Trabajo de Fin de Máster
Máster en Sistemas Inteligentes
Julio, 2023

Sistema de reconocimiento de patrones en imágenes satelitales para la detección de objetos

Sergio Fernández Marcos
Tutora: Vivian López Batista
Cotutor: Gabriel Villarubia González

Departamento de Informática y Automática
Universidad de Salamanca

Resumen

A lo largo de los últimos años, la inteligencia artificial (IA) ha experimentado un avance significativo en la detección de objetos en imágenes. La aplicación de técnicas de aprendizaje profundo y de *machine learning* ha revolucionado la capacidad de los algoritmos para identificar y localizar objetos en una variedad de entornos visuales.

En este artículo, se presenta un estudio detallado sobre la detección de objetos en imágenes satelitales utilizando la arquitectura YOLO (*You Only Look Once*). Se realiza el proceso de entrenamiento de varios modelos YOLO utilizando un conjunto de datos específico, con el objetivo de comparar y analizar los resultados obtenidos.

La investigación comienza con un marco teórico que aborda los conceptos clave de la IA, el *deep learning* y el *machine learning*, y se enfoca en las arquitecturas más destacadas para la detección de objetos en imágenes. Se destacan las características, las ventajas y la evolución de los modelos YOLO en particular, y se establece una base sólida para comprender el entrenamiento y evaluación de los algoritmos en el caso de estudio.

A continuación, se presenta un caso de estudio en el cual se detalla el proceso de entrenamiento de cada algoritmo YOLO paso a paso. Se describen las configuraciones utilizadas, el conjunto de datos empleados y los métodos de evaluación para garantizar la precisión y la eficiencia en la detección de objetos.

Finalmente, se presentan y analizan los resultados obtenidos al comparar los diferentes algoritmos YOLO entrenados.

A pesar de la probada eficacia de diversos métodos de detección en múltiples contextos, seleccionar el más adecuado para cada implementación concreta representa un reto. La comparación y evaluación de distintas técnicas, centrada en métricas de rendimiento y conjuntos de datos representativos, son fundamentales para determinar la mejor opción para una detección de objetos precisa y confiable en imágenes.

Palabras clave: reconocimiento de patrones, detección de objetos, imágenes satelitales, algoritmos de aprendizaje profundo, técnicas aprendizaje automático.

Abstract

Over recent years, artificial intelligence (AI) has made significant strides in object detection within images. The application of deep learning techniques and machine learning has revolutionized the ability of algorithms to identify and locate objects in a variety of visual environments. In this paper, a detailed study on object detection in satellite images using the YOLO (You Only Look Once) architecture is presented. The training process of various YOLO models using a specific dataset is carried out, with the aim of comparing and analyzing the results obtained.

The research begins with a theoretical framework that addresses the key concepts of AI, deep learning, and machine learning, focusing on the most prominent architectures for object detection in images. The features, advantages, and evolution of YOLO models in particular are highlighted, and a solid foundation is established to understand the training and evaluation of algorithms in the case study.

Next, a case study is presented in which the training process of each YOLO algorithm is detailed step by step. The settings used, the datasets employed, and the evaluation methods to ensure accuracy and efficiency in object detection are described.

Finally, the results obtained when comparing the different trained YOLO algorithms are presented and analyzed.

Despite the proven effectiveness of various detection methods in multiple contexts, selecting the most suitable for each specific implementation represents a challenge. The contrasting and evaluation of different techniques, centered on performance metrics and representative datasets, are fundamental to determine the best option for precise and reliable object detection in images.

Keywords: pattern recognition, object detection, satellite images, deep Learning algorithms, machine learning techniques.

Índice

Índice de figuras

Indice de tablas

1. Introducción

La detección de objetos en imágenes es un factor indispensable en el campo de la visión por ordenador o visión artificial, que consiste en identificar y localizar la presencia de objetos específicos en una imagen[3]. El objetivo es entrenar un modelo o algoritmo que pueda reconocer patrones visuales característicos de los objetos de interés y luego aplicar este modelo a nuevas imágenes para detectar y delimitar la ubicación de dichos objetos. En las últimas décadas, ha experimentado una gran evolución gracias al desarrollo de técnicas de aprendizaje automático y algoritmos de visión por ordenador[4]. Estos avances han permitido abordar con mayor eficacia y precisión la tarea de identificar y localizar objetos en imágenes. Los métodos más habituales y ampliamente utilizados incluyen:

Detección basada en redes neuronales convolucionales (CNN, *Convolutional Neural Networks*): Estas redes neuronales son especialmente eficaces para la detección de objetos en imágenes debido a su capacidad para aprender características discriminatorias de manera automática. Las arquitecturas de redes neuronales convolucionales más comunes para la detección de objetos incluyen el detector de objetos en imágenes (YOLO, *You Only Look Once*) y el detector de regiones con redes neuronales convolucionales (R-CNN, *Region-Based Convolutional Neural Networks*).

Detección basada en aprendizaje automático: Este enfoque implica el entrenamiento de modelos de aprendizaje automático, como las máquinas de vectores de soporte (SVM, *Support Vector Machines*), los bosques aleatorios (*Random Forests*) o los clasificadores basados en CNN, utilizando un conjunto de datos etiquetados que contienen ejemplos de objetos y fondos[5]. Estos modelos aprenden a discriminar entre objetos y fondos y pueden utilizarse para detectar objetos en nuevas imágenes.

Detección basada en características: Este enfoque implica la extracción manual o automática de características visuales relevantes de los objetos, como bordes, texturas o colores, y el uso de algoritmos de clasificación para identificar y localizar los objetos en función de estas características[6].

Aunque estos métodos han demostrado su eficacia en una amplia gama de aplicaciones, desde la detección de rostros y vehículos hasta la detección de objetos en imágenes médicas o satelitales. Sigue siendo un desafío encontrar el mejor enfoque que pueda adaptarse a cada implementación práctica. Siendo fundamental contrastar y evaluar los diferentes métodos de detección de objetos en imágenes, para determinar cuál funciona mejor en un caso de uso específico. La evaluación comparativa basada en métricas de rendimiento y el uso de conjuntos de datos representativos permiten identificar el método más apropiado para lograr una detección precisa y confiable de objetos en imágenes.

1.1. Objetivos

Este trabajo tiene como objetivo principal la investigación y evaluación de diferentes métodos de detección de objetos, con el objetivo de poder seleccionar el enfoque que mejor se adapte a una aplicación concreta.

Las principales tareas para conseguir este objetivo son las siguientes:

1. Realizar un estudio del estado del arte de los diferentes métodos de detección de objetos en imágenes existentes hasta el momento.

2. Crear y preparar un conjunto de entrenamiento formado por imágenes satelitales.

3. Entrenamiento de los modelos de detección de objetos con el conjunto de datos.

4. Evaluar el rendimiento de los algoritmos y métodos de detección implementados.

5. Contrastar y evaluar los diferentes métodos de detección de objetos con el objetivo de poder seleccionar el enfoque que mejor se adapte a una aplicación concreta.

2. Estado del arte

El objetivo principal de este apartado es explorar diferentes enfoques y técnicas para la detección de algoritmos y patrones de objetos en imágenes satelitales. Se explorarán algoritmos basados en aprendizaje automático, así como métodos de procesamiento de imágenes y segmentación para mejorar la precisión y eficiencia de la detección.

2.1. Inteligencia artificial

La inteligencia artificial (IA) es un campo de estudio y desarrollo de tecnología que se centra en la creación de sistemas y programas capaces de realizar tareas que normalmente requieren de la inteligencia humana. Estos sistemas son capaces de aprender, razonar, percibir, planificar y tomar decisiones, entre otras capacidades[7].

La IA se basa en la idea de que las máquinas pueden procesar y analizar grandes cantidades de datos para identificar patrones y aprender de ellos. Esto se logra a través de algoritmos y modelos matemáticos que permiten a las máquinas reconocer y comprender información, tomar decisiones y resolver problemas.

En el contexto de la IA, una de las ramas más relevantes y utilizadas es el aprendizaje automático o *machine learning*.

2.2. Aprendizaje automático

El aprendizaje automático, o *machine learning*, es una parte de la IA que se enfoca en el desarrollo de algoritmos y modelos que permiten a los ordenadores aprender y encontrar información útil automáticamente a partir de recopilar información y experiencias pasadas, sin una programación explícita[8].

En el aprendizaje automático, los algoritmos utilizan técnicas estadísticas y computacionales para reconocer patrones y tomar decisiones o hacer predicciones sin ser específicamente programados para cada situación[9]. Estos algoritmos son entrenados con datos de entrada y sus correspondientes resultados esperados, a través de un proceso llamado entrenamiento.

Existen diferentes enfoques en el aprendizaje automático, incluyendo:

- Aprendizaje supervisado: Se proporcionan datos de entrenamiento etiquetados, donde se conoce la respuesta esperada para cada ejemplo. El modelo aprende a mapear las características de entrada a las etiquetas correspondientes, y luego puede predecir las etiquetas para nuevas instancias.

- Aprendizaje no supervisado: Los datos de entrenamiento no están etiquetados y el objetivo es descubrir patrones o estructuras ocultas en los datos. El modelo aprende a agrupar o clasificar los datos basándose en similitudes o diferencias entre ellos.

- Aprendizaje por refuerzo: El modelo aprende a tomar decisiones secuenciales mediante la interacción con un entorno. Se le proporcionan recompensas o penalizaciones según las acciones que toma, y su objetivo es aprender a maximizar las recompensas a lo largo del tiempo.

2.3. Aprendizaje profundo

El aprendizaje profundo, o *deep learning*, es una rama del aprendizaje automático que se centra en el entrenamiento de redes neuronales profundas para aprender y extraer representaciones de alto nivel de los datos[10]. Las redes neuronales profundas están compuestas por múltiples capas de nodos interconectados, que se denominan unidades o neuronas, y cada capa procesa y transforma los datos de entrada de manera gradual y jerárquica.

El aprendizaje profundo ha revolucionado muchos campos de la inteligencia artificial, como el procesamiento de imágenes, el procesamiento del lenguaje natural y la visión por ordenador[11]. La capacidad de las redes neuronales profundas para aprender automáticamente características y patrones complejos en los datos ha permitido avances significativos en tareas como la clasificación de imágenes, el reconocimiento de voz, la traducción automática y la generación de texto.

Algunos conceptos clave en el *deep learning* incluyen:

Redes neuronales convolucionales: Son ampliamente utilizadas en el procesamiento de imágenes y reconocimiento visual. Estas redes están diseñadas específicamente para aprovechar la estructura espacial de los datos de imagen, utilizando filtros convolucionales y capas de agrupación para aprender características locales y luego combinarlas en representaciones más abstractas.

Redes neuronales recurrentes: Son utilizadas en el procesamiento del lenguaje natural y otras tareas secuenciales. Tienen conexiones recurrentes que les permiten mantener una memoria interna y capturar dependencias a largo plazo en los datos secuenciales. Esto las hace especialmente útiles para el análisis de texto, la generación de texto y la traducción automática.

Aprendizaje por transferencia: Es una técnica en la que se aprovecha el conocimiento aprendido en una tarea para mejorar el rendimiento en otra tarea relacionada. Mediante la transferencia de los pesos y conocimientos de una red neuronal entrenada en un conjunto de datos grande, se puede acelerar y mejorar el aprendizaje en un conjunto de datos más pequeño o diferente.

Además del aprendizaje supervisado, donde se proporcionan etiquetas de clase, el aprendizaje profundo también puede abordar el aprendizaje no supervisado. Esto implica el uso de técnicas como *autoencoders*, redes generativas adversarias y agrupamiento (*clustering*) para descubrir patrones ocultos y estructuras en los datos sin la necesidad de etiquetas previas.

2.4. Detección de objetos

La detección de objetos en imágenes es una tarea fundamental en el campo de la visión por ordenador. Se refiere al proceso de identificar y localizar objetos de interés en una imagen o secuencia de vídeo. A diferencia de la clasificación de imágenes, que solo identifica la presencia de un objeto, la detección de objetos también determina dónde se encuentra el objeto en la imagen y dibuja un cuadro delimitador alrededor del objeto. Tiene una amplia gama de aplicaciones, incluyendo la seguridad y vigilancia, el reconocimiento facial, la conducción autónoma, y la inspección y control de calidad en la fabricación, entre otros. Existen varios métodos y algoritmos para la detección de objetos, que van desde técnicas tradicionales basadas en características hasta enfoques más modernos basados en el aprendizaje profundo. Los métodos basados en el aprendizaje profundo, como las CNN, han demostrado ser muy efectivos para la detección de objetos. Las CNN son capaces de aprender automáticamente características de nivel inferior y superior de las imágenes, lo que las hace muy robustas a las variaciones en la iluminación, la orientación y la escala. Además, las redes neuronales convolucionales pueden ser entrenadas con grandes cantidades de datos, lo que les permite aprender a reconocer una amplia variedad de objetos.

2.5. Elaboración de un detector de objetos

En este apartado se va a realizar una descripción detallada de las partes que conforman la elaboración de un detector de objetos eficiente y preciso.

2.5.1. Preprocesamiento de datos

El primer paso para entrenar un detector de objetos es seleccionar un conjunto de datos adecuado. Este conjunto de datos debe contener imágenes que representen las clases de objetos que se desean detectar. Es esencial tener una variedad suficiente de imágenes que cubran diferentes condiciones, ángulos, tamaños y variaciones de iluminación para lograr un modelo robusto. Además, se debe realizar un proceso de filtrado para eliminar imágenes borrosas o seleccionadas incorrectamente, ya que podrían afectar la calidad y fiabilidad de los datos. Es esencial garantizar que el conjunto de datos utilizado esté compuesto por imágenes nítidas y representativas de cada clase de objeto, lo cual es fundamental para obtener resultados óptimos en el entrenamiento y la evaluación del modelo.

Una vez que se ha seleccionado el conjunto de imágenes, el siguiente paso es redimensionar todas las imágenes a un mismo tamaño. Esto se realiza con el objetivo de mejorar el rendimiento del modelo, al estandarizar el tamaño de las imágenes. Al tener todas las imágenes en el mismo tamaño, se minimiza la pérdida de la información visual y se evitan posibles sesgos causados por diferencias en las dimensiones. Esto permite que el modelo se enfoque en los patrones y características relevantes, sin verse afectado por las diferencias de escala, lo cual mejora la precisión y la robustez del modelo.

Una vez que se ha completado el redimensionamiento, se procede al etiquetado de las imágenes. En el contexto de imágenes satelitales, es importante tener en cuenta que cada clase de objeto tenga el mismo nivel de *zoom*, es decir, que se debe mantener una consistencia en el tamaño aparente de los objetos de interés en relación con el mapa satelital. Para lograr esto, se debe realizar un ajuste cuidadoso para que las imágenes de una misma clase tengan un tamaño y resolución similares.

El proceso de etiquetado consiste en asignar etiquetas o anotaciones a las imágenes, para indicar la presencia y la ubicación de los objetos que se pretenden detectar. En el contexto de esta investigación, al tratarse de un aprendizaje supervisado, se deben establecer previamente las diferentes clases de objetos. Además, separar y etiquetar cada una de ellas de manera precisa en cada imagen. Esto implica definir y delimitar las regiones correspondientes a cada clase mediante el uso de cajas delimitadoras o *bounding boxes*. El etiquetado preciso de cada imagen es esencial para entrenar al modelo de detección de objetos de manera efectiva.

2.5.2. Entrenamiento del modelo

En esta etapa, es común dividir el conjunto de datos etiquetados en conjuntos de entrenamiento y conjunto de prueba, mediante una división conocida como *train/test split*. El conjunto de entrenamiento se utiliza para entrenar el modelo, mientras que el conjunto de prueba se utiliza para evaluar su rendimiento una vez entrenado.

La división entre conjuntos de entrenamiento y prueba se realiza para evaluar la capacidad del modelo de generalizar y detectar objetos en nuevas imágenes que no ha visto durante el entrenamiento. La proporción de la división puede variar dependiendo del tamaño del conjunto de datos y de las necesidades específicas del problema. Por ejemplo, se puede utilizar una división del 80 % para entrenamiento y el 20 % restante para pruebas.

Además, es común aplicar técnicas de aumento de datos al conjunto de entrenamiento. Estas técnicas implican la aplicación de transformaciones a las imágenes existentes para aumentar la variabilidad y diversidad de los datos de entrada al modelo. Dos técnicas comunes de aumento de datos son el volteo y la rotación.

El volteo consiste en generar imágenes espejo, ya sea horizontal o verticalmente, para ampliar la variabilidad en la orientación y el ángulo de visión de los objetos presentes en las imágenes. Esto permite que el modelo aprenda a reconocer patrones desde diferentes perspectivas y mejore su capacidad de generalización.

La rotación se utiliza para simular diferentes puntos de vista en las imágenes. Al aplicar rotaciones aleatorias en varios ángulos, el modelo puede aprender a reconocer objetos desde diferentes orientaciones y ángulos de visión, lo que ayuda a mejorar su capacidad para detectar objetos en diversas condiciones.

Es importante destacar que los parámetros de entrenamiento pueden variar según el modelo que se esté utilizando. Cada modelo tiene sus propios hiperparámetros y configuraciones que afectan su rendimiento y capacidad de detección. Estos parámetros incluyen: la arquitectura de red, la tasa de aprendizaje, que es un hiperparámetro que controla la velocidad a la que el modelo aprende durante el entrenamiento y determina el tamaño de los pasos que se toman para ajustar los pesos de las conexiones en la red neuronal. Además, el número de capas que es la cantidad de capas en la red neuronal utilizadas para el modelo de detección de objetos. Por otra parte, el tamaño del lote o *batch size*, que se refiere al número de ejemplos de entrenamiento que se utilizan en una iteración, antes de que los pesos del modelo se actualicen. Finalmente, el número de épocas de entrenamiento, una época, se refiere a una pasada completa del conjunto de datos de entrenamiento durante el entrenamiento del modelo. Ajustar estos parámetros adecuadamente es fundamental para obtener un modelo óptimo en términos de precisión y eficiencia.

2.5.3. Detección de los objetos

Finalmente, llegamos al proceso de detección de objetos utilizando la red que ha sido entrenada específicamente para esta tarea. Una vez que se han ajustado los

pesos de la red durante el entrenamiento, se proporcionan a la red datos de entrada, que en este caso son imágenes que pueden contener el objeto que deseamos detectar.

En este punto, el sistema realiza la detección de los objetos presentes en las imágenes de entrenamiento, utilizando el proceso que hemos descrito anteriormente. Los cuadros delimitadores se dibujan alrededor de los objetos detectados, indicando su presencia. Además, en las arquitecturas que vamos a detallar más adelante, obtenemos información precisa sobre la ubicación de los objetos en la imagen, así como la clase a la que pertenecen.

2.6. Revisión sistemática de la literatura

Dentro del campo de la interpretación de imágenes satelitales, el reconocimiento de patrones desempeña un papel fundamental en la detección y análisis de objetos de interés. Estas imágenes contienen una gran cantidad de información geoespacial valiosa, pero su procesamiento y comprensión pueden resultar desafiantes debido a la complejidad de las interacciones presentes en el espacio-tiempo narrativo.

En la actualidad, existen diferentes plataformas que permiten obtener y manejar imágenes satelitales o capturadas por drones, lo que ha facilitado el acceso a este tipo de datos. Para aprovechar al máximo estas imágenes, se han desarrollado técnicas avanzadas de aprendizaje automático, especialmente algoritmos de *Deep Learning*, que permiten el reconocimiento automático de objetos en imágenes satelitales.

El objetivo principal de la revisión sistemática de la bibliografía, *mapping*, es realizar una revisión sistemática y exhaustiva de los métodos existentes en el campo. A través de esta revisión, se busca obtener una comprensión profunda de los algoritmos, técnicas y enfoques utilizados para el reconocimiento de objetos en imágenes satelitales.

2.6.1. Metodología

Para formular las preguntas de la revisión sistemática se ha utilizado el proceso PICO, cada letra de PICO[12] representa un componente clave de la pregunta:

- P: Problema o Población. Se refiere al grupo de interés o a la población objetivo que se estudiará en la revisión sistemática. Aquí se debe describir claramente las características demográficas, condiciones de salud u otros aspectos relevantes de la población en cuestión.
- I: Intervención o Exposición. Indica la intervención, tratamiento o exposición de interés que se examinará en la revisión sistemática. Puede ser un enfoque terapéutico, una intervención preventiva, una exposición a un factor de riesgo, etc.
- C: Comparación. Representa la intervención, tratamiento o exposición alternativa que se utilizará como punto de comparación en la revisión sistemática.

Puede ser un placebo, otro tratamiento, una intervención diferente o la ausencia de intervención.

- O: Resultado. Se refiere a los resultados o los desenlaces específicos que se esperan evaluar en la revisión sistemática. Puede ser una medida de eficacia, un resultado clínico, una mejora en la calidad de vida u otros resultados relevantes.

El marco metodológico para la realización de esta revisión sistematica de los métodos y algoritmos utilizados para la detección de objetos en imágenes satelitales es PRISMA (*Preferred Reporting Items for Systematic Reviews and Meta-Analyses*). El marco se compone de un conjunto de elementos que deben ser incluidos en el informe de una revisión sistemática, lo cual facilita la comprensión y la evaluación crítica del estudio. Estos elementos incluyen información sobre el diseño del estudio, los métodos utilizados para la selección y extracción de los datos, los criterios de inclusión y exclusión, los resultados obtenidos y el análisis de la calidad de los estudios incluidos.

Por último, se ha utilizado la herramienta *software* Parsifal para el proceso de la elaboración de la revisión de la literatura sistemática.

2.6.2. Proceso de *Mapping*

La detección de objetos en imágenes satelitales es un campo de investigación y aplicación que ha existido durante varias décadas. Sin embargo, ha habido avances significativos en los últimos años gracias al desarrollo de técnicas avanzadas de aprendizaje automático, especialmente en el ámbito de los algoritmos de *Deep Learning*. El uso de estas técnicas ha permitido mejorar significativamente la precisión y eficiencia de la detección de objetos en imágenes satelitales. Las preguntas que me he formulado según el método PICO (Población, Intervención, Comparación, Resultado) para mi proyecto son las siguientes.

1. Primera pregunta: ¿Los artículos estudian técnicas de identificación de objetos mediante imágenes satelitales?
2. Segunda pregunta: ¿Cuáles son los algoritmos utilizados para el reconocimiento de objetos en imágenes satelitales mediante el análisis de patrones?
3. Tercera pregunta: ¿Cuáles son las técnicas y enfoques utilizados para el reconocimiento de objetos en imágenes satelitales?
4. Cuarta pregunta: ¿Existen artículos donde se aplican los algoritmos de reconocimiento de imágenes en casos reales?
5. Quinta pregunta: ¿Cuáles son las métricas que definen mejor el rendimiento de los algoritmos de reconocimiento de patrones en imágenes satelitales?

El siguiente paso es extraer los artículos de investigación relacionados, en mi caso he seleccionado Web of Science, IEEE Xplore y Scopus como las fuentes principales

para obtener los artículos que serán objeto de estudio. Estas bases de datos son ampliamente reconocidas y contienen una amplia gama de revistas académicas en diversos campos. Además, la Universidad de Salamanca tiene licencias para acceder a estas bases de datos, lo que garantiza la disponibilidad de una gran cantidad de artículos relevantes para nuestra investigación. Al emplear *queries* específicas, que aparecen a continuación, es posible obtener resultados más precisos y relevantes, ya que se pueden ajustar los criterios de búsqueda según los términos y requisitos específicos de la investigación. Además, esta metodología permite ahorrar tiempo al eliminar la necesidad de revisar manualmente una gran cantidad de artículos irrelevantes.

- **Web of Science (WoS):**

```
(TS=("satellite images" OR "object detection"))
    AND (TS=("deep Learning algorithms" OR "machine learning techniques")
        AND (TS=("object recognition" OR "algorithm analysis" OR "techniques")))
```

- **IEEExplorer:**

```
((object detection* AND satellite images*) AND (deep Learning algorithms*
    OR machine learning techniques*) AND (object recognition* OR algorithm analysis
        OR techniques*)) AND (ContentType:Journals OR ContentType:Conference-Papers)
```

- **Scopus:**

```
TITLE-ABS-KEY ( ( "pattern recognition"  AND  "object detection" )  AND
    ( "satellite images" ) )  AND  ( TITLE-ABS-KEY ( "algorithms" )
        OR  TITLE-ABS-KEY ( "machine learning" )  OR  TITLE-ABS-KEY ( "deep learning" ) )
        AND  PUBYEAR  >  2016  AND  ( LIMIT-TO ( LANGUAGE ,  "English" )
            OR  LIMIT-TO ( LANGUAGE ,  "Spanish" ) )
                AND  ( LIMIT-TO ( DOCTYPE ,  "ar" ) )  AND
                    (  LIMIT-TO ( OA ,  "publisherfullgold" ) )
                        AND  ( LIMIT-TO ( PUBYEAR ,  2023 )  OR
                            LIMIT-TO ( PUBYEAR ,  2022 )  OR
                             LIMIT-TO ( PUBYEAR ,  2021 )
                               OR  LIMIT-TO ( PUBYEAR ,  2020 )
                                   OR  LIMIT-TO ( PUBYEAR ,  2019 )
                                    OR  LIMIT-TO ( PUBYEAR ,  2018 )
                                     OR  LIMIT-TO ( PUBYEAR ,  2017 ) )
```

Figura 1: Consultas utilizadas.

En la Figura 1 se pueden observar las peticiones realizadas, se ha intentado acotar la búsqueda lo máximo posible, ya que hay gran cantidad de artículos relacionados con la detección de objetos. En las tres bases de datos se han seguido unos criterios para seleccionar los artículos.

- Los artículos están disponibles para descargar o ver completamente con la licencia de la Universidad de Salamanca.
- Los artículos están escritos en inglés o español.
- Los artículos tienen que estar publicados entre 2017 y 2023.

A su vez, como se puede observar, tienen que tener palabras clave como *pattern recognition*, *object detection*, *satellite images*, *algorithms*, *Deep Learning algorithms* o *machine learning techniques*. El conjunto de artículos obtenidos contiene 142 artículos, en la Figura 2 se puede observar el porcentaje de artículos seleccionado de cada base de datos.

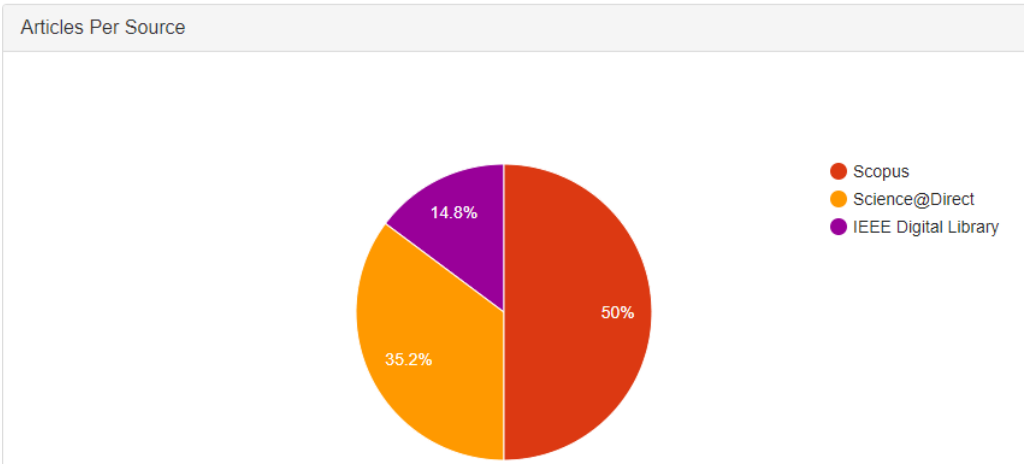

Figura 2: Número de artículos seleccionados

Después se han introducido los artículos obtenidos en plataforma de Parsifal, donde se realizó un segundo cribado de artículos entrando en el enlace de cada artículo y descartando los distintos artículos según los siguientes criterios:

- Eliminando artículos duplicados.
- Eliminando artículos que pertenecían a conferencias, simposios, congresos no relacionados con el tema de mi proyecto.
- Eliminando artículos que estaban relacionados parcialmente con el tema de mi proyecto de investigación.

En la Figura 3, se puede observar el número de artículos seleccionados de cada base de dados y los artículos que fueron aceptados en esta fase.

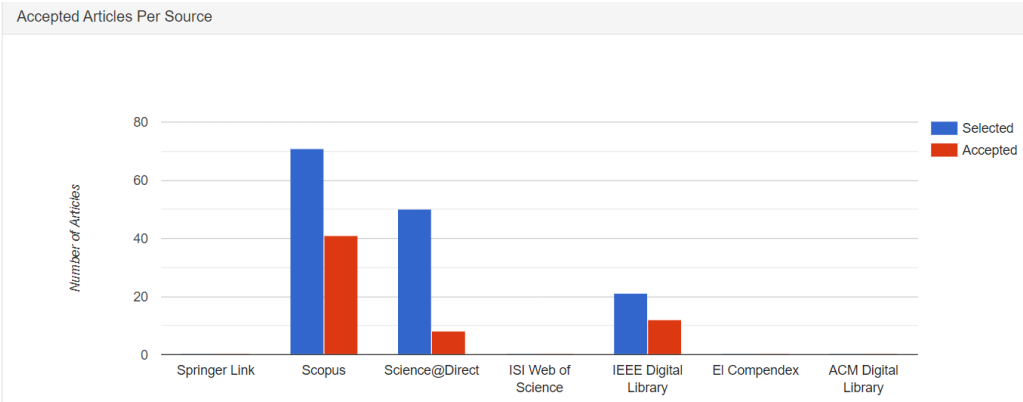

Figura 3: Número de artículos aceptados

Finalmente, se realizó el último cribado de estos 61 artículos con la herramienta de Parsifal, realizando 6 preguntas para cada artículo, donde cada artículo puede tener una puntuación del 0 al 6 dependiendo si cumple cada pregunta (1 punto), cumple cada pregunta parcialmente (0,5 puntos) o no cumple el objetivo de la pregunta (0 puntos). Las preguntas son las siguientes:

1. ¿El artículo tiene objetivos claros?
2. ¿El artículo compara diferentes algoritmos o reconocimiento de patrones en imágenes satelitales para la detección de objetos?
3. ¿El artículo describe de manera detallada la técnica utilizada para el reconocimiento de objetos en imágenes satelitales?
4. ¿Se proporciona información suficiente sobre las características del conjunto de datos, como el tipo de imágenes satelitales utilizadas?
5. ¿Se presentan ejemplos concretos de cómo se aplican estos algoritmos?

Al realizar este último cribado, se seleccionaron los artículos que tenían la puntuación máxima de 6 puntos, es decir, 33 de los 61 artículos seleccionados.

2.6.3. Resultados de *Mapping*

En el estudio de la literatura sistemática, se han identificado varios artículos relevantes que abordan la detección de objetos en imágenes satelitales utilizando dos enfoques populares: YOLO [13, 14, 15, 16] y R-CNN [17, 18, 19, 20, 21, 22, 23, 24, 25].

Las redes neuronales convolucionales que también se han utilizado en varios artículos [26, 27, 28, 29, 30] son la base fundamental tanto de YOLO como de R-CNN. Estas arquitecturas de red están diseñadas específicamente para el procesamiento y análisis de imágenes. Las CNN son capaces de aprender y extraer características relevantes de las imágenes a través de capas de convolución y *pooling*, lo que las

hace especialmente efectivas en tareas de visión por ordenador como la detección de objetos.

YOLO se destaca por su enfoque de detección en tiempo real, donde la red neuronal convolucional realiza predicciones directas de las regiones de interés y las clases de los objetos detectados en una sola pasada por la imagen completa. Esto permite una detección eficiente y precisa de múltiples objetos en tiempo real.

Por otro lado, R-CNN adopta un enfoque basado en regiones, donde se generan propuestas de regiones de interés en la imagen utilizando técnicas como *Selective Search*. Luego, cada región propuesta se procesa individualmente utilizando una red neuronal convolucional para clasificar y localizar los objetos presentes.

La principal diferencia entre YOLO y R-CNN radica en su arquitectura y enfoque de detección. YOLO es un método de detección de objetos en tiempo real que realiza la detección y clasificación de objetos en una sola pasada a través de una red neuronal convolucional. Por otro lado, R-CNN es un enfoque basado en regiones que divide el proceso de detección en etapas separadas, primero generando regiones de interés y luego clasificando esas regiones utilizando una red neuronal convolucional.

Aunque ambos enfoques tienen como objetivo la detección de objetos, difieren en su enfoque de procesamiento y la manera en que abordan el problema. YOLO se destaca por su rapidez y eficiencia, permitiendo la detección en tiempo real, mientras que R-CNN es conocido por su precisión y capacidad de detectar objetos con mayor detalle, aunque puede ser más lento en comparación.

A su vez hay artículos donde suelen utilizar varios algoritmos juntos como Yolo, R-CNN y SSD (*Single Shot MultiBox Detector*) y compararlos [31, 32, 33] o artículos que utilizan modelos que utilizan diferentes versiones de los algoritmos como Fast-R-CNN o RestNet, que se destacan por su capacidad para entrenar redes muy profundas, evitando el problema de la degradación del rendimiento que se encuentra en las redes más profundas convencionales [34, 35, 36, 37].

3. Métodos y algoritmos de detección de objetos

En esta sección discutiremos diversas estructuras arquitectónicas utilizadas en la detección de objetos en la actualidad. Es fundamental comprender su funcionamiento, ya que nos permite obtener una visión global de los diversos enfoques empleados para llevar a cabo un detector.

3.1. Sliding Window Systems

Este método es el que primero apareció y se empezó a utilizar, según Viola y Jones (2001) [38], implica deslizar una ventana rectangular de tamaño fijo sobre la imagen de entrada, examinando cada posición de la ventana para determinar la presencia de objetos de interés. Además, Dalal y Triggs (2005) [39] mencionaron que el sistema de ventana deslizante se basa en la extracción de características relevantes de la región cubierta por la ventana y la clasificación utilizando un clasificador previamente entrenado.

A continuación se explica cómo funciona el sistema de ventana deslizante con detalle:

- Tamaño y posición de la ventana: El tamaño de la ventana se elige en función del tamaño aproximado del objeto a detectar. La ventana se desliza sistemáticamente sobre la imagen, moviéndose píxel a píxel o en pasos predefinidos.
- Extracción de características: En cada posición de la ventana, se extraen características relevantes de la región de la imagen cubierta por la ventana. Estas características pueden incluir descriptores de texturas, colores, bordes u otros atributos visuales que sean útiles para la detección de objetos. Es común utilizar técnicas de procesamiento de imágenes y aprendizaje automático para extraer y representar estas características.
- Clasificación de la ventana: Una vez que se extraen las características, se utiliza un clasificador para determinar si la ventana contiene un objeto de interés o no. El clasificador se entrena previamente con ejemplos positivos (ventanas que contienen el objeto) y ejemplos negativos (ventanas que no contienen el objeto). Se aplican técnicas de aprendizaje automático, como clasificadores basados en SVM o redes neuronales, para realizar la clasificación.
- Umbral de detección: Después de la clasificación, se establece un umbral para decidir si una ventana se considera positiva (contiene el objeto) o negativa (no contiene el objeto). El umbral se determina según la confianza o probabilidad de que la ventana contenga el objeto. Si la puntuación de confianza supera el umbral establecido, se considera una detección positiva.

El sistema de ventana deslizante se repite en diferentes escalas y aspectos de la imagen para abordar la variabilidad en el tamaño y la orientación de los objetos.

3.2. Region-Based Convolutional Neural Networks

El R-CNN (*Region-based Convolutional Neural Network*) es un enfoque pionero en la detección de objetos que fue presentado por Girshick et al. en 2014 [5]. Este método introdujo una nueva forma de abordar la detección de objetos mediante el uso de redes neuronales convolucionales.

El R-CNN se compone de los siguientes pasos:

- Generación de propuestas de regiones: En primer lugar, se utilizan algoritmos de generación de propuestas, como Selective Search, para identificar regiones de la imagen que potencialmente contienen objetos. Estas regiones se consideran candidatas para la detección y se extraen para su posterior procesamiento.
- Extracción de características: Cada región propuesta se redimensiona a un tamaño fijo y se pasa a través de una red neuronal convolucional preentrenada, como AlexNet o VGG16. La red convolucional extrae características visuales de la región propuesta y las transforma en un vector de características.
- Clasificación y localización: El vector de características extraído de cada región propuesta se introduce en clasificadores individuales, como SVMs, que determinan la clase a la que pertenece la región (por ejemplo, perro, automóvil, persona, etc.). Además, se utiliza un regresor para ajustar las coordenadas de la región propuesta y mejorar la precisión de la localización del objeto dentro de la región.

El enfoque R-CNN introdujo el uso de redes neuronales convolucionales para la detección de objetos y demostró un rendimiento prometedor en términos de precisión. Sin embargo, este método tenía la desventaja de ser computacionalmente costoso y lento debido a la necesidad de procesar cada región propuesta de forma independiente.

Posteriormente, se han desarrollado mejoras al método R-CNN para abordar sus limitaciones. Algunas de estas mejoras incluyen:

- Fast R-CNN: Introducido por Girshick en 2015 [40], este enfoque mejoró la eficiencia del R-CNN al proponer una arquitectura en la que se pasaba la imagen completa a través de una red neuronal convolucional, lo que permitía compartir cómputos entre las regiones propuestas. Además, se utilizaron capas *RoI pooling* para adaptar las características extraídas a un tamaño fijo y facilitar el procesamiento.
- Faster R-CNN: Presentado por Ren et al. en 2015 [41], el Faster R-CNN introdujo la *Region Proposal Network* (RPN), que permitía generar propuestas de regiones de manera eficiente dentro de la propia red neuronal convolucional. Esto eliminó la necesidad de utilizar algoritmos externos para generar las propuestas, lo que mejoró significativamente la velocidad y eficiencia del método.

- Mask R-CNN: Esta variante, propuesta por He et al. en 2017 [42], amplió el R-CNN al agregar una capa adicional para la segmentación semántica de objetos. Además de la detección y localización, el Mask R-CNN es capaz de generar máscaras precisas para cada objeto detectado, lo que permite una segmentación fina y detallada.

3.3. Single Shot Detector

El *Single Shot Detector* (SSD) es un método popular para la detección de objetos introducido por Liu et al. en 2016 [43]. Este enfoque se caracteriza por ser rápido y preciso al detectar objetos en imágenes.

El SSD se destaca por las siguientes características:

- Diseño de red neuronal en una sola pasada: A diferencia de los métodos tradicionales que utilizan múltiples etapas para generar propuestas y refinar la detección, el SSD realiza la detección en una sola pasada. Esto se logra mediante la incorporación de múltiples capas de convolución que predicen la presencia de objetos en diferentes escalas y aspectos.

- Anclas y clasificación por niveles: El SSD utiliza anclas en diferentes niveles de la red para detectar objetos de diferentes tamaños y aspectos. Cada ancla se asigna a un objeto o fondo mediante la clasificación basada en confianza. Además, se utilizan capas de convolución adicionales para ajustar las coordenadas de las regiones propuestas y mejorar la precisión de la localización.

- Uso de características multi-escala: El SSD aprovecha las características de múltiples niveles de la red para capturar objetos de diferentes tamaños. Las capas de convolución más tempranas capturan características de nivel más alto y objetos más grandes, mientras que las capas posteriores capturan detalles finos y objetos más pequeños.

- Eficiencia y precisión: El diseño del SSD permite una detección rápida y precisa de objetos. Al realizar la detección en una sola pasada y utilizar características multi-escala, el SSD logra un equilibrio entre la eficiencia computacional y la precisión en la detección.

3.4. RetinaNet

El RetinaNet es un método de detección de objetos introducido por Lin et al. en 2017 [44]. Este enfoque se destaca por abordar el desafío de detectar objetos en diferentes escalas y tamaños con una alta precisión.

La arquitectura de RetinaNet es un modelo de detección de objetos basado en redes neuronales convolucionales que se destaca por su capacidad para detectar

objetos en una amplia gama de escalas sin sacrificar la precisión. Utiliza una combinación de características de alto nivel y características de nivel más bajo para realizar detecciones precisas.

La arquitectura de RetinaNet se compone de dos componentes principales: el extractor de características y la cabeza de detección. El extractor de características es generalmente una red convolucional profunda, como ResNet, que se encarga de extraer características de alto nivel de la imagen de entrada. Esta parte de la red se entrena previamente en grandes conjuntos de datos para aprender representaciones generales de las imágenes.

La cabeza de detección se conecta directamente al extractor de características y realiza las predicciones de detección de objetos. En RetinaNet, la detección se realiza a través de una arquitectura de pirámide de características (*feature pyramid network*, FPN) que combina múltiples niveles de características para abordar la detección de objetos a diferentes escalas. La FPN permite capturar tanto las características semánticas de alto nivel como las características espaciales detalladas para mejorar la precisión de la detección.

Para predecir las probabilidades de cada clase de objeto, RetinaNet utiliza un conjunto de anclas *anchors* en cada nivel de la pirámide de características. Las *anchors* son cuadros delimitadores predefinidos que abarcan diferentes aspectos y tamaños de objetos. Cada ancla se asocia con una probabilidad de objeto (objeto presente o no presente) y las probabilidades de cada clase de objeto.

La predicción de probabilidades se realiza a través de dos subredes en paralelo: la subred de clasificación y la subred de regresión. La subred de clasificación se encarga de predecir las probabilidades de cada clase de objeto para cada ancla, utilizando capas convolucionales y capas de clasificación específicas. La subred de regresión se encarga de predecir las coordenadas de los cuadros delimitadores para cada ancla, ajustando las ubicaciones de *anchors* para que se ajusten mejor a los objetos reales.

Durante el entrenamiento de RetinaNet, se utilizan varios parámetros importantes. Algunos de estos parámetros son la tasa de aprendizaje, que controla la magnitud de los ajustes realizados en los pesos durante el entrenamiento, el tamaño del lote *batch size*, que determina el número de ejemplos de entrenamiento utilizados en cada iteración, y el número de épocas de entrenamiento, que indica cuántas veces se pasa por el conjunto de datos completo durante el entrenamiento.

A continuación, se describen los aspectos clave de este método:

- Enfoque de detección de objetos en una sola pasada: A diferencia de los métodos anteriores que requerían múltiples etapas de generación de propuestas y refinamiento, el RetinaNet realiza la detección en una sola pasada. Esto se logra mediante la incorporación de un enfoque de *head* de múltiples escalas en la red.
- Característica de anclaje y clasificación por niveles: El RetinaNet utiliza anclas (*anchors*) en diferentes niveles de la pirámide de características para detectar objetos en diferentes escalas y tamaños. Cada (*anchor*) se asigna a un objeto

o un fondo mediante la clasificación basada en una función de pérdida denominada *focal loss*. Diseñada para abordar el desequilibrio de clases en la detección de objetos, donde la mayoría de las regiones son fondos en comparación con los objetos de interés.

- Redes de características compartidas: El RetinaNet utiliza una red de características compartidas para extraer características de la imagen en diferentes escalas. Esto permite una representación más rica y robusta de los objetos en la imagen, independientemente de su tamaño.
- Eficiencia y precisión: El diseño del RetinaNet permite lograr una detección de objetos precisa sin comprometer la eficiencia computacional. La utilización de una sola pasada y la clasificación basada en anclas ayudan a reducir la complejidad del método sin afectar la precisión.

Su enfoque de *head* de múltiples escalas y la *focal loss* han sido fundamentales para su éxito.

3.5. You Only Look Once

YOLO (*You Only Look Once*) es un enfoque popular en la detección de objetos introducido por Redmon et al. en 2016 [45]. Una de las características distintivas de YOLO es su enfoque en la detección de objetos en tiempo real, ya que realiza la detección de forma directa y eficiente en una sola pasada de la red neuronal sobre la imagen completa de la red neuronal convolucional.

En YOLO, la red neuronal se divide en dos partes principales: la parte de extracción de características y la parte de detección, como se observa en la Figura 4.

La parte de extracción de características es generalmente una red convolucional profunda, como Darknet, que se encarga de aprender representaciones de alto nivel de la imagen de entrada. Esta parte de la red se entrena previamente en grandes conjuntos de datos para capturar características generales de las imágenes.

La parte de detección es responsable de generar las predicciones de detección de objetos. Consiste en capas finales de la red neuronal que se conectan directamente a la parte de extracción de características. En YOLO, se divide la imagen de entrada en una cuadrícula de celdas y cada celda es responsable de predecir un conjunto de cuadros delimitadores y las probabilidades asociadas a las clases de objetos.

Para cada celda de la cuadrícula, se generan múltiples cuadros delimitadores que representan posibles detecciones de objetos. Cada cuadro delimitador está compuesto por un conjunto de valores que predicen las coordenadas del cuadro (esquina superior izquierda y esquina inferior derecha), la confianza de detección y las probabilidades de cada clase. La confianza de detección refleja la certeza de que un objeto está presente en el cuadro delimitador, mientras que las probabilidades de clase representan las probabilidades de que el objeto pertenezca a cada clase.

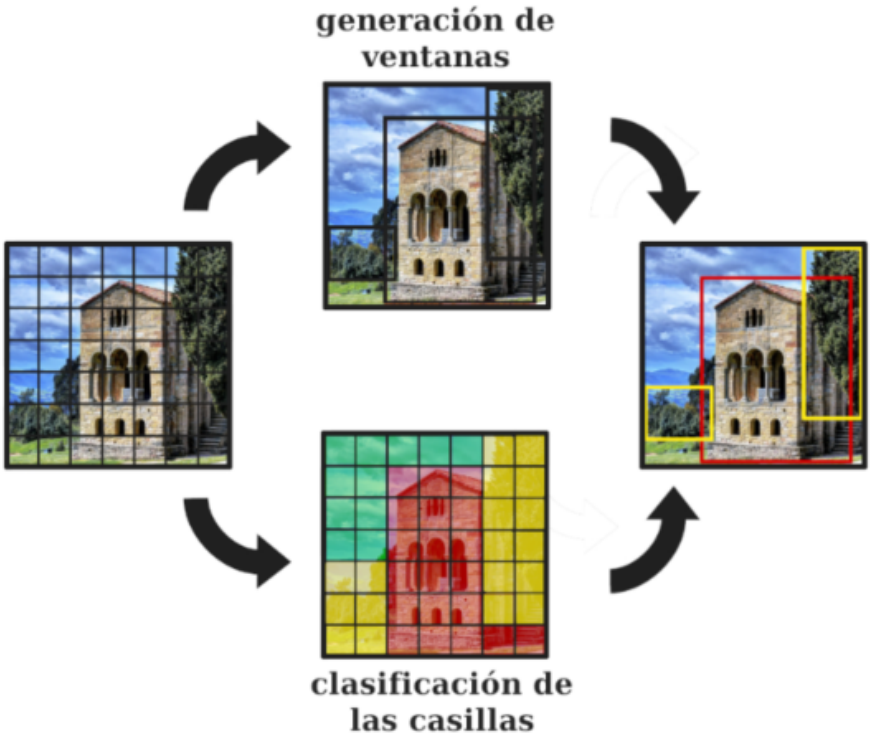

Figura 4: Detección de imágenes Yolo.[1]

Las principales características del método YOLO son:

- Detección en una sola pasada: A diferencia de los enfoques basados en ventanas deslizantes que requieren múltiples pasadas por la imagen, YOLO realiza la detección en una sola pasada. La red neuronal convolucional de YOLO divide la imagen de entrada en una cuadrícula y predice las cajas delimitadoras y las probabilidades de clase para cada celda de la cuadrícula.
- Predicción basada en anclas: YOLO utiliza anclas predefinidas que representan diferentes tamaños y aspectos de los objetos. Estas anclas se utilizan para generar predicciones de las cajas delimitadoras y las probabilidades de clase. El modelo de YOLO realiza ajustes finos en las coordenadas de las cajas y predice la confianza de cada predicción.
- Eficiencia y velocidad: Debido a que YOLO realiza la detección en una sola pasada y utiliza una cuadrícula para generar las predicciones, es un método muy eficiente y rápido. Esto lo hace adecuado para aplicaciones en tiempo real donde la velocidad de detección es crucial.
- *Trade-off* entre precisión y localización: Si bien YOLO es rápido, puede tener dificultades para detectar objetos pequeños o con detalles finos. Debido a que las cajas delimitadoras se predicen en una cuadrícula, YOLO puede tener limitaciones en la precisión de la localización en comparación con enfoques basados en regiones.

3.5.1. Evolución de Yolo

YOLO, desarrollado por Joseph Redmon, et al., fue presentado por primera vez en 2015 y fue una revolución en el campo de la detección de objetos en imágenes. YOLO adoptó un enfoque completamente diferente, proponiendo un solo paso para la detección de objetos en lugar de los dos pasos que utilizaban los modelos anteriores. YOLO dividía la imagen en una cuadrícula y cada celda de la cuadrícula predecía varios cuadros delimitadores y las probabilidades de clase para esos cuadros delimitadores. A pesar de su velocidad, YOLO tenía muchas falsas detecciones y luchaba con objetos pequeños.

3.5.2. Yolov2 (también conocido como Yolo 9000)

La versión YOLOv2 2.7.1 tiene como objetivo principal incrementar la precisión y acelerar el tiempo de detección en comparación con las versiones anteriores [46]. Se han realizado varios cambios significativos para lograr estos objetivos.

Uno de los cambios más importantes ha sido la incorporación de la Normalización por Lotes en las capas de convolución. Este proceso añade operaciones al modelo antes o después de la función de activación de cada capa oculta, normalizando cada entrada y centrando en cero el resultado, que luego se escala y cambia usando dos nuevos parámetros por capa. Esta técnica facilita al modelo aprender la escala y media óptimas para cada entrada de la capa.

Además, se ha mejorado el clasificador. Mientras que en la primera versión se entrenaba con imágenes de 224x224 y se realizaba la detección con imágenes de 448x448, en esta versión se entrena y se detecta utilizando imágenes de 448x448, lo que mejora la precisión media (mAP).

Otra mejora significativa ha sido la introducción de cajas de anclaje. Esto permite que el sistema pueda detectar más de un objeto por celda, solucionando un problema de las versiones anteriores donde solo se podía detectar un objeto por celda. Las formas y tamaños de las cajas de anclaje son seleccionadas por YOLO mediante el algoritmo de aprendizaje no supervisado K-Means, que organiza los datos en diferentes grupos o *clusters* en función de sus características.

En cuanto a las coordenadas de ubicación, YOLOv2 las predice en función de las celdas de la cuadrícula, generando hasta cinco cuadros delimitadores por celda y prediciendo cinco coordenadas para cada cuadro delimitador: $tx, ty, tw, thyto$.

Finalmente, en lo que respecta a la arquitectura de la red, YOLOv2 introduce un nuevo modelo de clasificación, *Darknet19*, que consta de 19 capas convolucionales y 5 capas de máxima agrupación. Esta arquitectura permite que YOLOv2 alcance una precisión notablemente superior a la de YOLOv1.

3.5.3. Yolov3

YOLOv3[47] es una versión actualizada y mejorada de YOLO, que mejora sus versiones anteriores en varios aspectos:

1. Velocidad y precisión: Es notablemente más rápido y más preciso que sus predecesores. Por ejemplo, a una resolución de 320 × 320, YOLOv3 se ejecuta en 22 ms a 28.2 mAP, lo que es tan preciso como SSD, pero tres veces más rápido.

2. Predicción de cuadros delimitadores: Predice cuadros delimitadores utilizando conglomerados de dimensiones como cajas de anclaje. Predice una puntuación de objetividad para cada cuadro delimitador usando regresión logística. Solo se asigna un cuadro delimitador previo para cada objeto de verdad de fondo.

3. Predicción de clase: En lugar de utilizar un *softmax*, utiliza clasificadores logísticos independientes para predecir las clases que el cuadro delimitador puede contener. Durante el entrenamiento, se utiliza la pérdida de entropía cruzada binaria para las predicciones de clase.

4. Predicciones a través de escalas: Predice cuadros en 3 escalas diferentes y extrae características de esas escalas usando un concepto similar a las redes de pirámides de características.

5. Extractor de características: Utiliza una nueva red para la extracción de características llamada *Darknet-53*, que es más poderosa que *Darknet-19*, utilizada en YOLOv2, y aún más eficiente que *ResNet-101* o *ResNet-152*.

6. Entrenamiento: Aún se entrena en imágenes completas. Utiliza entrenamiento a múltiples escalas, mucha ampliación de datos y normalización de lotes.

3.5.4. Yolov4

Los creadores de YOLOv4[48] evaluaron diversas arquitecturas de red troncal, incluyendo *CSPResNext50*, *CSPDarknet53* y *EfficientNet-B3*. Tras numerosos experimentos, decidieron implementar *CSPDarknet53* como la red troncal en la arquitectura final, basándose en su intuición y en sus resultados experimentales.

La combinación de esta red troncal con las técnicas y características antes mencionadas da lugar a YOLOv4. Esta arquitectura logra una precisión y velocidad superiores, convirtiéndola en una opción robusta y eficiente para la detección de objetos en tiempo real. En resumen, integra una serie de innovaciones y mejoras para proporcionar un rendimiento superior tanto en precisión como en velocidad.

Es una arquitectura de detección de objetos profunda que aprovecha diversas técnicas avanzadas para mejorar la precisión y la velocidad. La arquitectura incorpora un módulo de atención espacial (*Spatial Attention Module*, SAM) que dirige la atención de la red hacia áreas específicas en la imagen, y una red de agregación

de rutas (*Path Aggregation Network*, PAN) para combinar información de diferentes niveles de la red, lo que contribuye a su rendimiento mejorado.

Para la precisión de la localización de objetos, YOLOv4 utiliza una estrategia conocida como *Complete Intersection over Union*, CIOU), que mejora la forma en que la red delinea los objetos detectados. Adicionalmente, para optimizar la detección de objetos pequeños, se introduce una técnica llamada eliminación de mosaico. Esta consiste en crear mosaicos con múltiples imágenes que se introducen a la red, ayudando así a la red a identificar objetos que podrían ser más pequeños de lo usual.

En cuanto a su arquitectura, YOLOv4 es más grande que sus predecesores. Esto permite una detección de objetos en múltiples escalas y aspectos, mejorando su precisión. Sin embargo, a pesar de su complejidad, mantiene una alta velocidad de procesamiento, llegando a procesar imágenes a un ritmo de hasta 60 FPS en *hardware* de consumo.

Una de las principales tácticas para mejorar la robustez de la red es el aumento de datos. Este proceso implica aplicar una variedad de transformaciones a las imágenes de entrada, como voltear, rotar, recortar, y ajustar el brillo, la saturación y la exposición.

En el proceso de entrenamiento de YOLOv4, la red troncal se entrena previamente en la clasificación de ImageNet. Esto significa que los pesos de la red ya se han ajustado para identificar características relevantes en una imagen. Sin embargo, estos se modificarán para la nueva tarea de detección de objetos.

3.5.5. Yolov5

Es un modelo de visión por computadora que se usa comúnmente para la detección de objetos. Viene en cuatro versiones: pequeña (s), mediana (m), grande (l) y extra grande (x), cada una ofreciendo tasas de precisión cada vez mayores. Se deriva la mayoría de su mejora de rendimiento de los procedimientos de entrenamiento de *PyTorch*, manteniéndose cerca de la arquitectura de YOLOv4.

Se ha demostrado que todas las variantes de YOLOv5 se entrenan más rápido que *EfficientDet*, y el modelo más preciso, YOLOv5x, puede procesar imágenes varias veces más rápido con un grado similar de precisión.

La detección de objetos, implica crear características a partir de imágenes de entrada. Estas características luego se alimentan a través de un sistema de predicción para dibujar cajas alrededor de los objetos y predecir sus clases.

Los procedimientos de entrenamiento de YOLO incluyen la augmentación de datos y cálculos de pérdida. La augmentación de datos hace transformaciones en los datos de entrenamiento base para exponer al modelo a una mayor variación semántica. YOLO calcula una función de pérdida total a partir de las funciones de pérdidas GIoU, *obj* y de clase.

La mayor contribución de YOLOv5 es traducir el marco de investigación Darknet al marco PyTorch. Este proceso facilita la implementación de nuevos avances en

investigación.

La augmentación de datos en YOLOv5 incluye escalado, ajustes de espacio de color y augmentación de mosaico. Este último es especialmente útil para el popular *benchmark* de detección de objetos COCO, ayudando al modelo a aprender a abordar el conocido "problema de los objetos pequeños".

YOLOv5 formatea la configuración del modelo en .yaml, en lugar de los archivos .cfg utilizados en Darknet. Los modelos se entrenan extremadamente rápido, lo que ayuda a reducir los costos de experimentación al construir su modelo.

3.5.6. Yolov6

También conocido como *MT-YOLOv6*, es un modelo de detección de objetos en una sola etapa basado en la arquitectura YOLO, desarrollado por investigadores en Meituan. Supera en rendimiento a YOLOv5 cuando se realiza un *benchmark* contra el conjunto de datos MS COCO.

Los modelos YOLO toman una imagen de entrada y la pasan a través de una serie de capas convolucionales en la parte posterior. Luego, los modelos YOLO alimentan esas características de la parte posterior a través del cuello. Posteriormente, pasan las características del cuello a través de tres cabezas, donde predicen la objetividad, la clase y la regresión de la caja.

YOLOv6 itera en la parte posterior y el cuello de YOLO al rediseñarlos teniendo en cuenta el *hardware*. El modelo introduce lo que los autores llaman *EfficientRep Backbone* y *Rep-PAN Neck*.

En los modelos YOLO incluyendo YOLOv5, las cabezas de clasificación y regresión de cajas comparten las mismas características. En YOLOx y YOLOv6, la cabeza está desacoplada. Esto significa que la red tiene capas adicionales que separan estas características de la cabeza final. Este cambio ha demostrado empíricamente aumentar el rendimiento del modelo.

Además de los cambios arquitectónicos, el repositorio YOLOv6 también implementa algunas mejoras en el *pipeline* de entrenamiento, incluyendo el entrenamiento libre de anclajes (no libre de NMS), la asignación de etiquetas *SimOTA* y la pérdida de regresión de cajas *SIoU*.

YOLOv6 logra un mAP más alto en COCO a varias tasas de FPS en comparación con YOLOv5, YOLOX y PPE-YOLOE. Pero, es importante notar que el conjunto de datos COCO es una proxy, pero no un modelo perfecto de cómo estos modelos se desempeñarán en su conjunto de datos.

3.5.7. Yolov7

Los autores de YOLOv7 se propusieron mejorar la detección de objetos, creando una arquitectura de red que predeciría las cajas delimitadoras con mayor precisión que otros modelos a velocidades de inferencia similares.

Para lograr esto, implementaron varias innovaciones:

Agregación Extendida de Capas Eficientes (E-ELAN): Los autores optimizaron las capas convolucionales de la red YOLO para una inferencia más eficiente, basándose en investigaciones previas. Esta optimización considera tanto la memoria requerida por las capas como la distancia que recorren los gradientes durante la retropropagación.

Técnicas de Escalamiento del Modelo: Ajustaron el modelo para diferentes necesidades, escalando la profundidad y la anchura de la red al mismo tiempo que concatenaban capas. Esto resultó en una arquitectura óptima, independientemente del tamaño del modelo.

Planificación de Re-parametrización: Implementaron una estrategia de promediado de pesos del modelo para aumentar la robustez. También identificaron qué módulos de la red debían usar estrategias de re-parametrización y cuáles no, utilizando rutas de propagación de flujo de gradiente.

Cabeza Auxiliar de Grueso a Fino: Introdujeron una *cabeza auxiliar* en el medio de la red para mejorar las predicciones. Aunque esta cabeza auxiliar no se entrena tan eficientemente como la cabeza final, su supervisión durante el entrenamiento mejora el rendimiento del modelo.

3.5.8. Yolov8

YOLOv8 es la última versión del modelo YOLO hasta la fecha y trae varias mejoras y cambios notables en la arquitectura de red:

Detección libre de anclas: A diferencia de los modelos anteriores de YOLO que utilizaban cajas de anclas para hacer predicciones, YOLOv8 es un modelo libre de anclas. Esto significa que predice directamente el centro de un objeto en lugar del desplazamiento desde una caja de anclaje conocida, lo que reduce la complejidad y mejora la velocidad de la supresión no máxima (NMS).

Nuevas convoluciones: YOLOv8 introduce cambios en los bloques de construcción de la red, incluyendo la sustitución de la convolución 6x6 inicial por una 3x3 y la introducción del módulo C2f en lugar del C3. Esto resulta en una reducción del número de parámetros y del tamaño general de los tensores.

Cierre de la aumentación de mosaicos: YOLOv8 aplica aumentación de mosaicos durante el entrenamiento, donde se combinan cuatro imágenes, lo que obliga al modelo a aprender objetos en nuevas ubicaciones y en diferentes contextos. Sin embargo, se descubrió que esta técnica puede degradar el rendimiento si se utiliza durante todo el entrenamiento, por lo que se desactiva durante las últimas diez épocas de entrenamiento.

Mejoras en la precisión: YOLOv8 ha demostrado tener una precisión de vanguardia en la evaluación COCO, que es el estándar de la industria para evaluar los modelos de detección de objetos. Además, en la evaluación RF100 de Roboflow[49], YOLOv8 superó a YOLOv5 y YOLOv7 en términos de mAP@.50, mostrando menos

valores atípicos y un mejor mAP en general.

La investigación de YOLOv8 se ha centrado principalmente en la evaluación empírica, realizando experimentos para validar los cambios en la red y en la rutina de entrenamiento. Su código se encuentra disponible en un repositorio público, permitiendo a la comunidad usarlo y mejorarlo.

3.6. Evaluación de los métodos

Las predicciones de un modelo de visión por ordenador pueden tener uno de cuatro resultados, y es deseable maximizar los resultados verdaderos y minimizar los resultados falsos:

- **Verdadero positivo (TP, *True Positive)***: se refiere a una detección y clasificación correctas, donde el modelo logra dibujar un cuadro del tamaño correcto alrededor del objeto correcto.
- **Falso positivo (FP, *False Positive)***: se trata de una detección y clasificación incorrectas, donde el modelo dibuja un cuadro en un objeto que no debería ser detectado. Este caso ocurre con mayor frecuencia cuando los objetos son similares entre sí y puede reducirse mediante un mejor etiquetado de los objetos que confunden al modelo.
- **Verdadero negativo (TN, *True Negative)***: indica que el objeto no estaba presente en la imagen y el modelo no dibujó un cuadro. Es decir, el modelo reconoció que no había nada en esa ubicación y no generó una detección errónea.
- **Falso negativo (FN, *False Negative)***: se refiere a la situación en la que el objeto estaba presente en la imagen, pero el modelo no logró detectarlo y no dibujó un cuadro. Esto ocurre con mayor frecuencia cuando el objeto presenta ángulos, iluminación o condiciones inusuales que no se encuentran en los datos de entrenamiento del modelo.

Los cuatro grupos forman la denominada matriz de confusión que se puede observar en la siguiente Figura 5.

Una vez explicados los posibles resultados de la previsión, las métricas que se utilizan para determinar el comportamiento de los modelos entrenados son los siguientes:

- **Precisión**: La fórmula de precisión es Verdadero Positivo dividido por la suma de Verdadero Positivo y Falso Positivo.

$$P = \frac{TP}{TP + FP}$$

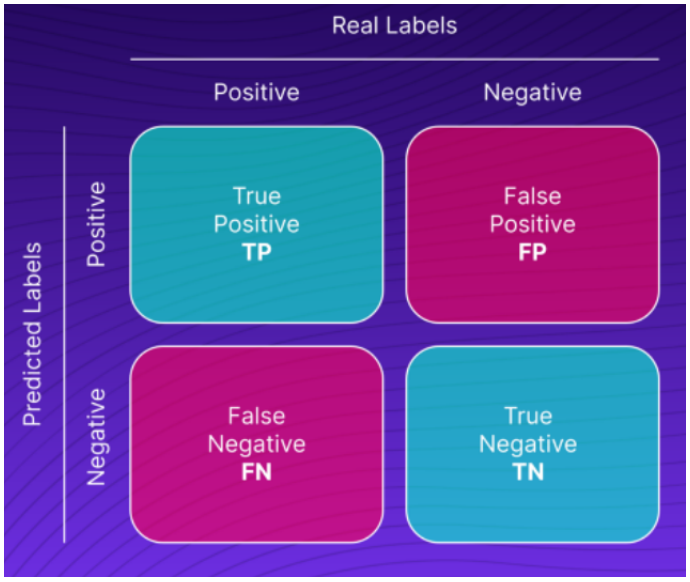

Figura 5: Matriz de confusión.[2]

- ***Recall***: La fórmula para recordar es Verdadero Positivo dividido por la suma de Verdadero Positivo y Falso Negativo.

$$P = \frac{TP}{TP + FN}$$

- ***Curva de Precisión-Recall***: Para construir la curva de precisión-*recall*, se varía el umbral de clasificación del modelo y se calculan los valores de precisión y recuperación correspondientes a cada umbral. Luego, estos valores se representan en un gráfico, donde el eje X corresponde al *recall* y el eje Y corresponde a la precisión, como se observa en la Figura 6.

- ***Boss loss***: Una métrica de pérdida, basada en una función de pérdida específica, que mide qué tan "ajustados" están los cuadros delimitadores predichos a los objetos reales (las etiquetas en las imágenes de su conjunto de datos). Un valor más bajo indica que su modelo está mejorando para la generalización y creando mejores cuadros delimitadores alrededor de los objetos que el conjunto de datos ha sido etiquetado para identificar.

- **Pérdida de clasificación**: Una métrica de pérdida, basada en una función de pérdida específica, que mide la corrección de la clasificación de todos los cuadros delimitadores previstos. Cada cuadro delimitador individual puede contener una clase de objeto (Imagen nula).

- **mAP**: La precisión media (mAP) se utiliza para medir el rendimiento de los modelos de visión artificial. Es igual al promedio de la métrica de precisión promedio en todas las clases de un modelo. Se puede usar mAP para comparar modelos diferentes en la misma tarea y diferentes versiones del mismo modelo. mAP toma valores entre 0 y 1.

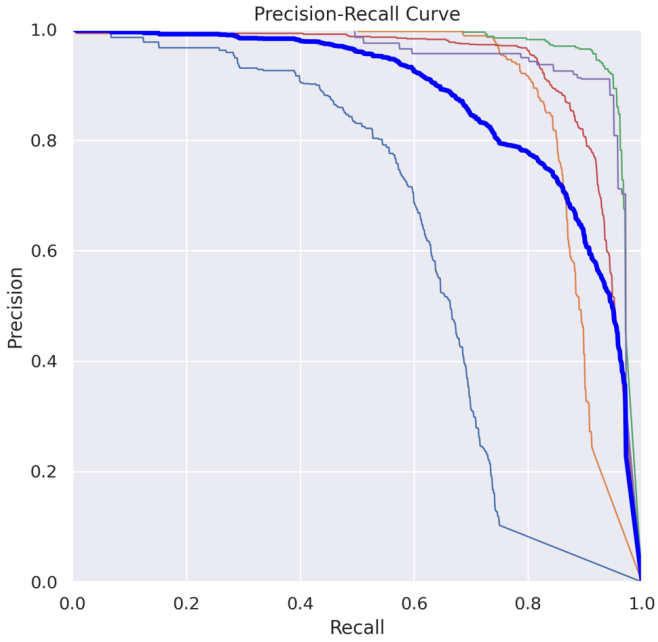

Figura 6: Curva Precisión-Recall.

Para comprender la precisión media promedio con más detalle, debemos dedicar algún tiempo a analizar las matrices de confusión, la precisión, la recuperación y la curva de precisión-recuperación.

- ***F-score***: Es una medida de la precisión y exhaustividad (*recall*) de un modelo en un problema de clasificación. Se calcula a partir de la combinación armónica de la precisión y el *recall*, y se utiliza comúnmente cuando las clases están desequilibradas en el conjunto de datos.

$$F1 = \frac{2 \cdot (precision \cdot recall)}{precision + recall}$$

- La **precisión** es la proporción de casos positivos correctamente clasificados sobre el total de casos clasificados como positivos.
- El ***recall*** es la proporción de casos positivos correctamente clasificados sobre el total de casos positivos en el conjunto de datos.

4. Caso de estudio

4.1. Obtención del conjunto de datos

El conjunto de datos utilizado en este estudio consiste en un total de 3510 imágenes satelitales. Estas imágenes se han clasificado en diferentes categorías que representan distintos tipos de objetos. Se han establecido cinco clases principales: *pool* (piscinas), *solar panel* (placas solares), *plane* (aviones), 0(barcos) y *tenniscourt* (pistas de tenis). Estas categorías fueron seleccionadas debido a que numerosos artículos científicos han demostrado que son los objetos más comúnmente detectados en mapas satelitales. La recopilación de un conjunto de datos tan extenso garantiza una amplia variedad de imágenes para entrenar y evaluar algoritmos de detección de objetos.

- Las piscinas, como objeto de interés, son especialmente relevantes debido a su presencia en áreas residenciales y recreativas. La detección precisa de piscinas en imágenes satelitales puede tener aplicaciones en la planificación urbana, la evaluación del uso de la tierra, detección de piscinas ilegales y la monitorización de la infraestructura.
- Las placas solares, por otro lado, han experimentado un crecimiento significativo en su adopción debido a su papel en la generación de energía renovable. Detectar y mapear la ubicación de estas instalaciones a través de imágenes satelitales puede ser útil para el análisis de la capacidad de generación de energía solar y la evaluación del potencial de expansión de fuentes de energía renovable.
- La detección de aviones y barcos en imágenes satelitales tiene implicaciones tanto en términos de seguridad como de la monitorización de tráfico y transporte. Identificar y rastrear estos objetos puede ser fundamental en situaciones de vigilancia y control de fronteras, así como en la supervisión del comercio marítimo y las rutas de navegación.
- Las pistas de tenis representan un objeto de interés debido a su presencia en áreas deportivas y recreativas. Detectar y mapear la ubicación de estas pistas a través de imágenes satelitales puede ser útil para la planificación de instalaciones deportivas y el análisis de la distribución de espacios recreativo.

4.1.1. Preprocesado y etiquetado de imágenes

Para la selección de imágenes en este estudio, se ha aplicado un proceso de preprocesamiento mediante la fusión de proyectos y la inclusión de imágenes del conjunto de datos de Roboflow Universe[50].

Además, se ha llevado a cabo un proceso de filtrado para eliminar imágenes borrosas o mal seleccionadas que podrían haber afectado la calidad y fiabilidad de los

datos. Esto ha garantizado que el conjunto de datos utilizado para el entrenamiento y evaluación esté compuesto por imágenes nítidas y representativas de cada clase de objeto.

Adicionalmente, se ha tenido en cuenta la importancia de mantener un nivel de *zoom* consistente en todas las imágenes de cada clase dentro del conjunto de datos. Para lograr esto, se ha realizado un ajuste cuidadoso para que las imágenes de una misma clase tengan un tamaño y resolución similares en relación con el mapa satelital. Este enfoque se ha implementado para mejorar el desempeño del modelo, ya que un *zoom* coherente facilita la detección y clasificación de objetos en el contexto espacial proporcionado por las imágenes satelitales.

Después de completar el proceso de preprocesamiento, se ha procedido al etiquetado de las imágenes utilizadas en el estudio. El etiquetado consiste en asignar etiquetas o anotaciones a las imágenes para indicar la presencia y la ubicación de los objetos de interés, como se observa en la Figura 7.

Figura 7: Etiquetado de paneles solares y piscinas.

Se han encontrado algunas clases de objetos que han tenido el mismo nombre o presentado inconsistencias en las etiquetas dentro del conjunto de datos. Esto ha generado ciertos problemas en la calidad e inestabilidad de las anotaciones.

Para resolver esta situación, se ha realizado un análisis y se ha llevado a cabo una revisión de las etiquetas. Se han solucionado las diferencias encontradas y se han asignado las etiquetas correctas a cada tipo de objeto, siguiendo la clasificación establecida para el estudio.

Posteriormente, se ha aplicado el redimensionamiento de las imágenes a un tamaño común de 600 x 600 píxeles. Esta medida es fundamental para asegurar que los algoritmos de detección funcionen de manera óptima. Al tener todas las imágenes en el mismo tamaño, se minimiza la variabilidad de la información visual y se evitan posibles sesgos causados por diferencias en las dimensiones. Esto permite que los algoritmos de detección de objetos se enfoquen en los patrones y características relevantes, sin verse afectados por diferencias de escala, lo cual mejora la precisión y robustez del modelo.

Finalmente, se ha aplicado la técnica de augmentación o aumento a las imágenes satelitales, con el fin de enriquecer el conjunto de datos y mejorar el rendimiento del modelo de detección. La augmentación es una técnica que genera nuevas muestras de entrenamiento mediante la aplicación de transformaciones a las imágenes existentes. En este estudio, se han utilizado dos técnicas de aumentación clave: el volteo y la rotación.

El volteo consiste en reflejar horizontal o verticalmente la imagen, creando así reflejos espejo. Esta técnica se ha aplicado para ampliar la variabilidad en la orientación y el ángulo de visión de los objetos presentes en las imágenes. Esto permite que el modelo aprenda a reconocer patrones desde diferentes perspectivas y mejore su capacidad de generalización.

La rotación se ha empleado para simular diferentes puntos de vista en las imágenes satelitales. Al aplicar rotaciones aleatorias en varios ángulos, el modelo puede aprender a reconocer objetos desde diferentes orientaciones y ángulos de visión, lo que ayuda a mejorar su capacidad para detectar objetos en diversas condiciones.

La combinación de estas técnicas de augmentación ha proporcionado un conjunto de datos más diverso y representativo, compuesto por 7,610 imágenes. Esto ha permitido que el modelo de detección de objetos aprenda de una variedad de escenarios.

4.2. Entrenamiento de los modelos

La configuración y el entrenamiento de los modelos se realizó siguiendo las instrucciones y el código proporcionados en los repositorios de GitHub de cada versión de YOLO. Estos repositorios proporcionan instrucciones detalladas y código fuente que permiten a los investigadores reproducir los resultados originales y adaptar los modelos para sus propios propósitos.

Para cada versión de YOLO, se siguió el siguiente proceso general:

- Clonación del repositorio de GitHub correspondiente a la versión de YOLO en cuestión.
- Preparación del entorno de Google Colaboratory para el entrenamiento, incluyendo la instalación de dependencias necesarias.
- Configuración del modelo para el entrenamiento, incluyendo la especificación de parámetros como el número de épocas, la tasa de aprendizaje y otros ajustes relacionados con el proceso de entrenamiento. Se ha realizado el entrenamiento de cada modelo con 200 épocas y se ha realizado un segundo entrenamiento con 50 épocas.
- Entrenamiento del modelo utilizando el conjunto de datos importado de Roboflow.
- Guardado del modelo entrenado para su posterior validación.

4.2.1. Yolov3

Lo primero es clonar el repositorio de Yolov3 de Ultralytics dentro de un cuaderno de Google Colab una vez se ha comprobado que se ha asignado una GPU.

Después de clonar exitosamente el repositorio YOLOv3 en el entorno de Google Colab, se descomprime el archivo .ZIP que contiene el conjunto de datos de imágenes que fue preparado previamente y contiene todas las imágenes que serán utilizadas para entrenar y probar el modelo YOLOv3.

Con todos los pasos completados, y el conjunto de datos listo y disponible en el entorno de trabajo, se procede al entrenamiento del modelo YOLOv3. Se seleccionaron los siguientes parámetros:

- *–img 640*: Este parámetro determina el tamaño de las imágenes que se utilizan para el entrenamiento. Todas las imágenes se redimensionan a 640x640 píxeles antes del entrenamiento
- *–batch 16*: Este es el tamaño del lote que se utiliza para el entrenamiento. En cada paso de entrenamiento, el modelo recibe un "lote"de 16 imágenes a la vez.
- *–epochs 200*: Una época es una pasada completa a través de todo el conjunto de datos de entrenamiento. Por lo tanto, con 200 épocas, se está haciendo pasar todo el conjunto de datos a través del modelo 200 veces.
- *–data dataset.location/data.yaml*: Este es el archivo que contiene la configuración para el conjunto de datos de entrenamiento, incluyendo la ubicación de las imágenes y las etiquetas, y el número de clases.
- *–weights /content/yolov3/yolov3.pt*: Este es el archivo que contiene los pesos preentrenados del modelo YOLOv3. Utilizando un modelo preentrenado puede acelerar significativamente el proceso de entrenamiento, ya que el modelo ya ha aprendido características útiles de un conjunto de datos grande y diverso.
- *–cache*: Esta bandera permite almacenar en caché las imágenes y las etiquetas para un acceso más rápido durante el entrenamiento. Esto puede acelerar el entrenamiento, especialmente si estás utilizando un disco de almacenamiento lento.

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
0/199	11.5G	0.0753	0.03656	0.02249	51	640

Tabla 1: Resultados de la primera época

Lo que se observa en la Tabla 1 es el resultado del entrenamiento después de una sola "época". Como mencioné anteriormente, una época es una pasada completa a través de todo el conjunto de datos de entrenamiento. Así que cuando ves "0/199", eso significa que acabas de completar la primera de las 200 épocas (la numeración comienza en 0).

Durante cada época, el modelo está tratando de aprender a minimizar tres tipos de pérdida: *box_ loss*, *obj_ loss*, y *cls_ loss*.

Tras completar el proceso de entrenamiento de 200 épocas, se obtuvieron los siguientes resultados:

```
200 epochs completed in 22.183 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 123.6MB
Optimizer stripped from runs/train/exp/weights/best.pt, 123.6MB
```

Esto indica que el proceso de entrenamiento de 200 épocas se completó en aproximadamente 22.183 horas. Al final del entrenamiento, se generan dos archivos de pesos en la carpeta *runs/train/exp/weights/: last.pt y best.pt*. El archivo *last.pt* contiene los pesos del modelo al final de la última época, mientras que el archivo best.pt contiene los pesos del modelo en la época que alcanzó la mejor precisión en el conjunto de validación durante el entrenamiento. La *optimización* de estos archivos se refiere al proceso de eliminar la información del optimizador, lo que reduce el tamaño del archivo y permite utilizar los modelos para inferencia sin necesidad de cargar el estado del optimizador.

En cuanto a los resultados se crea un fichero *results.csv* donde se recoge el conjunto de datos. A partir de este fichero se pueden sacar conclusiones sobre el entrenamiento del modelo.

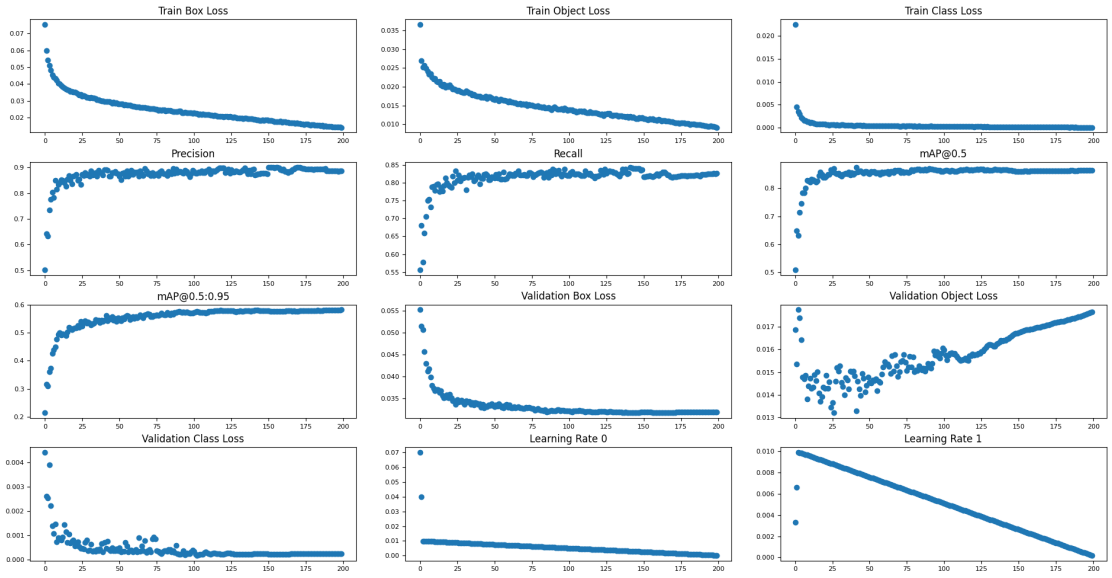

Figura 8: Curva entrenamiento YoloV3.

En la Figura 8 se presentan un conjunto de métricas que se han obtenido mediante la generación de gráficos utilizando un programa en el lenguaje de programación Python.

- **epoch:** 99.5
  Este valor representa el número medio de épocas durante las cuales se realizó el entrenamiento. Un valor alto sugiere que el modelo se ha entrenado a lo largo de un número significativo de iteraciones.

- **train/box_loss:** 0.024511
  Esta es la pérdida promedio en términos de ubicación y tamaño de las cajas delimitadoras durante el entrenamiento. Un valor bajo como este indica que el modelo ha aprendido a predecir las cajas delimitadoras con bastante precisión.

- **train/obj_loss:** 0.014552
  Esta es la pérdida promedio en términos de detección de objetos durante el entrenamiento. Un valor bajo también en este caso indica que el modelo es bastante preciso al detectar si un objeto está presente o no.

- **train/cls_loss:** 0.000514
  Esta es la pérdida promedio en términos de clasificación de objetos durante el entrenamiento. Un valor muy cercano a cero sugiere que el modelo es muy preciso en la clasificación de los objetos detectados en sus respectivas categorías.

- **metrics/precision:** 0.874514
  Una precisión promedio alta sugiere que, de todas las detecciones de objetos que el modelo realizó, una gran proporción eran detecciones correctas.

- **metrics/recall:** 0.814735
  Un alto *recall* promedio indica que el modelo fue capaz de detectar una gran proporción de los objetos presentes en las imágenes.

- **metrics/mAP_0.5:** 0.853922
  El valor promedio alto del mAP (Mean Average Precision) con un umbral de IoU (Intersection over Union) de 0.5 sugiere que el modelo ha tenido un buen rendimiento en términos de precisión y *recall*.

- **metrics/mAP_0.5:0.95:** 0.554961
  Este es el mAP calculado a diferentes niveles de umbral de IoU, desde 0.5 hasta 0.95. El valor es más bajo que el mAP con umbral de IoU de 0.5, lo que es esperable, ya que un umbral de IoU más alto es más estricto. Sin embargo, un valor mayor a 0.5 sugiere que el modelo todavía tiene un rendimiento razonablemente bueno, incluso con umbrales más altos.

- **val/box_loss, val/obj_loss, val/cls_loss**
  Estos valores representan las pérdidas promedio en la validación, y son comparables a las pérdidas durante el entrenamiento. Que estos valores sean bajos sugiere que el modelo no está sobreajustado y tiene un buen rendimiento también en los datos de validación.

- **x/lr0, x/lr1, x/lr2**
  Estos valores representan las tasas de aprendizaje promedio utilizadas en el entrenamiento. Una tasa de aprendizaje adecuada es crucial para un buen entrenamiento. No hay un valor çorrecto.º ïncorrecto.en este caso, ya que la tasa de aprendizaje óptima puede variar dependiendo del problema y de la configuración de entrenamiento.

En la Figura 9 se puede observar con más detalle la métrica de precisión.

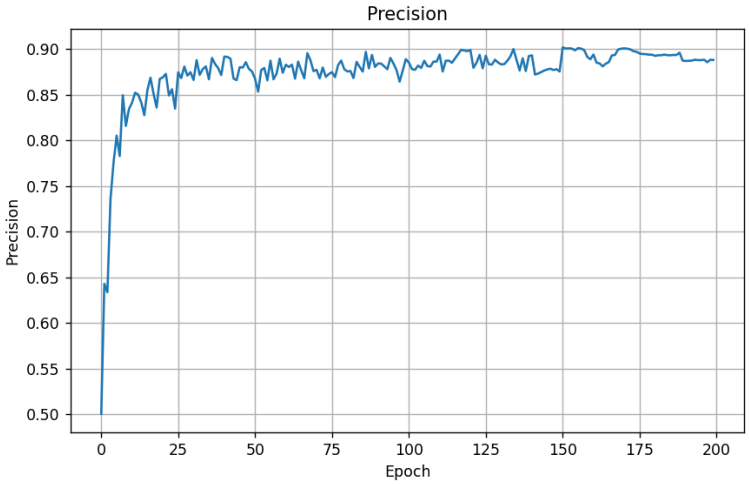

Figura 9: Curva de precisión YoloV3.

En la Figura 10 se puede observar con más detalle la métrica de *recall*.

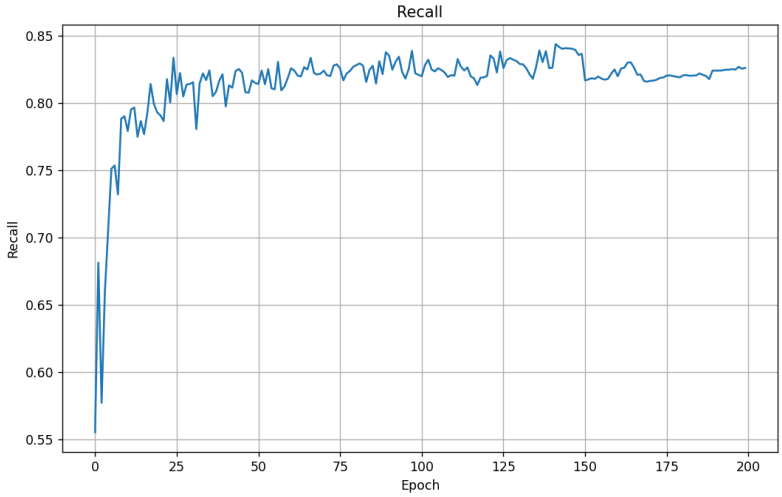

Figura 10: Curva de recall YoloV3.

En la Figura 11 se puede observar la implementación del modelo propuesto. Es evidente que el modelo identifica de manera efectiva las clases dentro de las imágenes satelitales proporcionadas. Las clases están asignadas con números que van desde 0 hasta 4, donde cada uno de ellos corresponde a una clase específica: 0 para los barcos, 1 para los aviones, 2 para las piscinas, 3 para las placas solares, y finalmente 4 para las pistas de tenis.

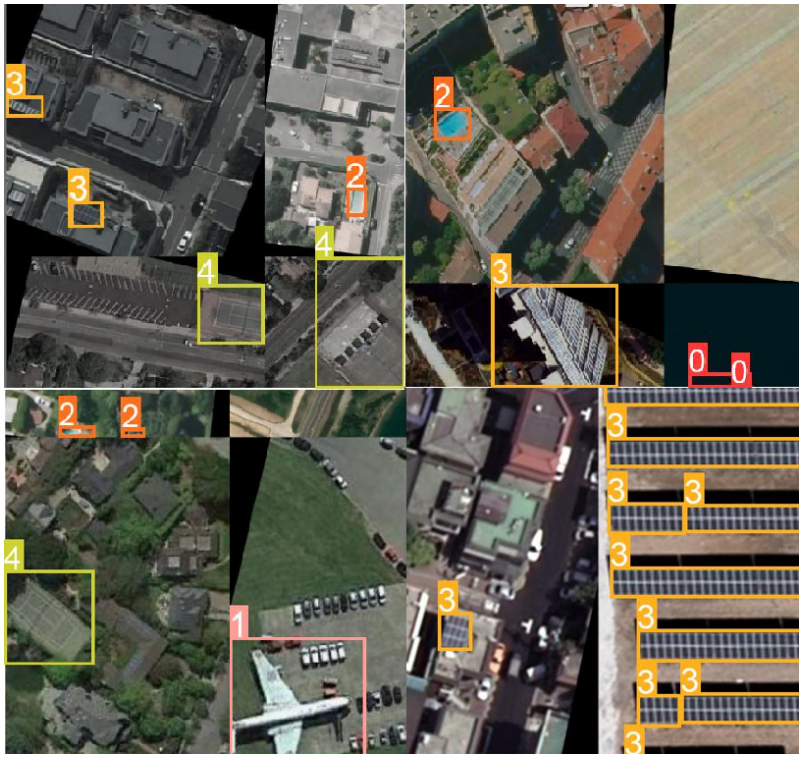

Figura 11: Validación YoloV3.

Al finalizar el entrenamiento del modelo se realizó un segundo entrenamiento del modelo Yolov3 con los mismos parámetros exceptuando el número de épocas, el cual se ha modificado de 200 a 50 épocas.

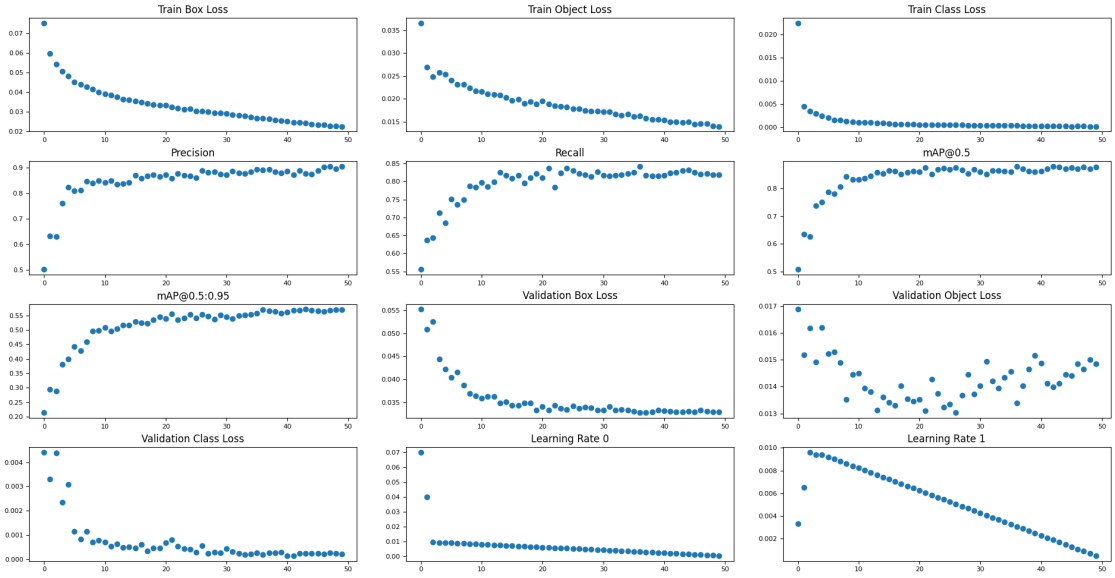

Figura 12: Resultados YoloV3 50 épocas.

En la Figura 12 se pueden observar los resultados del entrenamiento. **train/box_loss (0.033568)**: Una pérdida baja indica que el modelo ha logrado ajustar correctamente las predicciones de las ubicaciones de los objetos en las imágenes de entrenamiento.

train/obj_loss (0.018916): Una pérdida baja indica que el modelo ha logrado identificar y clasificar correctamente los objetos en las imágenes de entrenamiento.

train/cls_loss (0.001204): Una pérdida baja significa que el modelo ha logrado clasificar correctamente los objetos en las imágenes de entrenamiento en las diferentes clases o categorías.

metrics/precision: Un valor de 0.85 sugiere que el 85 % de las predicciones positivas realizadas por el modelo son correctas.

metrics/recall: Un valor de 0.795 indica que el modelo ha identificado correctamente el 79.5 % de las instancias positivas.

metrics/mAP_0.5: Un valor de 0.837811 sugiere que el modelo ha obtenido una precisión promedio del 83.8 % en la detección de objetos con un umbral de confianza del 50 %.

metrics/mAP_0.5:0.95: Un valor de 0.515674 indica que el modelo ha obtenido una precisión promedio del 51.6 % en este rango de umbrales.

4.2.2. Yolov5

El primer paso en el proceso de configuración para el entrenamiento del modelo es asegurarse de que Google Colab ha asignado una GPU para el uso durante el cuaderno.

Para comenzar con el entrenamiento del modelo, el primer paso consistió en clonar el repositorio oficial de YOLOv5 en el entorno de Google Colab. Tras clonar el repositorio, se cambió el directorio de trabajo al directorio recién clonado del repositorio YOLOv5.

```
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
```

Una vez ubicados en el directorio correcto, procedimos a instalar las dependencias necesarias para el entrenamiento del modelo, todas especificadas en el archivo *requirements.txt*. Este archivo incluye varias bibliotecas y módulos necesarios para la correcta ejecución de YOLOv5. Para instalar estas dependencias.

Como se observa en la Figura 13 el árbol del directorio */content/yolov5* representa la estructura de archivos y directorios del repositorio YOLOv5, los archivos y directorios más importantes son los siguientes:

1. *README.md*: Este es el archivo de lectura que proporciona información sobre el proyecto, incluyendo cómo instalarlo, cómo usarlo y referencias a información adicional relevante.
2. *requirements.txt*: Este archivo contiene todas las dependencias necesarias para ejecutar el código en este repositorio. Puedes instalar todas las dependencias con el comando *pip install -r requirements.txt*.
3. *train.py*: Este es el *script* principal para entrenar un modelo YOLOv5. Incluye argumentos para especificar parámetros como: el tamaño de la imagen, el tamaño del lote, el número de épocas, entre otros.
4. *detect.py*: Este *script* se utiliza para realizar la detección de objetos en imágenes o vídeos utilizando un modelo YOLOv5 entrenado.
5. *val.py*: Este *script* se utiliza para validar el rendimiento de un modelo YOLOv5 en un conjunto de datos de validación.
6. *data/*: Este directorio contiene varios archivos *YAML* que especifican los caminos a los conjuntos de datos para diferentes tareas de entrenamiento. También contiene *scripts* para descargar conjuntos de datos y archivos que definen los hiperparámetros para diferentes escenarios de entrenamiento.
7. *models/*: Este directorio contiene los archivos de definición del modelo para las diferentes versiones de YOLOv5 (por ejemplo, *yolov5s.yaml, yolov5m.yaml, yolov5l.yaml, y yolov5x.yaml*). También incluye definiciones para varias otras arquitecturas de modelos y variaciones.

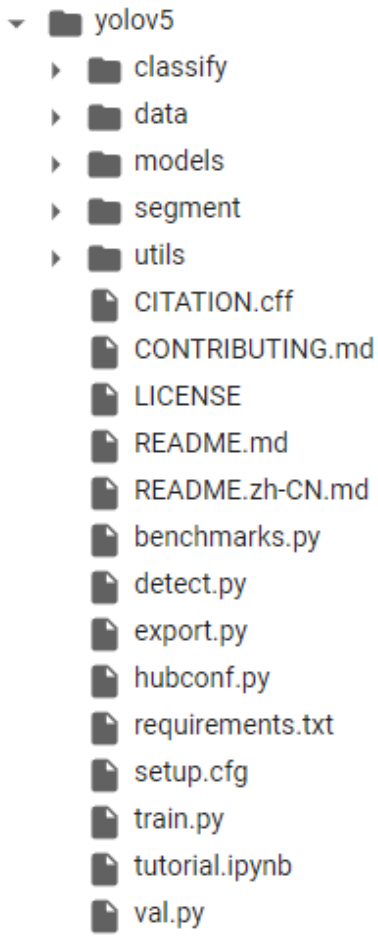

Figura 13: Etiquetado de paneles solares y piscinas.

8. *utils/*: Este directorio contiene varios scripts de utilidad para tareas como la descarga de datos, el registro de experimentos, la medición de métricas y otras operaciones relacionadas con el procesamiento de imágenes y el entrenamiento de modelos.

9. *classify/, segment/*: Estos directorios contienen scripts para tareas adicionales de clasificación y segmentación, respectivamente, que se pueden realizar con YOLOv5.

10. *export.py*: Este *script* se utiliza para exportar modelos YOLOv5 a otros formatos, como ONNX, TorchScript y TensorFlow.

Adicionalmente, instalamos el módulo `roboflow` para facilitar el manejo del conjunto de datos.

```
%pip install -qr requirements.txt
%pip install -q roboflow
```

Finalmente, importamos los módulos adicionales necesarios para el entrenamiento y verificamos que todo se había instalado correctamente, y que una GPU estaba disponible para el entrenamiento.

```
import torch
import os
from IPython.display import Image, clear_output  # to display images
```

Una vez creado el espacio de trabajo hay que exportar el conjunto de datos previamente procesado.

Para exportar el conjunto de datos de Roboflow en un formato que pueda ser utilizado con YOLOv5, se sigue el proceso de exportación de Roboflow y se selecciona *YOLOv5 PyTorch* como formato de exportación.

Esto genera un enlace de descarga para el conjunto de datos formateado para YOLOv5. Este enlace puede ser utilizado para descargar directamente el conjunto de datos en el entorno de Google Colab y extraerlo para su uso en el entrenamiento.

En este punto, el conjunto de datos está listo para ser utilizado con YOLOv5 y podemos proceder con el entrenamiento del modelo.

El siguiente paso es el entrenamiento del modelo seleccionado.

Una vez preparado el conjunto de datos, se ha procedido al entrenamiento del modelo. Para ello, se ha utilizado el script *train.py* proporcionado en el repositorio de YOLOv5. Este *script* admite diversos argumentos que definen las características del proceso de entrenamiento. En nuestro caso, se seleccionaron los siguientes argumentos:

```
!python train.py --img 640 --batch 16 --epochs 200 \
--data {dataset.location}/data.yaml --weights yolov5s.pt --cache
```

A continuación se detalla el significado de cada uno de los argumentos:

- *–img 640*: Define el tamaño de las imágenes que se introducirán en la red para el entrenamiento. En nuestro caso, se eligió un tamaño de 640x640, ya que las imágenes del conjunto de datos se redimensionaron a este tamaño previamente.
- *–batch 16*: Define el tamaño del lote para el entrenamiento. Se seleccionó un tamaño de lote de 16 para equilibrar la velocidad de entrenamiento y el uso de la memoria de la GPU.
- *–epochs 200*: Define el número de épocas para el entrenamiento. Se seleccionaron 200 épocas como un compromiso razonable entre la eficiencia del entrenamiento y el rendimiento del modelo.
- *–data dataset.location/data.yaml*: Especifica la ubicación del archivo YAML que contiene la información sobre el conjunto de datos de entrenamiento.

- *–weights yolov5s.pt*: Define los pesos iniciales del modelo, utilizando los pesos del modelo YOLOv5s preentrenado.
- *–cache*: Activa el almacenamiento en caché de las imágenes para acelerar el entrenamiento.

Después de un entrenamiento que ha durado aproximadamente 4.194 horas, se completaron las 200 épocas establecidas. Los pesos del optimizador almacenaron de los archivos *last.pt* y *best.pt*, dejando estos archivos con un tamaño de 14.5 MB.

Se validó el modelo utilizando el archivo de pesos *best.pt*, que representa los pesos del modelo que obtuvo los mejores resultados durante el entrenamiento. Los resultados de la validación mostraron un rendimiento sólido en todas las clases, con un mAP50 general de 0.875 y un mAP50-95 de 0.568.

En la Tabla 2, se presentan los resultados de validación por clase.

Clase	Imágenes	Instancias	Precisión	Recall	mAP50
0	700	579	0.808	0.591	0.656
plane	700	499	0.827	0.834	0.873
pool	700	335	0.928	0.955	0.969
solarpanel	700	780	0.960	0.822	0.935
tenniscourt	700	141	0.833	0.943	0.941

Tabla 2: Resultados de validación por clase para el entrenamiento de YOLOv5.

En la Figura 14 se observa la matriz de confusión de la clasificación de las 5 clases usando Yolov5. En el eje vertical se muestran las clases predichas, mientras que en el eje horizontal están las que son verdaderas. Las clases que mejor han sido clasificadas han sido las piscinas (*pool*) y las pistas de tenis (*tenniscourt*) aunque en general todas se han clasificado correctamente, exceptuando la clase más baja, la de los barcos(*0*).

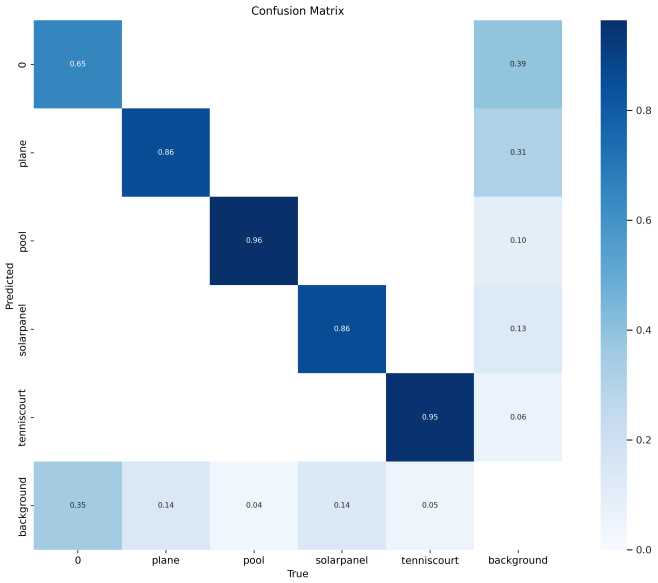

Figura 14: Matriz de confusión YoloV5.

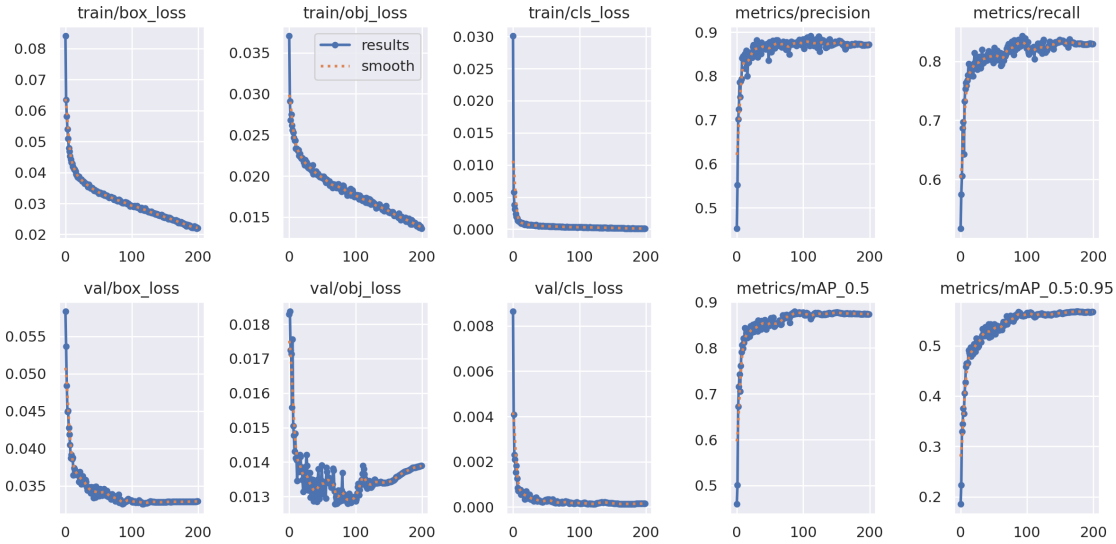

Figura 15: Resultados YoloV5.

En la Figura 15 se presentan un conjunto de métricas que se han obtenido mediante la generación de gráficos utilizando un programa en el lenguaje de programación en Python.

- **train/box_loss, train/obj_loss, train/cls_loss**: Estos valores son relativamente bajos, lo que indica que el modelo está aprendiendo bien de los datos de entrenamiento.
- **metrics/precision**: Este valor indica que el modelo tiene una precisión promedio de alrededor del 86 %, lo cual es bastante bueno. La precisión es la proporción de identificaciones positivas que fueron realmente correctas.
- **metrics/recall**: Este valor sugiere que el modelo tiene una sensibilidad o *recall* del 81 %. El *recall* es la proporción de verdaderos positivos que fueron identificados correctamente.
- **metrics/mAP_0.5, metricsmAP_0.5:0.95**: ç Estos valores también son buenos, lo que indica que el modelo está realizando una detección precisa de los objetos.
- **val/box_loss**, val/obj_loss, val/cls_loss: çEstos valores también son bajos, lo que sugiere que el modelo está generalizando bien y no está sobre ajustando los datos de entrenamiento.

En la Figura 16 se observa la curva Precisión-Confidence, es una representación gráfica que muestra cómo varía la precisión de un modelo de clasificación a medida que se ajusta el umbral de confianza para asignar las clases. Los resultados indican que al ajustar el umbral de confianza, se obtiene una precisión del 95.9 % (0.959) para todas las clases cuando el valor de confianza es 1.00. Esto significa que cuando se establece un umbral de confianza alto, el modelo es muy preciso al predecir las clases, lo que implica que tiene una alta tasa de predicciones positivas correctas.

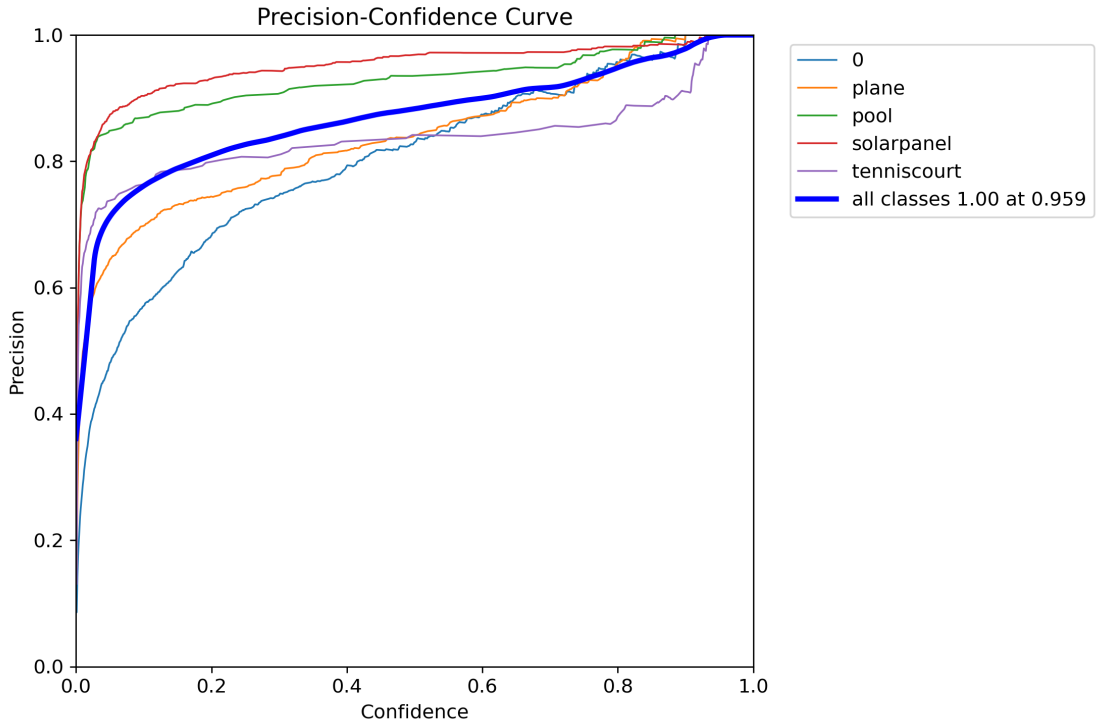

Figura 16: Curva Precision-Confidence YoloV5.

En la Figura 17 se observa la curva que muestra cómo varía la tasa de *recall* (también conocida como sensibilidad) del modelo a medida que se ajusta el umbral de confianza. Los resultados indican que obtienes un *recall* del 91 % (0.91) para todas las clases, cuando el valor de confianza es 0.000. Esto sugiere que, al disminuir el umbral de confianza, el modelo logra identificar correctamente una alta proporción de instancias positivas.

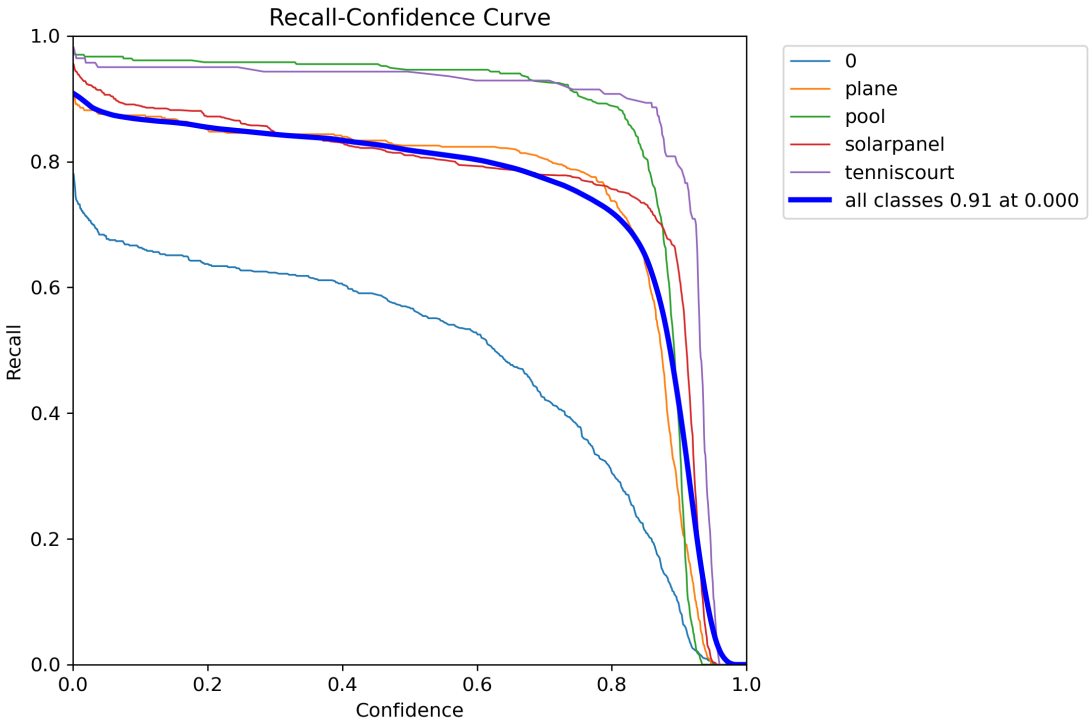

Figura 17: Curva de Recall-Confidence YoloV5.

En la Figura 18 se puede observar la implementación del modelo propuesto. Es evidente que el modelo identifica de manera efectiva las clases dentro de las imágenes satelitales proporcionadas. Las clases están asignadas con números que van desde 0 hasta 4, donde cada uno de ellos corresponde a una clase específica: 0 para los barcos, 1 para los aviones, 2 para las piscinas, 3 para las placas solares, y finalmente 4 para las pistas de tenis.

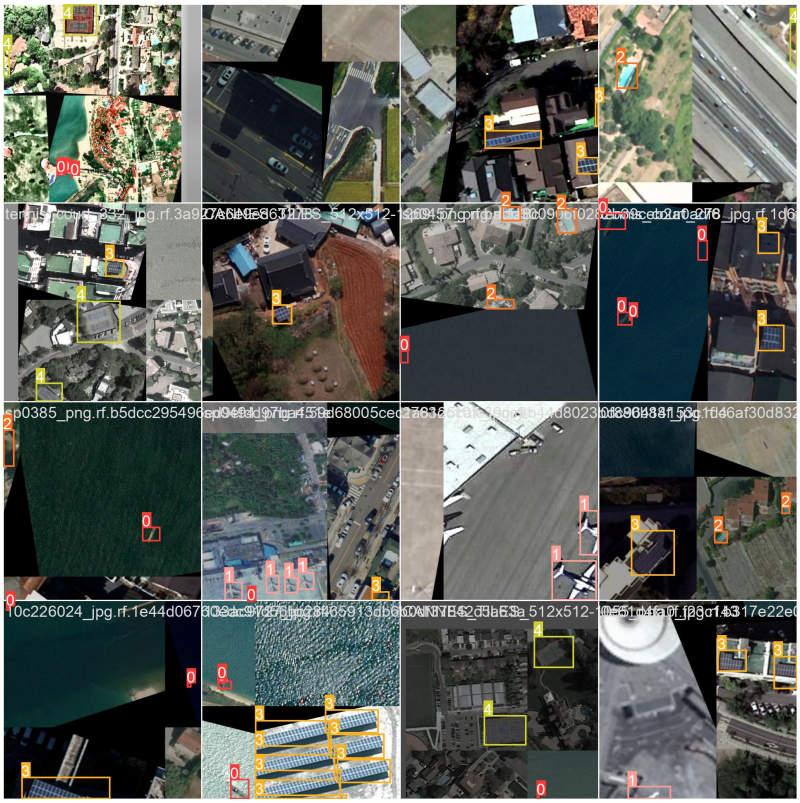

Figura 18: Validación YoloV5.

Al finalizar el entrenamiento del modelo se realizó un segundo entrenamiento del modelo Yolov5 con los mismos parámetros, exceptuando el número de épocas, el cual se ha modificado de 200 a 50 épocas.

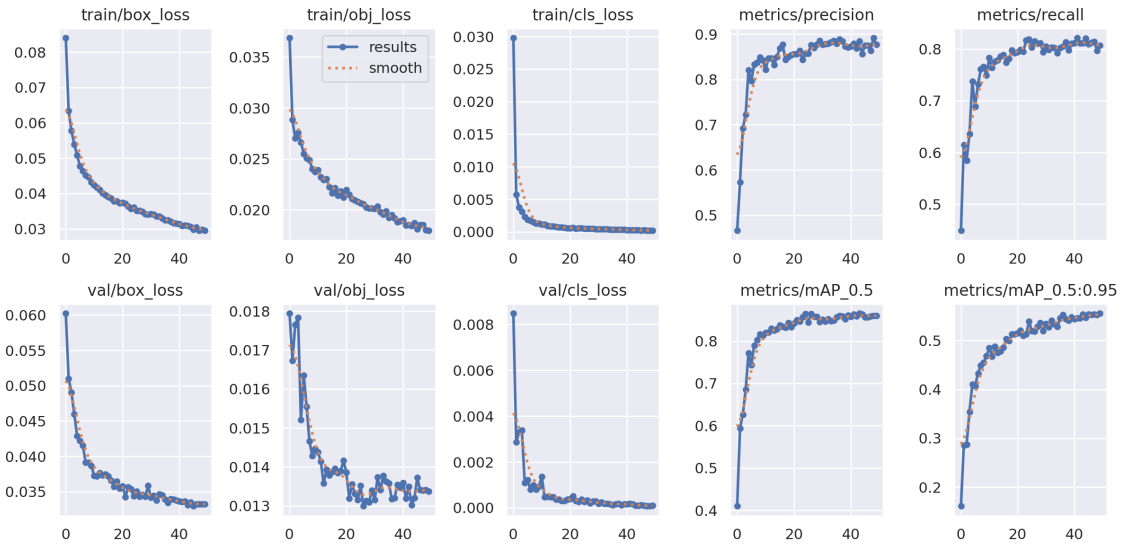

Figura 19: Resultados YoloV5 con 50 épocas.

En la Figura 19 se pueden observar los resultados del entrenamiento. La métrica **train/box_loss** representa la pérdida de caja, reflejando qué tan bien el modelo ha ajustado las predicciones de las ubicaciones de los objetos en las imágenes de entrenamiento. La métrica **train/obj_loss** indica la pérdida de objeto, que refleja qué tan bien el modelo ha identificado y clasificado correctamente los objetos en las imágenes de entrenamiento. Además, la métrica **train/cls_loss** indica la pérdida de clasificación, que refleja qué tan bien el modelo ha clasificado correctamente los objetos en las diferentes clases o categorías en las imágenes de entrenamiento. Una pérdida baja en estas métricas sugiere que el modelo ha logrado un buen ajuste y clasificación de los objetos durante el entrenamiento.

metrics/precision: La precisión es una medida de qué tan preciso es el modelo al predecir las clases. Un valor de 0.842288 sugiere que el 84.2 % de las predicciones positivas realizadas por el modelo son correctas.

metrics/recall: Un valor de 0.775106 indica que el modelo ha identificado correctamente el 77.5 % de las instancias positivas.

metrics/mAP_0.5: Un valor de 0.820441 sugiere que el modelo ha obtenido una precisión promedio del 82.0 % en la detección de objetos con un umbral de confianza del 50 %.

metrics/mAP_0.5:0.95: Un valor de 0.494896 indica que el modelo ha obtenido una precisión promedio del 49.5 % en este rango de umbrales.

4.2.3. Yolov7

Para llevar a cabo la detección de objetos con YOLOv7, primero clonamos el repositorio YOLOv7 e instalamos las dependencias necesarias utilizando el comando *!pip install -r requirements.txt*. Esta acción nos garantiza que contamos con todos los paquetes y bibliotecas necesarios para llevar a cabo el entrenamiento con YOLOv7.

Una vez que todas las dependencias están instaladas, procedemos a cargar el conjunto de datos desde Roboflow. Este conjunto de datos servirá como base para el entrenamiento de nuestro modelo YOLOv7.

Además, descargamos los pesos preentrenados de YOLOv7. Los pesos mismos son un punto de partida útil para el entrenamiento, ya que el modelo ya ha aprendido algunas características útiles de los datos de entrenamiento previos, lo que puede ayudar a acelerar el entrenamiento y a mejorar la precisión del modelo final.

Una vez que todo está listo, procedemos a entrenar el modelo utilizando el *script train.py* con los siguientes parámetros:

- *–batch 16*: Esto define el tamaño del lote para el entrenamiento.
- *–epochs 200*: Esto define el número de pasadas completas a través del conjunto de datos de entrenamiento.
- *–data {dataset.location}/data.yaml*: Esto especifica la ubicación de los datos de entrenamiento.
- *–weights 'yolov7_training.pt'*: Esto especifica los pesos preentrenados que se utilizarán para el entrenamiento.
- *–device 0*: Este parámetro especifica el dispositivo en el que se realizará el entrenamiento.

Tras 200 épocas de entrenamiento, en la Tabla 3 se pueden observar los siguientes resultados:

Class	Images	Labels	Precision	Recall	mAP@.5	mAP@.5:.95
all	700	2334	0.883	0.841	0.885	0.594
0	700	579	0.836	0.642	0.725	0.42
plane	700	499	0.86	0.842	0.874	0.544
pool	700	335	0.921	0.955	0.973	0.553
solarpanel	700	780	0.953	0.818	0.908	0.705
tenniscourt	700	141	0.843	0.95	0.943	0.75

Tabla 3: Resultados de entrenamiento tras 200 épocas con YOLOv7

Finalmente, se informa de que se ha completado el entrenamiento de 200 épocas y que el optimizador ha sido retirado de los últimos pesos, reduciendo así el tamaño del archivo de los pesos a 74 8MB.

En la Figura 20 se observa la matriz de confusión de la clasificación de las 5 clases usando Yolov7. En el eje vertical se muestran las clases predichas, mientras que en el eje horizontal están las que son verdaderas. Las clases que mejor han sido clasificadas han sido las piscinas (*pool*) y las pistas de tenis (*tenniscourt*) aunque en general todas se han clasificado correctamente, exceptuando la clase más baja, la de los barcos(*0*). El resultado es mejor que en el modelo de Yolov5 para todas las clases.

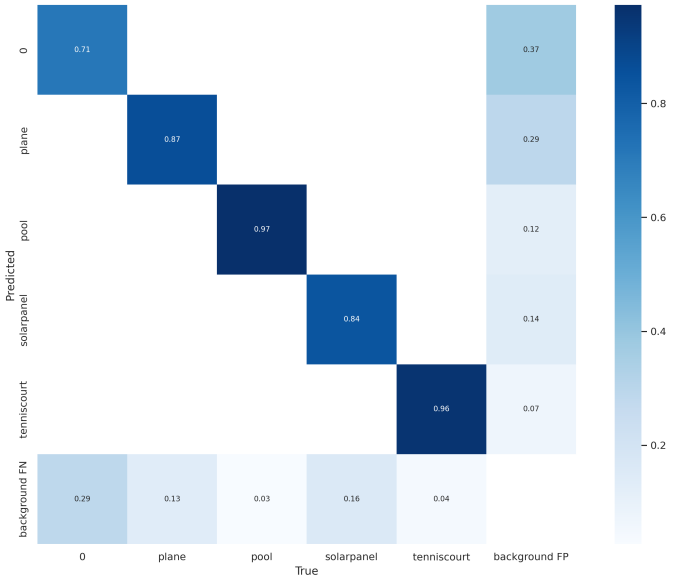

Figura 20: Matriz de confusión YoloV7.

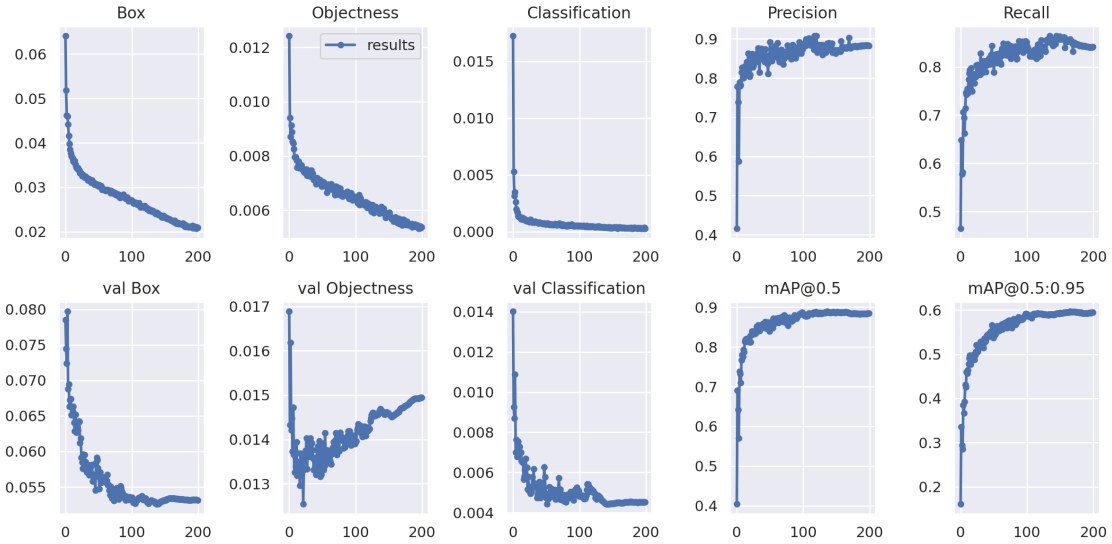

Figura 21: Resultados YoloV7.

En la Figura 21 se muestran los resultados obtenidos del rendimiento del modelo de detección de objetos Yolov7 después de 200 épocas de entrenamiento. Las métricas de rendimiento clave se describen a continuación:

- **Epochs (200):** El modelo ha sido entrenado durante 200 épocas.
- **Train/box_loss (0.027656):** Cuanto más bajo sea este valor, mejor será el rendimiento del modelo en la predicción de los límites de los objetos.
- **Train/obj_loss (0.006507):** Este es el error del modelo en términos de la identificación de si un objeto está presente o no. De nuevo, cuanto más bajo sea este número, mejor.
- **Train/cls_loss (0.000729):** Evalúa la precisión de clasificación del modelo, los objetos que ha detectado correctamente. Cuanto más bajo sea este número, mejor.
- **Total_loss (0.034892):** Es la suma de las tres pérdidas anteriores. Representa la función de coste total que el modelo intenta minimizar durante el entrenamiento.
- **Metrics/precision (0.862050):** En este caso, cuando el modelo dice que ha encontrado un objeto, tiene razón el 86,2 % de las veces.
- **Metrics/recall (0.822230):** Significa que el modelo es capaz de identificar correctamente el 82,2 % de los objetos.
- **Metrics/mAP_0.5 (0.861656):** Este valor alto indica que el modelo tiene un rendimiento sólido en la detección y clasificación de objetos.

- **Metrics/mAP_0.5:0.95 (0.559435):** Aunque este valor es más bajo que el anterior, sigue siendo un buen indicador de que el modelo tiene un rendimiento sólido en varias condiciones.
- **Val/box_loss (0.055734), Val/obj_loss (0.014189), Val/cls_loss (0.005099):** Estas son las pérdidas equivalentes a las de entrenamiento, pero calculadas sobre el conjunto de validación en lugar del conjunto de entrenamiento. Son indicadores de cómo se espera que el modelo se comporte con nuevos datos. La comparación de estos valores con los de las pérdidas de entrenamiento puede dar una indicación sobre si el modelo está sobre ajustado o no.

En la Figura 22 se observa la curva Precisión-Confidence, es una representación gráfica que muestra cómo varía la precisión de un modelo de clasificación a medida que se ajusta el umbral de confianza para asignar las clases. Los resultados indican que al ajustar el umbral de confianza, se obtiene una precisión del 96.1 % (0.961) para todas las clases cuando el valor de confianza es 1.00. Esto significa que cuando se establece un umbral de confianza alto, el modelo es muy preciso al predecir las clases, lo que implica que tiene una alta tasa de predicciones positivas correctas.

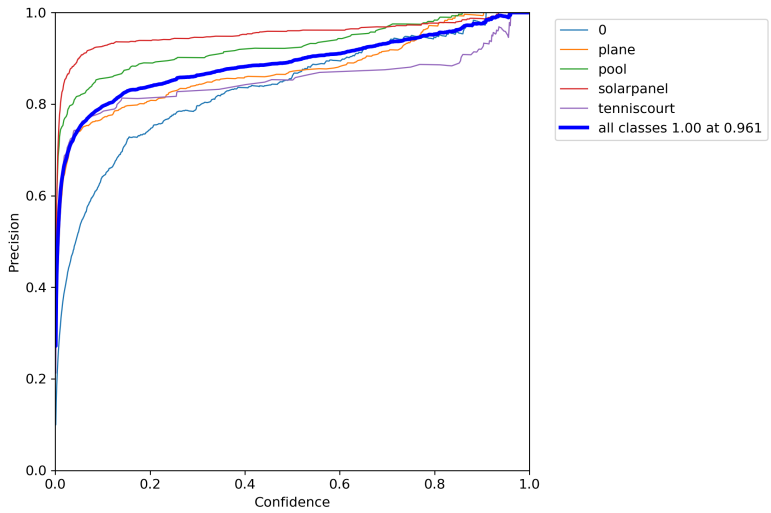

Figura 22: Curva de precisión YoloV7.

En la Figura 23 se observa la curva que muestra cómo varía la tasa de *recall* del modelo a medida que se ajusta el umbral de confianza. Los resultados indican que obtienes un *recall* del 93 % (0.91) para todas las clases cuando el valor de confianza es 0.000. Esto sugiere que, al disminuir el umbral de confianza, el modelo logra identificar correctamente una alta proporción de instancias positivas.

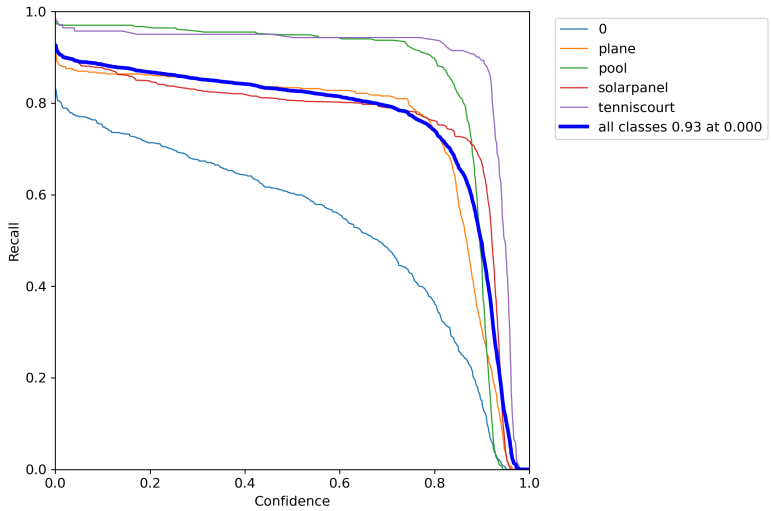

Figura 23: Curva de recall YoloV7.

En la Figura 24 se puede observar la implementación del modelo propuesto. Es evidente que el modelo identifica de manera efectiva las clases dentro de las imágenes satelitales proporcionadas. Las clases están asignadas con números que van desde 0 hasta 4, donde cada uno de ellos corresponde a una clase específica: 0 para los barcos, 1 para los aviones, 2 para las piscinas, 3 para las placas solares, y finalmente 4 para las pistas de tenis.

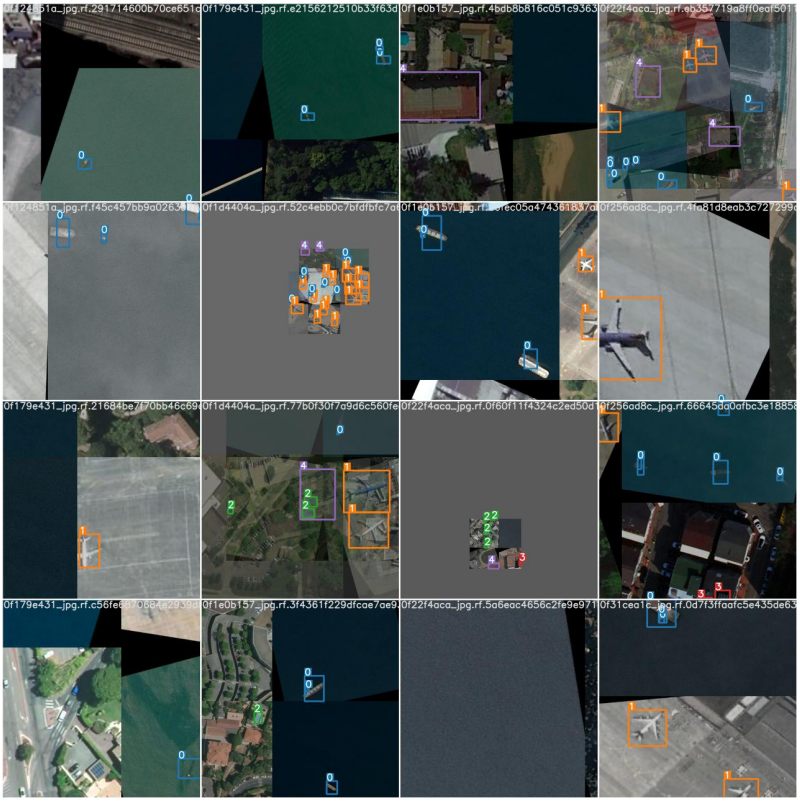

Figura 24: Validación YoloV7.

Al finalizar el entrenamiento del modelo se realizó un segundo entrenamiento del modelo Yolov7 con los mismos parámetros, exceptuando el número de épocas, el cual se ha modificado de 200 a 50 épocas.

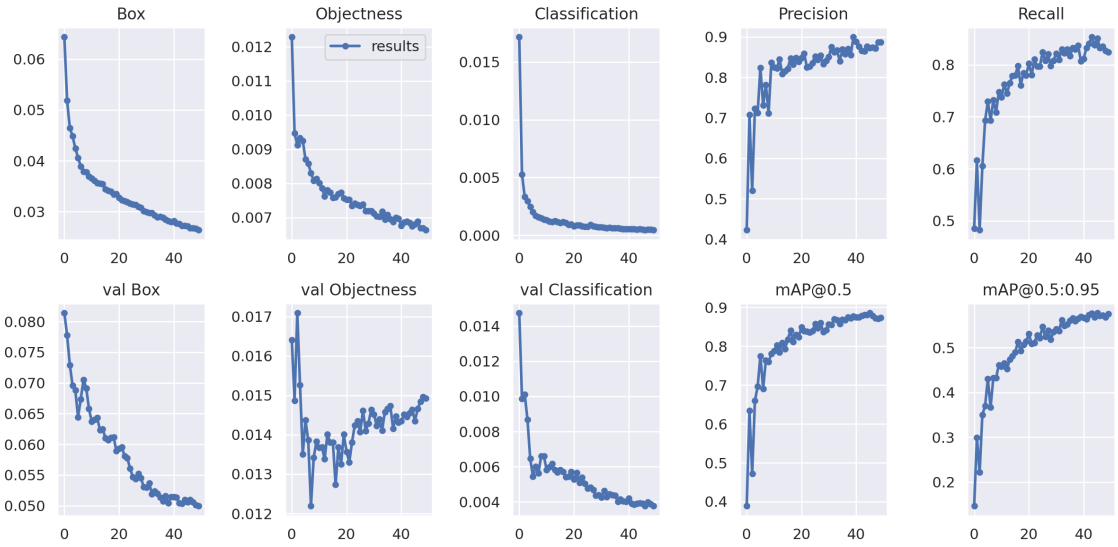

Figura 25: Resultados YoloV7 50 épocas.

En la Figura 25 se pueden observar los resultados del entrenamiento. El valor de **epoch** indica el número de épocas o iteraciones completadas. La métrica **train/box_loss** representa la pérdida de caja, reflejando qué tan bien el modelo ha ajustado las predicciones de las ubicaciones de los objetos en las imágenes de entrenamiento. La métrica **train/obj_loss** indica la pérdida de objeto, que refleja qué tan bien el modelo ha identificado y clasificado correctamente los objetos en las imágenes de entrenamiento. Además, la métrica **train/cls_loss** indica la pérdida de clasificación, que refleja qué tan bien el modelo ha clasificado correctamente los objetos en las diferentes clases o categorías en las imágenes de entrenamiento. Una pérdida baja en estas métricas sugiere que el modelo ha logrado un buen ajuste y clasificación de los objetos durante el entrenamiento.

metrics/precision: Un valor de 0.89960 sugiere que el 89.96 % de las predicciones positivas realizadas por el modelo son correctas. Esto indica que el modelo tiene una alta precisión en la clasificación de las instancias.

metrics/recall: Un valor de 0.85340 indica que el modelo ha identificado correctamente el 85.34 % de las instancias positivas. Esto sugiere que el modelo tiene una alta capacidad para detectar la mayoría de las instancias positivas.

metrics/mAP_0.5: Un valor de 0.88540 sugiere que el modelo ha obtenido una precisión promedio del 88.54 % en la detección de objetos con un umbral de confianza del 50 %. Esto indica que el modelo tiene un buen desempeño en la detección precisa de objetos.

metrics/mAP_0.5:0.95: Esta métrica mide la precisión promedio del modelo en la detección de objetos en un rango de umbrales de confianza del 50 % al 95 %.

Un valor de 0.57740 indica que el modelo ha obtenido una precisión promedio del 57.74 % en este rango de umbrales. Esto indica que el modelo tiene un rendimiento moderado en la detección de objetos en un rango más amplio de confianza.

4.2.4. Yolov8

El entrenamiento de YOLOv8 se realiza a lo largo de 200 épocas, es decir, se realiza un total de 200 pasadas a través de todo el conjunto de datos de entrenamiento. En la Tabla 4 se muestran los resultados tras finalizar el entrenamiento. En cada época, el modelo está tratando de minimizar tres tipos de pérdida: la pérdida de caja (*box_loss*), la pérdida de clase (*cls_loss*), y la pérdida de deformación de filtro (*dfl_loss*).

box_loss es la pérdida de localización del objeto. El modelo está intentando predecir correctamente las coordenadas de la caja delimitadora de cada objeto en la imagen. *box_loss* es la medida del error en estas predicciones.

cls_loss es la pérdida de clasificación del objeto. El modelo intenta predecir la clase del objeto dentro de cada cuadro delimitador (por ejemplo, avión, piscina, panel solar, etc.). *cls_loss* mide el error en estas predicciones de clase.

dfl_loss es la pérdida de deformación del filtro. YOLOv8 presenta un módulo de deformación del filtro que permite al modelo aprender transformaciones espaciales de las características, de manera más eficiente en resumen mide el error en estas transformaciones.

La columna *Instances* refiere al número de instancias (objetos) presentes en las imágenes del lote actual durante el entrenamiento. La columna *Size* indica el tamaño de las imágenes que se están procesando.

El entrenamiento toma un total de 5.309 horas, como se indica en la salida *200 epochs completed in 5.309 hours*. Después de esto, el optimizador es retirado de los pesos finales y se guardan los pesos del modelo, reduciendo así el tamaño del archivo de los pesos a 22.5 MB.

Posteriormente, se lleva a cabo una validación utilizando los mejores pesos guardados. Los resultados de esta validación se muestran en las tablas después del mensaje "Validating...". Estos resultados dan una visión detallada del rendimiento del modelo en datos que no se han utilizado durante el entrenamiento, lo que proporciona una indicación importante de cómo se puede esperar que el modelo se comporte en el mundo real.

Finalmente, se muestran las medidas de la velocidad de procesamiento del modelo, incluyendo el tiempo de preprocesamiento, inferencia, cálculo de pérdida y postprocesamiento por imagen. Esto proporciona información sobre la eficiencia del modelo en términos de tiempo de ejecución.

Aquí están los resultados finales del entrenamiento:

Por último, el modelo ha sido validado con los mejores pesos guardados, obte-

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95
all	700	2334	0.872	0.819	0.871	0.592
0	700	579	0.791	0.549	0.628	0.356
plane	700	499	0.853	0.832	0.883	0.553
pool	700	335	0.92	0.946	0.968	0.578
solarpanel	700	780	0.933	0.819	0.927	0.704
tenniscourt	700	141	0.863	0.95	0.949	0.768

Tabla 4: Resultados de entrenamiento tras 200 épocas con YOLOv8

niendo resultados similares a los del entrenamiento. También se ha medido el tiempo de preprocesamiento, inferencia, pérdida y postprocesamiento por imagen, que se encuentra en la línea que comienza con "Speed:", proporcionando información sobre la eficiencia del modelo en términos de tiempo de ejecución.

En la Figura 26 se observa la matriz de confusión de la clasificación de las 5 clases usando Yolov8. En el eje vertical se muestran las clases predichas, mientras que en el eje horizontal están las que son verdaderas. Las clases que mejor han sido clasificadas han sido las piscinas (*pool*) y las pistas de tenis (*tenniscourt*) aunque en general todas se han clasificado correctamente, exceptuando la clase más baja, la de los barcos(*0*). El resultado es mejor que en el modelo Yolov5 para todas las clases, pero peor que el modelo Yolov7.

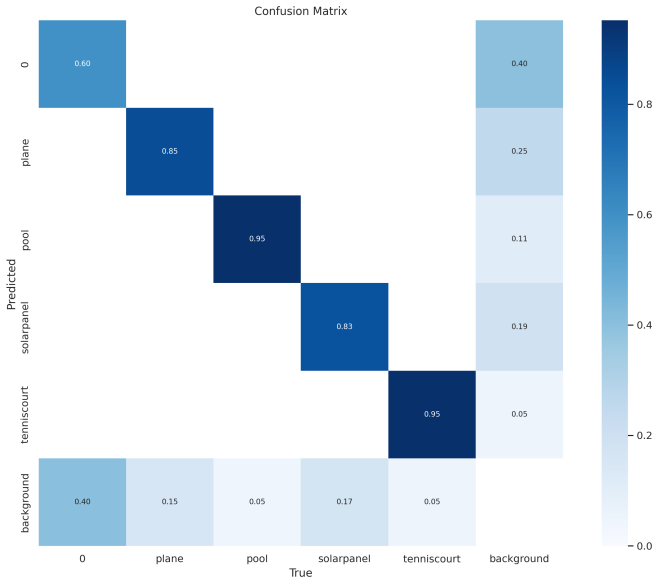

Figura 26: Matriz de confusión YoloV8.

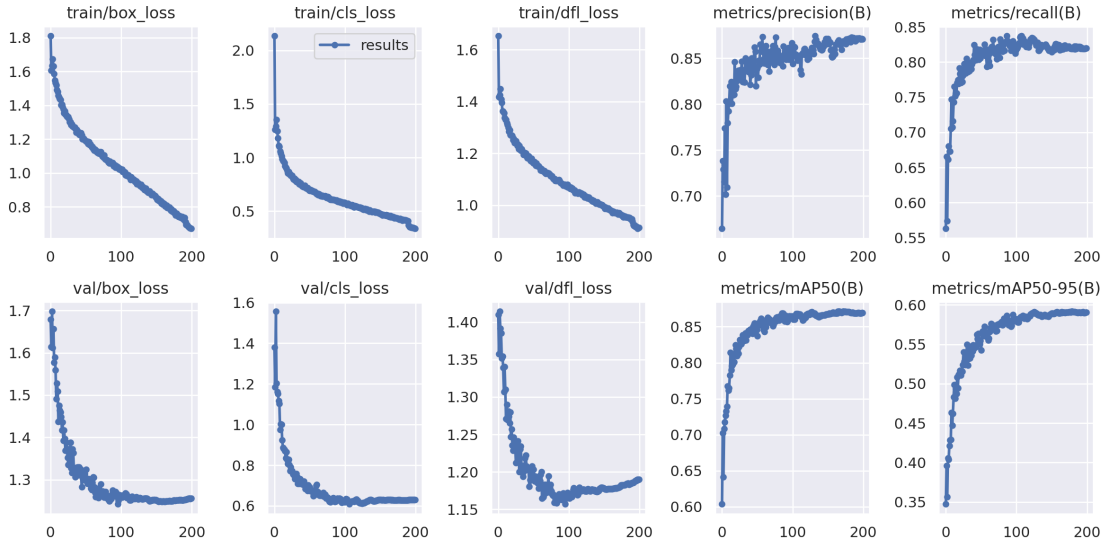

Figura 27: Resultados YoloV8.

- **Epochs**: Indica que has entrenado el modelo durante **200** épocas. En la Figura 27 se muestran los resultados del entrenamiento de manera gráfica.
- **train/box_loss** (**1.042255**): Este valor es relativamente alto, lo que sugiere que el modelo tiene dificultades para predecir las ubicaciones exactas de los objetos en las imágenes.
- **train/obj_loss** (**0.621270**): Al igual que la pérdida de la caja delimitadora, este valor también es bastante alto, lo que indica que el modelo está luchando para identificar correctamente los objetos.
- **train/cls_loss** (**1.093480**): Nuevamente, este valor es bastante alto, lo que sugiere que el modelo puede tener dificultades para clasificar correctamente los objetos.
- **metrics/precision** (**0.847831**): Este es un valor sólido para la precisión, lo que sugiere que el modelo es bastante bueno para predecir positivamente los objetos.
- **metrics/recall** (**0.806700**): Este también es un valor sólido para el *recall*, indicando que el modelo es bueno para identificar todos los ejemplos positivos en las imágenes.
- **metrics/mAP_0.5** (**0.849348**) y **metrics/mAP_0.5:0.95** (**0.563308**): Estos valores son decentes y muestran que el modelo está realizando un trabajo bastante bueno en la detección de objetos en general.
- **val/box_loss** (**1.297333**), **val/obj_loss** (**0.687272**) y **val/cls_loss** (**1.196873**): Estos valores son las pérdidas correspondientes en el conjunto de validación.

Son más altas que las pérdidas de entrenamiento, lo que sugiere que el modelo puede estar sobre ajustando a los datos de entrenamiento y generalizando menos efectivamente a los datos no vistos.

- **x/lr0, x/lr1, x/lr2** (**0.005575, 0.005073, 0.005073**): Estos son los valores de la tasa de aprendizaje para las diferentes partes de la red. Estos parecen ser valores estándar y no indican problemas particulares.

En la Figura 28 se observa la curva Precisión-Confidence, es una representación gráfica que muestra cómo varía la precisión de un modelo de clasificación a medida que se ajusta el umbral de confianza para asignar las clases. Los resultados indican que al ajustar el umbral de confianza, se obtiene una precisión del 94.5 % (0.955) para todas las clases cuando el valor de confianza es 1.00. Esto significa que cuando se establece un umbral de confianza alto, el modelo es muy preciso al predecir las clases, lo que implica que tiene una alta tasa de predicciones positivas correctas.

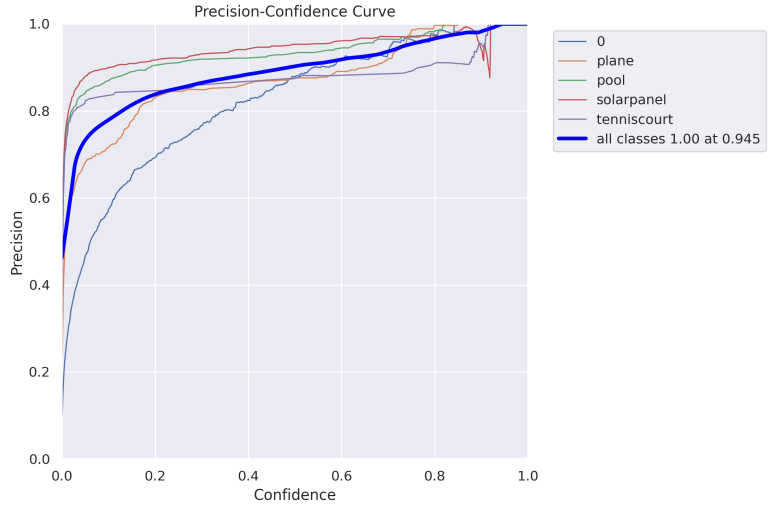

Figura 28: Curva de precisión YoloV8.

En la Figura 29 se observa la curva que muestra cómo varía la tasa de *recall* del modelo a medida que se ajusta el umbral de confianza. Los resultados indican que obtienes un *recall* del 90 % (0.90) para todas las clases cuando el valor de confianza es 0.000. Esto sugiere que, al disminuir el umbral de confianza, el modelo logra identificar correctamente una alta proporción de instancias positivas.

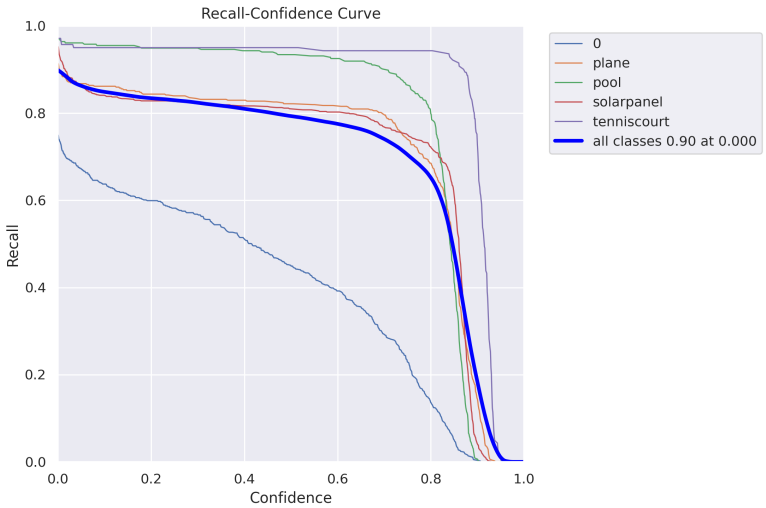

Figura 29: Curva de recall YoloV8.

En la Figura 30 se puede observar la implementación del modelo propuesto. Es evidente que el modelo identifica de manera efectiva las clases dentro de las imágenes satelitales proporcionadas. Las clases están asignadas con números que van desde 0 hasta 4, donde cada uno de ellos corresponde a una clase específica: 0 para los barcos, 1 para los aviones, 2 para las piscinas, 3 para las placas solares, y finalmente 4 para las pistas de tenis.

Figura 30: Validación YoloV8.

Al finalizar el entrenamiento del modelo se realizó un segundo entrenamiento del modelo Yolov8 con los mismos parámetros, exceptuando el número de épocas, el cual se ha modificado de 200 a 50 épocas.

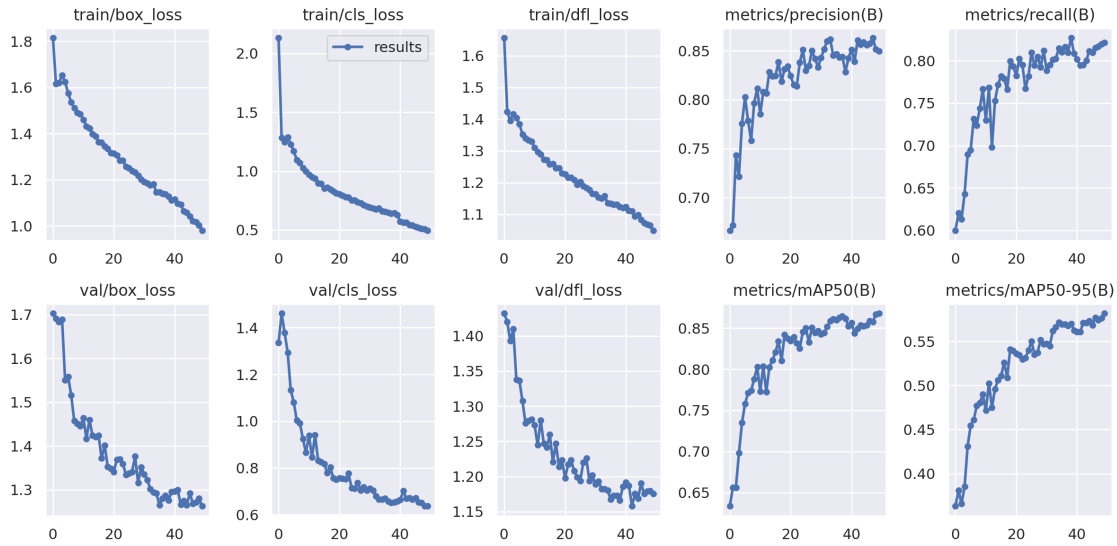

Figura 31: Resultados YoloV8 50 épocas.

En la Figura 31 se observan de manera gráfica los resultados del segundo entrenamiento. Durante el entrenamiento del modelo, el valor de ***epoch*** indica el número de épocas o iteraciones completadas. La métrica ***train/box_loss*** representa la pérdida de caja, reflejando qué tan bien el modelo ha ajustado las predicciones de las ubicaciones de los objetos en las imágenes de entrenamiento. La métrica **textit-train/obj_loss** indica la pérdida de objeto, que refleja qué tan bien el modelo ha identificado y clasificado correctamente los objetos en las imágenes de entrenamiento. Además, la métrica ***train/cls_loss*** indica la pérdida de clasificación, que refleja qué tan bien el modelo ha clasificado correctamente los objetos en las diferentes clases o categorías en las imágenes de entrenamiento. Una pérdida baja en estas métricas sugiere que el modelo ha logrado un buen ajuste y clasificación de los objetos durante el entrenamiento.

metrics/precision: Un valor de 0.821393 sugiere que el 82.1 % de las predicciones positivas realizadas por el modelo son correctas.

metrics/recall: Un valor de 0.771372 indica que el modelo ha identificado correctamente el 77.1 % de las instancias positivas.

metrics/mAP50: Un valor de 0.817963 sugiere que el modelo ha obtenido una precisión promedio del 81.8 % en la detección utilizando este umbral.

metrics/mAP50-95(B): Un valor de 0.522049 indica que el modelo ha obtenido una precisión promedio del 52.2 % en este rango de umbrales.

5. Resultados

En esta sección se ha realizado una comparación exhaustiva entre los resultados de cada modelo entrenado. Primero se realiza una comparación entre los resultados de cada modelo entrenado, para 200 y 50 épocas, posteriormente se comparan los modelos entre sí y por último se realiza una comparación entre los máximos que ha obtenido cada algoritmo en las métricas más relevantes.

YoloV3	**200 épocas**	**50 épocas**
train/box_loss	0.024511	0.033568
train/obj_loss	0.014552	0.018916
train/cls_loss	0.000514	0.001204
metrics/precision	0.874514	0.850224
metrics/recall	0.814735	0.795184
metrics/mAP_0.5	0.853922	0.837811
metrics/mAP_0.5:0.95	0.554961	0.515674
val/box_loss	0.033257	0.035766
val/obj_loss	0.015781	0.014280
val/cls_loss	0.000457	0.000730
x/lr0	0.005575	0.007137
x/lr1	0.005073	0.005132
x/lr2	0.005073	0.005132

Tabla 5: Resultados de entrenamiento para el modelo YOLOv3.

En la Tabla 5 se realiza una comparación del modelo Yolov3 entrenado 200 y 50 épocas.

Pérdida de Entrenamiento: En el modelo YOLOv3, el entrenamiento de 200 épocas arroja valores de pérdida de entrenamiento ('box', 'obj', 'cls') inferiores a los del entrenamiento de 50 épocas. Esto significa que el modelo con más épocas de entrenamiento es capaz de ajustar mejor los datos de entrenamiento.

Métricas de desempeño: Respecto a las métricas de precisión, *recall* ymAP, el modelo de 200 épocas tiene un rendimiento superior al de 50 épocas. Esto indica que con un mayor número de épocas, el modelo puede detectar con más precisión y exhaustividad los objetos de interés.

Pérdida de Validación: Similar a los resultados de entrenamiento, el modelo de 200 épocas exhibe una menor pérdida de validación para 'box' y 'cls'. Esto sugiere una mejor capacidad de generalización en datos no vistos para el modelo con más épocas de entrenamiento. Sin embargo, es interesante notar que la pérdida de 'obj' en la validación es ligeramente inferior para el modelo de 50 épocas.

Tasas de Aprendizaje: Las tasas de aprendizaje (lr0, lr1, lr2) son similares en ambos modelos. Hay una disminución marginal en las tasas de aprendizaje para el modelo de 200 épocas en comparación con el de 50 épocas, pero la diferencia es mínima y no afectaría significativamente la elección entre estos dos modelos.

Yolov5	**200 épocas**	**50 épocas**
train/box_loss	0.030749	0.038457
train/obj_loss	0.018133	0.021636
train/cls_loss	0.000588	0.001441
metrics/precision	0.862278	0.842288
metrics/recall	0.812028	0.775106
metrics/mAP_0.5	0.856920	0.820441
metrics/mAP_0.5:0.95	0.539615	0.494896
val/box_loss	0.034163	0.036847
val/obj_loss	0.013572	0.014038
val/cls_loss	0.000330	0.000701
x/lr0	0.005575	0.007137
x/lr1	0.005073	0.005132
x/lr2	0.005073	0.005132

Tabla 6: Resultados de entrenamiento para el modelo YOLOv5.

En la Tabla 6 se realiza una comparación del modelo Yolov5 entrenado 200 y 50 épocas.

- **Pérdida de Entrenamiento:** Al entrenar durante 200 épocas, se obtienen valores de pérdida de entrenamiento más bajos para 'box', 'obj' y 'cls', en comparación con el entrenamiento de 50 épocas. Esto indica que el modelo entrenado durante un mayor número de épocas se ajusta de manera más eficiente a los datos de entrenamiento.
- **Métricas de Desempeño:** En cuanto a la precisión, recall y mAP, el modelo entrenado durante 200 épocas supera al modelo entrenado durante 50 épocas. Esto sugiere que el modelo con más épocas de entrenamiento es capaz de detectar los objetos de interés con mayor precisión y completitud.
- **Pérdida de Validación:** De manera similar a los resultados de entrenamiento, el modelo entrenado durante 200 épocas muestra una menor pérdida de validación para 'box' y 'cls', lo que indica que este modelo tiene una mejor capacidad de generalización hacia datos no vistos. Sin embargo, la pérdida de 'obj' en validación es ligeramente menor en el modelo de 50 épocas.
- **Tasas de Aprendizaje:** Ambos modelos presentan tasas de aprendizaje similares (lr0, lr1, lr2). Aunque las tasas de aprendizaje son ligeramente más bajas en el modelo de 200 épocas, la diferencia es mínima y no sería un factor determinante en la elección entre estos dos modelos.

Yolov7	200 épocas	50 épocas
train/box_loss	0.027656	0.033356
train/obj_loss	0.006507	0.007599
train/cls_loss	0.000729	0.001399
total_loss	0.034892	0.042355
metrics/precision	0.862050	0.821928
metrics/recall	0.822230	0.776222
metrics/mAP_0.5	0.861656	0.811752
metrics/mAP_0.5:0.95	0.559435	0.497808
val/box_loss	0.055734	0.058317
val/obj_loss	0.014189	0.014244
val/cls_loss	0.005099	0.005372

Tabla 7: Resultados de entrenamiento para el modelo YOLOv7.

En la Tabla 7 se realiza una comparación del modelo Yolov7 entrenado 200 y 50 épocas.

Pérdida de Entrenamiento: En el modelo YOLOv7, se observa que el entrenamiento con 200 épocas resulta en valores de pérdida ('box', 'obj', 'cls', 'total') menores en comparación con el entrenamiento de 50 épocas. Esto sugiere que el modelo se ajusta mejor a los datos de entrenamiento cuando se entrena por más épocas.

Métricas de desempeño: El modelo de 200 épocas supera al de 50 épocas en todas las métricas de desempeño evaluadas (precisión, *recall*, mAP). Esto indica que el modelo de 200 épocas tiene mejor precisión y exhaustividad en la detección de objetos.

Pérdida de Validación: Para la pérdida de validación, los valores 'box', 'obj' y 'cls' son ligeramente menores en el modelo de 200 épocas. Esto implica que este modelo generaliza mejor a datos no vistos, aunque la diferencia no es sustancial.

Tasas de Aprendizaje: Las tasas de aprendizaje (lr0, lr1, lr2) son ligeramente más bajas en el modelo de 200 épocas, lo que podría sugerir una convergencia más suave del entrenamiento. Sin embargo, la diferencia es pequeña y probablemente no sea un factor crucial para elegir entre los dos modelos.

Yolov8	**200 épocas**	**50 épocas**
train/box_loss	1.042255	1.287606
train/obj_loss	0.621270	0.817990
train/cls_loss	1.093480	1.219344
metrics/precision	0.847831	0.821393
metrics/recall	0.806700	0.771372
metrics/mAP_0.5	0.849348	0.817963
metrics/mAP_0.5:0.95	0.563308	0.522049
val/box_loss	1.297333	1.380988
val/obj_loss	0.687272	0.808646
val/cls_loss	1.196873	1.231450
x/lr0	0.005575	0.007137
x/lr1	0.005073	0.005132
x/lr2	0.005073	0.005132

Tabla 8: Resultados de entrenamiento para el modelo YOLOv8.

En la Tabla 8 se realiza una comparación del modelo Yolov8 entrenado 200 y 50 épocas.

Pérdida de Entrenamiento: En el caso del modelo YOLOv8, se observa que el entrenamiento con 200 épocas resulta en valores de pérdida ('box', 'obj', 'cls', 'total') más bajos en comparación con el entrenamiento de 50 épocas. Esto sugiere que el modelo se ajusta de manera más precisa a los datos de entrenamiento cuando se le da más tiempo de entrenamiento.

Métricas de Desempeño: Al evaluar las métricas de desempeño, se encuentra que el modelo entrenado durante 200 épocas supera al modelo entrenado durante 50 épocas en todas las métricas evaluadas, como precisión, recall y mAP. Esto indica que el modelo entrenado durante 200 épocas tiene una mayor precisión y capacidad de detección exhaustiva de objetos.

Pérdida de Validación: Al analizar la pérdida de validación, se observa que los valores de 'box', 'obj' y 'cls' son ligeramente inferiores en el modelo entrenado durante 200 épocas. Esto sugiere que este modelo tiene una mejor capacidad de generalización hacia datos no vistos, aunque la diferencia no es significativa.

Tasas de Aprendizaje: En cuanto a las tasas de aprendizaje (lr0, lr1, lr2), también se observa que son ligeramente más bajas en el modelo entrenado durante 200 épocas. Sin embargo, al igual que en el caso de YOLOv7, esta pequeña diferencia probablemente no sea un factor decisivo al elegir entre estos dos modelos.

Métricas	Yolov3	Yolov5	Yolov7	Yolov8
train/box_loss	0.024511	0.030749	0.027656	1.042255
train/obj_loss	0.014552	0.018133	0.006507	0.621270
train/cls_loss	0.000514	0.000588	0.000729	1.093480
metrics/precision	0.874514	0.862278	0.862050	0.847831
metrics/recall	0.814735	0.812028	0.822230	0.806700
metrics/mAP_0.5	0.853922	0.856920	0.861656	0.849348
metrics/mAP_0.5:0.95	0.554961	0.539615	0.559435	0.563308
val/box_loss	0.033257	0.034163	0.055734	1.297333
val/obj_loss	0.015781	0.013572	0.014189	0.687272
val/cls_loss	0.000457	0.000330	0.005099	1.196873
x/lr0	0.005575	0.005575		0.005575
x/lr1	0.005073	0.005073		0.005073
x/lr2	0.005073	0.005073		0.005073
hours	22.183	4.194	21.018	5.309

Tabla 9: Resultados de entrenamiento para todos los modelos.

En la Tabla 9 se realiza una comparación entre todos los modelos entrenados con 200 épocas.

- **Pérdida de Entrenamiento:** Los modelos YOLOv3, YOLOv5, y YOLOv7 muestran pérdidas de entrenamiento similares, con YOLOv3 mostrando la menor pérdida para 'box' y 'obj', YOLOv5 la menor para 'cls', y YOLOv7 con pérdidas intermedias. Sin embargo, YOLOv8 muestra pérdidas considerablemente más altas en todas las categorías.

- **Métricas de Precisión:** YOLOv3 tiene la mayor precisión, seguido de cerca por YOLOv5 y YOLOv7. YOLOv8, a pesar de su pérdida significativamente más alta, no queda muy por detrás en términos de precisión.

- **Métricas de *Recall:*** YOLOv7 tiene el *recall* más alto, con YOLOv3 y YOLOv5 muy cerca. Nuevamente, YOLOv8, a pesar de las altas pérdidas, no se queda muy atrás.

- **Métricas *mAP:*** En términos de mAP@0.5, todos los modelos son bastante comparables, aunque YOLOv7 lidera ligeramente. Sin embargo, en mAP@0.5:0.95, YOLOv8 supera al resto, a pesar de su alta pérdida.

- **Pérdida de Validación:** YOLOv8 tiene una pérdida de validación mucho mayor que los otros modelos. Entre los otros, YOLOv7 tiene la mayor pérdida de 'box', mientras que YOLOv3 tiene las menores pérdidas de 'obj' y 'cls'.

- **Pérdida de Entrenamiento:** Resulta notable cómo la pérdida de 'cls' en el modelo YOLOv8 es órdenes de magnitud mayor en comparación con los otros modelos. Esto podría indicar problemas en la clasificación durante el entrenamiento, tal vez debido a la complejidad del modelo o la necesidad de ajustes adicionales en la configuración del entrenamiento.

- **Desempeño de Validación *versus* Entrenamiento:** YOLOv7 muestra una diferencia considerable entre la pérdida de entrenamiento y la de validación para 'box' y 'cls', sugiriendo un posible sobreajuste del modelo a los datos de entrenamiento. Por otro lado, YOLOv3 presenta una consistencia más alta entre los resultados de entrenamiento y validación, lo que podría indicar una mejor generalización del modelo.
- **Relación Precision-*Recall:*** Aunque YOLOv3 presenta la mayor precisión, su *recall* no es el más alto, siendo superado por YOLOv7. Este es un recordatorio de que la elección del modelo puede depender del equilibrio específico entre precisión y *recall* que se requiera para la aplicación en cuestión.
- **Performance en *mAP:*** Aunque YOLOv8 tiene la mayor puntuación en mAP@0.5:0.95, también tiene pérdidas de entrenamiento y validación mucho mayores. Esto podría indicar que, a pesar de su rendimiento en esta métrica, el modelo podría no ser el más robusto o el más generalizable.
- **Tasas de Aprendizaje:** Aunque las tasas de aprendizaje son las mismas para todos los modelos, esto no se traduce en un desempeño uniforme. Esto subraya que, aunque la tasa de aprendizaje es un parámetro importante en el entrenamiento de la red, no es el único factor que determina el éxito del entrenamiento.

Métricas	Yolov3	Yolov5	Yolov7	Yolov8
train/box_loss	0.033568	0.038457	0.033356	1.287606
train/obj_loss	0.018916	0.021636	0.007599	0.817990
train/cls_loss	0.001204	0.001441	0.001399	1.219344
metrics/precision	0.850224	0.842288	0.821928	0.821393
metrics/recall	0.795184	0.775106	0.776222	0.771372
metrics/mAP_0.5	0.837811	0.820441	0.811752	0.817963
metrics/mAP_0.5:0.95	0.515674	0.494896	0.497808	0.522049
val/box_loss	0.035766	0.036847	0.058317	1.380988
val/obj_loss	0.014280	0.014038	0.014244	0.808646
val/cls_loss	0.000730	0.000701	0.005372	1.231450
x/lr0	0.007137	0.007137		0.007137
x/lr1	0.005132	0.005132		0.005132
x/lr2	0.00513	0.005132		0.005132
hours	6.911	1.475	5.445	2.801

Tabla 10: Resultados de entrenamiento para todos los modelos.

En la Tabla 10 se realiza una comparación entre todos los modelos entrenados con 50 épocas.

- **Pérdida de Entrenamiento:** Los modelos YOLOv3, YOLOv5, y YOLOv7 muestran pérdidas de entrenamiento similares, con YOLOv3 mostrando la menor pérdida para 'box' y 'obj', YOLOv5 la menor para 'cls', y YOLOv7 con pérdidas intermedias. Sin embargo, YOLOv8 muestra pérdidas considerablemente más altas en todas las categorías.

- **Métricas de Precisión:** YOLOv3 tiene la mayor precisión, seguido de cerca por YOLOv5 y YOLOv7. YOLOv8, a pesar de su pérdida significativamente más alta, no queda muy por detrás en términos de precisión.

- **Métricas de Recall:** YOLOv7 tiene el *recall* más alto, con YOLOv3 y YOLOv5 muy cerca. Nuevamente, YOLOv8, a pesar de las altas pérdidas, no se queda muy atrás.

- **Métricas *mAP:*** En términos de mAP@0.5, todos los modelos son bastante comparables, aunque YOLOv7 lidera ligeramente. Sin embargo, en mAP@0.5:0.95, YOLOv8 supera al resto, a pesar de su alta pérdida.

- **Pérdida de Validación:** YOLOv8 tiene una pérdida de validación mucho mayor que los otros modelos. Entre los otros, YOLOv7 tiene la mayor pérdida de 'box', mientras que YOLOv3 tiene las menores pérdidas de 'obj' y 'cls'.

- **Pérdida de Entrenamiento:** Resulta notable cómo la pérdida de 'cls' en el modelo YOLOv8 es órdenes de magnitud mayor en comparación con los otros modelos. Esto podría indicar problemas en la clasificación durante el entrenamiento, tal vez debido a la complejidad del modelo o la necesidad de ajustes adicionales en la configuración del entrenamiento.

- **Desempeño de Validación *versus* Entrenamiento:** YOLOv7 muestra una diferencia considerable entre la pérdida de entrenamiento y la de validación para 'box' y 'cls', sugiriendo un posible sobreajuste del modelo a los datos de entrenamiento. Por otro lado, YOLOv3 presenta una consistencia más alta entre los resultados de entrenamiento y validación, lo que podría indicar una mejor generalización del modelo.
- **Relación Precision-*Recall:*** Aunque YOLOv3 presenta la mayor precisión, su *recall* no es el más alto, siendo superado por YOLOv7. Este es un recordatorio de que la elección del modelo puede depender del equilibrio específico entre precisión y *recall* que se requiera para la aplicación en cuestión.
- **Rendimiento en *mAP:*** Aunque YOLOv8 tiene la mayor puntuación en mAP@0.5:0.95, también tiene pérdidas de entrenamiento y validación mucho mayores. Esto podría indicar que, a pesar de su rendimiento en esta métrica, el modelo podría no ser el más robusto o el más generalizable.
- **Tasas de Aprendizaje:** Aunque las tasas de aprendizaje son las mismas para todos los modelos, esto no se traduce en un desempeño uniforme. Esto subraya que, aunque la tasa de aprendizaje es un parámetro importante en el entrenamiento de la red, no es el único factor que determina el éxito del entrenamiento.

	Yolov3	Yolov5	Yolov7	Yolov8
Precisión	0.90174	0.89157	0.9076	0.87427
Recall	0.84385	0.84305	0.8643	0.87427
mAP	0.87601	0.87968	0.8897	0.87133

Tabla 11: Resultados de 200 épocas

En la Tabla 11 se realiza una compración entre los valores máximos que han alcanzado los modelos con un entrenamiento con 200 épocas.

- **Precisión:** YOLOv7 alcanza la mayor precisión, seguido por YOLOv3 y YOLOv5 que tienen resultados bastante cercanos. YOLOv8 tiene el valor más bajo de precisión.
- ***Recall:*** YOLOv7 también lidera en *recall*, siendo YOLOv3 y YOLOv8 los que le siguen. YOLOv5 tiene el menor valor de *recall* en este caso.
- ***mAP:*** De nuevo, YOLOv7 lidera en mAP, con YOLOv5 y YOLOv3 siguiéndole muy de cerca. YOLOv8 tiene la menor puntuación.

	Yolov3	Yolov5	Yolov7	Yolov8
Precisión	0.90564	0.89176	0.8996	0.86331
Recall	0.84214	0.82183	0.8534	0.82692
mAP	0.88078	0.86555	0.8854	0.86786

Tabla 12: Resultados de 50 épocas

En la Tabla 12 se realiza una compración entre los valores máximos que han alcanzado los modelos con un entrenamiento con 50 épocas.

- **Precisión:** En este caso, YOLOv3 tiene la mayor precisión, seguido de cerca por YOLOv7. YOLOv5 y YOLOv8 presentan resultados similares y son los que tienen menor precisión.
- ***Recall:*** YOLOv7 tiene el *recall* más alto, seguido por YOLOv3, YOLOv8 y finalmente YOLOv5.
- ***mAP:*** De nuevo, YOLOv7 tiene la mayor puntuación en mAP, seguido por YOLOv3, YOLOv5 y finalmente YOLOv8.

6. Conclusiones

Para este trabajo se ha realizado un estudio detallado de los métodos de detección de objetos en imágenes, centrando el enfoque en los algoritmos más empleados hasta la fecha. Se ha llevado a cabo una exhaustiva revisión de la literatura, incluyendo una amplia gama de técnicas y arquitecturas. Por otro lado, se ha investigado la evolución del algoritmo YOLO (You Only Look Once), analizando las distintas versiones y mejoras a lo largo de su desarrollo. Esta revisión del estado del arte ha proporcionado una base sólida para el desarrollo y la evaluación posterior de los métodos de detección implementados.

A su vez, se ha creado un conjunto de datos personalizado, para el caso de estudio, compuesto por imágenes satelitales. Este conjunto fue cuidadosamente seleccionado y adaptado, con el objetivo de ser utilizado en la detección de objetos en los métodos implementados. Se ha realizado el preprocesamiento de las imágenes, el cual implicó la normalización, el redimensionado y la eliminación de imágenes para asegurar las condiciones adecuadas para el entrenamiento de los modelos. Además, se llevaron a cabo anotaciones para etiquetar los objetos de interés en las imágenes, permitiendo así el entrenamiento y validación de los modelos con una elevada precisión. La creación de este conjunto de datos personalizado demostró ser un pilar fundamental en el éxito del proyecto. Se ha realizado el entrenamiento de los modelos de detección de objetos con el conjunto de datos. Hemos logrado entrenar con éxito varios modelos de detección de objetos utilizando nuestro conjunto de datos. Posteriormente, se ha realizado la evaluación del rendimiento de los algoritmos y métodos de detección implementados: Hemos evaluado de manera exhaustiva los algoritmos y métodos de detección que implementamos, usando una variedad de métricas para obtener una comprensión clara de su rendimiento. Por último, se ha realizado el contraste y evaluación de los diferentes métodos de detección de objetos con el objetivo de poder seleccionar el enfoque que mejor se adapte a una aplicación concreta. En general, el entrenamiento de los modelos durante 200 épocas produjo mejores resultados en términos de pérdidas y métricas de rendimiento en comparación con 50 épocas. Este resultado es consistente en todos los modelos y sugiere que un mayor número de épocas permite al modelo aprender y ajustarse mejor a los datos de entrenamiento.

Sin embargo, en términos de eficiencia (tiempo y recursos de cómputo), entrenar durante 50 épocas puede ser una mejor opción. Aunque los modelos entrenados durante 50 épocas pueden no rendir tan bien como sus contrapartes de 200 épocas, aún proporcionan resultados razonablemente buenos y lo hacen en un tiempo más corto.

Al comparar todos los modelos entre sí, YOLOv7 y YOLOv3 tienden a proporcionar los mejores resultados en general. Estos dos modelos muestran menores pérdidas y métricas de rendimiento más altas en comparación con YOLOv5 y YOLOv8.

YOLOv8 se destacó por tener valores de pérdida significativamente más altos, lo que podría indicar un sobreajuste del modelo. Aunque las métricas de rendimiento de YOLOv8 no son las más bajas, la alta pérdida sugiere que el modelo puede tener dificultades para generalizar a datos nuevos. En términos de las métricas máximas

alcanzadas, YOLOv7 nuevamente tiende a liderar en la mayoría de las métricas para ambas duraciones de entrenamiento. YOLOv3 y YOLOv5 a menudo le siguen de cerca.

Estos resultados sugieren que YOLOv7 puede ser la mejor opción si se busca el máximo rendimiento, independientemente de la duración del entrenamiento. Sin embargo, la elección del modelo también puede depender de otros factores, como las necesidades específicas de la aplicación, la disponibilidad de recursos de cómputo y la cantidad de tiempo disponible para el entrenamiento.

Si la prioridad es obtener el mejor rendimiento posible, parece que un entrenamiento más largo (200 épocas) con YOLOv7 sería la mejor opción. Sin embargo, si la eficiencia y la economía de recursos son una consideración importante, entonces un entrenamiento más corto (50 épocas) con YOLOv3 podría ser una buena elección.

6.1. Líneas futuras

Como líneas de trabajo futuras planteamos algunas alternativas:

1. Ampliación de la comparación a otros modelos de detección de objetos: Existen otros modelos de detección de objetos más allá de los basados en YOLO, como SSD, Faster R-CNN, o EfficientDet. La inclusión de estos modelos en la comparativa proporcionaría una visión más completa del panorama de la detección de objetos.

2. Prueba con diferentes configuraciones de los modelos: Cada uno de los modelos de YOLO tiene diversas configuraciones que se pueden ajustar. Analizar cómo la modificación de ciertos parámetros afecta al rendimiento de los modelos podría aportar una mayor comprensión de su funcionamiento.

3. Investigar la resistencia de los modelos a las adversidades en el mundo real: Muchas aplicaciones de detección de objetos tienen que lidiar con condiciones desafiantes en el mundo real, como la poca luz, la obstrucción de objetos, las condiciones climáticas adversas, etc. Analizar cómo cada modelo se comporta frente a estos desafíos podría ser de interés.

4. Detección de objetos en vídeos: Investigar la detección de objetos en vídeos, donde se busca identificar y rastrear objetos en movimiento a lo largo del tiempo y entrenar y comparar los modelos con el nuevo conjunto de datos.

Referencias

[1] https://lamaquinaoraculo.com/computacion/deteccion-de-objetos/. [*Citado en págs.* VI *y 18.*]

[2] https://blog.roboflow.com/what-is-a-confusion-matrix/. [*Citado en págs.* VI *y 25.*]

[3] John Smith. Object detection in computer vision. *International Journal of Computer Vision*, 75(1):119–130, 2008. [*Citado en pág. 1.*]

[4] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. *Digital image processing using MATLAB*. Gatesmark Publishing, 2004. [*Citado en pág. 1.*]

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. pages 580–587, 2014. [*Citado en págs. 1 y 14.*]

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009. [*Citado en pág. 1.*]

[7] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2016. [*Citado en pág. 2.*]

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. [*Citado en pág. 3.*]

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. [*Citado en pág. 3.*]

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. [*Citado en pág. 3.*]

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. [*Citado en pág. 3.*]

[12] Connie Schardt, Melissa B Adams, Tom Owens, Sheri Keitz, and Paul Fontelo. Utilization of the pico framework to improve searching pubmed for clinical questions. *BMC Medical Informatics and Decision Making*, 7(1):16, 2007. [*Citado en pág. 7.*]

[13] P. Shi, Q. Jiang, C. Shi, J. Xi, G. Tao, S. Zhang, Z. Zhang, B. Liu, X. Gao, and Q. Wu. Oilwell detection via large-scale and high-resolution remote sensing images based on improved yolo v4. *Remote Sensing*, 13(16), 2021. cited By 7. [*Citado en pág. 11.*]

[14] R. La Grassa, G. Cremonese, I. Gallo, C. Re, and E. Martellato. Yololens: A deep learning model based on super-resolution to enhance the crater detection of the planetary surfaces. *Remote Sensing*, 15(5), 2023. cited By 0. [*Citado en pág. 11.*]

[15] K. Patel, C. Bhatt, and P.L. Mazzeo. Improved ship detection algorithm from satellite images using yolov7 and graph neural network. *Algorithms*, 15(12), 2022. cited By 2. [*Citado en pág. 11.*]

[16] Y. Wang, L. Cui, C. Zhang, W. Chen, Y. Xu, and Q. Zhang. A two-stage seismic damage assessment method for small, dense, and imbalanced buildings in remote sensing images. *Remote Sensing*, 14(4), 2022. cited By 11. [*Citado en pág. 11.*]

[17] O.L.F. de Carvalho, O.A. de Carvalho, A.O. Albuquerque, P.P. Bem, C.R. Silva, P.H.G. Ferreira, R.S. de Moura, R.A.T. Gomes, R.F. Guimarães, and D.L. Borges. Instance segmentation for large, multi-channel remote sensing imagery using mask-rcnn and a mosaicking approach. *Remote Sensing*, 13(1):1–24, 2021. cited By 38. [*Citado en pág. 11.*]

[18] M. Sakeena, E. Stumpe, M. Despotovic, D. Koch, and M. Zeppelzauer. On the robustness and generalization ability of building footprint extraction on the example of segnet and mask r-cnn. *Remote Sensing*, 15(8), 2023. cited By 0. [*Citado en pág. 11.*]

[19] D. Yan, H. Zhang, G. Li, X. Li, H. Lei, K. Lu, L. Zhang, and F. Zhu. Improved method to detect the tailings ponds from multispectral remote sensing images based on faster r-cnn and transfer learning. *Remote Sensing*, 14(1), 2022. cited By 9. [*Citado en pág. 11.*]

[20] M. Amo-Boateng, N. Ekow Nkwa Sey, A. Ampah Amproche, and M. Kye-reh Domfeh. Instance segmentation scheme for roofs in rural areas based on mask r-cnn. *Egyptian Journal of Remote Sensing and Space Science*, 25(2):569–577, 2022. cited By 2. [*Citado en pág. 11.*]

[21] Qifan Wu, Daqiang Feng, Changqing Cao, Xiaodong Zeng, Zhejun Feng, Jin Wu, and Ziqiang Huang. Improved mask r-cnn for aircraft detection in remote sensing images. *SENSORS*, 21(8), APR 2021. [*Citado en pág. 11.*]

[22] Emmanuel Maggiori, Guillaume Charpiat, Yuliya Tarabalka, and Pierre Alliez. Recurrent neural networks to correct satellite image classification maps. *IEEE Transactions on Geoscience and Remote Sensing*, 55(9):4962–4971, 2017. [*Citado en pág. 11.*]

[23] Lior Rubanenko, Sebastian Pérez-López, Joseph Schull, and Mathieu G. A. Lapôtre. Automatic detection and segmentation of barchan dunes on mars and earth using a convolutional neural network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:9364–9371, 2021. [*Citado en pág. 11.*]

[24] F. Doğan and I. Turkoğlu. Comparison of deep learning models in terms of multiple object detection on satellite images. *Journal of Engineering Research (Kuwait)*, 10(3):89–108, 2022. cited By 0. [*Citado en pág. 11.*]

[25] Z. Chen, K. Lu, L. Gao, B. Li, J. Gao, X. Yang, M. Yao, and B. Zhang. Automatic detection of track and fields in china from high-resolution satellite images using multi-scale-fused single shot multibox detector. *Remote Sensing*, 11(11), 2019. cited By 6. [*Citado en pág. 11.*]

[26] M. Krinitskiy, P. Verezemskaya, K. Grashchenkov, N. Tilinina, S. Gulev, and M. Lazzara. Deep convolutional neural networks capabilities for binary classification of polar mesocyclones in satellite mosaics. *Atmosphere*, 9(11), 2018. cited By 16. [*Citado en pág. 11.*]

[27] S. Yeşilmen and B. Tatar. Efficiency of convolutional neural networks (cnn) based image classification for monitoring construction related activities: A case study on aggregate mining for concrete production. *Case Studies in Construction Materials*, 17, 2022. cited By 5. [*Citado en pág. 11.*]

[28] Mainak Bandyopadhyay. Multi-stack hybrid cnn with non-monotonic activation functions for hyperspectral satellite image classification. *NEURAL COMPUTING & APPLICATIONS*, 33(21):14809–14822, NOV 2021. [*Citado en pág. 11.*]

[29] Elena C. Rodriguez-Garlito, Abel Paz-Gallardo, and Antonio Plaza. Automatic detection of aquatic weeds: A case study in the guadiana river, spain. *IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING*, 15:8567–8585, 2022. [*Citado en pág. 11.*]

[30] S. Jiang, W. Yao, M.S. Wong, G. Li, Z. Hong, T.-Y. Kuc, and X. Tong. An optimized deep neural network detecting small and narrow rectangular objects in google earth images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:1068–1081, 2020. cited By 17. [*Citado en pág. 11.*]

[31] D.-G. Stuparu, R.-I. Ciobanu, and C. Dobre. Vehicle detection in overhead satellite images using a one-stage object detection model. *Sensors (Switzerland)*, 20(22):1–18, 2020. cited By 10. [*Citado en pág. 12.*]

[32] L. Courtrai, M.-T. Pham, and S. Lefèvre. Small object detection in remote sensing images based on super-resolution with auxiliary generative adversarial networks. *Remote Sensing*, 12(19):1–19, 2020. cited By 43. [*Citado en pág. 12.*]

[33] Yi Wang Syed Muhammad Arsalan Bashir. Small object detection in remote sensing images with residual feature aggregation-based super-resolution and object detector network. *Remote Sensing*, 13(9), 2021. cited By 18. [*Citado en pág. 12.*]

[34] Xinyang Song, Zhen Hua, and Jinjiang Li. Pstnet: Progressive sampling transformer network for remote sensing image change detection. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 15:8442–8455, 2022. [*Citado en pág. 12.*]

[35] Y. Yu, H. Ai, X. He, S. Yu, X. Zhong, and M. Lu. Ship detection in optical satellite images using haar-like features and periphery-cropped neural networks. *IEEE Access*, 6:71122–71131, 2018. cited By 15. [*Citado en pág. 12.*]

[36] S. Wiam, T. Khouloud, H. Bouchra, S.M. Nabil, and K. Adil. Hybrid deep learning architecture for land use: Land cover images classification with a comparative and experimental study. *International Journal of Advanced Computer Science and Applications*, 13(12):899–910, 2022. cited By 0. [*Citado en pág. 12.*]

[37] N.N. Navnath, K. Chandrasekaran, A. Stateczny, V.M. Sundaram, and P. Panneer. Spatiotemporal assessment of satellite image time series for land cover classification using deep learning techniques: A case study of reunion island, france. *Remote Sensing*, 14(20), 2022. cited By 1. [*Citado en pág. 12.*]

[38] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. 1:I–511, 2001. [*Citado en pág. 13.*]

[39] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 1, 2005. [*Citado en pág. 13.*]

[40] Ross Girshick. Fast r-cnn. pages 1440–1448, 2015. [*Citado en pág. 14.*]

[41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. pages 91–99, 2015. [*Citado en pág. 14.*]

[42] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. pages 2980–2988, 2017. [*Citado en pág. 15.*]

[43] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. pages 21–37, 2016. [*Citado en pág. 15.*]

[44] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988. IEEE, 2017. [*Citado en pág. 15.*]

[45] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. [*Citado en pág. 17.*]

[46] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. [*Citado en pág. 19.*]

[47] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. [*Citado en pág. 20.*]

[48] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. [*Citado en pág. 20.*]

[49] Roboflow. `https://roboflow.com/`. [*Citado en pág. 23.*]

[50] Roboflow universe. `https://universe.roboflow.com/`. [*Citado en pág. 27.*]