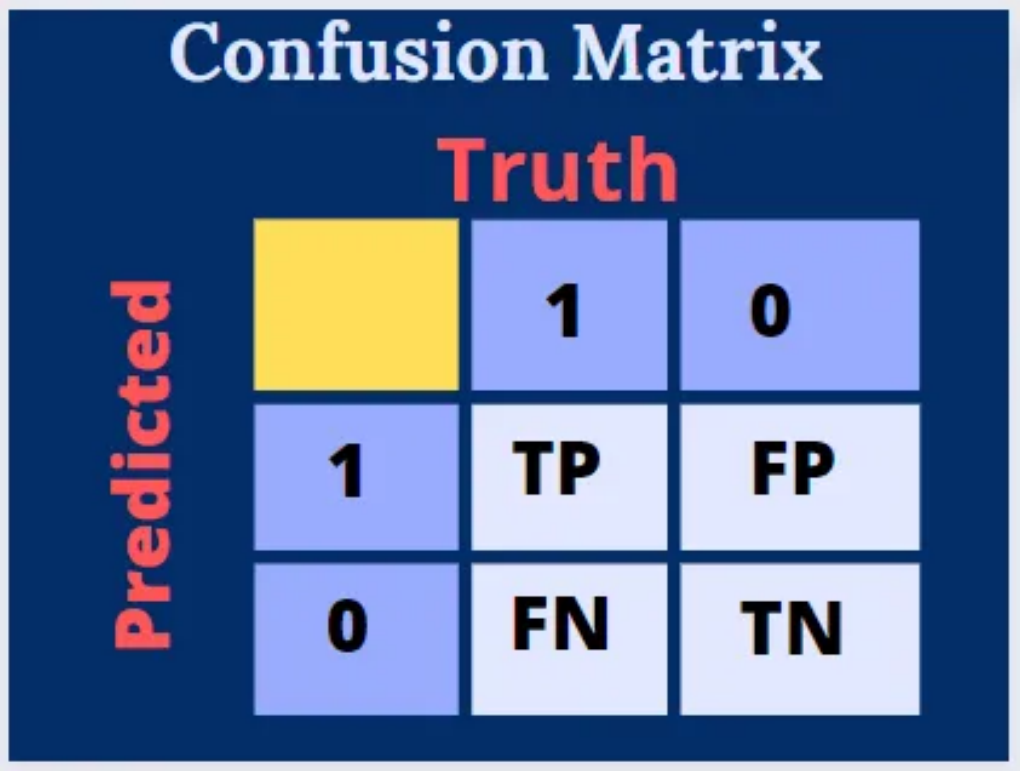

Figura 3.17: Matriz de confusión [41]

- **Precisión (*Precision*):** La precisión es una métrica que representa la cantidad de elementos clasificados como positivos que realmente son positivos. El cálculo de la misma se realiza de la siguiente manera:

$$Precision = TP/(TP + FP)$$

Esta métrica se utiliza especialmente cuando se quieren detectar valores anormales de falsos positivos, es decir, cuando elementos de la clase negativa son clasificados como positivos.

- **Sensibilidad (*Recall*):** La sensibilidad es la métrica utilizada para medir la cantidad de elementos de la clase positiva que realmente fueron clasificados como positivos

$$Recall = TP/(TP + FN)$$

Esta métrica es utilizada principalmente cuando se pretende centrarse en el valor de los falsos negativos, es decir, cuando los elementos de la clase positiva no están siendo detectados como tal.

- **F1-Score:** Se trata de una métrica ampliamente utilizada en el mundo de la inteligencia artificial, pues se trata de un valor que mide el rendimiento de un modelo de una manera más acertada que la precisión o la sensibilidad por separado. La puntuación F1 es un valor obtenido mediante el cálculo de la media armónica de la precisión y la sensibilidad. Se calcula de la siguiente manera:

$$F1Score = 2 * Precision * Recall/(Precision + Recall)$$

- **IoU (*Intersection over Union*):** Se trata de un valor obtenido a partir de las *bounding boxes* de la predicción y la verdadera. Como su propio nombre indica (Intersección sobre Unión), la IoU se obtiene dividiendo la intersección de las *bounding boxes* entre la unión de las mismas. Sin embargo, inicialmente no se podrían obtener valores como la precisión o la sensibilidad a partir de la IoU, por lo que se crean los denominados umbrales. En la detección de objetos se utilizan "umbrales de confianza" para determinar si una predicción es positiva o negativa. Estos son valores que crean una división en el dominio de la función de la IoU, por lo que si por ejemplo, para un umbral de 0,5 se obtiene un valor de IoU de 0,8, este se clasificaría como positivo, y, al contrario, si dicho valor estuviese por debajo de 0,5 sería clasificado como negativo, pudiendo obtener así todas las métricas previamente mencionadas.

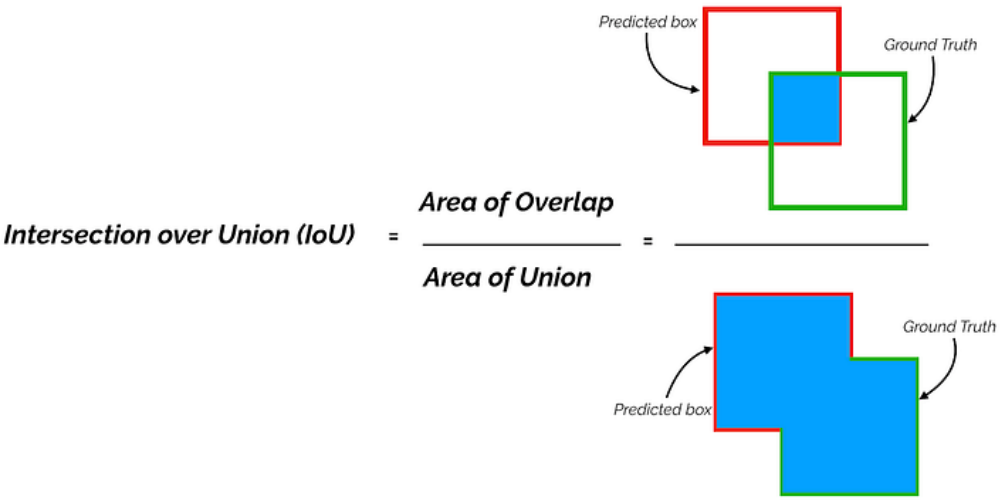

Figura 3.18: Matriz de confusión [54]

En la figura 3.18 se puede observar la fórmula utilizada para la obtención de la IoU y una representación de los valores utilizados, donde la caja roja es la *bounding box* predicha y la verde es la verdadera.

- **mAP:** El promedio de la precisión media (mean Average Precision) es una métrica ampliamente utilizada en el ámbito de la detección de objetos. Esta métrica se obtiene realizando la media de las precisiones medias de cada categoría y se sigue el siguiente proceso:

  1. Se calcula la curva Precisión/Recall, donde para cada valor de Recall(r) se toma el valor de precisión más alto para cualquier sensibilidad $r' \geq r$.
  2. El área bajo la curva calculada es considerada la precisión media. Así pues, se calcula la precisión media para cada categoría y una vez se tienen todas se calcula la media.

  Otra opción al calcular la mAP es tomar diferentes umbrales de confianza y realizar la media de las precisiones medias obtenidas en cada umbral, de manera que no se dependa así de las clases existentes en el dataset.

- **Matriz de confusión:** Más que una métrica es una manera de representar visualmente el rendimiento de un modelo. Como se puede observar en la figura 3.17, la matriz de confusión representa todas las predicciones realizadas por el modelo y si estas son correctas o no. Sin embargo, la matriz de la figura es para un problema binario, es decir, en el que solo se tengan dos clases, por lo

que para un problema en el que haya más de dos clases habría que ampliar la matriz a una como la de la figura 3.19 donde se producen los siguientes cambios para cada clase:

- Los positivos son la clase en la que nos centramos.
- Los negativos son cualquier clase distinta a la positiva en vez de una sola

Esta tabla resulta muy útil para comprender el rendimiento de un modelo de manera rápida y sencilla, pudiendo ver en qué aspectos falla y en cuales desempeña mejor.

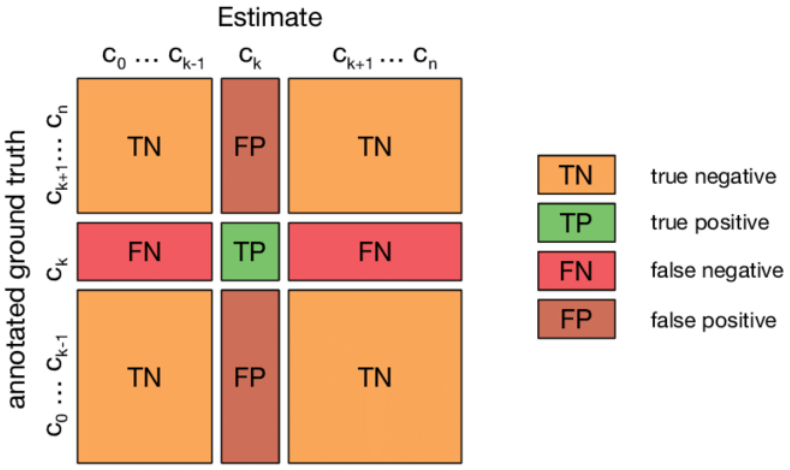

Figura 3.19: Matriz de confusión NxN [32]

3.2.3. Implementación

A lo largo de esta sección se presentan y explican las diferentes herramientas y librerías utilizadas para llevar a cabo la implementación de los modelos. Además, se exponen las razones por las que se ha decidido utilizar cada herramienta en concreto y no otra similar.

a) Lenguaje

Python [17]

Para la implementación de los modelos utilizados se ha utilizado el lenguaje de programación Python, el cual es el más popular a la hora de trabajar en el campo de la inteligencia artificial y más concretamente, de la visión artificial. Se trata de un

lenguaje de programación de alto nivel, fácil de comprender y de gran utilidad para diversidad de campos dada la amplia cantidad de librerías creadas por la comunidad. Dentro del campo de la visión artificial destacan librerías como keras, tensorflow o pytorch que ofrecen la posibilidad de implementar gran cantidad de modelos de manera no muy compleja. Algunas de las librerías mencionadas fueron utilizadas durante la prueba de modelos.

b) Herramientas

- **Anaconda [2]:** Anaconda es una distribución libre de Python y R que facilita la gestión de paquetes. Permite crear los denominados entornos virtuales sobre los cuales se pueden instalar paquetes que funcionarán únicamente para dichos entornos. Esta característica de anaconda permite gestionar varios proyectos a la vez, o en este caso, para la gestión de distintos modelos que requieran versiones distintas de alguna librería.
- **Visual Studio Code [16]:** Visual Studio Code es un editor de código desarrollado por Microsoft, disponible tanto para Windows como para Linux, sistemas operativos utilizados durante el desarrollo del proyecto. Se trata de un editor de código que permite la edición de todo tipo de archivos de texto, entre los cuales se incluyen los cuadernos de *Jupyter*. Además, incluye soporte para llevar a cabo depuración de código, soporte para el control de versiones de Git, el cual ha sido utilizado, y cuenta con una gran variedad de extensiones desarrolladas por la comunidad que favorecen la comodidad al desarrollar código. Debido a la gran comodidad y versatilidad del que ofrece el editor se ha elegido como editor de código principal para el proyecto aunque también se ha hecho uso del editor de *Jupyter* y de Google Colab.
- **Google Colab [6]:** Google Colaboratory es un servicio ofrecido por Google mediante el cual se pueden ejecutar cuadernos de *Jupyter* evitando así la utilización de recursos propios. Además, gracias a que el servicio ofrece la oportunidad de utilizar GPU dedicadas a la ciencia de datos se pueden entrenar modelos mucho más rápido que en local.
- **Jupyter Notebook [1]:** Jupyter Notebook es una aplicación cliente-servidor con la cual se pueden editar los documentos denominados cuadernos. Estos documentos permiten la ejecución de código por secciones en vez de ejecutar todo de una sola vez, lo que resulta bastante útil cuando se pretende dividir un programa en diferentes secciones.

- **Roboflow [8]:** Roboflow es una plataforma dedicada a facilitar la creación de sistemas de visión artificial. En concreto, su uso durante este proyecto fue la conversión de etiquetas de un formato a otro y la obtención de conjuntos de datos con los que probar los modelos. Gracias a esta plataforma, el formato de la etiqueta no supone un problema demasiado grande, ya que permite cambiar dicho formato de manera rápida y sencilla, característica muy útil dado el formato de salida de las etiquetas de LabelMe (herramienta de etiquetado de imágenes).

- **LabelMe [52]:** LabelMe es una herramienta de anotación escrita en python para el etiquetado de imágenes tanto para segmentación como para detección. La herramienta permite la apertura de directorios de imágenes y facilita el etiquetado de las mismas mediante la creación de las *bounding boxes* señalando únicamente dos puntos y la clase por cada caja. Resulta una herramienta bastante sencilla de utilizar que agiliza el proceso de etiquetamiento de imágenes. Sin embargo, la salida de las etiquetas creadas es en formato JSON, por lo que en un principio no serían aptas para el uso con modelos de YOLO. Por otra parte, gracias a Roboflow, la conversión del formato JSON al formato de YOLO resulta inmediata. En la figura 3.20 se puede observar una imagen etiquetada usando LabelMe.

Figura 3.20: Ejemplo de etiquetado de una imagen con LabelMe

c) Librerías

- **Ultralytics [7]:** Como ya se ha mencionado en alguna sección del documento, Ultralytics es la organización desarrolladora de los modelos de la familia YOLO

y dado el impacto de sus modelos, han desarrollado una librería para python mediante la cual se pueden utilizar sus modelos, ya sea para entrenar, predecir o evaluarlos. Por otra parte, ultralytics permite la utilización de diferentes comandos por consola para entrenar, realizar inferencia o validar modelos para las distintas tareas que puedan realizar dichos modelos. Además de ofrecer una implementación directa de los modelos de YOLO, resulta bastante fácil de manejar comparada con librerías como keras o pytorch, por lo que se ha considerado una librería imprescindible para el desarrollo del proyecto.

- **Super-gradients [18]:** Dado que YOLO NAS no ha sido desarrollado directamente por Ultralytics, se ha de hacer uso de esta librería, la cual incluye la implementación de los modelos de YOLO NAS y las funciones para entrenar, evaluar e inferir utilizando dichos modelos.

- **Tensorflow [14]:** Tensorflow es una librería de código abierto desarrollada por Google y destinada al aprendizaje automático. Dicha librería facilita la creación de modelos de inteligencia artificial gracias a la implementación de gran cantidad de modelos ya existentes y de los componentes de dichos modelos, así como funciones para evaluar dichos modelos. Por otra parte, Tensorflow es multiplataforma, es decir, puede trabajar haciendo uso tanto de la CPU como de la GPU o TPU (Unidad de Procesamiento de Tensores). Dado lo intuitiva que resulta dicha librería y lo bien organizada que está la documentación de la misma, se ha hecho uso de esta para probar modelos estudiados durante la fase de análisis. Otra opción alternativa a Tensorflow igualmente válida es Keras, sin embargo, dada la previa familiaridad con Tensorflow se optó por esta última.

- **NumPy [10]:** Se trata de una librería fundamental a la hora de trabajar en cualquier campo que haga uso del cálculo numérico y el análisis de datos. Se caracteriza por el incremento en la velocidad a la que se realizan operaciones numéricas, llegando a ser hasta 50 veces más rápido, pues la estructura *array* definida en esta librería resulta ser mucho más eficiente que las listas de python. Se ha optado por su uso dada la gran cantidad de operaciones que se han de realizar en el ámbito de la visión artificial además de que resulta ser una de las dependencias de librerías como Tensorflow.

- **Matplotlib [9]:** Matplotlib es probablemente la librería más popular en cuanto a la creación de gráficas dado que resulta sumamente fácil de utilizar y crea gráficas con gran rapidez y con un nivel estético bastante bueno. Sin embar-

go, cuando se pretende obtener gráficas con un mayor nivel de detalle o se pretenden personalizar un poco más, se suele optar por Seaborn.

- **PyTorch [12]:** PyTorch es una librería de python de gran popularidad en el ámbito de la inteligencia artificial ya que, al igual que tensorflow, incluye la implementación de gran cantidad de funciones y modelos utilizados de manera habitual. En este caso se utilizó por la función que permite la utilización de la GPU para procesar código y la que permite generar un esquema de la arquitectura de un modelo.
- **Overleaf [5]:** La herramienta utilizada para la generación de la documentación ha sido Overleaf, un editor de LaTeX utilizado para escribir documentos científicos. El formato utilizado por este editor facilita en gran medida procesos como la creación de tablas, la organización del documento o la generación de referencias.

d) Notebooks

Para llevar a cabo la implementación de ambos modelos se han definido una serie de *Jupyter Notebooks* para cada modelo, los cuales han sido ejecutados en un entorno remoto proporcionado por Google Colab. El código incluido en estos notebooks y algunas de las salidas más importantes se pueden encontrar en el Anexo C.

Capítulo 4

Evaluación

Con el fin de demostrar la validez de los modelos utilizados en el contexto del problema, se ha procedido a realizar una serie de experimentos con ambos modelos en varios conjuntos de datos.

4.1. Experimentos realizados

A lo largo de esta sección se detallan los experimentos realizados así como los resultados obtenidos y para poder realizar una comparación más directa de los resultados se dedicará una sección a cada dataset.

Los experimentos llevados a cabo consisten en realizar un proceso de entrenamiento de 20 épocas con los modelos yolov8-m y yolo-nas-m sobre los datasets mencionados en la sección 3.2.2 a excepción del último (UAV Sheep Images) para el cual se han dedicado 10 dado su tamaño. Se ha tomado un *batch size* de 16 y un tamaño de imagen de 640. Finalmente, se realiza una posterior evaluación de dichos modelos sobre los conjuntos de test de los datasets que se detallará en la sección 4.2.

Cabe destacar que los resultados obtenidos durante el entrenamiento de ambos modelos vienen dados en formatos distintos, por lo que algunas gráficas no tendrán el mismo formato. A continuación se exponen los principales resultados obtenidos por cada modelo en cada dataset, incluyendo las matrices de confusión obtenidas por YOLOv8, ya que es el único modelo que las proporciona tras el entrenamiento. Otro tipo de resultados proporcionados en especial por el modelo YOLOv8 se pueden encontrar en la carpeta de Google Drive indicada en el Anexo A.

4.1.1. Dataset propio

En las figuras 4.1 y 4.3 se muestran las gráficas obtenidas durante el entrenamiento de los modelos YOLOv8 y YOLO-NAS. Se puede observar una mejora sustancial a lo largo de las primeras épocas y que se alcanza una mAP con el set de validación muy cercana a la unidad (0.99 para ambos modelos). Sin embargo, aunque en el caso de YOLOv8 la precisión y el recall van a la par alcanzando valores muy altos, con YOLO-NAS, la precisión no llega a 0,2 al final del entrenamiento mientras que el recall es prácticamente 1. Esto se debe a que crea muchas *bounding boxes* y aunque algunas de ellas se correspondan con las instancias que se encuentren en la imagen, otras muchas no se corresponden con nada. Nótese que en la matriz de confusión de YOLOv8 se observa cómo alguna instancia de "fondo" es clasificada como "oveja"; esto es así porque las ovejas del grupo presente en el dataset están muy juntas y algunas no han sido etiquetadas como tal por estar detrás de otras. Sin embargo, YOLOv8 ha sido capaz de detectarlas a pesar de la oclusión, lo que demuestra las altas capacidades del modelo.

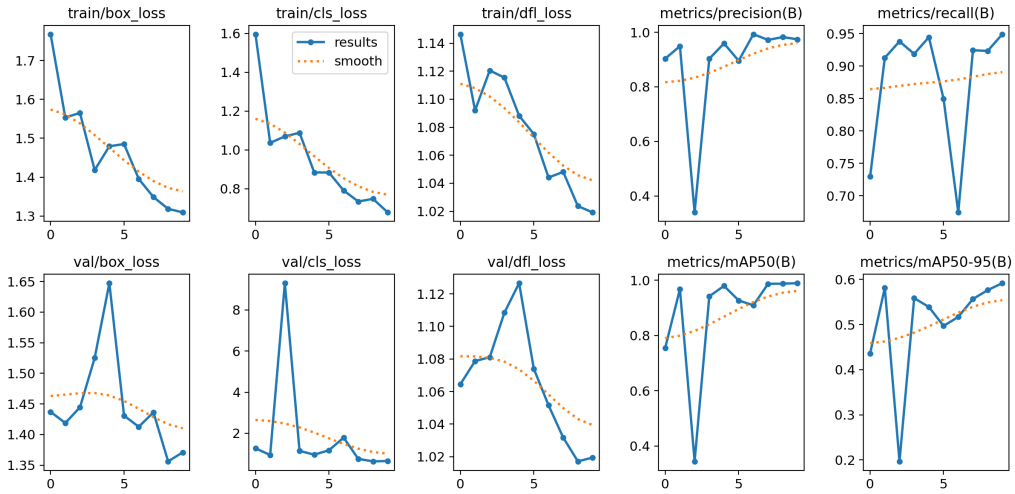

Figura 4.1: Resultados del entrenamiento de YOLOv8 con dataset propio

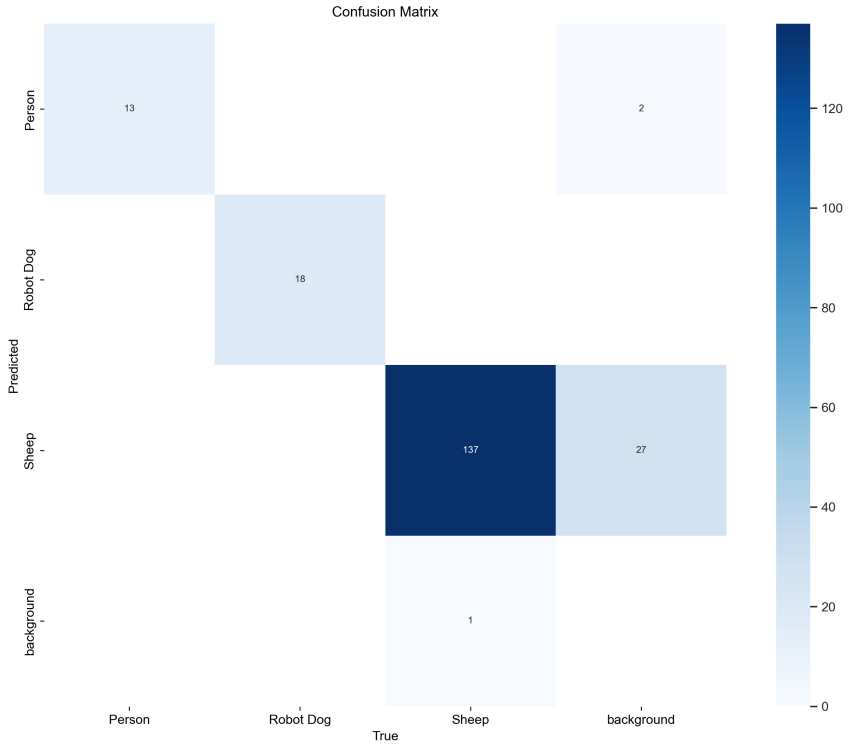

Figura 4.2: Matriz de confusión obtenida del entrenamiento de YOLO-NAS con dataset propio

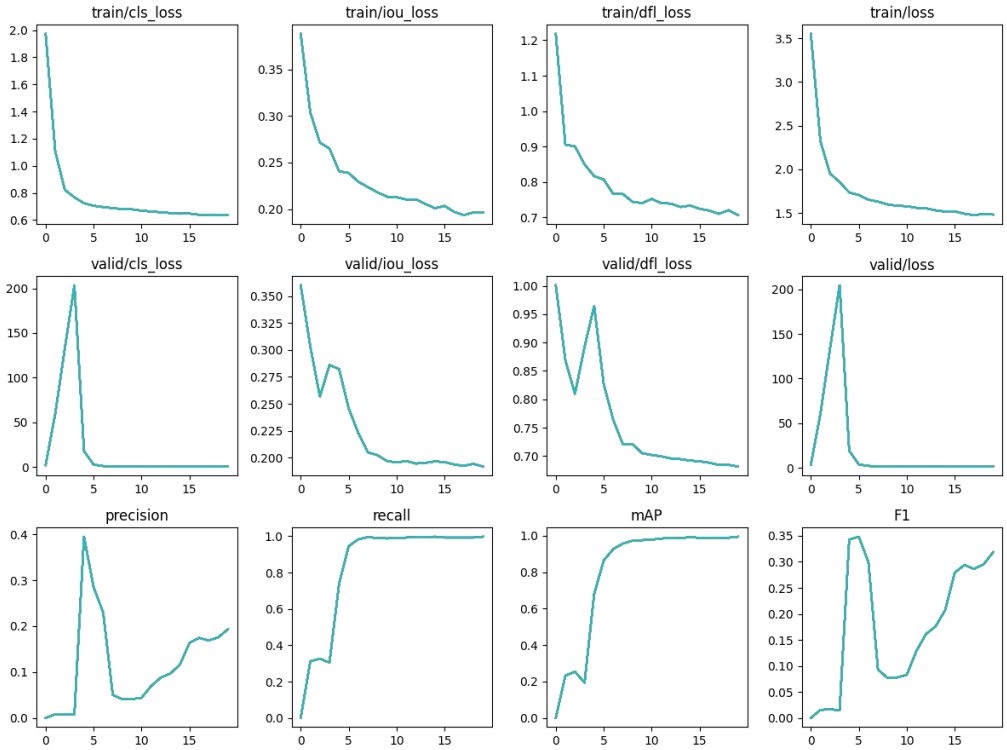

Figura 4.3: Resultados del entrenamiento de YOLO-NAS con dataset propio

4.1.2. Aerial Sheep Image Dataset [45] - Roboflow

En las figuras 4.4 y 4.6 se muestran las gráficas obtenidas durante el entrenamiento de los modelos YOLOv8 y YOLO-NAS

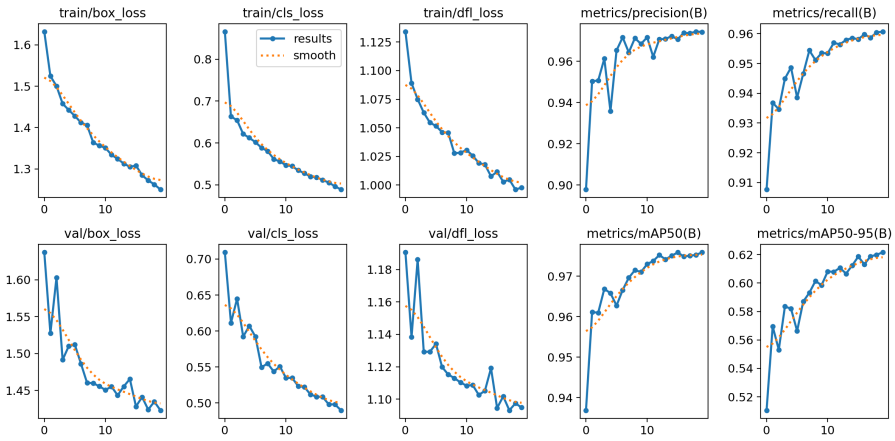

Figura 4.4: Resultados del entrenamiento de YOLOv8 con dataset Aerial Sheep Image Dataset

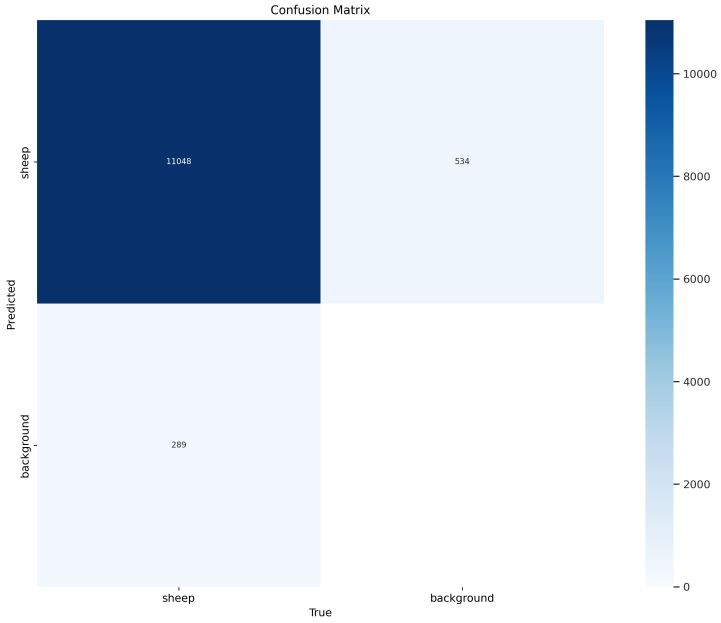

Figura 4.5: Matriz de confusión obtenida del entrenamiento de YOLO-NAS con dataset Aerial Sheep Image Dataset

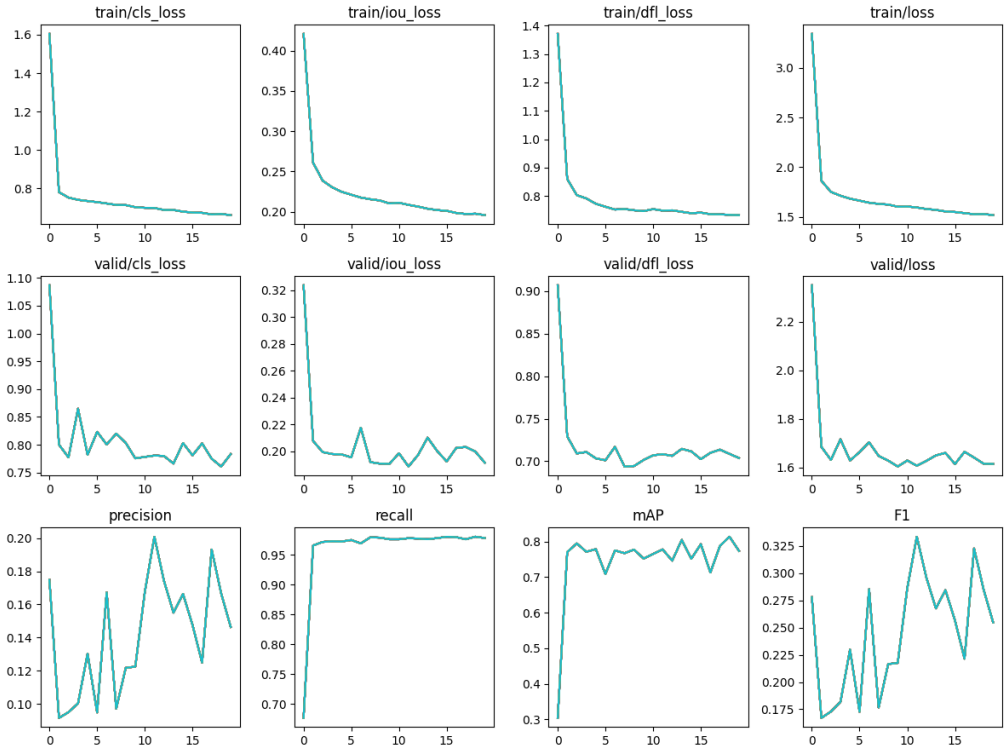

Figura 4.6: Resultados del entrenamiento de YOLO-NAS con dataset Aerial Sheep Image Dataset

Al contrario que con el dataset anterior, se puede observar en este que los modelos aunque mejoran de manera progresiva con respecto al conjunto de entrenamiento, tiene variaciones muy grandes en las pérdidas del conjunto de validación, lo que puede llevar a pensar que se está produciendo alguna clase de sobreajuste. Sin embargo, se puede apreciar que aunque las métricas para YOLO-NAS no sean demasiado buenas, sucediendo lo mismo que en el caso anterior, para YOLOv8 los resultados son considerablemente mejores, alcanzando una mAP de 0,976 a la par que mantiene un recall y una precisión similares.

4.1.3. Sheeps Image Dataset [27] - Roboflow

En las figuras 4.7 y 4.9 se muestran las gráficas obtenidas durante el entrenamiento de los modelos YOLOv8 y YOLO-NAS

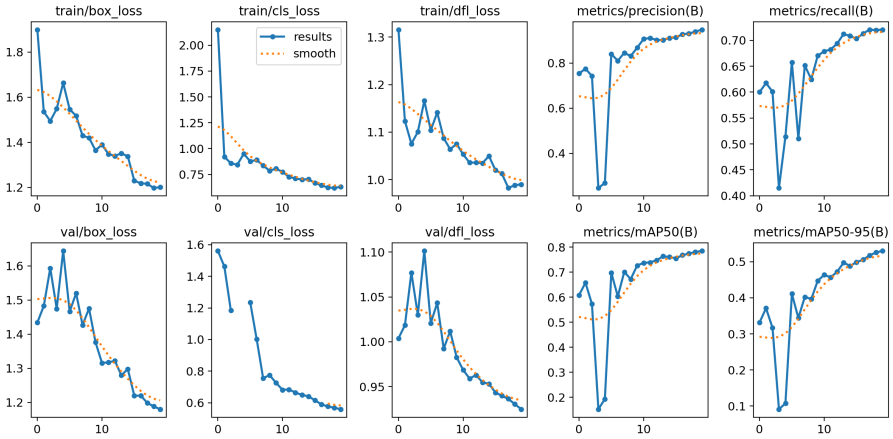

Figura 4.7: Resultados del entrenamiento de YOLOv8 con dataset Sheeps Image Dataset

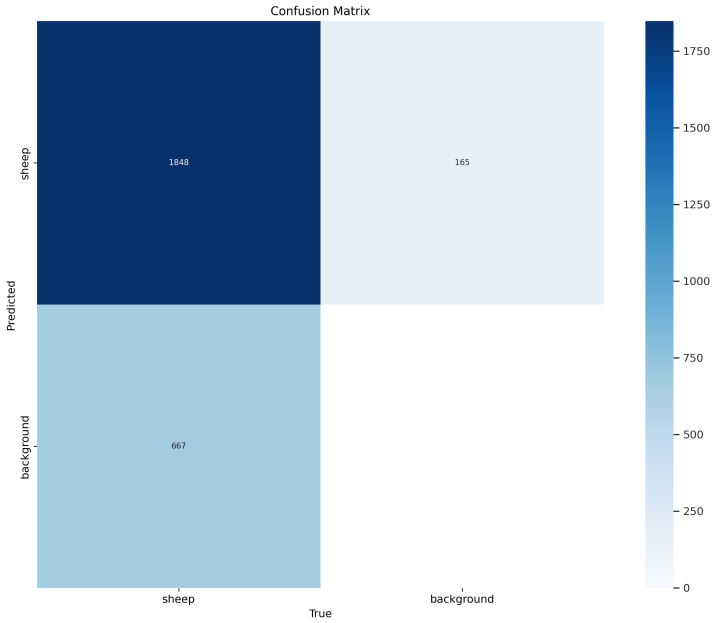

Figura 4.8: Matriz de confusión obtenida del entrenamiento de YOLO-NAS con dataset Sheeps Image Dataset

Los resultados para este dataset son bastante pobres en comparación con el anterior debido en gran medida a la pequeña cantidad de imágenes que posee el dataset (la menor de los cuatro). Sin embargo, dado el bajo número de imágenes y el número de épocas que es bastante pequeño en comparación con lo que se suele dedicar, los modelos presentan resultados bastante aceptables, alcanzando entre un 0,7 y un 0,8 de mAP. En este apartado, se puede observar en la matriz de confusión de la Figura 4.8 que una parte considerable de las ovejas han sido clasificadas como "fondo", es

decir, que no han sido detectadas. Esto puede deberse como se ha mencionado al principio a la falta de datos.

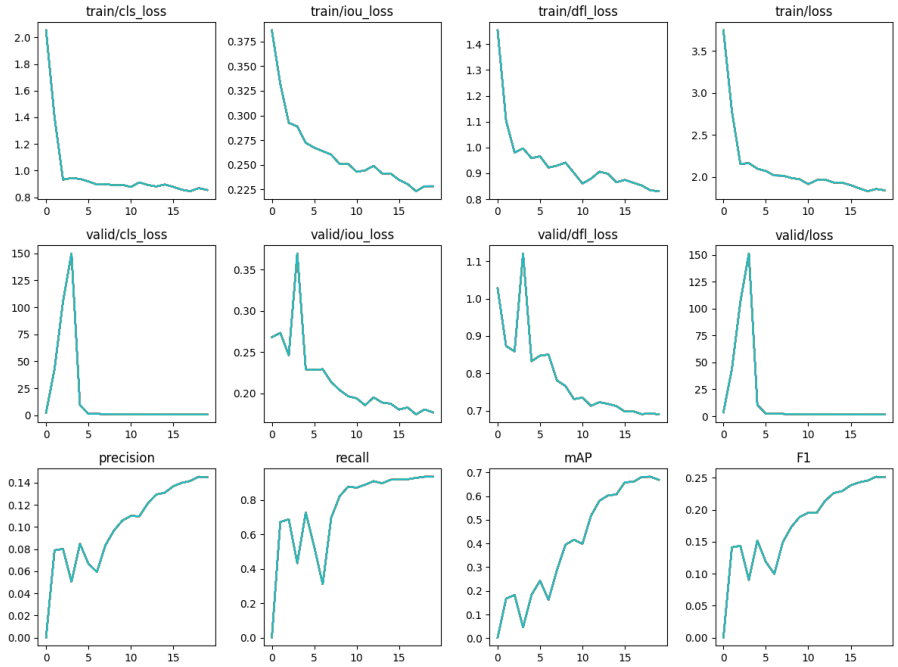

Figura 4.9: Resultados del entrenamiento de YOLO-NAS con dataset Sheeps Image Dataset

4.1.4. UAV sheep images [27] - Roboflow

En las figuras 4.7 y 4.9 se muestran las gráficas obtenidas durante el entrenamiento de los modelos YOLOv8 y YOLO-NAS

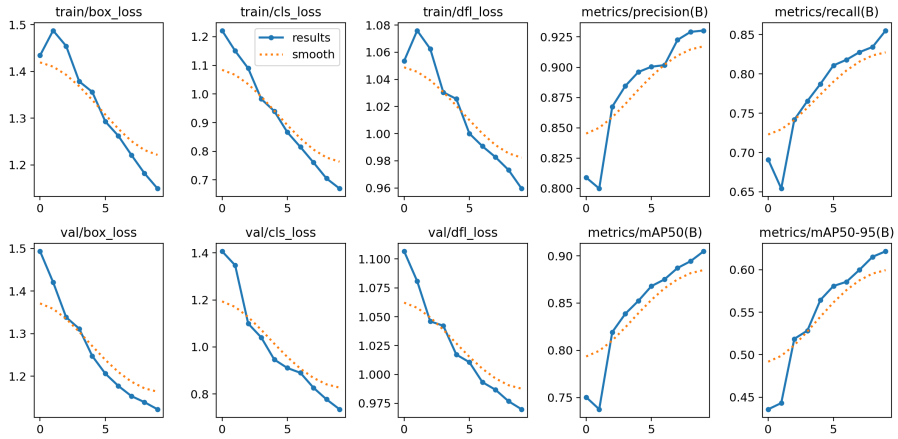

Figura 4.10: Resultados del entrenamiento de YOLOv8 con dataset UAV sheep images

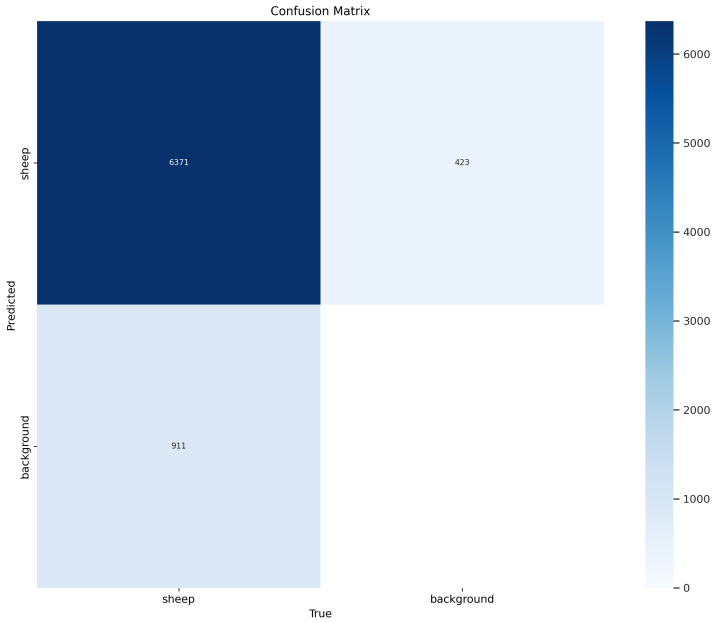

Figura 4.11: Matriz de confusión obtenida del entrenamiento de YOLO-NAS con dataset UAV sheep images

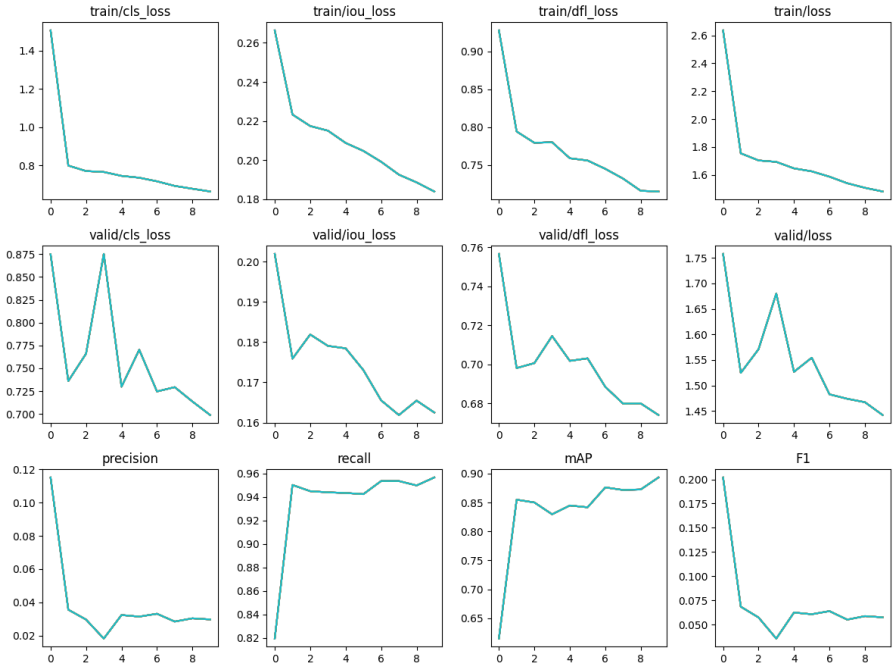

Figura 4.12: Resultados del entrenamiento de YOLO-NAS con dataset UAV sheep images

Para este dataset, ambos modelos desempeñan realmente bien dado el número de épocas que se ha tomado, dado que alcanzan una mAP de hasta un 0,9 en la última

época del entrenamiento. Sin embargo, la precisión para el modelo de YOLO-NAS es realmente baja, lo que quita mérito al valor de la mAP.

4.2. Análisis de resultados

Con el fin de determinar qué modelo es mejor, se ha procedido a realizar una evaluación de ambos modelos tras el entrenamiento en cada dataset. La mAP obtenida en dichas evaluaciones se puede ver en la tabla 4.1.

Tabla 4.1: Resultados evaluación

	Dataset			
Modelo	**Propio**	**Aerial Sheep Images**	**Sheeps Image**	**UAV Sheeps Images**
YOLOv8-m	0,99	0,98	0,79	0,90
YOLO-NAS-m	0,99	0,77	0,69	0,89

Como se puede deducir de los resultados presentes en la tabla, el modelo YOLOv8 resulta mucho más eficiente que YOLO-NAS, incluso en el dataset más pequeño, para el cual ambos tuvieron valores bajos de mAP, YOLOv8 supero en un 10 % a YOLO-NAS.

Así pues, como modelo final se escogería YOLOv8, dada su estabilidad, la facilidad de uso y la eficiencia que presenta al trabajar tanto con datasets grandes como pequeños. Cabe destacar también que la facilidad de uso que presenta YOLOv8 tanto para entrenamiento como para evaluación e inferencia es muy superior a la de YOLO-NAS, lo que favorece uno de los aspectos que se persiguen con el desarrollo de este proyecto. A continuación se muestra la inferencia realizada sobre dos imágenes por los modelos de YOLOv8 entrenados con los datasets de imágenes aéreas.

(a) Imagen aérea de ovejas para inferencia [3]

(b) Imagen aérea de ovejas para inferencia [15]

Figura 4.13: Imágenes para inferencia

(a) Inferencia sobre imagen 1 **(b)** Inferencia sobre imagen 2

Figura 4.14: Resultados inferencia de YOLOv8 entrenado con dataset Aerial Sheep Image Dataset

(a) Inferencia sobre imagen 1 **(b)** Inferencia sobre imagen 2

Figura 4.15: Resultados inferencia de YOLOv8 entrenado con dataset Sheeps Image Dataset

(a) Inferencia sobre imagen 1 **(b)** Inferencia sobre imagen 2

Figura 4.16: Resultados inferencia de YOLOv8 entrenado con dataset UAV Sheep Images

Conclusión

En esta sección se expondrán las conclusiones a las que se ha llegado tras la realización del proyecto, especificando los problemas encontrados durante la misma, el grado de cumplimentación de los requisitos propuestos y las posibilidades a futuro del trabajo realizado.

Aportaciones realizadas

Con el desarrollo del presente Trabajo de Fin de Grado se han cumplido todos los requisitos planteados en la sección 3.2.1. Incluyendo la creación de la guía para el uso y reentreno del modelo. Además, se proporcionan los datasets y los notebooks utilizados para facilitar la comprensión del trabajo realizado.

Trabajos futuros

Como posibilidades a futuro del modelo utilizado, se plantean las siguientes:

1. Integración del modelo con un dron para la completitud del sistema, de manera que el proceso sea completamente automático
2. Integración del modelo con un robot como el que se ve en los videos del dataset propio para la monitorización a nivel de suelo
3. Cambio del sistema de detección a uno de segmentación para un análisis más exhaustivo
4. Mejora del rendimiento del modelo mediante la continuación del entrenamiento en los datasets.
5. Extrapolación del modelo a otro ámbito similar como puede ser la monitorización de animales salvajes.

Problemas encontrados

Durante la realización del proyecto han surgido principalmente dos problemas:

- Durante la prueba de modelos, se intentaron utilizar otros cuatro modelos aparte de los presentados en la sección 3.2.2. Sin embargo, dada la poca documentación proporcionada por los autores de dichos modelos o por la incompatibilidad de versiones que se dio durante el montaje de los entornos para llevar a cabo la ejecución, se tuvieron que abandonar dichos modelos, teniendo que utilizar finalmente los presentados en esta memoria.
- Dado que la versión gratis del servicio de Google Colab solo ofrece la posibilidad de utilizar una GPU durante una cantidad de tiempo determinado al día, la duración de los entrenamientos tuvo que ser reducida, y el *batch size* utilizado para algún dataset tuvo que ser también reducido. Sin embargo, esto ha servido para determinar la capacidad de los modelos para trabajar con pocas horas de entrenamiento.

Opiniones personales

El desarrollo de este Trabajo de Fin de Grado ha sido de gran ayuda para aprender más sobre las distintas aplicaciones que puede tener la inteligencia artificial además de la clasificación o la regresión. Además, el trabajo de investigación llevado a cabo durante este proyecto ha resultado muy útil para aprender a realizar una investigación exhaustiva de manera eficiente y para aprender a documentar de manera correcta dicha investigación, los cuales considero aspectos imprescindibles a la hora de realizar tareas similares a la realizada en este proyecto.

Agradecimientos

Me gustaría agradecer a mi tutora Lidia por la paciencia y dedicación que ha tenido conmigo. Además, sus aportaciones y correcciones han resultado muy útiles a la hora de desarrollar el proyecto.

Lista de referencias

[1] 1. what is the jupyter notebook? — jupyter/ipython notebook quick start guide 0.1 documentation. https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html, (Accessed on 06/28/2023)

[2] About anaconda | anaconda. https://www.anaconda.com/about-us, (Accessed on 06/29/2023)

[3] Aerial sheep image (265×190). https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR-OJdO95EAHRJ3DeSlPNWn1_paPDzhvo7rO-BUMUpFzwQo0ShqQ-qNwl5lWA2zrwdnc_O&usqp=CAU, (Accessed on 07/06/2023)

[4] Control y seguimiento de animales con drones | dronintegra. https://dronintegra.com/control-de-animales-con-drones/, (Accessed on 07/05/2023)

[5] Documentación - overleaf, editor de latex online. https://es.overleaf.com/learn, (Accessed on 07/04/2023)

[6] Google colaboratory. https://colab.research.google.com/?hl=es, (Accessed on 06/27/2023)

[7] Home - ultralytics yolov8 docs. https://docs.ultralytics.com/, (Accessed on 06/26/2023)

[8] Introduction - roboflow docs. https://docs.roboflow.com/, (Accessed on 06/27/2023)

[9] Matplotlib — visualization with python. https://matplotlib.org/, (Accessed on 06/28/2023)

[10] Numpy - about us. https://numpy.org/about/, (Accessed on 06/28/2023)

[11] The pascal visual object classes challenge 2012 (voc2012). http://host.robots.ox.ac.uk/pascal/VOC/voc2012/#testdata, (Accessed on 06/25/2023)

[12] Pytorch. https://pytorch.org/, (Accessed on 07/04/2023)

[13] Qué son las redes neuronales y sus funciones | atria innovation. https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/, (Accessed on 06/29/2023)

[14] Tensorflow core | aprendizaje automático para principiantes y expertos. https://www.tensorflow.org/overview?hl=es-419, (Accessed on 06/28/2023)

[15] Vista aérea del rebaño de ovejas. paisaje con animales de drone. | foto premium. https://www.freepik.es/fotos-premium/vista-aerea-rebano-ovejas-paisaje-animales-drone_14391908.htm, (Accessed on 07/06/2023)

[16] Visual studio code - code editing. redefined. https://code.visualstudio.com/, (Accessed on 06/26/2023)

[17] Welcome to python.org. https://www.python.org/, (Accessed on 06/26/2023)

[18] Aharon, S., Louis-Dupont, Ofri Masad, Yurkova, K., Lotem Fridman, Lkdci, Khvedchenya, E., Rubin, R., Bagrov, N., Tymchenko, B., Keren, T., Zhilko, A., Eran-Deci: Super-gradients (2021). https://doi.org/10.5281/ZENODO.7789328, https://zenodo.org/record/7789328

[19] Alonso, J.: Transfer learning en deep learning: Más allá de nuestros modelos | by josu alonso | medium. https://medium.com/@josumsc/transfer-learning-en-deep-learning-m%C3%A1s-all%C3%A1-de-nuestros-modelos-c304c3a77d4, (Last accessed on 06/25/2023)

[20] Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M., Farhan, L.: Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. Journal of Big Data **8**(1), 53 (Mar 2021). https://doi.org/10.1186/s40537-021-00444-8, https://doi.org/10.1186/s40537-021-00444-8

[21] Arce, J.I.B.: La matriz de confusión y sus métricas – inteligencia artificial –. https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/, (Accessed on 06/29/2023)

[22] Bochkovskiy, A., Wang, C., Liao, H.M.: Yolov4: Optimal speed and accuracy of object detection. CoRR **abs/2004.10934** (2020), https://arxiv.org/abs/2004.10934

[23] Bozinovski, S.: Reminder of the first paper on transfer learning in neural networks, 1976. Informatica (Slovenia) **44** (2020)

[24] de Producciones Ganaderas y Cinegéticas, S.G.: Estudio sobre el sector vacuno de carne en España. Datos SITRAN 2021. Segmento: Engorde de Terneros. Tech. rep., Dirección General de Producciones y Mercados Agrarios. Ministerio de Agricultura, Pesca y Alimentación (2021), https://www.mapa.gob.es/es/ganaderia/temas/produccion-y-mercados-ganaderos/2022_estudioengordeternerosdatossitran2021_pub_tcm30-512343.pdf

[25] Donges, N.: What is transfer learning? a guide for deep learning | built in. https://builtin.com/data-science/transfer-learning, (Last accessed on 06/25/2023)

[26] Elfwing, S., Uchibe, E., Doya, K.: Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. CoRR **abs/1702.03118** (2017), http://arxiv.org/abs/1702.03118

[27] hardee: Sheeps dataset. https://universe.roboflow.com/hardee/sheeps-6bmti (mar 2022), https://universe.roboflow.com/hardee/sheeps-6bmti, visited on 2023-07-04

[28] He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. CoRR **abs/1406.4729** (2014), http://arxiv.org/abs/1406.4729

[29] Hui, J.: Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and yolov3) | by jonathan hui | medium. https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo (Accessed on 06/25/2023)

[30] Jocher, G., Chaurasia, A., Qiu, J.: Ultralytics yolov8 (2023), https://github.com/ultralytics/ultralytics

[31] Jocher, G., Waxmann, S.: Object detection datasets overview - ultralytics yolov8 docs. https://docs.ultralytics.com/datasets/detect/, (Accessed on 06/26/2023)

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

[33] Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. CoRR **abs/1405.0312** (2014), http://arxiv.org/abs/1405.0312

[34] Mehra, A.: Understanding yolov8 architecture, applications & features. https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/, (Accessed on 06/29/2023)

[35] MTandJL: Uav sheep images dataset. https://universe.roboflow.com/mtandjl/uav_sheep_images (may 2023), https://universe.roboflow.com/mtandjl/uav_sheep_images, visited on 2023-07-05

[36] NATHAN, A.R.: Object recognition vs object detection vs image segmentation | kaggle. https://www.kaggle.com/discussions/getting-started/169984, (Last accessed on 06/25/2023)

[37] Norouzzadeh, M.S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M.S., Packer, C., Clune, J.: Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. Proceedings of the National Academy of Sciences (2018). https://doi.org/10.1073/pnas.1719367115, http://www.pnas.org/content/early/2018/06/04/1719367115

[38] O'Shea, K., Nash, R.: An introduction to convolutional neural networks. CoRR **abs/1511.08458** (2015), http://arxiv.org/abs/1511.08458

[39] Park, S.: A guide to two-stage object detection: R-cnn, fpn, mask r-cnn | by sieun park | codex | medium. https://medium.com/codex/a-guide-to-two-stage-object-detection-r-cnn-fpn-mask-r-cnn-and-more-54c2e16843 (Accessed on 06/25/2023)

[40] Qiu, M., Huang, L., Tang, B.: Asff-yolov5: Multielement detection method for road traffic in uav images based on multiscale feature fusion. Remote Sensing **14**, 3498 (07 2022). https://doi.org/10.3390/rs14143498

[41] R, G.S.: confusion matrix| recall| precision| tpr,tnr,fpr,fnr | towards ai. https://pub.towardsai.net/confusion-matrix-179b9c758b55, (Accessed on 06/29/2023)

[42] RangeKing: Brief summary of yolov8 model structure · issue #189 · ultralytics/ultralytics. https://github.com/ultralytics/ultralytics/issues/189#issue-1527158137, (Last accessed on 06/22/2023)

[43] Rančić, K., Blagojević, B., Bezdan, A., Ivošević, B., Tubić, B., Vranešević, M., Pejak, B., Crnojević, V., Marko, O.: Animal detection and counting from uav images using convolutional neural networks. Drones **7**(3) (2023). https://doi.org/10.3390/drones7030179, https://www.mdpi.com/2504-446X/7/3/179

[44] Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR **abs/1506.02640** (2015), http://arxiv.org/abs/1506.02640

[45] Riis: Aerial sheep dataset. https://universe.roboflow.com/riis/aerial-sheep (jun 2022), https://universe.roboflow.com/riis/aerial-sheep, visited on 2023-07-03

[46] Rizzoli, A.: Object detection: Models, architectures & tutorial [2023]. https://www.v7labs.com/blog/object-detection-guide#h1, (Last accessed on 06/25/2023)

[47] Rusk, N.: Deep learning. Nature Methods **13**(1), 35–35 (1 2016), https://doi.org/10.1038/nmeth.3707

[48] Sarwar, F., Griffin, A., Rehman, S.U., Pasang, T.: Detecting sheep in uav images. Computers and Electronics in Agriculture **187**, 106219 (2021). https://doi.org/https://doi.org/10.1016/j.compag.2021.106219, https://www.sciencedirect.com/science/article/pii/S0168169921002362

[49] Singh, A., Pietrasik, M., Natha, G., Ghouaiel, N., Brizel, K., Ray, N.: Animal detection in man-made environments. CoRR **abs/1910.11443** (2019), http://arxiv.org/abs/1910.11443

[50] Team, D.R.: Yolo-nas by deci achieves state-of-the-art performance on object detection using neural architecture search. https://deci.ai/blog/yolo-nas-object-detection-foundation-model/, (Accessed on 07/04/2023)

[51] Tsang, S.H.: Review — cspnet: A new backbone that can enhance learning capability of cnn | by sik-ho tsang | medium. https://sh-tsang.medium.com/review-cspnet-a-new-backbone-that-can-enhance-learning-capability-of-cnn-da7ca (Accessed on 06/29/2023)

[52] Wada, K.: Labelme: Image Polygonal Annotation with Python. https://doi.org/10.5281/zenodo.5711226, https://github.com/wkentaro/labelme

[53] Willi, M., Pitman, R.T., Cardoso, A.W., Locke, C., Swanson, A., Boyer, A., Veldthuis, M., Fortson, L.: Identifying animal species in camera trap images using deep learning and citizen science. Methods in Ecology and Evolution **10**(1), 80–91 (2019). https://doi.org/https://doi.org/10.1111/2041-210X.13099, https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13099

[54] Yohanandan, S.: map (mean average precision) might confuse you! | by shivy yohanandan | towards data science. https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2, (Accessed on 06/29/2023)

[55] Zhu, H., Wei, H., Li, B., Yuan, X., Kehtarnavaz, N.: A review of video object detection: Datasets, metrics and methods. Applied Sciences **10**(21) (2020). https://doi.org/10.3390/app10217834, https://www.mdpi.com/2076-3417/10/21/7834

Anexo A

Control de versiones

Para llevar a cabo un control preciso de todo el código utilizado se ha hecho uso del controlador de versiones Git, así como de su versión web GitHub.

Todos los notebooks utilizados pueden ser encontrados en el siguiente repositorio: `https://github.com/msuarf02/herd-monitorization`. Además, para llevar a cabo la reproducción de todos los experimentos realizados durante este proyecto se pueden descargar los datasets de la página Roboflow utilizando los enlaces disponibles en la sección 4.2. Por otra parte, en el siguiente enlace: `https://drive.google.com/drive/folders/1sGQ398iqD71uM7gHs8sadLGh0TW7djbo?usp=sharing`, se puede acceder a la carpeta de drive conteniendo los videos a partir de los cuales se ha creado el dataset y los datasets utilizados así como los notebooks y los resultados de la ejecución de estos para cada dataset.

Anexo B

Seguimiento de proyecto fin de grado

En esta sección se describen aspectos del proyecto como el seguimiento y la diferencia entre la planificación y el desarrollo real.

B.1. Forma de seguimiento

Para realizar el seguimiento del trabajo se han llevado a cabo reuniones semanales con la tutora en las que se revisaba el trabajo realizado y los objetivos a conseguir para la siguiente semana, llevando así un seguimiento riguroso del avance del proyecto.

B.2. Planificación inicial

La planificación inicial se puede visualizar en la sección 2.2.

B.3. Planificación final

En cuanto a los periodos de estudio del estado de la cuestión y el de prueba de modelos, se cumplieron los plazos para ambos sprints, encontrando los estudios realizados, las tecnologías más utilizadas y los datasets a utilizar. Sin embargo, el sprint dedicado al reentreno de modelos tuvo que ser ampliado una semana dada la limitación que impone Google Colab para la utilización de GPU de manera gratuita.

Finalmente, la evaluación y comparación de los modelos pudo hacerse a tiempo dada la brevedad que estas supusieron

Anexo C

Manual de usuario

C.0.1. YOLOv8

```
[ ]  1 from ultralytics import YOLO
     2
     3 root = "/content/drive/MyDrive/TFG/animal-detection/"
     4 dataset = "aerial_sheep_2"
```

Figura C.1: Definición del directorio raíz del proyecto y del dataset a utilizar

```
[ ]  1 from google.colab import drive
     2 drive.mount('/content/drive')

     Mounted at /content/drive
```

Figura C.2: Montaje de drive en el entorno virtual

```
[ ]  1 model = YOLO('yolov8m.pt')
     2 results = model.train(
     3    data= root + 'datasets/' + dataset +'/data.yaml',
     4    imgsz=640,
     5    epochs=20,
     6    batch=16,
     7    name=root + 'YOLOv8/'+ dataset +'/yolov8m'
     8 )
```

Figura C.3: Elección de modelo y entrenamiento del mismo

```
[ ]  1 model = YOLO(root +'YOLOv8/' + dataset +'/yolov8m/weights/best.pt')
     2 test_results = model.val(
     3     data = root + 'datasets/' + dataset +'/data.yaml',
     4     imgsz=640,
     5     batch=32,
     6     name= root+ 'YOLOv8/'+ dataset +'/test'
     7 )
```

Figura C.4: Evaluación del mejor modelo del entrenamiento con el conjunto de test del dataset

```
[19]  1 model.predict(source, save=True, save_conf=False, hide_labels=True, conf=0.5, project=root, name='YOLOv8/'+ dataset +'/Results')
```

Figura C.5: Inferencia sobre imágenes y video de un directorio

C.0.2. YOLO-NAS

```
[ ]  1 dataset_params = {
     2     'data_dir': root +'datasets/'+ dataset,
     3     'train_images_dir':'train/images',
     4     'train_labels_dir':'train/labels',
     5     'val_images_dir':'valid/images',
     6     'val_labels_dir':'valid/labels',
     7     'test_images_dir':'test/images',
     8     'test_labels_dir':'test/labels',
     9     'classes': classes[dataset]
    10 }
```

Figura C.6: Definición de los directorios de imágenes y etiquetas

```python
from IPython.display import clear_output

import collections
collections.Iterable = collections.abc.Iterable

train_data = coco_detection_yolo_format_train(
    dataset_params={
        'data_dir': dataset_params['data_dir'],
        'images_dir': dataset_params['train_images_dir'],
        'labels_dir': dataset_params['train_labels_dir'],
        'classes': dataset_params['classes']
    },
    dataloader_params={
        'batch_size':16,
        'num_workers':2
    }
)

val_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': dataset_params['data_dir'],
        'images_dir': dataset_params['val_images_dir'],
        'labels_dir': dataset_params['val_labels_dir'],
        'classes': dataset_params['classes']
    },
    dataloader_params={
        'batch_size':16,
        'num_workers':2
    }
)

test_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': dataset_params['data_dir'],
        'images_dir': dataset_params['test_images_dir'],
        'labels_dir': dataset_params['test_labels_dir'],
        'classes': dataset_params['classes']
    },
    dataloader_params={
        'batch_size':16,
        'num_workers':2
    }
)

clear_output()
```

Figura C.7: Definición de los parámetros de los conjuntos de datos

```python
from super_gradients.training import models
model = models.get(nombre_modelo,
                   num_classes=len(dataset_params['classes']),
                   pretrained_weights="coco"
                   )
```

```python
from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import DetectionMetrics_050
from super_gradients.training.models.detection_models.pp_yolo_e import PPYoloEPostPredictionCallback

train_params = {
    'silent_mode': False,
    "average_best_models":True,
    "warmup_mode": "linear_epoch_step",
    "warmup_initial_lr": 1e-6,
    "lr_warmup_epochs": 3,
    "initial_lr": 5e-4,
    "lr_mode": "cosine",
    "cosine_final_lr_ratio": 0.1,
    "optimizer": "Adam",
    "optimizer_params": {"weight_decay": 0.0001},
    "zero_weight_decay_on_bias_and_bn": True,
    "ema": True,
    "ema_params": {"decay": 0.9, "decay_type": "threshold"},
    "max_epochs": 10,
    "mixed_precision": True,
    "loss": PPYoloELoss(
        use_static_assigner=False,
        num_classes=len(dataset_params['classes']),
        reg_max=16
    ),
    "valid_metrics_list": [
        DetectionMetrics_050(
            score_thres=0.1,
            top_k_predictions=300,
            num_cls=len(dataset_params['classes']),
            normalize_targets=True,
            post_prediction_callback=PPYoloEPostPredictionCallback(
                score_threshold=0.01,
                nms_top_k=1000,
                max_predictions=300,
                nms_threshold=0.7
            )
        )
    ],
    "metric_to_watch": 'mAP@0.50'
}
```

Figura C.8: Obtención del modelo y definición de los parámetros de entrenamiento

```python
trainer.train(model=model,
              training_params=train_params,
              train_loader=train_data,
              valid_loader=val_data)
```

Figura C.9: Entrenamiento del modelo

```python
best_model = models.get(nombre_modelo,
                        num_classes=len(dataset_params['classes']),
                        checkpoint_path=root + "YoloNAS/"+ dataset +"/ckpt_best.pth")
```

```
[2023-07-05 18:35:33] INFO - checkpoint_utils.py - Successfully loaded model weights from /content/drive/MyDrive/TFG/animal-detection/YoloNAS/UAV_sheep/ckpt_best.pth EMA checkpoint.
```

```python
trainer.test(model=best_model,
             test_loader=test_data,
             test_metrics_list=DetectionMetrics_050(score_thres=0.1,
                                                    top_k_predictions=300,
                                                    num_cls=len(dataset_params['classes']),
                                                    normalize_targets=True,
                                                    post_prediction_callback=PPYoloEPostPredictionCallback(score_threshold=0.01,
                                                                                                           nms_top_k=1000,
                                                                                                           max_predictions=300,
                                                                                                           nms_threshold=0.7)
                                                   ))
```

Figura C.10: Obtención del mejor modelo del entrenamiento y evaluación con conjunto de test