UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Active Roads Pipeline (ARP): pipeline de detección de caminos activos

Informe de Proyecto de Grado presentado por

Genaro Nadile, Martín Marr, Tatiana Perera

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Gonzalo Tejera
Mercedes Marzoa

Montevideo, 19 de agosto de 2025

Agradecimientos

Queremos expresar un agradecimiento especial a nuestras familias y parejas por el apoyo incondicional brindado a lo largo de este proceso y de toda la carrera. En especial, a quienes nos acompañaron y también a quienes, lamentablemente, ya no están, pero siguen presentes en nuestro recuerdo. Sin ustedes a nuestro lado, esto no hubiera sido posible.

Agradecemos también a nuestros tutores por su guía y compromiso durante el desarrollo del proyecto.

Finalmente, extendemos nuestro agradecimiento a todas las personas que contribuyeron a nuestra formación y hicieron posible que hoy lleguemos a esta etapa.

Resumen

El control de plagas agrícolas representa un desafío constante para los productores, no solo por los elevados costos que implica realizar un monitoreo continuo y aplicar medidas eficaces para su erradicación, sino también por las significativas pérdidas económicas.

Particularmente en América del Sur, las hormigas cortadoras de hojas, pertenecientes principalmente a los géneros Atta y Acromyrmex, representan una amenaza considerable para múltiples cultivos. Se estima que una sola colonia adulta puede dañar hasta un 10 % de una hectárea cultivada. Cuando varias colonias actúan en simultáneo, especialmente en etapas críticas de crecimiento como la floración o el brote inicial, el daño puede incrementarse, afectando severamente la producción agrícola anual.

Ante esta problemática, es indispensable implementar herramientas tecnológicas efectivas que permitan monitorear, detectar y dar seguimiento a estas plagas en tiempo real. Este proyecto aborda dicha necesidad mediante el desarrollo de un sistema basado en técnicas de inteligencia artificial, a través de la implementación de un *pipeline* que segmenta caminos y detecta sobre ellos la presencia de hormigas cortadoras, facilitando así su control.

Los resultados del proyecto muestran que los modelos de segmentación y detección (YOLOv11) empleados en el sistema alcanzan métricas satisfactorias cuando se evalúan de forma aislada, pero su desempeño se degrada significativamente al integrarlos en el sistema ARP y aplicarlos sobre el conjunto de evaluación del mismo. En particular, la segmentación logra resultados aceptables, mientras que la detección de hormigas dentro de los caminos presenta un desempeño limitado, lo que afecta la efectividad global del sistema. Esto evidencia la necesidad de mejorar la calidad y diversidad de los datos de entrenamiento para obtener mejores resultados en escenarios reales.

Palabras clave: Hormigas cortadoras de hojas (Leaf Cutting Ants), Detección de hormigas, Segmentación de caminos, Proyectos de Grado, Computación, Inteligencia artificial

Índice general

Capítulo 1

Introducción

1.1. Presentación del problema y contexto

La agricultura es considerada uno de los sectores productivos más importantes a nivel global, siendo especialmente relevante en América del Sur debido a su gran diversidad y riqueza en recursos naturales (Food y Organization, 2019). Sin embargo, uno de los principales desafíos que enfrentan los productores agrícolas en esta región es el manejo efectivo de las plagas. Este problema genera no solo altos costos relacionados con el monitoreo constante y las acciones necesarias para su control o erradicación, sino también importantes pérdidas económicas derivadas de los daños ocasionados a las cosechas ((IAEA), 2025).

Dentro de este contexto, las hormigas cortadoras de hojas, también conocidas como hormigas jardineras o cultivadoras, pertenecientes a los géneros Atta y Acromyrmex, constituyen un grupo de insectos que recolectan material vegetal (forraje) para cultivar un hongo simbiótico, del cual se alimentan.

Las hormigas cortadoras de hojas son consideradas una de las plagas agrícolas de mayor relevancia en Latinoamérica debido al amplio espectro de plantas que atacan, incluyendo aproximadamente 47 tipos de cultivos agrícolas y hortícolas, 13 especies de pastos forrajeros y unas 50 especies forestales, tanto autóctonas como introducidas. El impacto de estas hormigas sobre la agricultura es considerable, llegando a ocasionar pérdidas en productividad y rentabilidad de hasta un 35 % (Tafur, 2019). Además, se estima que una colonia adulta puede afectar hasta el 10 % del área de cultivo, agravándose cuando múltiples colonias actúan en simultáneo en períodos críticos de la planta, como lo son la floración o el brote inicial (Dell'Onte, 2023).

Actualmente, los mecanismos utilizados para combatir esta plaga incluyen métodos como la localización y fumigación directa del hormiguero, así como

la colocación estratégica de cebos granulados que, al ser recolectados por las hormigas, modifican las condiciones del hongo simbiótico, limitando así su fuente principal de alimento. Otros métodos utilizados son la aplicación de barreras físicas para impedir el acceso de las hormigas a los cultivos, la inundación o destrucción mecánica de hormigueros, y el control biológico mediante hongos o microorganismos patógenos específicos contra hormigas cortadoras.

Dado el impacto ambiental, económico y productivo que puede generar el uso inadecuado de pesticidas, la Organización de las Naciones Unidas para la Alimentación y la Agricultura (FAO) promueve el enfoque del Manejo Integrado de Plagas (MIP) como una estrategia clave para reducir la dependencia del uso intensivo de pesticidas. El MIP combina distintas prácticas complementarias (como el control biológico, las técnicas culturales, el monitoreo sistemático y el uso racional de agroquímicos) con el objetivo de mantener las poblaciones de plagas por debajo de los niveles económicamente dañinos. Además de contribuir a la sostenibilidad ambiental, esta estrategia busca proteger la salud humana y fomentar prácticas agrícolas compatibles con un desarrollo productivo y ambientalmente equilibrado (Food and Agriculture Organization of the United Nations, 2025).

Si bien muchas de las técnicas actualmente empleadas para el control de hormigas cortadoras se alinean parcialmente con los principios del MIP, su implementación suele enfrentar obstáculos económicos, operativos o ambientales. Por esta razón, existe un creciente interés en desarrollar métodos tecnológicos basados en inteligencia artificial y visión computacional, capaces de detectar y monitorear de forma temprana la presencia y actividad de estas hormigas, promoviendo así un manejo preventivo más efectivo, sustentable y menos invasivo.

1.2. Objetivos de la investigación

En base al problema planteado, se establecen los siguientes objetivos.

1.2.1. Objetivo general

Desarrollar una herramienta basada en modelos de aprendizaje profundo para la detección de hormigas y segmentación de sus caminos con el propósito de proporcionar información que permita respaldar la toma de decisiones para un control más eficiente de esta plaga en el sector agrícola.

1.2.2. Objetivos específicos

Se plantean los siguientes objetivos para lograr el objetivo principal del proyecto:

- Entrenar, evaluar y contrastar modelos de segmentación aplicados a la detección de caminos de hormigas.
- Entrenar, evaluar y contrastar modelos de detección de objetos para identificar hormigas individuales en imágenes.

1.3. Estructura del documento

Este documento está organizado en diferentes capítulos para facilitar su lectura.

En el segundo capítulo, *revisión de antecedentes*, se analizan los trabajos previos más relevantes relacionados con la detección y segmentación de plagas, con énfasis en investigaciones que se centran en hormigas.

Posteriormente, en el tercer capítulo denominado *marco teórico*, se presentan los fundamentos conceptuales necesarios para respaldar el proyecto, incluyendo aspectos esenciales sobre visión por computadora, procesamiento de imágenes, extracción de características y redes neuronales convolucionales (CNN, por su sigla en inglés).

Luego, en el cuarto capítulo, *metodología*, se describe el proceso empleado para la obtención y preparación del conjunto de datos, la implementación específica de los algoritmos de detección y segmentación seleccionados, así como las herramientas, frameworks y tecnologías utilizadas durante la ejecución del proyecto.

El quinto capítulo, *Ants Roads Pipeline (ARP)*, describe en detalle el diseño de la arquitectura propuesto, así como su funcionamiento interno. Se explica cómo se integran los distintos módulos del sistema, desde la segmentación de caminos hasta la detección de hormigas, incluyendo el flujo de datos entre etapas, las decisiones de diseño adoptadas y las características técnicas de cada componente.

En el sexto capítulo, *experimentación y resultados*, se presentan los escenarios experimentales abordados, así como los resultados específicos obtenidos en las etapas de detección y segmentación.

Por último, en el capítulo siete, *conclusiones y trabajos futuros*, se exponen las conclusiones alcanzadas en el proyecto. Además, se presentan posibles líneas de investigación a ser exploradas para continuar profundizando en el estudio del control automatizado de hormigas.

Capítulo 2

Revisión de antecedentes

En esta sección se presenta una síntesis del documento del estado del arte (Nadile, Marr, y Perera, 2025) sobre el monitoreo y control de plagas, con énfasis en las hormigas cortadoras de hojas.

2.1. Hormigas cortadoras de hojas

2.1.1. Biología y organización social

Las hormigas cortadoras de hojas, también conocidas como hormigas jardineras, pertenecen a los géneros *Atta* y *Acromyrmex*, ambos integrantes de la tribu *Attini*. Como su nombre sugiere, estas hormigas cortan hojas que transportan hasta el nido; sin embargo, dichas hojas no forman parte de su dieta, sino que se utilizan para cultivar un hongo. Una característica a destacar de estas especies es su relación simbiótica con hongos del género *Leucoagaricus*, los cuales cultivan en cámaras subterráneas del nido, lo que da origen al término jardineras. Si bien las hormigas obreras adultas complementan su dieta con savia vegetal, este hongo constituye la principal fuente de alimento de la colonia, especialmente para las larvas que se encuentran en etapa de desarrollo.

El nivel de organización requerido para desempeñar tareas como el cultivo del hongo, recolección de hojas, defensa del nido, cuidado de la reina y cría de las larvas, requiere una organización social altamente estructurada. Por este motivo, estas hormigas presentan una organización eusocial compleja, con castas reproductoras (reina y machos alados) y no reproductoras (obreras). En *Atta*, las obreras se dividen en castas definidas según su tamaño (polimorfismo), lo cual determina funciones específicas dentro del nido. En cambio, en *Acromyrmex*, la variabilidad morfológica es menor y la asignación de tareas es más flexible.

En cuanto a la estructura del nido, este puede ser **subterráneo** o **epigeo** (sobresale de la superficie de la tierra), y está compuesto por una red de túneles y cámaras dedicadas al cultivo del hongo, la cría de larvas, alojamiento de la reina o el depósito de residuos.

2.1.2. Comportamiento de forrajeo

El comportamiento de forrajeo en las hormigas cortadoras de hojas responde a una estructura colectiva. Este proceso tiene como objetivo principal recolectar material vegetal para alimentar al hongo simbionte que sustenta a la colonia, y puede dividirse en tres etapas: búsqueda y selección de recursos, manipulación del material vegetal y transporte hacia el nido. Durante estas actividades, las hormigas recorren redes de caminos estables y adaptables a las condiciones del entorno, superando en muchos casos áreas de forrajeo de 1000 m^2.

En cuanto a la actividad diaria, se observa un patrón con mayor intensidad durante las primeras horas de la mañana y al atardecer, cuando las condiciones ambientales son más favorables. Anualmente, la actividad de forrajeo se incrementa en períodos de desarrollo de la colonia, como la producción de nuevas obreras o de individuos alados, destacando los meses de febrero a abril y de septiembre a noviembre en Uruguay.

2.1.3. Métodos de control

Las hormigas cortadoras de hojas, a pesar de su rol ecológico en el reciclado de nutrientes y la modificación de la vegetación, representan una amenaza a los cultivos por su capacidad para defoliar plantas jóvenes y causar daños a su estructura.

Entre los métodos de control más utilizados se encuentran:

- *Insecticidas de contacto*: su eficacia es limitada ya que no siempre alcanzan a la reina y presentan alta toxicidad ambiental.
- *Termonebulización*: permite la distribución de insecticidas de contacto a alta presión dentro del nido, aunque no garantiza la erradicación completa y puede afectar negativamente al suelo.
- *Cebos tóxicos*: son transportados por las obreras al interior del nido. Este método es el más empleado en contextos forestales, aunque su efectividad depende del tamaño del cebo y su ubicación respecto a los caminos de forrajeo.
- *Control biológico*: utiliza hongos entomopatógenos, bacterias y nematodos, que buscan afectar directamente a las hormigas o al hongo simbionte. Sin

embargo, en condiciones de campo, su eficacia se ve limitada debido a los mecanismos de defensa y limpieza de estas especies.

- *Control físico o mecánico*: consiste en remover el nido o instalar barreras, es ambientalmente seguro pero poco viable a gran escala.

Es importante tener en cuenta la planificación de las acciones de control en función del ciclo biológico de las hormigas, aplicándolas preferentemente antes de la implantación del cultivo y en los períodos de desarrollo de larvas y hormigas voladoras, para evitar la formación de nuevas colonias.

2.2. Detección y clasificación de insectos y plagas

La detección automática de plagas en cultivos ha sido ampliamente abordada mediante el uso de visión por computadora combinada con técnicas de aprendizaje profundo. Estas herramientas permiten analizar imágenes de plantas para identificar signos de presencia de plagas, reduciendo así la dependencia de inspecciones manuales.

En esta sección se presentan las principales técnicas utilizadas para la detección y clasificación de plagas, así como las métricas más comunes para evaluar su desempeño.

2.2.1. Técnicas basadas en visión por computadora y aprendizaje profundo

Los trabajos analizados utilizan métricas comunes como AUC (Area Under the Curve), mAP (mean Average Precision) y F1-score para evaluar el desempeño de modelos aplicados a la detección y clasificación de plagas.

En los trabajos analizados que emplean técnicas de detección de objetos en plagas, se obtienen métricas de desempeño variables según el tipo de modelo y el contexto experimental. Entre estos, el estudio *A Systematic Review on Automatic Insect Detection Using Deep Learning* (Teixeira, Ribeiro, Morais, Sousa, y Cunha, 2023) recopila y compara numerosos métodos de detección, clasificándolos según su estructura base y sus resultados. En dicho análisis, se reportan buenos desempeños para YOLOv5 y Faster R-CNN, alcanzando un mAP de 94, 7 % y 94,6 % respectivamente. Cabe destacar que estos modelos se aplicaron sobre conjuntos de datos heterogéneos, y sus resultados demuestran una buena capacidad de detección en escenarios variados. Dentro del mismo trabajo, se destaca una variante denominada TPest-RCNN, basada en Faster

R-CNN, la cual alcanza el mejor desempeño con un mAP de 95,2 %, aunque se evalúa sólo sobre dos clases de insectos.

Por otro lado, el estudio denominado *An Effective Data Augmentation Strategy for CNN-Based Pest Localization and Recognition in the Field* (Li y cols., 2019) propone una estrategia de *data augmentation* combinada con una red CNN y una *Region Proposal Network* (RPN), alcanzando un mAP de 81,4 %. Si bien el resultado es inferior al de las arquitecturas previamente mencionadas, representa una mejora significativa respecto a métodos convencionales como HR-ResNet y FPN, lo que refuerza la utilidad de las técnicas de aumento de datos para abordar la detección en condiciones complejas.

En contraste, el estudio *Deep learning for automated detection of Drosophila suzukii: potential for UAV-based monitoring* (Roosjen, Kellenberger, Kooistra, Green, y Fahrentrapp, 2020) propone un enfoque basado en una ResNet-18 modificada para la detección de la mosca del vinagre de alas manchadas (*Drosophila suzukii*) en imágenes capturadas tanto desde posiciones estáticas como mediante vehículos aéreos no tripulados (UAV). Los resultados obtenidos muestran valores de AUC relativamente bajos, entre 0,506 y 0,603 para imágenes estáticas, y entre 0,086 y 0,284 para imágenes obtenidas con UAVs. Esto evidencia las limitaciones que pueden surgir debido a la baja calidad de las imágenes o a la inestabilidad del dispositivo de captura.

En resumen, los modelos basados en arquitecturas profundas como YOLOv5 y Faster R-CNN demuestran ser los más efectivos para tareas de detección de plagas en condiciones generales. Asimismo, la incorporación de técnicas de aumento de datos y procesamiento multiescala representa un complemento relevante cuando se busca mejorar la precisión en escenarios donde la adquisición de datos resulta limitada o compleja.

2.3. Estudios enfocados en detección de hormigas

Dentro de la bibliografía existente se analizaron varios trabajos enfocados en la detección y el seguimiento de hormigas cortadoras de hojas, debido a su importancia como plaga agrícola. Estos estudios emplean técnicas de visión por computadora y aprendizaje profundo para identificar individuos, ubicar nidos y registrar su actividad.

2.3.1. Técnicas específicas aplicadas a hormigas

En los trabajos centrados específicamente en hormigas cortadoras de hojas, se han desarrollado métodos que combinan visión por computadora y aprendi-

zaje profundo para la detección, segmentación y seguimiento de individuos en distintos entornos.

Uno de los enfoques destacados es AntTracker (2023) (Sabattini, Sturniolo, Bollazzi, y Bugnon, 2023), un pipeline que permite segmentar hormigas individualmente, hacer el seguimiento de su trayectoria y clasificarlas según si transportan carga. El proceso se basa en técnicas de extracción de fondo, segmentación mediante filtros y una CNN para la clasificación. El seguimiento se implementa mediante un algoritmo que asocia regiones a trayectorias de forma continua a lo largo de los fotogramas del video.

Otro estudio relevante, titulado *Accurate detection and tracking of ants in indoor and outdoor environments* (Wu, Cao, y Guo, 2020), propone un modelo basado en Track Before Detect, que incluye una etapa de detección con una RPN conectada a una ResNet-50 para generar regiones de interés, y una segunda etapa de clasificación y refinamiento de esas regiones. Para el seguimiento, se utilizan descriptores de apariencia (generados con ResNet-50) combinados con filtros de Kalman para predecir trayectorias.

En la investigación *Remote detection and measurement of leaf-cutting ant nests using deep learning and and unmanned aerial vehicle* (Dos Santos y cols., 2022), también se explora el uso de imágenes capturadas mediante drones (UAVs) para detectar nidos epígeos de hormigas cortadoras en plantaciones de eucalipto. En este caso, se compara un enfoque clásico, basado en redes MLP con descriptores de color y textura, con modelos YOLOv5 entrenados con subimágenes del ortomosaico. El enfoque basado en YOLOv5 alcanza un AUC de 97,65 %, superando ampliamente al método MLP.

Finalmente, en el marco de un proyecto de grado de la Facultad de Ingeniería de la Universidad de la República, el trabajo titulado *Desarrollo de un oso hormiguero artificial* (Berois, de Oliveira, y Gastelú, 2024), presenta una solución autónoma orientada al monitoreo de hormigas cortadoras mediante visión artificial. El sistema consiste en un robot hexápodo equipado con un modelo YOLOv8 entrenado con imágenes reales, sintéticas y externas, capaz de detectar y seguir hormigas en diversos entornos. Esta propuesta busca ofrecer una alternativa ecológica al uso de pesticidas en el manejo de plagas agrícolas.

2.3.2. Evaluación y limitaciones identificadas

Los métodos aplicados a la detección y seguimiento de hormigas han mostrado buenos resultados en términos de precisión y seguimiento, específicamente en ambientes controlados. En el caso del modelo basado en Track Before Detect, se obtuvo un mAP de 90,5 % en vídeos bajo condiciones controladas y 85,8 % en ambientes no controlados. Además, se registraron valores elevados de precisión en seguimiento, como un mMOTA de 85,8 % y un mMOTP de 83,3 %.

AntTracker, por su parte, logra un F2-score del 83 % en segmentación, un MOTA del 97 % en seguimiento y un F1-score del 82 % en la detección de hormigas que transportaban hojas. Estos resultados evidencian un alto desempeño del pipeline en entornos con fondo uniforme (ver figura 2.1).

Figura 2.1: Ejemplo de captura del dispositivo AntTracker colocado sobre un camino de hormigas con fondo uniforme. Imagen tomada de *AntTracker: A low-cost and efficient computer vision approach to research leaf-cutter ants behavior*, por Sabattini et al. (Sabattini y cols., 2023).

Sin embargo, también se identificaron limitaciones. En condiciones no controladas, factores como la calidad de imagen, la oclusión entre individuos, los movimientos cercanos simultáneos y la falta de contraste dificultan la detección precisa y el seguimiento continuo. En particular, cuando varias hormigas ingresan o salen del campo de visión al mismo tiempo, pueden producirse errores en la asociación de trayectorias.

En cuanto a la detección remota de hormigueros epigeos con UAVs, si bien los modelos YOLOv5 alcanzaron altos valores de AUC en condiciones adecuadas, los métodos clásicos como Multilayer Perceptron (MLP) presentaron numerosos falsos positivos y negativos, reflejando su baja capacidad para adaptarse a variaciones del entorno. Esto sugiere que, aunque los enfoques basados en aprendizaje profundo ofrecen resultados prometedores, su desempeño sigue dependiendo en gran medida de las condiciones del entorno y de la calidad de los datos utilizados para el entrenamiento.

Capítulo 3

Marco teórico

Para el desarrollo del sistema de segmentación de caminos y localización de hormigas en imágenes, se hace uso de modelos avanzados de detección y segmentación de objetos, así como de otras herramientas propias del campo de visión por computadora. En este contexto, resulta pertinente establecer una base teórica sólida que permita comprender los fundamentos conceptuales y técnicos sobre los cuales se construye la solución propuesta.

3.1. Visión por computadora

3.1.1. Definición

Según Szeliski, tanto la visión por computadora como la computación gráfica "modelan cómo se mueven y animan los objetos, cómo la luz se refleja en sus superficies, se dispersa por la atmósfera, se refracta a través de las lentes de la cámara (o de los ojos humanos) y, finalmente, se proyecta sobre un plano de imagen (o curvo)" (Szeliski, 2010, p. 3).

Si bien estos campos utilizan modelos físicos para describir el comportamiento de la luz y los objetos, en el caso de la computación gráfica, estos modelos permiten generar imágenes a partir de una representación del mundo, como ocurre al renderizar escenas o animaciones. En cambio, la visión por computadora es un campo de la inteligencia artificial que realiza el proceso inverso; esta "describe el mundo que vemos en una o varias imágenes y reconstruye sus propiedades, como la forma, la iluminación y las distribuciones de color". (Szeliski, 2010, p. 4). Un ejemplo claro de esto es que, a partir de una fotografía de un paisaje, un sistema de visión por computadora intenta determinar dónde hay aves, cuál es su forma, cómo está iluminada la escena, entre otras cualidades.

Esto deja implícito ciertas interrogantes: ¿Cómo hace una máquina para alcanzar este nivel de entendimiento e interpretación visual? ¿Cómo logra determinar qué objetos están presentes en una imagen, en qué ubicación se encuentran y bajo qué condiciones de iluminación? Para explicar esto, se divide dicho proceso en tres etapas (Sucar y Gómez, 2020):

- Procesamiento de bajo nivel: se recibe la imagen como entrada y se utilizan los píxeles de la misma para extraer información como gradiente, profundidad, colores, texturas, etc.
- Procesamiento de nivel intermedio: agrupa la información generada en el nivel anterior para obtener líneas y regiones.
- Procesamiento de alto nivel: se interpretan los datos obtenidos en los niveles anteriores con el propósito de asignarles un significado o descripción a la imagen (predicados, geometría, entre otros).

Este proceso se ve potenciado al integrarse con técnicas de *deep learning* (aprendizaje profundo), un enfoque de la inteligencia artificial que permite a los computadores aprender de la experiencia estableciendo una jerarquía de conceptos (conceptos complejos se definen utilizando conceptos más simples) (Pateria, Subagdja, Tan, y Quek, 2021; Goodfellow, Bengio, y Courville, 2016). Específicamente, esto es posible mediante la construcción de redes neuronales con múltiples capas, inspiradas en el concepto de aprendizaje de neuronas humanas, las cuales se encuentran interconectadas entre sí (Elgendy, 2020).

3.1.2. Redes neuronales

Una red neuronal [1]se define como un conjunto de *neuronas* organizadas en capas. Esta neurona o neurona artificial es una unidad computacional que calcula su salida de forma matemática a partir de los valores que recibe (ya sea directamente de los datos del problema en la primera capa, o de las neuronas de la capa anterior). Estos valores de entrada conforman un vector de características, donde cada posición del vector se pondera con un peso que representa la importancia de esa característica en el proceso de decisión (la salida de la neurona).

Un caso particular de redes neuronales es la arquitectura feedforward. Esta red neuronal se caracteriza por dividir sus capas jerárquica y funcionalmente de la siguiente manera: una capa de entrada (primera capa), capas ocultas (capas intermedias) y una capa de salida (capa final) (ver figura 3.1.2.1).

[1]El contenido de esta sección se basa en el libro *Deep Learning for Vision Systems* (Elgendy, 2020), salvo que se indique lo contrario.

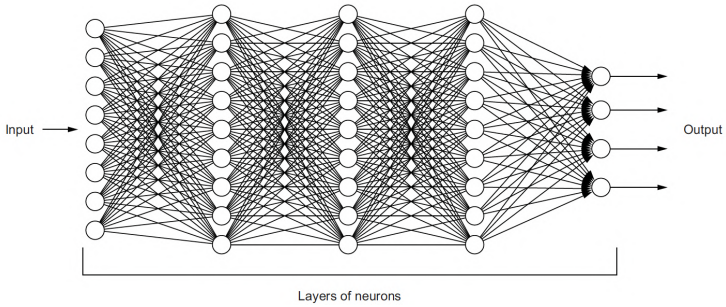

Figura 3.1.2.1: Representación esquemática de una red neuronal artificial (ANN) basada en una arquitectura feedforward. La red está compuesta por una capa de entrada, varias capas intermedias y una capa de salida. Cada círculo representa una neurona, y las líneas indican las conexiones entre las neuronas de una capa a la siguiente. Imagen tomada de *Deep Learning for Vision Systems*, Elgendy (Elgendy, 2020, p. 9).

La profundidad del modelo está dada por la cantidad de capas que tiene esta arquitectura. En una arquitectura feedforward, el flujo de datos avanza de forma secuencial: parte de la capa inicial, luego alimenta las capas ocultas (capa a capa) y finalmente llega a la capa de salida, que es la que genera la predicción del modelo. Durante el entrenamiento, esta predicción se compara con el valor de referencia mediante una función de pérdida, la cual determina qué tan consistente es la salida con el valor esperado.

Este tipo de red neuronal constituye un componente fundamental de las redes neuronales convolucionales (CNN, por su sigla en inglés), que son la base de la mayoría de los modelos de clasificación de imágenes disponibles en la actualidad.

3.1.2.1. Red neuronal convolucional

Una arquitectura CNN es una red neuronal, que se caracteriza por utilizar *capas convolucionales* en lugar de capas tradicionales para la extracción de características. Estas capas están completamente conectadas localmente, y son alimentadas con la imagen en crudo para realizar la extracción de características (mapa de características). La salida de estas capas es luego aplastada para ser enviada a las capas completamente conectadas encargadas de clasificar las características obtenidas. Por último, la red neuronal activa la salida correspondiente a la clase predicha para la imagen (en la figura 3.2, esto sería la clase tres o siete).

Entrando más en detalle en cada una de las secciones de la red, una convolución consiste en aplicar dos funciones para obtener una tercera: en el caso de una CNN, la primera función es la imagen de entrada y la segunda es el filtro

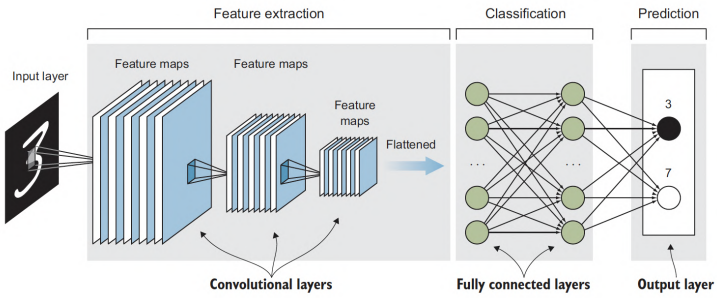

Figura 3.2: Arquitectura de una red neuronal convolucional. Imagen tomada de *Deep Learning for Vision Systems*, (Elgendy, 2020, p. 9).

convolucional.

El filtro de convolución (o kernel) es una matriz que recorre la imagen original, pixel a pixel realizando cálculos matemáticos para producir una nueva imagen *"convolucionada"*, que servirá como entrada a la siguiente capa. Esta matriz de convolución representa los pesos de la red, que se inicializan aleatoriamente y se ajustan durante el entrenamiento.

El tamaño de los filtros depende del nivel de detalle que se quiera capturar (normalmente varía entre 2×2 y 5×5): tamaños grandes pasan por alto detalles finos, mientras que tamaños pequeños los capturan mejor. Además, los filtros contienen los pesos que la red aprende durante el entrenamiento: cuanto mayor sea su tamaño más profunda y compleja será la red, lo que también aumenta el riesgo de sobreajuste y la demanda de recursos computacionales.

Junto con el tamaño de los filtros, la cantidad de pasos (stride) y el relleno (padding) son otros hiperparámetros importantes que controlan la salida de una capa convolucional. Por ejemplo, si se desea procesar todos los píxeles, se tendría que fijar la cantidad de pasos en uno, lo que devuelve la imagen *convolucionada* del mismo tamaño que la imagen original. Si la cantidad de pasos se establece en dos, la salida tendrá aproximadamente la mitad del tamaño, como se muestra en la figura 3.3.

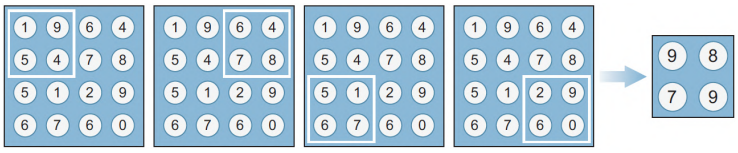

Figura 3.3: Ejemplo de operación de *max pooling* con *stride* de 2. Se observa cómo la ventana de pooling recorre la matriz de características en pasos de 2, reduciendo su tamaño final. Imagen tomada de *Deep Learning for Vision Systems*, (Elgendy, 2020, p. 115).

El relleno, por su parte, se utiliza normalmente para preservar el tamaño espacial del volumen de entrada, de modo que el ancho y la altura de salida sean iguales (véase un ejemplo en la figura 3.4). Esto ayuda a construir redes más profundas, de no ser así las dimensiones se reducirían al avanzar a capas más profundas.

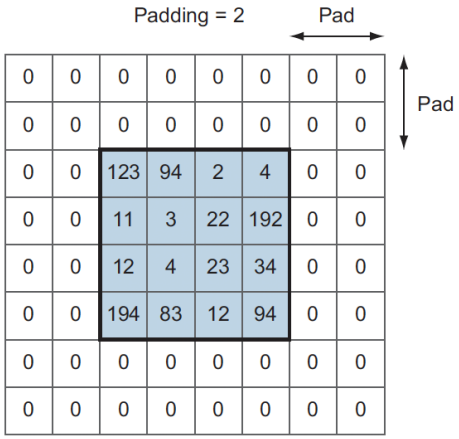

Figura 3.4: Ejemplo de *padding* aplicado a un mapa de características: se añaden filas y columnas con ceros alrededor del mapa original (en azul), aumentando su tamaño en 2 unidades en cada dirección. Imagen tomada de *Deep Learning for Vision Systems*, (Elgendy, 2020, p. 114).

El propósito de utilizar estos dos hiperparámetros es conservar los detalles importantes de la imagen y transferirlos a la siguiente capa, o bien ignorar la información espacial de la imagen para reducir el costo computacional.

A su vez, entre las capas convolucionales se encuentran las capas de agrupación (pooling layers), cuyo objetivo es reducir el muestreo de los mapas de características generados por las capas convolucionales a un menor número de parámetros. Esta reducción se logra aplicando una función estadística de resumen, como puede ser el máximo de los valores (max pooling, ver figura 3.3) o el promedio (average pooling). Estas capas forman parte de la sección convolucional de la red pero no incrementan su profundidad, dado que comprimen las características más importantes en un espacio de menor dimensión.

Finalmente, luego de que la imagen es procesada por las capas convolucionales y de pooling, encargadas de aprender y resumir sus características más relevantes, es el momento de usar estas características para clasificar las imágenes. En este punto entran las capas completamente conectadas, una red neuronal en la que todos los nodos de una capa están conectados a todos los nodos de la capa anterior. Estas corresponden a las redes de arquitectura feedforward mencionadas anteriormente.

3.2. Técnicas de detección de objetos

La detección de objetos en imágenes es una técnica de visión por computadora que consiste en identificar la posición de uno o varios objetos en una imagen y asignarles una categoría. Para ello, los modelos de detección, a partir de la imagen de entrada, generan una o más cajas delimitadoras y asignan probabilidades de que cada uno pertenezca a determinada clase (Elgendy, 2020).

A continuación, se explica el funcionamiento de distintos modelos empleados para la detección descrita anteriormente.

3.2.1. You Only Look Once

You Only Look Once (YOLO) [2], es un modelo de inteligencia artificial diseñado para la detección y segmentación de objetos en imágenes. A diferencia de otros métodos que dividen este proceso en varias fases, YOLO toma la imagen de entrada y, en una sola iteración, predice cajas delimitadoras y calcula las probabilidades de clase para cada objeto. Este enfoque le permite ser extremadamente rápido y eficiente, lo que lo convierte en una excelente opción para aplicaciones en tiempo real.

En el diagrama de la fig 3.5 representa el procesamiento en alto nivel que se le aplica a las imágenes desde su formato original hasta las predicciones finales.

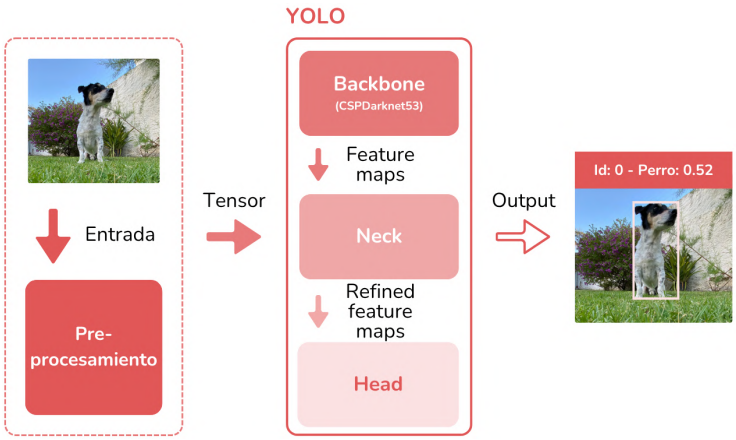

Figura 3.5: Flujo de alto nivel del proceso de detección de objetos en una imagen.

Dado que existen múltiples versiones del modelo, esta explicación se centra

[2]El contenido de esta sección se basa en la documentación técnica oficial de YOLOv8 (Ultralytics, 2024), salvo que se indique lo contrario.

específicamente en la arquitectura de YOLOv8, ya que es la versión más reciente, con mejoras tanto en precisión como en eficiencia en comparación con versiones anteriores (Stereolabs, 2023).

3.2.1.1. Pre procesamiento

Antes de ingresar al modelo, la imagen pasa por una etapa de preprocesamiento. Esto es típicamente realizado por las bibliotecas que utilizan YOLO, como Ultralytics, que primero re-dimensionan la imagen mediante letterbox a un tamaño estándar (640 x 640 píxeles), luego la normalizan y por último la convierten a un tensor numérico (Ultralytics, 2025).

Aunque las imágenes digitales ya están representadas como matrices numéricas desde el momento en que son capturadas o almacenadas, los modelos de inteligencia artificial no las interpretan de la misma forma; es necesario transformar la imagen digital en una estructura de datos (tensor) adecuada para el procesamiento de una red neuronal. Esta representación tensorial es necesaria para que la red neuronal pueda procesar la imagen, dado que contiene la información visual organizada de manera que el modelo pueda identificar patrones y características (LearnPyTorch.io, 2023).

Al finalizar el procesamiento de la imagen, se obtiene un tensor numérico que es utilizado para alimentar al modelo.

3.2.1.2. Arquitectura

En cuanto a la arquitectura de YOLO, esta consta de tres componentes principales: Backbone, Neck y Head.

Backbone

El Backbone es la primera etapa del modelo. Se encarga de procesar el tensor generado tras el preprocesamiento de la imagen (Ultralytics Team, 2025a). Su objetivo es extraer características jerárquicas y relevantes de la imagen. Estas características son esenciales para que el modelo pueda identificar objetos en etapas posteriores.

Particularmente, YOLOv8 utiliza como backbone la CNN CSPDarknet53, la cual aplica operaciones convolucionales para generar mapas de características (features maps). Estos mapas incluyen desde patrones básicos, como bordes y texturas, hasta características complejas como formas y relaciones espaciales. Un aspecto a destacar de esta red es la implementación de conexiones Cross-Stage Partial (CSP), las cuales permiten "mejorar el flujo de información entre las diferentes etapas de la red" (Ultralytics, 2024), incrementando así la exactitud del modelo.

Una vez obtenidos los mapas de características, estos se propagan hacia la siguiente etapa del modelo, denominada Neck.

Neck

El Neck refina las características obtenidas para mejorar la detección. Para ello, esta etapa emplea una estructura basada en una PANet (Path Aggregation Network), una red neuronal que captura información en múltiples escalas al integrar los mapas de características provenientes de diferentes etapas del backbone (Ultralytics Team, 2025b).

Head

Por último, el Head es la etapa final del modelo. Su principal función es transformar los mapas de características generados por el Backbone, y refinados por el Neck en predicciones concretas: las coordenadas de los objetos detectados, las clases a las que pertenecen y su nivel de confianza.

3.2.2. Faster R-CNN

Faster R-CNN[3] es un modelo de detección de objetos en imágenes basado en un proceso de dos etapas: la primera realiza un relevamiento de la imagen para identificar regiones de interés (RoI), y la segunda realiza la detección y clasificación de los elementos presentes en dichas regiones.

3.2.2.1. Arquitectura

Este modelo se construye sobre la arquitectura de Fast R-CNN, la cual, a partir de una imagen y un conjunto de regiones de interés, es capaz de detectar objetos ubicados dentro de estas regiones. Faster R-CNN amplía esta arquitectura al agregar al inicio del procesamiento de la imagen una Region Proposal Network (RPN), con el fin de generar regiones de interés (RoIs) en las que es probable encontrar los objetos a detectar. Esta comparte capas convolucionales con Fast R-CNN, conocidas como Backbone, como se explica a continuación.

Backbone

El procesamiento de la imagen comienza pasando por un conjunto de capas convolucionales, conocidas como backbone o modelo base, encargadas de generar un mapa de características. Esta salida es utilizada tanto por la RPN como por el módulo de detección de Fast R-CNN. Varios modelos han sido utilizados como backbone en distintas investigaciones, entre ellos se pueden destacar MobileNet,

[3]El contenido de esta sección se basa en el artículo *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* (Ren, He, Girshick, y Sun, 2015), salvo que se indique lo contrario.

ResNet y SqueezeNet por haber obtenido buenos resultados (véase 3.4).

Region Proposal Network

La primera etapa comienza con la RPN, que opera directamente sobre el mapa de características generado por el backbone, y se encarga de generar las RoIs necesarias para el módulo Fast R-CNN. Además, esta red asigna indicadores a las regiones que señalan la probabilidad de que las mismas contengan un objeto o formen parte del fondo de la imagen.

Una particularidad de la RPN, cuando se utiliza como parte de la arquitectura Faster R-CNN, es que comparte capas convolucionales con el módulo de detección de objetos Fast R-CNN. De esta forma, se reducen los tiempos de cálculo, ya que los mapas de características se calculan una sola vez y se utilizan en dos puntos distintos del proceso.

Fast R-CNN

Luego del procesamiento de la imagen realizado por las capas convolucionales, y de la generación de las RoIs por RPN, comienza la segunda etapa del modelo al ingresar los datos previamente calculados al módulo Fast R-CNN. Este módulo procesa las RoIs con una capa llamada RoI pooling, la cual convierte todas las regiones en mapas del mismo tamaño. Finalmente, los mapas pasan por redes neuronales completamente conectadas y por dos ramas: una de clasificación para determinar la categoría del objeto detectado, y otra para realizar la regresión de la caja, es decir, para predecir y ajustar sus coordenadas. El procedimiento previamente descrito puede ser observado en la figura 3.6.

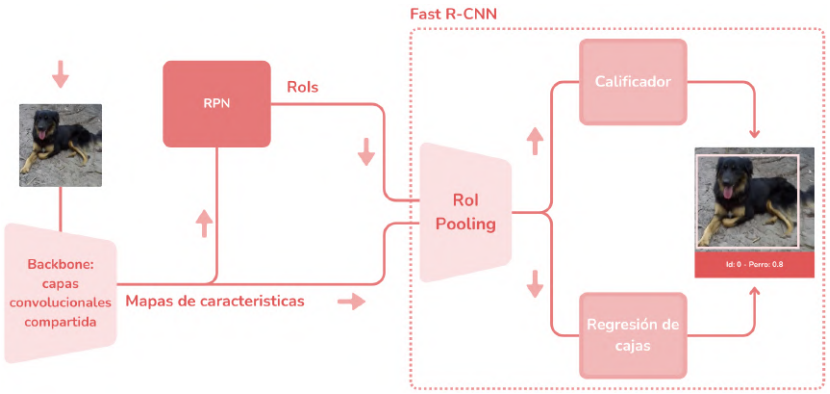

Figura 3.6: Flujo general de Faster R-CNN, que combina la propuesta de regiones mediante RPN con el procesamiento y clasificación realizado por Fast R-CNN.

3.3. Técnicas de segmentación de instancias

La segmentación de instancias en una imagen es una técnica de visión por computadora que permite localizar objetos mediante máscaras de píxeles que abarcan el área que estos ocupan. Para ello, se asigna a cada píxel un objeto específico, lo que permite determinar las dimensiones y la cantidad de instancias que hay en una imagen. (Szeliski, 2022).

A continuación, se explica el funcionamiento de distintos modelos empleados para la segmentación descrita anteriormente.

3.3.1. Mask R-CNN

Mask R-CNN[4] es un modelo utilizado para la segmentación de instancias en imágenes. Utiliza el modelo de dos etapas explicado en la sección Faster R-CNN , donde primero se identifican regiones de interés (RoIs) y luego se realiza la detección y clasificación de los elementos presentes en dichas regiones.

3.3.1.1. Arquitectura

Mask R-CNN es una extensión del modelo Faster R-CNN, pero con dos modificaciones clave. Por un lado, la capa de RoI pooling de la RPN es reemplazada por una nueva capa llamada RoI align. Por otro lado, se modifica el head del modelo para permitir el cálculo de la máscara. Estas diferencias, y la evolución general de la arquitectura R-CNN, se ilustran en la figura 3.7.

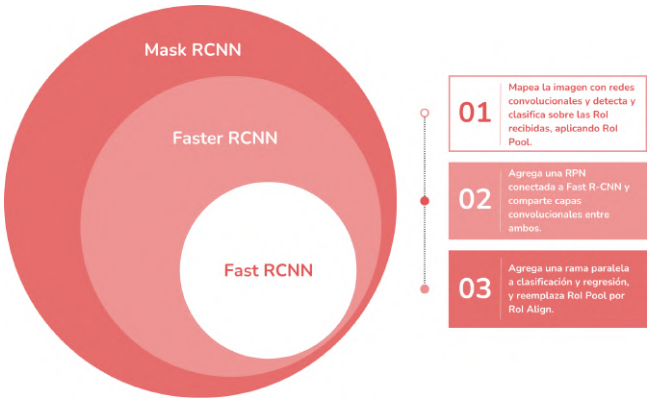

Figura 3.7: Evolución de la familia R-CNN.

[4]El contenido de esta sección se basa en el artículo *Mask R-CNN* (He, Gkioxari, Dollár, y Girshick, 2017), salvo que se indique lo contrario.

Region Proposal Network

Una limitación importante de la capa RoI pooling utilizada en la RPN es que no mantiene alineada la información de la RoI sobre el mapa de características. Esto se vuelve evidente cuando los bordes de la región no coinciden con los límites de la celda del mapa (véase figura 3.8). En estos casos, RoI pooling realiza un alineamiento de la RoI con las celdas y, por ende, desplaza la zona de interés en la imagen. Aunque esto no es tan relevante en el caso de detección, sí lo es en segmentación, debido a que exige una precisión píxel a píxel, a diferencia de la primera.

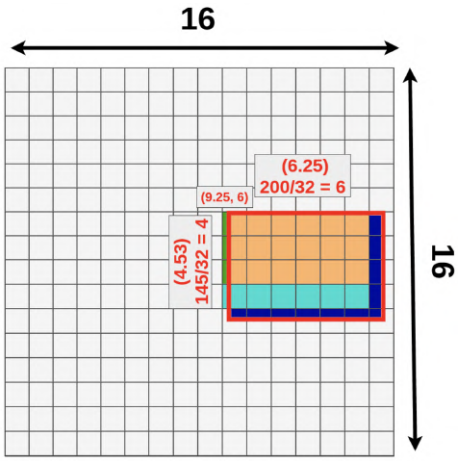

Figura 3.8: Ejemplo visual de cómo *RoI Align* evita pérdidas por redondeo presentes en *RoI Pooling* , preservando mejor la información espacial. Imagen tomada de (Erdem, 2020).

Para resolver este problema, los creadores de Mask R-CNN introdujeron la capa RoI Align, que, en lugar de alinearse respecto a las celdas para obtener un mapa de características, realiza un cálculo respecto a los valores de estas utilizando interpolación bilineal. El proceso consiste en dividir la RoI en un conjunto de zonas de igual tamaño, luego se determinan cuatro puntos internos en cada una de las regiones, de forma que se asigna un valor a cada uno con interpolación bilineal. Finalmente, se aplica el método de pooling seleccionado (max o average) utilizando los valores calculados.

Esto mejora los resultados obtenidos en comparación con el uso de RoI Pooling, que no solo descarta parte de su información original, sino que también agrega información irrelevante de celdas externas.

Head

El cabezal de Faster R-CNN está formado por dos ramas que trabajan en

paralelo: una se encarga de la clasificación en clases de la RoI, y otra se encarga del refinamiento de la caja donde se encuentra el objeto. A su vez, Mask R-CNN agrega una tercera rama que opera en paralelo para calcular la máscara binaria del objeto dentro de la RoI.

Esta rama, en lugar de ser una red completamente conectada, es un conjunto de convoluciones que genera una salida de $n \times n$ a partir de una entrada del mismo tamaño. De esta forma, se conserva la espacialidad de la entrada, a diferencia de las otras ramas que colapsan el resultado en un vector. Otro aspecto a destacar es el hecho de que esta rama trabaje en paralelo, lo cual permite a Mask R-CNN mantener una precisión similar a la de Faster R-CNN en detección y clasificación de objetos.

3.3.2. SAM-CLIP

A medida que los modelos de visión por computadora han avanzado, surgen arquitecturas especializadas con capacidades complementarias. CLIP, por ejemplo, se utiliza para comprensión semántica, mientras que SAM se especializa en la segmentación de objetos. Frente a este escenario, se introduce un modelo unificado que permite combinar de manera eficiente ambos enfoques. En este contexto, se presenta a SAM-CLIP como una arquitectura unificada que reúne las capacidades de SAM y CLIP en un mismo modelo, facilitando su implementación en dispositivos con recursos limitados y ampliando su aplicabilidad en tareas de segmentación guiadas por texto (Wang y cols., 2023).

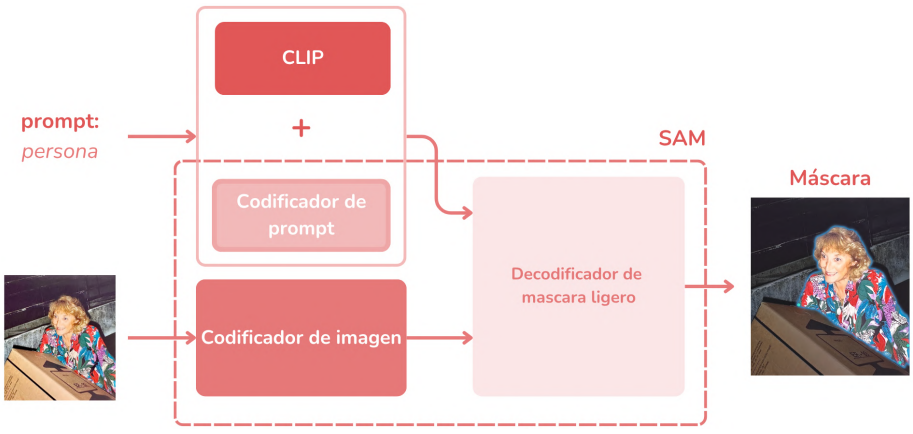

Figura 3.9: Diagrama de alto nivel que ilustra el funcionamiento de SAM con CLIP integrado, inspirado en una figura del artículo *Segment Anything* de Kirillov et al. (Kirillov y cols., 2023).

3.3.2.1. Contrastive Language-Image Pre-training

Contrastive Language-Image Pre-training (CLIP[5]) es un modelo desarrollado por OpenAI que utiliza una estrategia de aprendizaje contrastivo para alinear representaciones visuales y textuales en un espacio de características compartido. El entrenamiento contrastivo se basa en asociar imágenes con tokens textuales, maximizando la similitud entre pares correctos de imágenes y texto, y minimizando los pares que no coinciden.

Para lograr esto, CLIP emplea dos componentes principales: un codificador de imágenes y un codificador de texto. Ambos generan representaciones vectoriales en un espacio de características común, con el objetivo de comparar directamente imágenes y descripciones textuales.

El codificador de imágenes puede estar basado en diferentes arquitecturas, entre ellas se encuentran las siguientes:

- ResNet: redes neuronales convolucionales utilizadas en visión por computadora. En este caso, se emplean variantes como Resnet-50, Resnet-101 y otras versiones ampliadas que aumentan la profundidad, el ancho y la resolución de entrada para mejorar el rendimiento del modelo.
- Vision Transformer (ViT): arquitectura basada en transformadores que divide imágenes en parches. Cada parche es procesado individualmente para obtener vectores de características normalizados, y les agrega codificación espacial para conservar la información posicional de los parches. Luego, se procesan los vectores mediante capas transformadoras para obtener el vector de características final de la imagen.

El codificador de texto, por su parte, también está basado en un transformador, similar a los utilizados en los modelos de lenguaje como GPT. Este codificador toma la secuencia de texto de entrada, la divide en subpalabras según un vocabulario de 49.152 tokens y las procesa para generar una representación que combina el significado de toda la secuencia. Los tokens constituyen una representación interna de las palabras del texto, las cuales refieren a los objetos o conceptos que aparecen en la imagen. Como resultado, se obtiene una representación vectorial en el mismo espacio de características compartido con el codificador de imágenes.

Las representaciones generadas por los codificadores se proyectan en un espacio de características compartido mediante una capa lineal (proyección lineal), y se le aplica una normalización L2 con el fin de facilitar el cálculo de similitudes.

Una de las características más importantes de este modelo es la transferencia sin entrenamiento (Zero-shot). Esto significa que puede clasificar imágenes sin

[5]El contenido de esta sección se basa en el artículo *Learning Transferable Visual Models From Natural Language Supervision* (Radford y cols., 2021), salvo que se indique lo contrario.

requerir un fine tuning (costoso en términos de datos o cómputo), dado que el modelo generaliza suficientemente como para realizar clasificaciones de imágenes sin necesidad de un entrenamiento específico.

3.3.2.2. Segment Anything Model

Segment Anything Model (SAM[6]) es un modelo de segmentación desarrollado por Meta AI que permite generar máscaras a partir de diferentes tipos de prompts (como puntos, cajas, etc.). Para su entrenamiento, se construyó el conjunto de datos SA-1B (Segment Anything 1-Billion Mask Dataset), que contiene más de un billón de máscaras generadas semiautomáticamente.

En el artículo oficial, se identifican tres bloques funcionales que componen la arquitectura de SAM:

- Codificador de imagen: utiliza un Vision Transformer (ViT) pre entrenado con el método Masked Autoencoder (MAE) para obtener un vector de características de la imagen. Este vector es calculado una única vez por imagen y se reutiliza en combinación con los diferentes tipos de prompts, logrando así mayor eficiencia.
- Codificador de prompts: permite identificar que se desea segmentar dependiendo de los diferentes tipos de entrada disponibles. SAM admite múltiples tipos de prompts, incluyendo puntos (posiciones espaciales), cajas delimitadoras (codificadas por los puntos de las esquinas) o máscaras (procesadas por convoluciones). Opcionalmente, es posible incorporar prompts textuales mediante el uso de CLIP, el cual genera el vector de características correspondiente a partir del texto.
- Decodificador de máscaras: combina las representaciones generadas por el codificador de imagen y el codificador de prompts para obtener una o más máscaras de segmentación válidas, cada una con un puntaje de confianza asociado. Ante posibles ambigüedades derivadas de los vectores de características de los prompts, el decodificador puede generar hasta tres máscaras por entrada, seleccionando la más adecuada en función del puntaje de confianza, que mide la compatibilidad entre las características visuales y la información proporcionada por el prompt.

Una vez identificados los codificadores que forman parte de la arquitectura del modelo, es posible resumir el flujo de funcionamiento de SAM de la siguiente manera. Primero, el codificador de imagen procesa la imagen de entrada para generar un vector de características. Simultáneamente, el codificador de prompts traduce las indicaciones del usuario (puntos, cajas, etc.) a un vector

[6]El contenido de esta sección se basa en el artículo *Segment Anything* (Kirillov y cols., 2023), salvo que se indique lo contrario.

de características correspondiente. Luego, ambos vectores se combinan dentro del decodificador de máscaras, donde interactúan a través de mecanismos de atención para generar las máscaras de segmentación.

Gracias a esta arquitectura, SAM está diseñado para la interacción con usuarios en tiempo real. Este modelo es capaz de generar resultados en aproximadamente 50 milisegundos en un navegador, y se integra fácilmente a sistemas interactivos y a diferentes tipos de aplicaciones de visión por computadora.

3.4. Backbones en modelos de detección y segmentación

Como se menciona anteriormente, los modelos de detección y segmentación utilizan como estructura principal ciertas arquitecturas de redes convolucionales, conocidas como backbones. Estas arquitecturas extraen las características más relevantes de la imagen, que las etapas posteriores del modelo emplean para localizar, clasificar o segmentar los objetos presentes. En esta sección se describen los backbones que se consideran en el proceso de experimentación.

3.4.1. MobileNet

Los modelos MobileNet[7] se desarrollaron con el propósito de ejecutarse en sistemas embebidos o dispositivos móviles. Para lograrlo, los diseñadores de esta arquitectura propusieron modelos compactos que consumen pocos recursos y logran un desempeño comparable a redes más demandantes, como AlexNet, pero con menor latencia. Este enfoque es diferente al que se suele usar, dado que generalmente se opta por complejizar y usar modelos más profundos que utilizan más recursos.

3.4.1.1. Arquitectura

La arquitectura de este modelo se basa en el concepto de convoluciones separables en profundidad. En este tipo de operación, el modelo aplica un filtro convolucional a cada canal de entrada de forma independiente y luego apila los resultados, manteniendo separados los canales durante el procesamiento inicial. A continuación, utiliza un filtro 1×1 (punto por punto) que combina las salidas de los distintos canales para obtener una representación unificada. Este proceso de dos etapas permite reducir significativamente la cantidad de cálculos y

[7]Esta sección se basa en el artículo *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* (Howard y cols., 2017), salvo que se indique lo contrario.

parámetros necesarios para procesar las imágenes, lo que disminuye la latencia y el tamaño del modelo.

MobileNet utiliza este tipo de convolución en la mayoría de sus capas, con filtros de 3×3. Solo la primera capa realiza una convolución completa, mientras que la última corresponde a una red totalmente conectada que finaliza en una SoftMax utilizada para la clasificación.

3.4.1.2. MobileNetV2

Tal como su nombre sugiere, MobileNetV2 introduce mejoras respecto a la versión anterior, como se explica en el artículo *MobileNetV2: Inverted Residuals and Linear Bottlenecks* de Sandler et al. (Sandler, Howard, Zhu, Zhmoginov, y Chen, 2018). Estas mejoras buscan la reducción de recursos computacionales, manteniendo o incluso mejorando el rendimiento del modelo.

Respecto a la arquitectura, los desarrolladores proponen el uso de un nuevo bloque como elemento básico; este elemento se denomina bloque residual invertido con cuello de botella lineal. En arquitecturas como ResNet, la estructura de bloques residuales consiste en reducir la dimensionalidad de la entrada usando convoluciones 1×1, luego se procesan los datos, se expanden nuevamente y, por último, se suman a la entrada. En cambio, MobileNetV2 propone un enfoque opuesto para estos bloques (de aquí deriva el nombre del bloque): primero expande la entrada con convoluciones 1×1, luego se procesa la información y, finalmente, reduce y suma la entrada inicial a los datos obtenidos.

Este cambio tiene como objetivo evitar la pérdida de información que puede producirse al aplicar funciones como ReLU después de reducir la cantidad de canales. Al observar el funcionamiento de las redes residuales, los investigadores advierten que reducir los canales antes del procesamiento y luego expandirlos puede provocar pérdida de información. En cambio, al realizar primero la expansión, la información se reparte en un espacio de mayor dimensionalidad, lo cual facilita la preservación de características importantes, incluso cuando algunas activaciones son anuladas por la función no lineal.

3.4.2. SqueezeNet

SqueezeNet[8] es una CNN que fue desarrollada con el objetivo de alcanzar un desempeño similar a otros modelos como Alex-Net, con el desafío de obtener una estructura que permita un uso más eficiente de la comunicación y los recursos. Como resultado, se obtuvo un modelo comparable en rendimiento, pero con una cantidad de parámetros cincuenta veces menor.

[8]El contenido de esta sección se basa en el artículo de *SqueezeNet* (Iandola y cols., 2016), salvo que se indique lo contrario.

3.4.2.1. Arquitectura

A partir del objetivo definido, los autores adoptaron una serie de decisiones de diseño que resultaron en la creación de un nuevo módulo.

Decisiones de diseño

Para lograr el objetivo, utilizan tres estrategias primordiales al diseñar el modelo:

1. Se remplazan la mayor parte de los filtros de 3×3 por filtros de 1×1, ya que esto reduce de 9 a 1 la cantidad de parámetros necesarios por filtro.

2. Se reducen los canales de entrada. De la misma forma que reducir el número de filtros 3×3 disminuye la cantidad de parámetros, decrementar los canales de entrada también reduce el tamaño del modelo. Para esto se usaron capas llamadas Squeeze, concepto profundizado en la próxima sección.

3. Se atrasa la reducción del tamaño de los datos con el fin de que las capas convolucionales posean mapas de activación más grandes, lo cual le permite al modelo conservar información a lo largo del procesamiento de la imagen por más tiempo.

Módulos Fire

En base a las decisiones de diseño planteadas, se definen los módulos Fire, compuestos por una capa convolucional squeeze conectada a una capa de expansión que combina filtros convolucionales de 1×1 y 3×3. Las capas squeeze están compuestas únicamente por filtros 1×1 que disminuyen el número de canales de entrada al combinarlos en un menor número de canales. En esta estructura, se establece como condición que la cantidad de filtros de la capa squeeze es menor que la de expansión.

Estas decisiones responden a las estrategias de diseño mencionadas anteriormente: el uso de filtros 1×1 corresponde a la aplicación de la estrategia número 1, ya que permite reducir la cantidad de parámetros. Por otra parte, el uso de las capas squeeze, junto con la condición previamente establecida al inicio de los módulos, se relaciona con la estrategia número dos, dado que reduce el número de entradas a los filtros 3×3.

Estructura

A partir de los módulos Fire y las decisiones de diseño previamente descritas, SqueezeNet organiza su estructura de la siguiente forma (ver figura 3.10): comienza con una capa convolucional, continúa el procesamiento con ocho módulos fire en secuencia, y finaliza con una última capa convolucional. Este modelo

utiliza el método de max pooling para reducir la información luego de la ejecución de las siguientes capas: la primera capa convolucional, el tercer y séptimo módulo fire, y la última capa convolucional. Esta aplicación en etapas relativamente tardías de max pooling tiene como objetivo aplicar el punto tres de las estrategias de diseño mencionadas anteriormente.

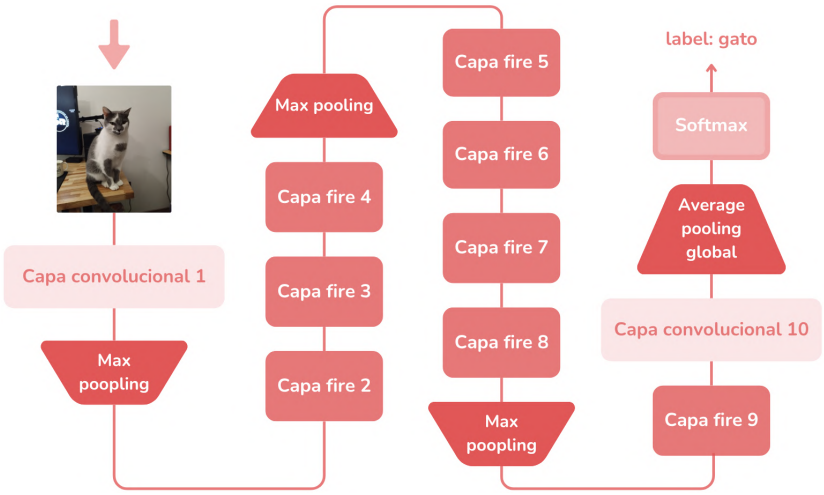

Figura 3.10: Ejemplo del flujo de procesamiento en SqueezeNet, desde la imagen de entrada hasta la clasificación.

3.4.3. Residual Networks

Las redes convolucionales profundas han sido fundamentales en la resolución de diversos problemas. En particular, los modelos más profundos han obtenido mejores resultados en distintas tareas. Sin embargo, el uso de una mayor cantidad de capas no garantiza obtener un buen resultado, dado que puede afectar el rendimiento del modelo. Un posible problema es la saturación de la exactitud, donde el desempeño alcanza un límite o incluso puede degradarse cuando incrementa la complejidad del modelo (número de capas) a medida que converge.

Para abordar este problema surgen las Residual Networks (ResNets[9]), redes convolucionales que utilizan un framework de aprendizaje residual desarrollado con el objetivo de permitir la generación de modelos más profundos. Estas redes evitan el problema de saturación de la exactitud al no aproximar directamente la función objetivo, sino una función residual que se construye a partir de ella.

El funcionamiento del aprendizaje residual se puede entender a partir de cómo las ResNet modelan esta función. Tomando x como el valor de entrada a

[9]Esta sección se basa en el artículo *Deep Residual Learning for Image Recognition* (He, Zhang, Ren, y Sun, 2015), salvo que se indique lo contrario.

un conjunto de capas, en lugar de aproximar una función objetivo $H(x)$ durante el entrenamiento, el modelo se aproxima mediante una función residual definida como $F(x) = H(x) - x$. El razonamiento detrás de esta formulación es que resulta más sencillo aproximar una función residual que aproximar una función sin información inicial, especialmente si involucra transformaciones no lineales, como ocurre con $H(x)$.

Para obtener la función objetivo $H(x)$ a partir de la función residual $F(x)$, se debe sumar x a la salida de $F(x)$ al final del bloque de capas. Esto se logra alimentando la salida con el valor x de entrada utilizado al principio del conjunto de capas, sumando sus valores elemento a elemento. Si la salida y la entrada coinciden en dimensiones, el procedimiento anteriormente detallado puede realizarse de forma directa; en caso contrario se debe utilizar una convolución para igualar las dimensiones. A este paso dentro del modelo, en el que se suma el valor x a la salida de las capas, se le conoce como conexiones de atajo (shortcut connections, véase figura 3.11), un concepto base del framework utilizado por las ResNets.

Figura 3.11: Representación de un shortcut connection en una ResNet.

3.5. Frameworks y utilidades

En las secciones anteriores se presentan los conceptos fundamentales de visión por computadora, junto con las principales arquitecturas de redes neuronales utilizadas para tareas de detección y segmentación. En esta sección se describen frameworks, componentes adicionales y utilidades que complementan dichas arquitecturas, mejoran su desempeño, facilitan su entrenamiento o permiten adaptarlas a distintas necesidades.

3.5.1. Feature Pyramid Network

Feature Pyramid Network (FPN)[10] es una arquitectura que facilita a los modelos el reconocimiento de objetos de diferentes tamaños en la imagen a procesar. Su diseño toma como base la estructura de las redes convolucionales profundas, que a medida que operan sobre la imagen, generan mapas de características de distintos tamaños.

[10]El contenido de esta sección se basa en el artículo *Feature Pyramid Networks for Object Detection* (Lin y cols., 2016), salvo que se indique lo contrario.

Para lograr esta capacidad, FPN aprovecha no solo la estructura jerárquica de estas redes, sino que también las capas finales, que contienen las características de mayor riqueza semántica. Estas son combinadas con las salidas de las capas anteriores para generar mapas de diferentes escalas. De esta forma, se evita procesar los mapas puros de las primeras etapas, que aún poseen características de bajo nivel en sus datos, debido a que el filtrado no es completo en esas etapas.

El proceso concreto mediante el cual FPN construye estos mapas multiescala se basa en dividir la red en etapas. Cada etapa agrupa, de forma ordenada, las capas que generan mapas de características del mismo tamaño. A su vez, de cada etapa se toma el mapa de la última capa, ya que este contiene las características de mayor nivel. A continuación, se combina el último mapa de características con el correspondiente a la etapa anterior, redimensionando el mapa más profundo mediante upsampling y realizando una suma elemento a elemento entre ambos. Este procedimiento se repite de forma sucesiva utilizando el resultado de esta combinación con los de la etapa anterior y así sucesivamente.

Esta forma de fusionar los mapas permite combinar las características con mayor significado semántico junto a las de menor nivel, pero más precisas en términos de localización, ya que han sufrido menos compresión al pasar por menos capas convolucionales.

3.5.2. Low-Rank Adaptation of Large Language Models

Low-Rank Adaptation of Large Language Models (LoRA-Rank)[11] es una técnica de ajuste fino diseñada para reducir la cantidad de parámetros necesarios al adaptar un modelo grande a una tarea específica. En lugar de modificar directamente los parámetros del modelo original, LoRA-Rank introduce un conjunto de parámetros adicionales, significativamente más pequeños, con los que se entrena mientras los pesos originales (que suelen encontrarse en el orden de miles o millones) permanecen congelados.

Esta técnica tiene múltiples ventajas, entre ellas se encuentran:

- Reduce la cantidad de parámetros en varios órdenes de magnitud en comparación a si se tuviera que ajustar el modelo original.
- Por lo anterior, al entrenar menos parámetros, se reduce la necesidad de recursos computacionales así como los requerimientos de memoria. Que deriva en una convergencia más rápida durante el entrenamiento.
- Los parámetros adicionales introducidos por esta técnica no incrementan la latencia en cuanto a la inferencia original, lo que permite mantener la

[11] El contenido de esta sección se basa en el artículo *Low-Rank Adaptation for Foundation Models: A Comprehensive Review* (Yang y cols., 2025), salvo que se indique lo contrario.

eficiencia.

- Al mantener congelados los pesos originales, se evita que el modelo pierda el conocimiento aprendido previamente al incorporar nueva información durante el entrenamiento.

Por todo lo anterior, LoRA-Rank ofrece una forma práctica y eficiente de adaptar modelos de gran tamaño (como SAM) a nuevas tareas, sin sacrificar su rendimiento y con una reducción en los costos computacionales.

3.5.3. Sliding Aided Hyper Inference and Fine-Tuning

Como mencionan los desarrolladores de este framework, la mayoría de los modelos diseñados para la detección o segmentación de objetos asumen que los elementos a detectar o segmentar ocupan un gran porcentaje de la imagen. Sin embargo, este supuesto no siempre se cumple, como ocurre al intentar detectar hormigas en un camino, o autos en fotos satelitales.

Para abordar este tipo de problemas, donde los objetos son pequeños y ocupan sólo una fracción de la imagen, se introdujo Sliding Aided Hyper Inference and Fine-Tuning (SAHIF [12]), un framework que permite tanto el entrenamiento de modelos para la detección de objetos pequeños, como la inferencia sobre imagen sin necesidad de una especialización previa del modelo. A lo que concierne este informe, sólo se hace foco en la etapa de inferencia conocida como SAHI.

SAHI, como su nombre lo indica, es una herramienta que permite realizar inferencias. En particular, mejora este proceso bajo las condiciones mencionadas anteriormente. Para lograrlo, divide la imagen en una grilla y realiza la inferencia sobre cada una de las celdas generadas. A continuación, se detalla el flujo de ejecución paso a paso de SAHI:

1. Cada celda es reescalada, respetando su relación de aspecto, y sobre cada sección se realiza la detección de los objetos con el modelo seleccionado.
2. Se divide la imagen en una grilla, según un ancho y alto definidos.
3. Se escalan las detecciones y se ubican en la imagen de entrada.
4. Se aplica un Non-Maximum Suppression (NMS) para eliminar las inferencias repetidas que se superponen.
5. Se filtran aquellos resultados que no alcancen un mínimo de certeza en la predicción.

[12] El contenido de esta sección se basa en el artículo *Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection* (Akyon, Altinuc, y Temizel, 2022), salvo que se indique lo contrario.

Opcionalmente, se puede realizar una inferencia sobre la imagen entera buscando cubrir aquellos objetos que son más grandes o que no entran en las celdas de la grilla. Estas detecciones se juntan con las de las celdas (una vez reescaladas) y se las procesa en conjunto con el NMS y el filtro de certeza.

Cuando se divide la imagen para aplicar SAHI, esta se particiona en múltiples regiones rectangulares, llamadas celdas, que conforman una grilla sobre la imagen original. El tamaño de cada celda se define en función de un ancho y un alto específicos, los cuales pueden ser calculados automáticamente por la herramienta. Además, se incorpora un factor de superposición. Este factor determina cuánto debe crecer una celda para superponerse con sus vecinas y alcanzar el porcentaje de superposición indicado. Según los experimentos realizados por los desarrolladores, esta estrategia mejoró la detección de objetos de tamaño mediano a pequeño, lo que se traduce en la disminución de la probabilidad de tener objetos repartidos entre dos secciones.

Respecto a los resultados obtenidos, se reporta una mejora de hasta un 6,8 % en el Average Precision (AP) al aplicar SAHI sobre los conjuntos de datos VisDrone y xView, para un conjunto determinado de detectores. Incluso se observa una mejora superior al utilizar el método de especialización de modelos SAHF: con una superposición del 25 %, el AP aumenta en un 2.9 %. Esta aproximación al problema obtiene mejores resultados sin aumentar el uso de memoria del sistema, ya que opera sobre secciones de la imagen que son menores que la imagen completa. No obstante, esta mejora provoca un aumento del tiempo de inferencia, debido a los pasos extras que requiere el proceso.

3.6. Entrenamiento y evaluación de modelos

Para entrenar un modelo o una red neuronal se divide el conjunto de datos etiquetados en tres subconjuntos: *entrenamiento*, *validación* y *prueba*. Cada subconjunto cumple una función específica:

- **Entrenamiento:** se utiliza para ajustar los parámetros del modelo.
- **Validación:** contiene ejemplos que el modelo no utiliza durante el entrenamiento. Este conjunto permite al modelo realizar controles al final de cada época al calcular el error entre las predicciones y la etiqueta para afinar los hiperparámetros.
- **Prueba:** se utiliza cuando finaliza el entrenamiento para medir la capacidad de generalización del modelo y obtener métricas que permitan evaluar su desempeño.

Es fundamental *"nunca usar los datos de prueba durante el entrenamiento"* (Elgendy, 2020, p. 152), de lo contrario, la evaluación no será confiable. Además,

es importante que cada subconjunto contenga muestras variadas y equilibradas de todas las clases. Si un conjunto queda dominado por una sola clase, el modelo podría sobreajustarse a esa clase y afectar a las demás.

En la práctica, para materializar este proceso de entrenamiento y validación, un algoritmo de aprendizaje supervisado recibe un conjunto de pares entrada-salida etiquetados y ajusta los parámetros del modelo para que las predicciones se aproximen lo más posible a las salidas esperadas. El ajuste se realiza minimizando una función de pérdida que cuantifica la diferencia entre las predicciones y las etiquetas verdaderas en el conjunto de entrenamiento. Durante la fase de entrenamiento, el algoritmo recorre repetidamente los datos y actualiza los parámetros con el objetivo de reducir dicha función; este proceso de optimización permite que el modelo aprenda representaciones que generalicen más allá de los ejemplos vistos (Szeliski, 2010). Una vez completado este proceso, los parámetros aprendidos se mantienen fijos ("congelados") y la red ejecuta solo la propagación hacia delante de datos de entrada no vistos previamente. Si el modelo se ajusta correctamente (sin sobreajuste ni subajuste), las salidas producidas por la capa final tienden a coincidir, o a tener un error muy pequeño, entre las predicciones y los valores deseados (Szeliski, 2010).

3.6.1. Métricas de evaluación

Siempre que se ajusta un modelo de aprendizaje automático, es necesario contar con métricas que permitan evaluar su desempeño tanto durante el entrenamiento como al finalizar su ejecución. Estas métricas son esenciales para monitorear el aprendizaje y guiar el ajuste de hiperparámetros, la selección del modelo más óptimo y otras decisiones que influyen en la mejora del rendimiento.

A continuación, se presentan las principales métricas utilizadas para evaluar el desempeño de los modelos considerados en este proyecto [13].

3.6.1.1. Precisión

La precisión determina cuántas de las predicciones positivas realizadas por el modelo corresponden realmente a casos positivos. Además, permite identificar si el modelo está realizando predicciones positivas incorrectas, es decir, cuando detecta una instancia donde en realidad no la hay (falso positivo).

$$\text{Precisión} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos positivos}} \tag{3.1}$$

[13] El contenido de esta sección se basa en el libro *Deep Learning for Vision Systems* (Elgendy, 2020), salvo que se indique lo contrario.

Un valor alto de precisión significa que la mayoría de las predicciones positivas son correctas. Llevado a un caso práctico, la mayoría de las detecciones de hormigas corresponden realmente a una hormiga; en otras palabras, el modelo comete pocos falsos positivos.

Por el contrario, un valor bajo de precisión indica que el modelo identifica instancias negativas como positivas . En el caso de la detección de hormigas, esto implica que el modelo confunde elementos del entorno (como hojas, piedras o sombras) con hormigas, generando una cantidad considerable de falsos positivos.

Por estas razones, esta métrica es útil para evaluar si las predicciones positivas realizadas por el modelo son confiables, y permite detectar la presencia de falsos positivos con el fin de ajustar su comportamiento.

3.6.1.2. Sensibilidad

Esta métrica mide en qué proporción los casos positivos son predichos correctamente por el modelo. Además, permite analizar si el modelo está fallando al no reconocer instancias positivas, es decir, cuando una instancia que debería predecirse como positiva se predice incorrectamente como negativa (falso negativo).

$$\text{Sensibilidad} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos negativos}} \tag{3.2}$$

De esta forma, una sensibilidad alta indica que el modelo detecta la mayoría de los casos positivos reales. Visto de otra forma, se detecta correctamente la mayoría de las hormigas presentes en una imagen y pocas hormigas reales pasan desapercibidas.

En cambio, una sensibilidad baja indica que el modelo no detecta una parte importante de los casos positivos. En el caso de la detección de hormigas, esto significa que pasa por alto varias hormigas presentes en la imagen y solo identifica una parte de las reales.

Por estas razones, esta métrica resulta especialmente importante cuando se busca minimizar la cantidad de instancias positivas que el modelo no logra detectar.

3.6.1.3. Puntaje F1

Esta métrica combina los valores de precisión y sensibilidad (ver fórmula 3.3) mediante una media armónica. Esto implica que penaliza fuertemente el desequilibrio entre ambos: si uno de los dos valores es bajo, el puntaje también lo será. Por esta razón, resulta útil cuando se quiere evitar pasar por alto instancias

positivas (por ejemplo, objetos de interés que deben ser detectados) y minimizar la cantidad de falsos positivos.

$$\text{Puntaje F1} = \frac{2(\text{Precisión} \times \text{Sensibilidad})}{\text{Precisión} + \text{Sensibilidad}} \tag{3.3}$$

Llevado a un ejemplo práctico, un *puntaje F1 alto* indica que el modelo es confiable y equilibrado: detecta correctamente la mayoría de las hormigas presentes (alta sensibilidad) y comete pocos errores al clasificar elementos del entorno como hormigas (alta precisión).

Por el contrario, un *puntaje F1 bajo* indica un mal equilibrio entre precisión y sensibilidad, y puede deberse a que el modelo comete muchos falsos positivos (detecta incorrectamente objetos como hormigas), a que omite numerosas hormigas reales (muchos falsos negativos), o a una combinación de ambos casos.

3.6.1.4. Intersección sobre la unión

La Intersección sobre la unión (IoU, por su sigla en inglés) es una métrica que evalúa la superposición entre dos cajas delimitadoras: una caja representa el **valor de referencia** de la detección o clasificación real, mientras que la otra representa lo que el modelo predice. Esto permite determinar si una predicción es válida (verdadero positivo) o no (falso positivo).

Como se puede ver en la ecuación 3.4, para calcular el IoU de una predicción se utilizan las cajas delimitadoras definidas, dividiendo el área de superposición de las cajas sobre el área de la unión.

$$\text{IoU} = \frac{B_{\text{valor de referencia}} \cap B_{\text{prediccion}}}{B_{\text{valor de referencia}} \cup B_{\text{prediccion}}} \tag{3.4}$$

El valor calculado se encuentra entre cero y uno, siendo cero que no hay superposición y uno que las cajas delimitadoras coinciden, como se muestra en la figura 3.12.

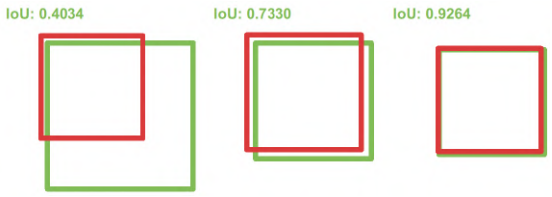

Figura 3.12: Ejemplos visuales del cálculo de (IoU) entre una caja delimitadora predicha (rojo) y una de referencia o *ground truth* (verde). Se muestran distintos niveles de superposición, con IoU de 0.4034 (bajo), 0.7330 (medio) y 0.9264 (alto). Imagen tomada de *Deep Learning for Vision Systems*, Elgendy (Elgendy, 2020, p. 290).

Para determinar si una predicción es válida, es necesario definir un umbral de confianza límite. Este se compara con el valor del IoU: si el IoU es mayor o igual al umbral, la predicción se considera correcta (verdadero positivo); en caso contrario, se considera incorrecta (falso positivo). Este umbral es ajustable al tipo de tarea, pero se utiliza como estándar el valor 0,5.

Capítulo 4

Metodología

En esta sección se describe el enfoque metodológico adoptado para abordar el problema de detección de hormigas cortadoras y segmentación de caminos. Se detallan los procesos de adquisición de datos, entrenamiento de los modelos y evaluación de resultados.

4.1. Conjuntos de datos

Para cumplir el objetivo de desarrollar y evaluar el sistema de detección de actividad en caminos de hormigas, es necesario contar con conjuntos de datos que permitan realizar un ajuste fino de los modelos a utilizar, así como evaluar el funcionamiento del sistema ARP en su totalidad. En concreto, se requieren tres conjuntos de datos:

- Uno para el entrenamiento y evaluación de los modelos de detección de hormigas.
- Otro para el entrenamiento y evaluación de los modelos de segmentación de caminos de hormigas.
- Finalmente, uno para evaluar el sistema de detección de caminos activos (ARP).

Con ese propósito, se realiza una revisión exhaustiva tanto de artículos como de repositorios públicos (Roboflow y Kaggle) con el objetivo de encontrar conjuntos de datos útiles para entrenar y evaluar los modelos. Como resultado, se encuentra un conjunto en Roboflow etiquetado para la tarea de detección de hormigas (de Gier, 2023).

Sin embargo, sus imágenes son capturadas en primer plano y no presentan el contexto del camino, lo que las hace inapropiadas para la tarea de segmentación, pero útiles para la detección. Tampoco se encuentran conjuntos que incluyan máscaras de caminos, ni que combinen hormigas y caminos en una misma imagen, lo cual es necesario para evaluar el sistema ARP.

En base a lo anterior, se decide utilizar el conjunto de Roboflow para entrenar los modelos de detección, y crear dos conjuntos adicionales en ambientes no controlados: uno para entrenar modelos de segmentación, con caminos de hormigas anotados, y otro para evaluar el sistema ARP, que contiene anotaciones tanto de hormigas como de caminos en las mismas imágenes.

4.1.1. Fuente de datos

Las imágenes capturadas para la conformación de los conjuntos de datos son capturadas en ambientes no controlados mediante un iPhone 14 Pro, en distintas áreas verdes del departamento de Montevideo. Los sitios específicos donde se recolectaron los datos incluyen:

- Jardín botánico.
- Jardín del Museo de bellas artes Juan Manuel Blanes.
- Parque Francisco Lecocq.
- Parque del Prado.
- Parque Rivera.

Estos lugares se eligieron para contar con una variedad de entornos verdes típicos de la ciudad, que aportan diversidad visual al conjunto de datos, dado que la variabilidad en texturas, iluminación y vegetación puede contribuir a mejorar la generalización del modelo.

4.1.2. Anotaciones

Para el proceso de etiquetado se decide utilizar *Computer Vision Annotation Tool* (CVAT), una herramienta *open source* que provee una interfaz de usuario para la anotación de conjuntos de datos en tareas de visión por computadora. Esta herramienta se selecciona en base a la recomendación realizada por los supervisores del proyecto, su amplia documentación y su baja curva de aprendizaje.

Teniendo en cuenta el tiempo que insume el etiquetado manual de caminos y de hormigas, se opta por incorporar herramientas de etiquetado automático. Específicamente, se opta por utilizar el servicio Nuclio provisto por CVAT, que permite ejecutar tareas de preanotación automática mediante modelos de inteligencia artificial. Para realizar esta preanotación se decide utilizar YOLO, debido a su amplia documentación y facilidad de uso. Para ello, se realiza un ajuste fino al modelo con el conjunto de datos obtenido hasta el momento y se utilizan los hiperparámetros con su valor por defecto. En el momento de integrar YOLO con el servicio de NUCLIO se modifica el código fuente de CVAT[1] debido a que el servicio no funciona específicamente para la segmentación con el modelo seleccionado.

Estas acciones permiten que CVAT genere preanotaciones automáticas a partir del modelo entrenado, agilizando considerablemente el proceso de etiquetado y reduciendo la carga de trabajo manual.

4.1.3. Proceso de generación del conjunto de datos

Como parte del desarrollo del conjunto de datos, se plantea un flujo de trabajo iterativo con el objetivo de mejorar en cada instancia la precisión de las anotaciones. En cada ciclo del proceso, se incorporan las nuevas imágenes capturadas. Una vez anotadas, estas imágenes se suman al conjunto y se utilizan para volver a entrenar el modelo encargado de generar anotaciones automáticas. Cabe destacar que en cada iteración el modelo se entrena nuevamente desde cero, sin reutilizar pesos ni configuraciones de entrenamientos anteriores. También, se debe tener en cuenta que las imágenes que conforman el conjunto de datos siempre presentan una instancia a detectar o segmentar.

A continuación, se describen los pasos que componen cada iteración del proceso:

1. **Se recolectan imágenes** en los entornos mencionados previamente.
2. **Se revisan manualmente las imágenes** capturadas para descartar aquellas que no presentan visibilidad adecuada de la instancia a ser etiquetada.
3. **Se divide el conjunto de datos** en tres subconjuntos: entrenamiento, validación y prueba.
4. **Se crean tareas de anotación** en CVAT.
5. **Se generan anotaciones automáticas** utilizando el servicio Nuclio integrado en CVAT, utilizando un modelo YOLO parcialmente entrenado para la tarea objetivo.

[1] CVAT modificado para usar YOLO con segmentación disponible en el link.

6. **Se validan las anotaciones automáticas** y agregan etiquetas manualmente si es necesario.

7. **Se realiza una validación cruzada entre anotadores** para evitar incoherencia en el etiquetado.

8. **Se re entrena el modelo con YOLO**, basado en las nuevas imágenes con las anotaciones realizadas, con el objetivo de mejorar la eficiencia de las nuevas anotaciones automáticas.

Este proceso se reitera hasta alcanzar un conjunto de datos que se considera adecuado para las tareas de ajuste de los modelos, entendido esto como obtener al menos un puntaje F1 de 70 % con sensibilidad y precisión equilibradas en alguno de los modelos evaluados. Además, se tuvo en cuenta el tiempo y los recursos disponibles para llevar a cabo el proceso antes mencionado.

4.1.4. Generación de conjunto de datos

Siguiendo el procedimiento descrito anteriormente, se obtienen los conjuntos que se presentan a continuación.

4.1.4.1. Conjunto de datos de caminos de hormigas

El conjunto de datos de caminos de hormigas se utiliza en la tarea de segmentación de caminos. Este tiene las siguientes características:

- **Composición del conjunto**: 1.208 imágenes en total.
  - **Entrenamiento**: representa el 71,1 % (859) del total de imágenes etiquetadas y se utiliza durante el proceso de entrenamiento del modelo.
  - **Validación**: representa el 19,4 % (234) del total y se emplea para ajustar los hiperparámetros y prevenir el sobre ajuste durante el entrenamiento.
  - **Prueba**: representa el 9,5 % (115) del total y se utiliza para evaluar el desempeño final del modelo entrenado.
- **Características de las imágenes (imágenes de referencia en la figura 4.1)**:
  - Resolución de imagen: 3024 × 4032 píxeles.
  - Formato de imagen: *JPEG*.

- Protocolo de captura: se mantiene una altura aproximada de entre 80 y 120 cm respecto al suelo. La orientación del dispositivo es mayormente paralela al plano del suelo, aunque en algunos casos se permitió una inclinación de hasta 20 grados.
- Diversidad visual: se capturan imágenes en distintos momentos del día y bajo diferentes condiciones climáticas, con el objetivo de introducir variación en aspectos como la iluminación, la presencia de bifurcaciones u obstáculos en los caminos.

Figura 4.1: Ejemplos de imágenes del conjunto de datos de caminos de hormigas.

4.1.4.2. Conjunto de datos para evaluar ARP

Este conjunto se utiliza para la evaluación del pipeline de detección de caminos activos y tiene las siguientes características:

- **Composición del conjunto**: 122 imágenes en total, empleadas para evaluar el desempeño final del sistema ARP. A su vez, cada imagen incluye

anotaciones simultáneas de **caminos de hormigas** (máscaras de segmentación) y **hormigas individuales** (cajas delimitadoras) como se aprecia en la figura 4.2).

- **Características de las imágenes**:
  - Resolución: 3024 × 4032 píxeles.
  - Formato: *JPEG*.
  - Protocolo de captura: se mantiene una altura aproximada de entre 30 y 50 cm respecto al suelo. La orientación del dispositivo es mayormente paralela al plano del suelo. La razón de este cambio en el protocolo es que las cámaras disponibles no son lo suficientemente potentes como para capturar las hormigas.
  - Diversidad visual: las tomas se realizan en exteriores a distintas horas del día y bajo diversas condiciones climáticas, por lo que varía la iluminación y la presencia de bifurcaciones u obstáculos en los caminos.

Figura 4.2: Imagen del conjunto de evaluación con segmentación del camino (perímetro en fucsia) y detección de hormigas (cajas rojas) junto con acercamientos de las detecciones (fila inferior).

4.2. Entrenamiento de modelos

Se opta por realizar un ajuste fino sobre modelos de aprendizaje automático. En particular, se emplean técnicas de detección para las hormigas y de segmentación para los caminos, debido a los distintos requerimientos en cada caso. Para los caminos es necesario utilizar técnicas de segmentación, ya que se requiere una delimitación píxel a píxel con el fin de determinar con exactitud qué hormigas se encuentran dentro de ellos. En cambio, para las hormigas no es necesario ese nivel de detalle: basta con conocer su ubicación general, por lo que una caja delimitadora resulta suficiente.

En total, se entrenan cuatro modelos basados en los pesos preentrenados de COCO: dos orientados a segmentación de caminos (YOLO y Mask R-CNN) y dos a la detección de hormigas (YOLO y Faster R-CNN). Se opta por utilizar estos pesos preentrenados con el fin de acelerar el entrenamiento y favorecer la convergencia de los modelos hacia sus respectivas tareas, permitiendo una mejor adaptación a los datos disponibles. Para cada modelo se detallan el entorno computacional (hardware, software y configuración), los hiperparámetros ajustados, el método de ajuste y las métricas de evaluación utilizadas.

4.2.1. Condiciones generales

Para el desarrollo del proyecto se empleó Visual Studio Code como IDE, debido a su facilidad de uso y a la familiaridad adquirida por el equipo gracias a su uso previo. Como lenguaje de programación principal se utiliza Python 3, dado que es utilizado generalmente para el entrenamiento de modelos y su disponibilidad de bibliotecas especializadas como *Ultralytics*, *Pytorch*, *Pandas*, *OpenCV*, entre otras.

En cuanto al ajuste de los valores de los hiperparámetros, esta es una etapa fundamental en el entrenamiento de modelos de aprendizaje automático. Estos valores afectan directamente el rendimiento, la capacidad de generalización y la velocidad de convergencia del modelo. Por este motivo, se opta por utilizar dos técnicas con el fin de optimizar el modelo:

- **Grid search (búsqueda en malla):** es un método que permite entrenar sistemáticamente el modelo hasta encontrar la configuración que optimice el desempeño según una métrica establecida. Consiste en seleccionar los hiperparámetros que se desean ajustar o refinar, definir un conjunto de valores para cada uno, y entrenar el modelo con todas las configuraciones posibles. Esto genera una especie de grilla con las combinaciones, donde cada dimensión corresponde a un hiperparámetro y sus valores definidos. El algoritmo evalúa sistemáticamente todas las combinaciones, seleccionando un valor por cada hiperparámetro en cada iteración.

- **Greedy Best First Search (GBFS):** consiste en ajustar un hiperparámetro a la vez según una métrica de evaluación previamente definida. El proceso comienza fijando un valor inicial para todos los hiperparámetros que se desean ajustar (usualmente el valor por defecto del modelo). Luego, se selecciona uno de ellos y se entrena el modelo con distintos valores para ese único hiperparámetro, manteniendo los demás fijos. Finalmente, se fija el valor que obtiene el mejor resultado. El proceso se repite iterativamente con los demás hiperparámetros hasta completar el ajuste.

Estas dos estrategias presentan diferencias importantes. Mientras que Grid Search puede considerarse un algoritmo de fuerza bruta, GBFS realiza un ajuste secuencial y toma decisiones basadas en los resultados obtenidos en cada iteración. Esto refleja que Grid Search prioriza la exhaustividad, mientras que Greedy se basa en un enfoque más metódico. Por esta razón, se prioriza la utilización de GBFS en la mayoría de los casos, aunque este puede quedar atrapado en mínimos locales. Por lo tanto, se recurre a Grid Search únicamente cuando no se consideran satisfactorios.

Cabe aclarar que, en ambos casos, se definen previamente los hiperparámetros sobre los cuales se iterará, y luego se ejecuta una secuencia de entrenamientos automatizados mediante scripts que implementan el procedimiento anteriormente descrito.

4.2.2. Método de evaluación

Definidas las estrategias de ajuste, es necesario un criterio para determinar la mejor combinación de hiperparámetros que optimice el modelo. Por ello, se implementa un módulo de métricas que utiliza la biblioteca scikit-learn (sklearn), que toma como referencia el código fuente del proyecto de grado *Pipeline de detección y seguimiento de manzanas para geolocalización de anomalías* (Sheppard, s.f.).

Este módulo evalúa las predicciones utilizando un umbral de confianza de 0,5 y un IoU de 0,5 para determinar si una detección es verdadera positiva. Establecer una confianza mínima del 50 % permite garantizar que la detección o segmentación de los objetos en el pipeline sea lo suficientemente fiable. De este modo, se reduce la probabilidad de que el sistema confunda otros elementos con hormigas o caminos, lo que mejora la precisión de las detecciones y la segmentación, y, en consecuencia, la efectividad en la identificación de un camino activo.

Bajo este lineamiento se clasifican las predicciones en verdaderos positivos, falsos positivos y falsos negativos. En cuanto a los verdaderos negativos (son instancias que no deben ser predichas y efectivamente no lo son), esta categorización no aplica a la detección y segmentación de objetos. En estos casos,

solo se evalúan las instancias positivas (objetos presentes en la imagen) y las falsas detecciones; no se considera explícitamente a las regiones sin objetos que el modelo ignora correctamente.

En este contexto se plantea utilizar las siguientes métricas para la evaluación de los modelos de segmentación y detección seleccionados:

- **Puntaje F1:** se emplea como métrica principal, ya que pondera de igual forma la *precisión* y la *sensibilidad*. A su vez, el puntaje F1 penaliza tanto las falsas detecciones como las omisiones, lo cual es importante cuando ambos errores afectan la tarea.
- **Precisión:** se emplea como métrica complementaria para evaluar la capacidad del modelo para identificar correctamente los verdaderos positivos, y detectar si tiende a generar falsos positivos. A su vez, al ser uno de los componentes del puntaje F1, resulta útil analizarla por separado para entender mejor el rendimiento del modelo.
- **Sensibilidad:** se emplea como métrica complementaria para evaluar si el modelo omite verdaderos positivos. Al utilizarla en conjunto con la *precisión*, permite obtener una visión más completa sobre el comportamiento del modelo frente a los falsos negativos. A su vez, al ser uno de los componentes del puntaje F1, resulta útil analizarla por separado para entender mejor el rendimiento del modelo.

Estas métricas también se consideran al momento de comparar el rendimiento entre los modelos evaluados.

4.2.3. Detección de hormigas

Para la detección de hormigas, se opta por utilizar Faster R-CNN y YOLO, ya que representan dos enfoques distintos para esta tarea: el primero corresponde a un modelo de dos etapas y el segundo, a uno de una sola etapa. A su vez, durante la elaboración del estado del arte (Nadile y cols., 2025), se identifica el artículo *A Systematic Review on Automatic Insect Detection Using Deep Learning*, de Teixeira et al. (Teixeira y cols., 2023), el cual compara diversos modelos aplicados a la detección de insectos. En dicho trabajo, tanto YOLOv5 como Faster R-CNN obtienen buenos resultados, lo que respalda la elección de ambas arquitecturas en el presente proyecto, especialmente por estar orientadas a tareas de detección en insectos, un dominio con características similares al abordado.

Respecto al conjunto de datos, se decide utilizar el conjunto *Ant Object Detection* (de Gier, 2023) en los procesos de entrenamiento, validación y prueba para los modelos YOLO y Faster R-CNN seleccionados. Si bien se piensa en un

principio crear un dataset propio de hormigas, esta idea se descarta por la gran cantidad de tiempo y recursos que esto insume.

(a) Hormigas en ambientes controlados.

(b) Hormigas en ambientes no controlados.

Figura 4.3: Imágenes representativas del conjunto de datos *Ant Object Detection* (de Gier, 2023) a utilizar en la tarea de detección de hormigas.

Este conjunto está compuesto por 14.210 imágenes con anotaciones para tareas de detección de hormigas. Las imágenes se encuentran divididas en tres subconjuntos: entrenamiento con 10.320 imágenes (73 %), validación con 2.893 imágenes (20 %) y prueba con 997 imágenes (7 %). Además, incluyen preprocesamientos básicos definidos en el trabajo original: las imágenes fueron automáticamente orientadas, redimensionadas a 640×640 píxeles y completadas en los

bordes mediante una técnica de reflexión. Cabe señalar que los autores del conjunto no aplicaron técnicas adicionales de data augmentation al mismo. En la figura 4.3 se puede apreciar una muestra de este conjunto.

A continuación, se detallan las configuraciones particulares utilizadas para el entrenamiento de los modelos de detección, incluyendo tanto los sistemas de cómputo empleados (dado que se recurrió a dos entornos distintos) como los métodos específicos aplicados durante el entrenamiento de cada uno.

YOLO

- **Sistema:** PC de escritorio con Windows 11 Pro, procesador AMD Ryzen 9 7950X, GPU Nvidia Gigabyte RTX 4070 con 12 GB de VRAM y 128 GB de RAM.
- **Implementación del modelo:** se utilizan las versiones YOLOv8 y YOLOv11 provistas por la biblioteca de Ultralytics.
- **Método de entrenamiento:** se utiliza el método GBFS y el Tuner (optimizador de rendimiento) provisto por la biblioteca Ultralytics para YOLO.
- **Hiperparámetros ajustados:** número de épocas, tamaño del lote, tasa de aprendizaje inicial, tasa de aprendizaje final, variante de modelo (n,s, m o l), decaimiento de pesos y transformaciones de data augmentation.

Faster R-CNN

- **Sistema:** PC de escritorio con Windows 11 Pro, procesador Intel Core i7-14700F de 14ª generación (2.10 GHz), GPU Nvidia Gigabyte RTX 4070 con 12 GB de VRAM y 64 GB de RAM.
- **Implementación del modelo:** se utiliza las implementación provista por PyTorch.
- **Método de entrenamiento:** se emplea la estrategia Greedy Search.
- **Hiperparámetros ajustados:** backbone, taza de aprendizaje, número de épocas, tamaño del lote, decaimiento de pesos y momento.

4.2.4. Segmentación de caminos

Para la segmentación de caminos, se utilizan Mask R-CNN y YOLO. La elección de estos modelos se encuentra alineada con las decisiones tomadas para la tarea de detección de hormigas, ya que Mask R-CNN extiende la arquitectura de Faster R-CNN incorporando una rama adicional para la predicción de máscaras de segmentación.

Respecto al conjunto de datos, ambos modelos emplean para el proceso de entrenamiento, validación y test el conjunto de imágenes de caminos de hormigas generado durante esta investigación, ya que no se dispone de un conjunto público equivalente.

A continuación, se detallan configuraciones particulares utilizadas para el entrenamiento de los modelos de segmentación:

YOLO

- **Sistema:** PC de escritorio con Windows 11 Pro, procesador Intel Core i7-11700KF de 11ª generación (3.60 GHz), GPU Gigabyte GeForce RTX 4080 con 16 GB de VRAM y 128 GB de RAM.
- **Implementación del modelo:** se utilizan las versiones YOLOv8 y YOLOv11 provistas por la biblioteca de Ultralytics.
- **Método de entrenamiento:** se utiliza el método GBFS y el Tuner (optimizador de rendimiento) provisto por Ultralytics para YOLO.
- **Hiperparámetros ajustados:** variante de modelo, tamaño de lote, número de épocas, tasa de aprendizaje inicial, taza de aprendizaje final, decaimiento de pesos, momento y paciencia.

Mask R-CNN

- **Sistema:** PC de escritorio con Windows 11 Pro, procesador AMD Ryzen 9 7950X, GPU Nvidia Gigabyte RTX 4070 con 12 GB de VRAM y 128 GB de RAM.
- **Implementación del modelo:** se utiliza la implementación provista por PyTorch.
- **Método de ajuste:** se prueban ambas estrategias. Dado que el método GBFS no arroja buenos resultados, se opta por una estrategia más exhaustiva mediante Grid Search.
- **Hiperparámetros ajustados:** número de épocas, tamaño del lote, tasa de aprendizaje, tamaño de anclaje, relación de aspecto de anclaje, Backbone y transformaciones de data augmentation.

Capítulo 5

Active Roads Pipeline (ARP)

Este capítulo presenta la arquitectura general del sistema propuesto, denominado *Active Roads Pipeline* (ARP), desarrollado con el objetivo de detectar caminos activos de hormigas a partir de imágenes capturadas en entornos naturales. ARP integra diferentes etapas de procesamiento, combinando modelos de segmentación y detección para identificar tanto los caminos como la presencia de hormigas sobre ellos.

En principio, se describen las decisiones de diseño que orientan la construcción del sistema y las tecnologías utilizadas. A partir de ellas, se presenta el *flujo del sistema*, detallando las etapas principales que componen el pipeline y su orden de ejecución. Posteriormente, se aborda la *arquitectura del sistema*, donde se explican los componentes implementados y su interacción.

5.1. Arquitectura del sistema

5.1.1. Decisiones de diseño

Se toman una serie de decisiones de diseño que buscan flexibilizar y mejorar el desempeño del sistema. Estas decisiones se basan en los desafíos encontrados durante el desarrollo de la investigación, como la dificultad para detectar objetos pequeños, la calidad de las imágenes o la presencia de elementos que distraen del área realmente importante. A continuación, se detallan los lineamientos de diseño adoptados:

- **Abstracción de los modelos de segmentación:** se diseña el pipeline con el objetivo de que pueda ser utilizado independientemente del modelo seleccionado. De esta forma, se logra una implementación flexible, que no queda restringida a un modelo específico y facilita cambios a futuro para integrarse con otros modelos.
- **Inclusión de SAHI:** dado que se trabaja con objetos pequeños como las hormigas, se incorpora la técnica *Slicing Aided Hyper Inference* (SAHI) con el objetivo de mejorar su detección. Esto se debe a que, en las imágenes, las hormigas suelen ocupar un área significativamente menor en comparación con los caminos. Esta diferencia de escala puede dificultar su reconocimiento por parte de los modelos, especialmente si no fueron entrenados para detectar objetos tan pequeños. Al dividir la imagen en secciones más pequeñas, SAHI permite al modelo enfocarse en regiones locales, aumentando así la probabilidad de detectar correctamente las hormigas.
- **Introducir un factor de nitidez**: como se aprecia en la figura 5.1 en algunas capturas las hormigas aparecen difuminadas debido a su velocidad de movimiento o comportamiento errático. Para reducir este efecto, se incorpora la opción de aplicar un filtro de nitidez a la imagen, con el objetivo de mejorar la definición de los objetos pequeños y así facilitar su detección por parte del modelo.
- **Focalizar la detección de hormigas sobre la máscara del camino:** esta opción permite reducir la información no relevante de la imagen y destacar únicamente los píxeles de interés, al limitar la detección de las hormigas al área segmentada. Esto contribuye a disminuir posibles falsas detecciones fuera del camino y puede llegar a mejorar la precisión del sistema. Cabe mencionar que, aunque es posible que algunas hormigas se encuentren fuera del camino, en este trabajo se prioriza la detección sobre el camino, por ser el área más relevante para el análisis. Además, al aplicar el módulo de detección sobre la imagen completa y luego específicamente sobre la máscara del camino, se busca comparar ambos enfoques. El motivo es que al focalizar la detección sobre dicha región, las hormigas presentes pueden apreciarse con mayor detalle, lo que potencialmente puede contribuir a obtener mejores resultados.

Figura 5.1: Ejemplos de hormigas difuminadas pertenecientes al conjunto de datos de evaluación del sistema.

5.1.2. Implementación

En base a las decisiones anteriores, se decide construir el sistema alrededor de una clase abstracta llamada *ActiveRoadsPipeline*. Esta clase define el flujo general del sistema y delega a sus clases concretas la responsabilidad de implementar la lógica específica para la segmentación de caminos de hormigas, así como la conversión de los resultados de detección al formato establecido. La estructura general de esta arquitectura puede observarse en la Figura 5.2.

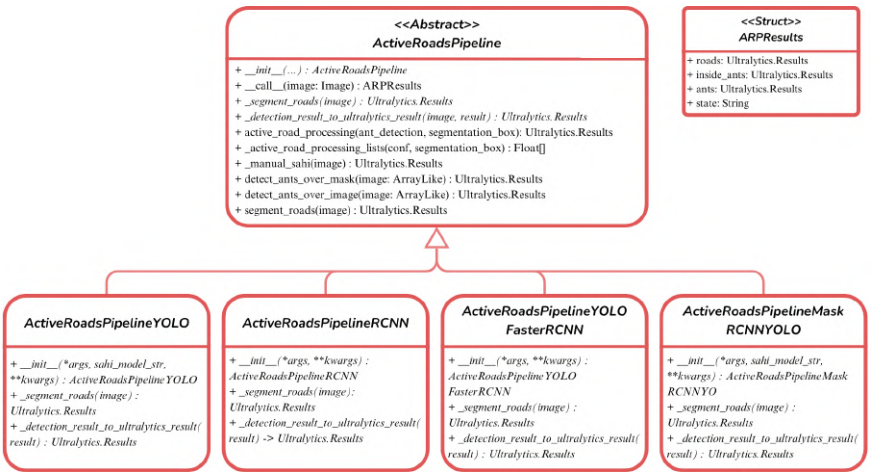

Figura 5.2: Diagrama de clases de la arquitectura del sistema ARP. Se muestran las clases principales que componen el pipeline, incluyendo la clase abstracta `ActiveRoadsPipeline`, sus implementaciones concretas para distintos modelos de detección y segmentación, y la estructura `ARPResults` que encapsula los resultados.

Asimismo, la clase *ActiveRoadsPipeline* permite configurar el comportamiento del sistema a través de un conjunto de parámetros. A continuación, se presenta un listado con las distintas variantes disponibles, las cuales permiten ajustar aspectos específicos del funcionamiento del pipeline:

- **Modelo de segmentación:** modelo previamente entrenado para la segmentación de caminos.
- **Modelo de detección:** modelo previamente entrenado para la detección de hormigas.
- **Uso de SAHI:** desactivar o habilitar el uso de SAHI en el módulo de detección de hormigas. Actualmente, solo es compatible con modelos YOLO, pero si el modelo de detección seleccionado es compatible con SAHI y se desea utilizar este mecanismo, es necesario asignar el atributo `sahi_model`

con una instancia del modelo construida mediante los métodos proporcionados por la biblioteca.

- **Detección sobre la máscara del camino:** permite decidir si las hormigas deben detectarse exclusivamente sobre la región segmentada del camino. En caso afirmativo, si SAHI está encendido, este también habilita la opción de seccionar la máscara, con dos posibles estrategias:

  - **Automática:** basada en las propiedades de la máscara, permite configurar:
    - Factor de solapamiento en el ancho de las secciones.
    - Factor de solapamiento en el alto de las secciones.
  - **Manual:** permite definir los parámetros en función de la relación de aspecto de la máscara:
    - Valor de solapamiento en ancho y alto.
    - Cantidad de columnas y filas utilizadas para seccionar la máscara.

- **Aplicación de nitidez:** permite ajustar un valor de *nitidez* que mejora la definición de objetos en la imagen. Para ello se utiliza el método *adjust_sharpness* provisto por la biblioteca de PyTorch.

- **Factor de expansión de la máscara:** dilata la máscara del camino con el objetivo de incluir hormigas que se encuentren en los márgenes del mismo.

Por último, al finalizar el procesamiento, el sistema produce una estructura de salida que contiene:

- **Máscaras segmentadas:** clasificadas como camino activo o inactivo según la detección de hormigas. En el caso de los caminos activos, su relevancia depende de la presencia efectiva de hormigas, por lo que se considera más representativo calcular la confianza como el promedio de las confianzas de las hormigas detectadas dentro de la máscara, reflejando tanto su cantidad como la certeza de sus detecciones. En cambio, para los caminos inactivos (donde no hay hormigas detectadas) no es posible calcular un promedio de confianzas de hormigas, por lo que se adopta la confianza con la que fue detectada la máscara del camino en sí.

- **Cajas delimitadoras generales:** delimitaciones de todas las hormigas detectadas en la imagen completa.

- **Cajas delimitadoras sobre el camino:** delimitaciones de las hormigas detectadas específicamente dentro del área segmentada como camino.

- **Clasificación de la actividad:** se devuelve como una cadena de texto (*string*) que puede incluir uno o más de los siguientes valores, teniendo en cuenta que una hormiga se considera real si su predicción tiene una confianza mínima de 0,5:

  - **Camino activo:** se detectan hormigas sobre el camino. Un camino se considera activo si contiene al menos un número mínimo (umbral) de hormigas dentro de su máscara. A su vez, se considera que una hormiga está dentro del camino si al menos la mitad de su caja delimitadora se encuentra dentro de los límites del mismo.

  - **Camino inactivo:** no se detectan hormigas sobre el camino o la cantidad de hormigas detectadas no supera el número mínimo.

  - **Actividad de hormigas:** se detectan hormigas en la imagen, independientemente de que se detecte un camino o no.

  - **Sin actividad de hormigas:** no se detectan hormigas en la imagen.

5.2. Flujo del sistema

La figura 5.3 presenta el flujo general del sistema ARP (*Active Roads Pipeline*), encargado de determinar si un camino está activo en función de la presencia de hormigas.

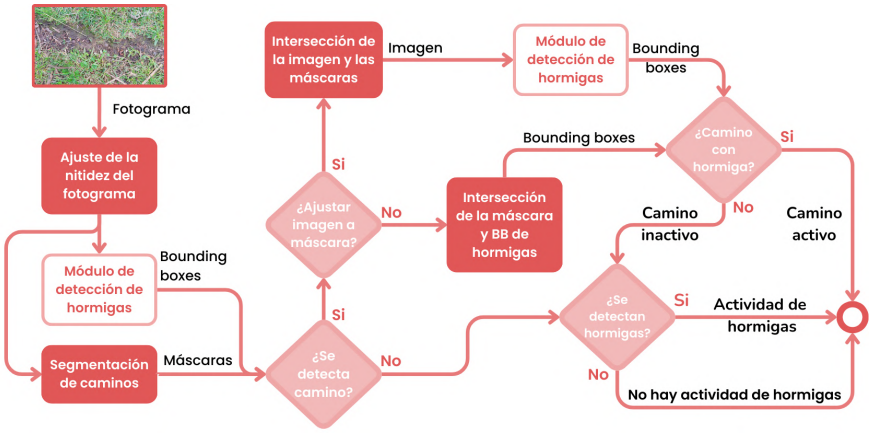

Figura 5.3: Flujo general del sistema ARP para la detección de caminos activos.

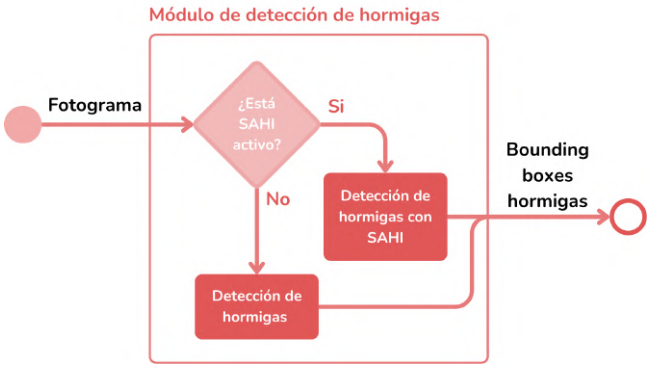

Figura 5.4: Detalle del módulo de detección de hormigas, con soporte para detección estándar y SAHI.

El proceso comienza con la recepción de un fotograma, al cual se le aplica un ajuste de nitidez. Esta operación, si bien no modifica la estructura de la imagen, ayuda a resaltar objetos en la misma.

Acto seguido, se invoca el módulo de detección de hormigas. Como se detalla en la Figura 5.4, este módulo puede operar en dos modos distintos, en función de la configuración establecida por el usuario y la naturaleza del modelo de detección utilizado. Cuando el sistema se encuentra en modo estándar, se realiza una única inferencia sobre la imagen completa, utilizando en este caso YOLO o Faster R-CNN. Sin embargo, cuando se activa SAHI, el comportamiento del sistema cambia de forma que la imagen es fragmentada en múltiples secciones solapadas, sobre cada una de las cuales se ejecuta el modelo de detección. Esta etapa retorna un conjunto de cajas delimitadoras con las posiciones de las hormigas.

Con las detecciones iniciales realizadas, el pipeline continúa con la segmentación de caminos. Esta tarea se lleva a cabo utilizando un modelo de segmentación, que puede ser YOLO o Mask R-CNN. El resultado de esta etapa es un conjunto de máscaras de caminos de hormigas.

Una vez obtenidas las máscaras, el sistema evalúa si existen caminos segmentados. En caso de que no se haya detectado ninguno, se verifica la existencia de hormigas sobre la imagen en base a las cajas delimitadoras obtenidas. Si en esta revisión se identifican hormigas, el sistema infiere que existe actividad de hormigas en el fotograma. Por el contrario, si no se detectan hormigas, el sistema determina que no hay actividad en ese fotograma.

Cuando sí se han segmentado caminos, el sistema profundiza su análisis evaluando individualmente cada una de las máscaras obtenidas. En este punto, se bifurca el flujo según el modo de búsqueda activado. En la búsqueda localizada, el sistema recorta la imagen original a la región cubierta por la máscara,

definida como el rectángulo envolvente mínimo (bounding box) que abarca completamente la máscara binaria. Sobre esta región recortada se aplica nuevamente el módulo de detección de hormigas con el objetivo de mejorar la precisión, reducir el contexto visual a analizar y enfocar la inferencia en la zona de interés. Si en esta nueva inferencia se detectan hormigas dentro del área segmentada, el camino correspondiente es clasificado como activo. En caso contrario, se lo considera inactivo.

En el caso de que la búsqueda localizada no esté activada, se utiliza una estrategia basada en intersección geométrica. En este enfoque, se analiza si alguna de las cajas delimitadoras obtenidas en la detección global de la imagen intersecta con la región delimitada por la máscara. Si la intersección existe, el camino también es marcado como activo.

Al finalizar el procesamiento, el sistema devuelve la estructura de salida detallada en el apartado anterior.

El siguiente pseudocódigo resume el flujo completo descrito anteriormente:

```
# Paso 1
Inicializar estructura vacía para guardar resultados:
cajas, máscaras y detecciones.

# Paso 2
Aplicar ajuste de nitidez a la imagen original.

# Paso 3
Detectar hormigas en la imagen completa.

# Paso 4
Realizar segmentación de caminos.

# Paso 5
if se utiliza búsqueda basada en máscaras:
    for cada máscara segmentada:
        Expandir la máscara
        Detectar hormigas dentro de la región expandida

        if cantidad de detecciones ≥ umbral:
            Crear una caja activa combinando la detección
            y la máscara
            Guardar la caja y las detecciones asociadas
        else:
            Guardar la caja segmentada original

# Paso 6
else:
    for cada máscara segmentada:
        Expandir la máscara
```

```
        Filtrar detecciones dentro de la máscara

        if cantidad de detecciones ≥ umbral:
            Filtrar las de confianza alta
            Calcular confianza promedio
            Crear y guardar una caja activa con dicha
            confianza
        else:
            Guardar la caja segmentada original

# Paso 7
Combinar las cajas obtenidas en estructuras finales de
resultados.

# Paso 8
Construir objetos de resultado para detecciones, máscaras
y estado.

return resultado
```

5.3. Métricas de evaluación

Se plantea evaluar dos problemas: por un lado, la segmentación multiclase de los caminos, clasificándolos en caminos activos, caminos inactivos o fondo (todo píxel que no pertenece a las categorías anteriores), con el objetivo de identificar no solo la ubicación de los caminos, sino también su estado de actividad. Por otro lado, se plantea evaluar la segmentación de caminos y la detección de hormigas, considerando en esta última tanto su presencia sobre los caminos como en el resto de la imagen.

Para el cálculo de las métricas de segmentación multiclase, se opta por implementar una nueva clase abstracta. La decisión se basa en que el problema pasa de segmentación binaria (por instancias) a una multiclase, lo que implica que la matriz de confusión deja de ser de 2×2 y pasa a ser de 3×3. Esta clase se encarga de comparar las predicciones con los valores de verdad y de asociar cada predicción con su etiqueta correspondiente, siempre que ambas pertenezcan a la misma clase, la intersección sobre unión (IoU) sea mayor o igual a 0,5, y la confianza de la predicción también sea mayor o igual a 0,5. De este modo, solo se consideran aquellas predicciones que coinciden en la categoría de camino con la etiqueta, cuya confianza indica al menos un 50 % de certeza de que efectivamente corresponden a un camino, y en las que más de la mitad de los píxeles de la máscara predicha se superponen con los de la máscara de verdad. A su vez, esta clase se encarga del cálculo de las métricas F1, precisión y sensibilidad utilizando el promedio macro, que consiste en calcular la métrica de manera independiente para cada categoría y luego promediar los resultados,

dando igual peso a todas ellas. Se decide utilizar esta métrica porque resulta igualmente importante detectar la presencia de un camino, como identificar si está activo, y además es adecuado en contextos donde el conjunto de datos está desbalanceado. Por último, el cálculo del IoU se delega en clases concretas.

Cabe resaltar que al igual que en los anteriores problemas de segmentación y detección, no se tienen en cuenta los verdaderos negativos, debido a que no tienen sentido en este contexto. Un verdadero negativo es un camino activo o inactivo que no existe y a la vez no se debe registrar.

Por último, para evaluar la tarea de segmentación y detección, se decide utilizar el mismo conjunto de métricas utilizado para el entrenamiento de los modelos (ver sección 4.2.2).

Capítulo 6

Experimentación y resultados

Este capítulo expone los experimentos que se efectúan con el fin de alcanzar el objetivo central del proyecto: desarrollar una herramienta basada en aprendizaje profundo capaz de detectar hormigas cortadoras e identificar sus caminos.

Particularmente, la experimentación se organiza en tres secciones. En la sección 6.1 se detalla los experimentos de segmentación, incluyendo la configuración de las arquitecturas escogidas, el rango de hiperparámetros explorado y el análisis comparativo de sus resultados. La sección 6.2 recoge la evaluación de los modelos de detección de hormigas y el correspondiente análisis comparativo. Finalmente, la sección 6.3 presenta la evaluación del sistema ARP sobre los datos de prueba y justifica la configuración adoptada como óptima.

6.1. Evaluación de los modelos de segmentación

Como parte de la creación de un sistema que permita determinar caminos activos de hormigas, es necesario generar modelos para **segmentar dichos caminos**. Para abordar esta tarea, se plantea experimentar con distintos modelos de segmentación sobre el conjunto de caminos de hormigas presentado en la sección 4.1.4.1, con el fin de determinar el modelo que mejor se ajuste al sistema.

6.1.1. Mask R-CNN

Se comienza la experimentación utilizando GBFS, pero debido al pobre desempeño alcanzado, se opta por Grid Search (ver sección 4.2.1) para encontrar

la combinación óptima de hiperparámetros.

Para ello, se llevan a cabo dos búsquedas. La primera explora el espacio de valores propuestos en la tabla 6.1 para los hiperparámetros:

Hiperparámetro	**Valores evaluados**
Número de épocas	100
Tasa de aprendizaje	{0,00005, 0,0005, 0,005}
Backbone	{MobileNetV2, SqueezNet-1_1, ResNet-50V1, ResNet-50V2, ResNet-101}
Tamaño de lote	{1, 5, 10, 15}

Tabla 6.1: Valores definidos para Mask R-CNN por hiperparámetro para la primera Grid Search. En todos los casos se utiliza una red FPN acoplada al *backbone*.

Para las configuraciones descritas en la tabla 6.1, se aplican técnicas de data augmentation sobre el conjunto de entrenamiento con el objetivo de aumentar la diversidad de los datos y mejorar la capacidad de generalización del modelo. Para ello, se utilizan las siguientes transformaciones de la librería PyTorch:

- Rotación aleatoria.
- Espejado horizontal aleatorio.
- Perspectiva aleatoria.
- Re dimensionamiento.

Asimismo, se mantienen los parámetros por defecto de tamaño y relación de aspecto del generador de anclajes, los cuales varían según el backbone utilizado (ver tabla 6.2).

Adicionalmente, se decide también realizar una segunda búsqueda variando los anclajes del modelo dada la característica alargada de los caminos. Se seleccionan estos valores tras un análisis cuantitativo del tamaño y relación de aspecto de las cajas delimitadoras del conjunto de datos de entrenamiento, donde cada valor en la tabla 6.3 corresponde a:

Backbone	**Tamaños de anclaje**	**Relaciones de aspecto**
MobileNetV2	(128, 256, 512)	(0.5, 1, 2.0)
SqueezeNet-1_1	(64, 128, 256, 512)	(0.5, 1, 2.0)
ResNet-50V1 ResNet-50V2 ResNet-101	(32, 64, 128, 256, 512)	(0.5, 1.0, 2.0)

Tabla 6.2: Parámetros por defecto del generador de anclajes (tamaños y relaciones de aspecto) utilizados en Mask R-CNN para cada backbone durante la primera búsqueda.

- **Tamaño de anclaje**: corresponde al tamaño total del anclaje utilizado por cada capa de la FPN, definido como el producto del largo por el ancho del anclaje. Se agrega una posición a la n-upla de tamaños por cada capa. Los valores utilizados son:
  - Tamaño máximo de las cajas delimitadoras en el conjunto de entrenamiento: 320.672.
  - Tamaño máximo incrementado en un 10 % para incluir mayor contexto del entorno del camino: 352.739.
  - Tamaño promedio de las cajas delimitadoras: 63.916.
  - Tamaño promedio incrementado en un 25 %: 79.895.
  - Tamaño promedio incrementado en un 50 %: 95.874.
- **Relación de aspecto**: utiliza una n-upla que se recorre para obtener las RoIs. En este caso, solo se utilizan los siguientes valores:
  - Para los caminos *verticales*, se considera el mayor valor de relación de aspecto de las cajas delimitadoras en el conjunto de entrenamiento: 10,50.
  - Para los caminos *diagonales*, se considera el valor 1.
  - Para los caminos horizontales, se considera el inverso del valor anterior: 0,09.

En base a esto, se modifica el comportamiento del generador de anclajes al modificar los valores por defecto. Se presentan en la tabla 6.3 los valores de prueba utilizados para cada uno de los backbones.

Backbone	**Tamaños de anclaje**	**Relaciones de aspecto**
MobileNetV2	{(320672, 320672, 320672), (352739, 352739, 352739), (63916, 63916, 63916), (79895, (79895, 79895), (95874, (95874, 95874)}	(0.09, 1, 10.5)
SqueezeNet-1_1	{(320672, 320672, 320672, 320672), (352739, 352739, 352739, 352739), (63916, 63916, 63916, 63916), (79895, 79895, 79895, 79895), (95874, 95874, 95874, 95874)}	(0.09, 1, 10.5)
ResNet-50V1 ResNet-101	{(320672, 320672, 320672, 320672, 320672), (352739, 352739, 352739, 352739, 352739), (63916, 63916, 63916, 63916, 63916), (79895, 79895, 79895, 79895, 79895), (95874, 95874, 95874, 95874, 95874)}	(0.09, 1, 10.5)

Tabla 6.3: Combinaciones de tamaños y relaciones de aspecto del generador de anclajes evaluadas con Mask R-CNN para cada backbone durante la segunda búsqueda. La modificación de estos hiperparámetros no es compatibles con la implementación de Mask R-CNN con ResNet-50v2 en PyTorch, por lo que este modelo fue descartado en esta etapa.

A su vez, se modifican las transformaciones de data augmentation aplicadas con el fin de evitar que estas reduzcan el tamaño de los caminos en la imagen. En particular, se elimina la transformación de perspectiva aleatoria y se incorporan dos nuevas: la variación aleatoria del brillo (para simular distintas condiciones de iluminación) y la generación de sombras aleatorias sobre la imagen.

En base a los experimentos planteados, las mejores combinaciones obtenidas para cada backbone en cada una de las búsquedas se presentan en la tabla 6.4 y en la tabla 6.5.

Los resultados presentados en la tabla 6.4 permiten identificar diferencias en el desempeño de los modelos evaluados. En primer lugar, se observa que dos modelos sobresalen sobre el resto: ResNet-50v1 y ResNet-101. Según las métricas, ambos presentan un valor similar de puntaje F1, a pesar de que ResNet-101 es más profundo (101 capas) que ResNet-50v1 (50 capas). Sin embargo, no se puede afirmar lo mismo del backbone ResNet-50v2, que a pesar de compartir la misma profundidad que la versión uno, muestra un rendimiento inferior. Esto sugiere una conexión entre el problema a resolver y la estructura de los

backbones utilizados, y no únicamente con la profundidad de los mismos.

Modelo	Tamaño de lote	Tasa de aprendizaje	Puntaje F1 (%)	Precisión (%)	Sensibilidad (%)
MobileNetV2	5	0,00005	12,94	11,4	14,97
SqueezeNet	15	0,0005	24,78	21,5	29,25
ResNet50V1	15	0,00005	31,13	23,83	44,9
ResNet50V2	1	0,00005	27,10	22,52	34,01
ResNet101	**1**	**0,00005**	**31,44**	**25,31**	**41,5**

Tabla 6.4: Mejores resultados de *F1-score*, junto con sus respectivos valores de *precisión* y *sensibilidad* por *backbone*, utilizando anclajes por defecto y considerando todas las combinaciones evaluadas en la primera búsqueda.

Modelo	Tamaño de lote	Tasa de aprendizaje	Tamaño de anclaje	Puntaje F1 (%)	Precisión (%)	Sensibilidad (%)
MobileNetV2	1	0,00005	63,916	14,21	28	9,52
SqueezeNet	15	0,0005	63,916	14,93	16,53	13,61
ResNet50V1	10	0,0005	63,916	28,97	24,53	35,37
ResNet101	**5**	**0,00005**	**63,916**	**41,81**	**42,86**	**40,82**

Tabla 6.5: Mejores resultados de *F1 score*, junto con sus respectivos valores de *precisión* y *sensibilidad* por *backbone*, variando los anclajes en base a la tabla 6.3 y considerando todas las combinaciones evaluadas en la segunda búsqueda.

Asimismo, la tabla 6.4 muestra una diferencia de desempeño entre los modelos MobileNetV2 y SqueezeNet, ambos diseñados con arquitecturas ligeras orientadas a la reducción del consumo de recursos. En este caso, SqueezeNet presenta un rendimiento superior, lo cual podría atribuirse a su estructura, que evita aplicar reducciones tempranas en la resolución espacial, lo que permite conservar por más tiempo la información espacial de la imagen. Por el contrario, MobileNetV2 tiende a realizar una reducción agresiva de la resolución en las primeras etapas del procesamiento, lo cual podría conducir a la pérdida temprana de información relevante. Como consecuencia, los mapas de características generados podrían no ser adecuados para su aprovechamiento por parte de la arquitectura Mask R-CNN, afectando negativamente el desempeño general del sistema.

Luego, en la tabla 6.5, se observa que el desempeño del modelo con SqueezeNet empeora sustancialmente. Se considera que esto está relacionado con el hecho de que, aunque la arquitectura no reduce tempranamente la resolución de los datos, carece de la complejidad necesaria para generar mapas de características que conecten información a lo largo de la imagen. Esta limitación dificulta el funcionamiento de la RPN, que no logra proponer regiones de interés (RoIs) adecuadas.

Por otro lado, en la misma tabla, también se aprecia que la diferencia de desempeño entre ResNet-101 y ResNet-50V1 aumenta, siendo ResNet-101 el

modelo con mejores resultados. Esto sugiere que, en la primera búsqueda, no se aprovecha completamente la capacidad adicional de ResNet-101.

Finalmente, dado que ResNet-101 fue el modelo con mejores resultados, se realiza una comparación entre su configuración con anclajes por defecto y con anclajes modificados (tablas 6.4 e 6.5, respectivamente). Se interpreta que la mejora obtenida al ajustar los anclajes se debe a que el modelo, con esta configuración, es capaz de capturar mayor información espacial relevante que con los valores por defecto. En particular, la precisión aumenta en 17 puntos porcentuales, lo que indica que el modelo se vuelve más eficaz para distinguir instancias positivas de negativas y, en consecuencia, mejora el puntaje F1 de manera significativa.

6.1.2. YOLO

La exploración se inicia con dos experimentos independientes en paralelo, uno con YOLOv8 (experimento 1) y otro con YOLOv11 (experimento 2), en los que se aplica la estrategia de GBFS. En cada iteración se ajusta un único hiperparámetro, evaluando en ambos modelos el mismo conjunto de valores para dicho parámetro, aunque las mejores configuraciones se fijan de forma independiente para cada uno.

Hiperparámetro	**Valores evaluados**
Variante de modelo	{yolov8n-seg, yolov8m-seg, yolo11n-seg, yolo11m-seg}
Tamaño de lote	{1, 2, 4, 8, 16, 32}
Momento	{0.222, 0.555, 0.89, 0.91, 0.93, 0.937, 0.96, 0.999}
Tasa de aprendizaje inicial	{0.00001, 0.00005, 0.0001, 0.0003, 0.0006, 0.0009, 0.0009999, 0.00099999, 0.001, 0.00100011, 0.00101, 0.004, 0.01 0.03, 0.06, 0.09, 0.1}
Tasa de aprendizaje final	{0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 0.25, 0.125, 0.0625}
Decaimiento de pesos	0.0005
Épocas	{50, 100, 150, 200, 250, 300}
Paciencia	20

Tabla 6.6: Hiperparámetros y valores utilizados durante los dos primeros experimentos.

El orden de ajuste es: variante de modelo, tamaño de lote, momento, tasa de aprendizaje inicial (*lr0*), tasa de aprendizaje final (*lrf*) y número de épocas utilizando inicialmente los valores por defecto para cada uno. Para cada iteración, se utilizan los valores por defecto para los hiperparámetros no ajustados. Además, para optimizar recursos computacionales, todos los entrenamientos incorporan un criterio de *parada temprana* con *paciencia=20*, que detiene el entrenamiento cuando la métrica de validación deja de mejorar tras veinte épocas. De esta forma, se define el espacio de búsqueda cuyos valores se resumen en la tabla 6.6.

Siguiendo los lineamientos anteriores se tiene que la búsqueda sobre YOLOv8 alcanza un puntaje F1 de 69,49 %, con una precisión del 92,13 % y una sensibilidad del 55,78 %, mientras que la búsqueda simultánea sobre YOLOv11 eleva el puntaje F1 a 76,49 % y la sensibilidad al 65,31 %, manteniendo la precisión (ver tabla 6.7). Si bien ambos experimentos presentan un valor alto de *precisión*, YOLOv11 supera a YOLOv8 en este parámetro y además alcanza una sensibilidad superior; la combinación de ambos factores le permite detectar más caminos reales y, por ende, alcanzar un puntaje F1 mayor.

Modelo	**Hiperparámetros**	**Puntaje F1 (%)**	**Precisión (%)**	**Sensibilidad (%)**
yolov8	modelo: yolov8n-seg tamaño de lote: 16 épocas: 150 lr0: 0,001 lrf: 0,001 momento: 0,937 decaimiento de pesos: 0,0005	69,49	92,13	55,78
yolov11	modelo: yolo11n-seg tamaño de lote: 32 épocas: 250 lr0: 0,001 lrf: 0,01 momento: 0,937 decaimiento de pesos: 0,0005	76,49	92,31	65,31
yolov11 (tuner)	modelo: yolo11n-seg tamaño de lote: 32 épocas: 150 lr0: 0,00577 lrf: 0,01019 momento: 0,77283 decaimiento de pesos: 0,00037 data augmentation [1]	71,60	90,63	59,18

Tabla 6.7: Resultados y métricas de los modelos para la tarea de segmentación con sus respectivos hiperparámetros.

[1] Valores obtenidos: hsv_h = 0.01778; hsv_s = 0.69647; hsv_v = 0.46592; perspectiva = 0.0; mosaico = 1.0

Identificada la mejor configuración en YOLOv11, se inicia un tercer experimento secuencial con *tune* (Ultralytics, 2025), el módulo de afinamiento automático de Ultralytics. Este módulo combina una fase inicial de búsqueda aleatoria con una estrategia basada en el rendimiento reciente para guiar la exploración del espacio de hiperparámetros. En este caso, el procedimiento parte de la configuración óptima obtenida previamente con *YOLOv11 + GBFS*, y centra su búsqueda en los hiperparámetros más sensibles o difíciles de ajustar, con el objetivo de afinar aún más el rendimiento del modelo. Para ello, se configuran 100 iteraciones del *tuner*, cada una correspondiente a un entrenamiento con una combinación distinta de los hiperparámetros seleccionados. Los hiperparámetros a manipular con el *tuner* son:

- *Decaimiento del peso* (rango: 0–0,001), para controlar el sobreajuste y mejorar la capacidad de generalización del modelo.
- *Momento* (rango: 0,60–0,98), que ayuda a escapar de mínimos locales. Dado que el mejor valor encontrado fue el valor por defecto, se busca comprobar si existe alguna otra combinación que permita optimizar el rendimiento.
- *Data augmentation* mediante ajustes en el tono (*hsv_h*, rango: 0–0,1), la saturación (*hsv_s*, rango: 0–0,9), el brillo (*hsv_v*, rango: 0–0,9) y la aplicación del efecto mosaico (*mosaico*, rango: 0–1). Estas transformaciones se ajustan porque la rutina de aumento está siempre activa en YOLO y, debido a un error de la librería, no puede desactivarse por completo. Además, la configuración por defecto podría no ser la más adecuada para la detección de caminos. Se decide utilizar únicamente estos efectos para intentar resaltar los caminos en la imagen.

Al analizar los resultados obtenidos, se observa que el *tuner* un descenso en el puntaje F1 (71,60 %), en la precisión (90,63 %) y la sensibilidad (59,18 %) si se compara con el segundo experimento. En este caso, estamos ante un escenario análogo al presentado anteriormente para el experimento 1 (menor puntaje F1, precisión y sensibilidad que el experimento 2), por lo que la combinación YOLOv11 + GBFS muestra un mejor desempeño que YOLOv11 + tuner. La Tabla 6.7 resume los resultados finales de las tres configuraciones evaluadas.

En síntesis, si bien estos modelos aún tienen margen de mejora, YOLOv11 + GBFS se posiciona como el que logra el mejor desempeño, con una baja cantidad de falsos positivos. Es decir, si YOLO segmenta un camino, hay una alta probabilidad de que efectivamente lo sea.

6.1.3. Análisis comparativo

Al examinar la tabla 6.8 se observa una clara ventaja de YOLOv11 sobre Mask R-CNN. YOLOv11 se posiciona como el más confiable, ya que nueve de ca-

da diez veces que predice un camino lo hace correctamente (precisión 92,31 %). El modelo Mask R-CNN, en cambio, apenas supera el 42 % de precisión, lo que equivale a decir que más de la mitad de las veces que predice un camino, este se equivoca.

En cuanto a la *sensibilidad* de YOLOv11, esta indica que no se identifica un tercio de los caminos reales (falsos negativos). En cambio, Mask R-CNN pierde casi seis de cada diez instancias (aproximadamente el doble).

Luego, si se analiza el equilibrio entre precisión y sensibilidad, **YOLOv11** alcanza un puntaje F1 de **76.49 %**, mientras que **Mask R-CNN** se queda en **41.81 %**. Este resultado prueba que YOLOv11 logra un mejor balance entre falsos positivos y falsos negativos. Dicho de otro modo, no solo detecta con menor frecuencia caminos inexistentes, sino que además omite muchos menos caminos reales en comparación con Mask R-CNN.

Métrica	**YOLOv11 (%)**	**Mask R-CNN (%)**	**Diferencia neta**
Puntaje F1	76,5	41,8	+34,7
Precisión	92,3	42,9	+49,4
Sensibilidad	65,3	40,8	+24,5

Tabla 6.8: Visión global de las métricas por instancia (segmentación de caminos).

Si la prioridad operativa reside en no pasar por alto caminos de hormigas, ninguno de los dos modelos alcanza aún la meta deseada, pero YOLOv11 parte de una base mucho más prometedora.

Se entiende que esta distinción se explica por la diferencia en sus arquitecturas: mientras que YOLO pertenece a la familia de algoritmos de una etapa, Mask R-CNN pertenece a la familia de dos etapas. Según el estudio *"Comparing One-Stage and Two-Stage Learning Strategy in Object Detection"* (Bi, Wen, y Xu, 2023), los modelos de una etapa como YOLO son menos propensos a confundirse con el fondo, lo cual es importante para el problema de segmentar caminos en entornos no controlados. Aunque el estudio hace referencia a modelos de detección, se considera que sus conclusiones pueden aplicarse también a la tarea de segmentación, ya que la arquitectura base es la misma en ambos casos, con Mask R-CNN implementado sobre la base de Fast R-CNN. Además, el mismo estudio establece que los modelos de dos etapas requieren una mayor cantidad de datos, lo cual contrasta con el conjunto de datos utilizado, que tan solo cuenta con 1.208 imágenes.

Cabe resaltar que, además, existe un trabajo práctico con Mask R-CNN sobre imágenes satelitales de ciudades (Boesch, 2022), en el cual se menciona

que este modelo resulta más adecuado para identificar campos de béisbol y otras instalaciones deportivas que para detectar calles, ya que los primeros presentan formas tipo gota y no lineales, a diferencia de las calles, cuya estructura se asemeja a los caminos de hormigas (objetos largos y finos en la imagen).

Estas razones nos llevan a utilizar el modelo YOLOv11 para la segmentación de caminos en la arquitectura ARP.

6.2. Evaluación de los modelos de detección

Como parte de la creación de un sistema que permita determinar caminos activos de hormigas, es necesario generar modelos para **detectar hormigas** que se encuentren en él. Para abordar esta tarea, se plantea experimentar con distintos modelos de detección sobre el conjunto de hormigas *Ant Object Detection* (de Gier, 2023), con el fin de determinar el modelo que mejor se ajuste al sistema.

6.2.1. Faster R-CNN

La experimentación comienza con la definición de los hiperparámetros a evaluar durante el ajuste fino del modelo con GBFS. Los hiperparámetros, junto con el espacio de búsqueda de sus valores, se resumen en la Tabla 6.9.

Hiperparámetro	**Valores evaluados**
Backbone	{ResNet-50, ResNet-101, MobileNetV2, SqueezeNet-1_1}
Tamaño de lote	{1, 2, 5, 10, 15}
Tasa de aprendizaje	{0.05, 0.005, 0.0005, 0.00005}
Momento	{0.9, 0.95, 0.98}
Decaimiento de pesos	$\{0.0005, 1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}\}$
Épocas	100

Tabla 6.9: Espacio de hiperparámetros explorado. Estos fueron ajustados en el siguiente orden: tamaño de lote, taza de aprendizaje, momento, decaimiento de pesos.

Para comenzar con la ejecución, se utilizan los siguientes valores, que corresponden a la configuración por defecto del modelo antes de aplicar el proceso de búsqueda:

- Tasa de aprendizaje: 0,005.
- Tamaño de lote: 10.

- Decaimiento de pesos: 0,0005.
- Momento: 0,9.

En esta primera instancia, se mantienen estos valores por defecto y se evalúa qué backbone ofrece el mayor desempeño para establecer una base. Los resultados de esta evaluación, presentados en la Tabla 6.10, muestran que los backbones de la familia ResNet son los que alcanzan el mejor rendimiento.

Backbone	**Puntaje F1** (%)	**Precisión** (%)	**Sensibilidad** (%)
MobileNetV2	58,13	55,33	61,23
SqueezeNet	70,05	77,55	63,87
ResNet50	76,16	74,35	78,05
ResNet101	77,94	77,89	77,98

Tabla 6.10: Comparación de métricas para distintos backbones en la tarea de detección de hormigas. En todos los casos se utiliza una red FPN acoplada al backbone.

El backbone con menor rendimiento es **MobileNetV2**. A pesar de contar con una profundidad de 53 capas (ligeramente superior a las 50 capas de **ResNet-50** su arquitectura está diseñada para la eficiencia computacional y la reducción del tamaño del modelo, mediante el uso de convoluciones separables en profundidad. Si bien esta estrategia arquitectónica reduce significativamente el número de parámetros y operaciones, también disminuye la capacidad del modelo para extraer características semánticamente ricas, también necesarias para la tarea planteada.

En segundo lugar, en términos de rendimiento, se encuentra **SqueezeNet**. Esto es esperado, dado que se trata de la arquitectura menos profunda, con aproximadamente 18 capas. Este diseño prioriza la eficiencia y la reducción de parámetros, lo que limita su capacidad para aprender jerarquías de características complejas y abstractas necesarias para la detección de hormigas.

Dentro de la familia ResNet, el modelo que obtuvo los mejores resultados con los hiperparámetros iniciales fue **ResNet-101**. Esto puede explicarse por su arquitectura más profunda (101 capas), donde una mayor profundidad permite jerarquizar características más complejas y ricas, generando mapas con información semántica de mejor calidad. Este aspecto resulta importante para la detección de objetos en un conjunto de datos como el de las hormigas, donde es necesario captar detalles finos y distinciones sutiles entre objetos para lograr predicciones precisas y confiables.

Continuando con la búsqueda, se evalúan los hiperparámetros restantes utilizando el backbone **ResNet-101**. La tabla 6.11 presenta las configuraciones que producen mejoras en el puntaje F1 con respecto a la iteración anterior.

Tasa de aprendizaje	Tamaño de lote	Decaimiento de pesos	Momento	Puntaje F1 (%)	Precisión (%)	Sensibilidad (%)
0,005	10	0,0005	0,9	77,92	77,89	77,96
0,005	10	0,0005	0,95	78,76	79,33	78,20
0,0005	1	0,0005	0,9	79,78	80,10	79,45

Tabla 6.11: Resultados de las mejores configuraciones de hiperparámetros para ResNet-101.

Como se observa en la tabla 6.11, si bien no hay gran diferencia respecto a las otras combinaciones, el mejor resultado se obtiene con la tercera configuración de ResNet-101. Este resultado surge como una desviación del proceso de GBFS, ya que, aunque se fija en una de las iteraciones la tasa de aprendizaje en 0,005, al experimentar con un tamaño de lote de 1 el entrenamiento no logra converger. Para resolver este problema, se decide probar con una tasa de aprendizaje menor (0,0005) para el tamaño de lote 1. Este ajuste resultó exitoso, no solo estabilizando el entrenamiento, sino también alcanzando un mejor rendimiento, con un puntaje F1 del 79,78 %. Se presume que, en este caso, la reducción en la tasa de aprendizaje es necesaria para evitar la divergencia del modelo ante la alta variabilidad introducida por el tamaño de lote pequeño.

Por último, al analizar los resultados finales, se concluye que, si bien se consigue mejorar el puntaje F1, los valores alcanzados muestran que el modelo todavía no logra adaptarse completamente a la complejidad y diversidad de las características de las hormigas en el conjunto de datos.

6.2.2. YOLO

Al igual que en las secciones anteriores, en este caso se busca ajustar los hiperparámetros del modelo con el objetivo de optimizar su desempeño. Para realizar esta optimización se decide utilizar el método GBFS. Como excepción, se aplica Grid Search exclusivamente sobre los valores de tamaño de lote y tasa de aprendizaje inicial, dado que se entiende que están estrechamente vinculados entre sí. El proceso de ajuste se realiza en el siguiente orden:

- Se realiza GBFS para determinar la variante del modelo que optimiza el desempeño, usando los valores por defecto del mismo:
  - Taza de aprendizaje inicial: 0,01.
  - Taza de aprendizaje final: 0,01.
  - Tamaño de lote: 16.
  - Decaimiento de pesos: 0,0005.

- Luego, se aplica Grid Search con el tamaño de lote y la tasa de aprendizaje inicial.
- Por último, se realizan dos iteraciones de GBFS: primero con la tasa de aprendizaje final y luego con el decaimiento de pesos.

En base a esto, el espacio de hiperparámetros explorado en el procedimiento previamente descrito se resume en la tabla 6.12, mientras que las mejores configuraciones encontradas durante esta exploración se presentan en la tabla 6.13.

Hiperparámetro	**Valores evaluados**
Variante de modelo	{YOLOv8-n, YOLOv8-s, YOLOv8-m, YOLOv8-l, YOLOv11-n, YOLOv11-s, YOLOv11-m, YOLOv11-l}
Tasa de aprendizaje inicial	{ 0.00001, 0.0001, 0.001, 0.01 }
Tamaño de lote	{1, 2, 4, 8, 10, 16}
Tasa de aprendizaje final	{0.00001, 0.0001, 0.001, 0.01, 0.0625, 0.125, 0.25, 0.5, 0.1}
Decaimiento de pesos	{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5}
Épocas	100

Tabla 6.12: Hiperparámetros y valores evaluados durante la búsqueda con YOLO.

Hiperparámetros	**Puntaje F1 (%)**	**Precisión (%)**	**Sensibilidad (%)**
tamaño de lote: 16 lr0: 0,01 lrf: 0,01 decaimiento de pesos: 0,0005	80,05	93,06	70,93
tamaño de lote: 16 lr0: 0,0001 lrf: 0,01 decaimiento de pesos: 0,0005	80,68	92,60	71,48
tamaño de lote: 16 lr0: 0,0001 lrf: 0,0001 decaimiento de pesos: 0,0005	81,29	92,39	72,57

Tabla 6.13: Combinaciones de hiperparámetros con mejor desempeño para la variante YOLOv11-l.

Particularmente, la decisión de realizar una experimentación adicional con el decaimiento de pesos utilizando la tercera configuración de la tabla 6.13 , se basa en evaluar el efecto de este hiperparámetro sobre el rendimiento del modelo, dado que no se observa una mejora sustancial respecto a los parámetros iniciales (primera configuración de la misma tabla). Esta experimentación evidencia que la modificación de este hiperparámetro no conduce a mejoras en el desempeño del modelo, por lo que la tercera configuración sigue siendo la que ofrece los mejores resultados.

Además, en la figura 6.1 se muestra la función de pérdida para distintas configuraciones de decaimiento de pesos. Se observa que los valores de pérdida en el conjunto de validación son superiores a los del conjunto de entrenamiento, lo que sugiere que el modelo podría estar sufriendo un sobreajuste. Siempre se utiliza el cálculo de la pérdida sobre el conjunto de validación, con el objetivo de medir la capacidad del modelo de generalización y no limitarse a los datos de entrenamiento, es decir, ser capaz de predecir correctamente en datos no vistos anteriormente. Una posible causa son las tasas de aprendizaje seleccionadas, ya que sus valores son demasiado bajos y pueden impedir que el modelo se ajuste correctamente a los datos o provocar que lo haga de forma excesivamente lenta.

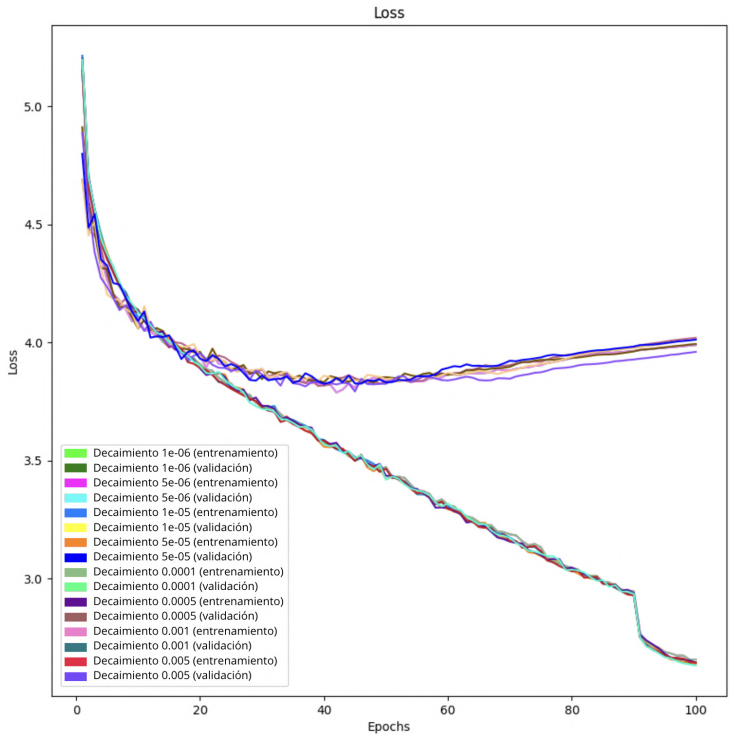

Figura 6.1: Comportamiento de la función de pérdida para distintos valores de decaimiento de pesos

Se considera que este problema puede mitigarse con tamaños de lote mayores, bajo la hipótesis de que un lote más grande permitiría al modelo enfocarse en las características comunes de las distintas especies de hormigas durante el entrenamiento. Esto reduciría el ruido durante la etapa de entrenamiento y podría permitir el uso de tasas de aprendizaje más altas, lo que da como resultado una reducción en el sobreajuste observado. Sin embargo, un tamaño de lote excesivo no siempre mejora el rendimiento y puede incluso perjudicar la

generalización del modelo.

Con respecto al mejor modelo obtenido y sus valores, se puede concluir que el mismo tiene un comportamiento conservador: el modelo sólo predice una hormiga cuando la certeza es alta, dado que la precisión supera a la sensibilidad en aproximadamente 20 puntos porcentuales. Además, el valor del puntaje F1 refleja que, aunque existe una buena relación entre precisión y sensibilidad, la última puede mejorar y, por ende, el modelo se puede optimizar.

6.2.3. Análisis comparativo

Tras obtener los resultados óptimos de la búsqueda de hiperparámetros, se realiza un análisis comparativo de los modelos de detección para seleccionar el que se va a utilizar en el sistema ARP. La tabla 6.14 presenta los mejores resultados de desempeño para cada modelo evaluado.

Métrica	**YOLOv11 (%)**	**Faster R-CNN (%)**	**Diferencia neta**
Puntaje F1	81,29	79,78	+1,51
Precisión	92,39	80,10	+12,29
Sensibilidad	72,57	79,45	–6,88

Tabla 6.14: Visión global de las métricas por instancia para los modelos de detección evaluados.

Al analizar la tabla 6.14 se evidencia que YOLOv11 presenta una ventaja de 1,51 % en puntaje F1 respecto a Faster R-CNN, lo que sugiere una ligera ventaja en su capacidad de detección de hormigas. Aunque esta diferencia es pequeña, las demás métricas revelan contrastes más significativos. En precisión, YOLO supera a Faster R-CNN en unos 12 puntos porcentuales, lo que implica una menor tasa de falsos positivos; cuando el modelo detecta una hormiga, es muy probable que la detección sea correcta. En contraste, Faster R-CNN puede confundir hormigas con otros objetos del entorno con mayor frecuencia. En sensibilidad, Faster R-CNN supera a YOLO en 7 puntos porcentuales, evidenciando una menor probabilidad de omitir hormigas presentes en la imagen.

Un aspecto importante a considerar al elegir el modelo de detección a integrar en el sistema ARP, es la priorización de la *precisión* sobre las otras métricas complementarias (sensibilidad y puntaje F1). Las zonas más relevantes en un camino son aquellas con mayor actividad de hormigas, lo que implica que, cuando hay muchas hormigas presentes, una *sensibilidad* más baja no resulta tan problemática. Esto se debe a que, aunque no se detecten todas las hormigas, las inferencias positivas son suficientes para identificar correctamente las áreas

con actividad. En cambio, un modelo con baja *precisión* podría señalar como activas zonas donde en realidad no hay hormigas, lo que terminaría afectando la confianza en el sistema.

Modelo	Tiempo total (s)	Tiempo medio por imagen (ms)
YOLOv11	15,83	15,97
Faster R-CNN (ResNet-101)	33,76	34,07

Tabla 6.15: Velocidad promedio de inferencia, calculada en el mismo equipo[1], sobre el conjunto de pruebas (991 imágenes).

Finalmente, la tabla 6.15 muestra que YOLO ofrece una mayor velocidad de inferencia en comparación con las demás arquitecturas evaluadas, lo cual es un factor importante si se desea que el sistema ARP opere en tiempo real. Si bien estos tiempos se calcularon en un dispositivo con una alta capacidad de cómputo, cabe resaltar que YOLOv11 infiere en la mitad de tiempo que Faster R-CNN. Esta diferencia se atribuye a que YOLO emplea una arquitectura de una sola etapa que realiza detección de manera más eficiente, mientras que modelos como Faster R-CNN requieren dos etapas y son más lentos. Esto se debe tener en cuenta en el momento de la elección del modelo a utilizar si el entorno en que opera el sistema tiene recursos limitados.

6.3. Evaluación del sistema ARP

En esta instancia se evalúan distintas combinaciones del sistema ARP sobre el conjunto de datos presentado en la sección 4.1.4.2 para el mejor modelo de segmentación y detección (YOLOv11 en ambos casos), con el objetivo de determinar la configuración que lo optimiza. Para ello, se ejecuta el sistema sobre el conjunto de datos generado y se procesa la información para identificar qué caminos están activos.

En un principio, se realiza una prueba para contrastar el comportamiento del pipeline sin modificaciones con una versión que activa SAHI en el módulo de detección de hormigas. Se parte de la hipótesis de que las hormigas dentro del conjunto de datos ocupan una menor proporción de espacio en la imagen respecto al conjunto de datos de entrenamiento utilizado (ver figura 4.3); lo que puede ocasionar que los modelos entrenados con estos datos sean más propensos a ignorar hormigas en el contexto descrito.

[1] PC de escritorio con Windows 11 Pro, procesador Intel Core i7-14700F de 14ª generación (2.10 GHz), GPU Nvidia Gigabyte RTX 4070 con 12 GB de VRAM y 64 GB de RAM.

Métrica	Estándar (%)	Con SAHI (%)	Diferencia neta
Puntaje F1	19,48	45,56	+26,08
Precisión	62,14	51,40	-10,74
Sensibilidad	39,14	46,46	+7,32

Tabla 6.16: Comparación de métricas entre el pipeline estándar y el pipeline con SAHI en el módulo de detección. Las métricas corresponden a los promedios macro de la segmentación multiclase de caminos activos e inactivos.

Al analizar los resultados de la tabla 6.16, se comprueba que la hipótesis planteada se cumple, ya que los resultados del puntaje F1 obtenidos con SAHI superan en 26 puntos porcentuales a los del pipeline sin modificaciones. Esto se explica porque, al utilizar métricas macro, aunque la precisión general de la configuración con SAHI disminuye (en 10 puntos porcentuales), los resultados por clase de precisión y sensibilidad son más equilibrados. Este mejor balance entre clases se refleja en una mejora del puntaje F1.

Rango de AR	Filas	Columnas	Superposición ancho	Superposición alto
$0,15 \leq \text{AR} \leq 1,85$	3	3	0,25	0,25
$0,07 \leq \text{AR} < 0,15$	3	2	0,25	0,375
$1,85 < \text{AR} \leq 1,92$	2	3	0,375	0,25
$0 < \text{AR} < 0,7$	3	1	0,25	0,50
$\text{AR} > 1,92$	3	1	0,50	0,25

Tabla 6.17: Configuraciones manuales de SAHI según el rango de relación de aspecto (AR). Dependiendo del AR de la caja delimitadora de la máscara, la imagen se fragmenta en el número especificado de filas y columnas para generar los parches. Además, se define el factor de superposición entre parches en función del ancho y alto.

En base a esto, se establece el uso de SAHI en el módulo de detección de hormigas y se plantea realizar Grid Search sobre las posibles configuraciones del sistema. En particular, se explora:

- Factor de nitidez: para evaluar si la mejora en la claridad de las imágenes implica una detección más precisa de hormigas por el sistema.
- Búsqueda en máscara: se aplica el módulo de detección de hormigas con SAHI activo a la mascara del camino para determinar si la detección aislada dentro del mismo produce buenos resultados.

- Modo: se prueba el funcionamiento de SAHI con la configuración por defecto frente a una configuración específica (ver tabla 6.17).

Como punto de partida, se establece un factor de expansión de la máscara igual a 1, lo que equivale a extenderla 100 píxeles alrededor de la máscara del camino. Asimismo, se fija un porcentaje de superposición del 25 % tanto en alto como en ancho para los parches generados por SAHI.

Los resultados de la tabla 6.18 posicionan a la primera configuración como la mejor. Esto se debe a que aplicar el algoritmo de nitidez sobre la imagen tiende a degradar el desempeño general del modelo. En particular, las métricas de detección de hormigas se ven afectadas negativamente cuando se incrementa el nivel de nitidez, como se observa en la tabla 6.19. Este descenso en el rendimiento se explica porque, si bien algunas hormigas aparecen visualmente más definidas, el aumento del factor de nitidez también intensifica el parecido con el fondo, dificultando su diferenciación, como se ilustra en la figura 6.2. Además, se observa que la variación en el nivel de nitidez no afecta los resultados en la tarea de segmentación de caminos (mismos valores en ambos casos).

Factor de nitidez	**Búsqueda en máscara**	**Modo**	**Puntaje F1 (%)**	**Precisión (%)**	**Sensibilidad (%)**
1	No	Defecto	45,56	51,40	46,46
1	Si	Defecto	35,20	39,39	40,66
1	Si	Específica	35,20	39,39	40,66
5	No	Defecto	37,94	46,07	37,62
5	Si	Defecto	17,60	22,12	17,68
5	Si	Específica	16,77	17,79	21,97

Tabla 6.18: Resultados macro de la segmentación multiclase utilizando Grid Search para determinar los caminos activos, con SAHI activado en el módulo de detección de hormigas.

Tarea	**Factor de nitidez**	**Puntaje F1 (%)**	**Precisión (%)**	**Sensibilidad (%)**
Segmentación de caminos	1	67.36	90.28	53.72
Segmentación de caminos	5	67.36	90.28	53.72
Detección de hormigas	1	57.54	77.41	45.79
Detección de hormigas	5	40.37	49.51	34.08

Tabla 6.19: Resultados de las tareas de segmentación de caminos y detección de hormigas en la imagen completa, evaluando el efecto del factor de nitidez. Un valor de 1 indica que no se aplicó ningún ajuste de nitidez (valor neutro).

(a) Imagen sin aumento de nitidez. (b) Imagen con aumento de nitidez.

Figura 6.2: Comparación visual entre la imagen original (izquierda) y una con nitidez aumentada (derecha). El aumento de nitidez intensifica detalles del fondo, lo que puede dificultar la detección de hormigas.

A su vez, la búsqueda sobre las máscaras también afecta los resultados. Se presume que esto ocurre debido a que la imagen obtenida tiene dimensiones distintas a la original. Esto provoca que la subdivisión que realiza SAHI no sea la más efectiva y, en consecuencia, no detecte de manera eficiente las hormigas, afectando así el correcto funcionamiento del pipeline.

En base a esto, se realiza una segunda búsqueda como se muestra en la tabla 6.20, fijando la mejor configuración obtenida hasta el momento (tabla 6.18) y experimentando con el factor de expansión de la máscara del camino, el umbral de hormigas para considerar un camino como activo y el valor de superposición de parches utilizado por SAHI, con el objetivo de mejorar el resultado anterior.

Superposición	**Factor de expansión**	**Umbral de hormigas**	**Puntaje F1 (%)**	**Precisión (%)**	**Sensibilidad (%)**
0	1	1	45,56	51,40	46,46
0,125	1	1	45,56	51,40	46,46
0,375	1	1	45,56	51,40	46,46
0,5	1	1	45.56	51,40	46,46
0,25	0	1	42,96	49,24	45,96
0,25	2	1	47,91	54,10	46,97
0,25	2	3	36,66	45,74	39,93
0,25	2	5	29,90	45,34	30,61

Tabla 6.20: Resultados macro de la segmentación multiclase para la detección de caminos activos utilizando SAHI en su configuración por defecto; sin búsqueda en máscara y con el factor de nitidez desactivado.

La mejor combinación está dada por un factor de expansión de 2, un valor mínimo de hormigas de 1, y manteniendo el valor por defecto de 0,25 para la superposición de parches de SAHI, ya que las variaciones en este último no generan mejoras. Se sospecha que el aumento del factor de expansión contribuye a la mejora de las métricas, ya que incrementa el área de superposición entre la máscara predicha por el modelo de segmentación y la máscara real, lo que permite asociar correctamente un mayor número de hormigas presentes en el camino.

Por otro lado, se observa un decaimiento en las métricas al aumentar el número mínimo de hormigas necesarias para determinar si un camino está activo (umbral de hormigas). Este deterioro parece estar relacionado con la baja sensibilidad observada en la tarea de detección de hormigas dentro de los caminos (ver tabla 6.21).

Tarea	**Puntaje F1** (%)	**Precisión** (%)	**Sensibilidad** (%)
Segmentación de caminos de hormigas	67,36	90,28	53,72
Detección de hormigas en la totalidad de la imagen	57,54	77,41	45,79
Detección de hormigas en la máscara del camino	20,92	81,62	12,00

Tabla 6.21: Resultados globales de tarea para caminos activos con factor de expansión 2 para la configuración de SAHI.

Para la mejor configuración obtenida, si se analiza el comportamiento del sistema al procesar cada tarea de forma independiente (ver tabla 6.21), se observa que la detección de hormigas dentro del camino presenta una baja tasa de falsos positivos, con una tasa de acierto de 8 de cada 10 hormigas. Sin embargo, la sensibilidad de esta detección es baja: aproximadamente 9 de cada 10 hormigas presentes en el camino no son detectadas.

Este bajo rendimiento puede atribuirse a dos factores principales. En primer lugar, se encuentra la pérdida de caminos segmentados, ya que la sensibilidad en la tarea de segmentación de caminos es del 53,72 % (es decir, 5 de cada 10 caminos no son identificados). Esto impacta directamente en la detección de hormigas sobre la máscara del camino, ya que si un camino no es segmentado, el algoritmo de detección no se aplica sobre esa región y, por tanto, todas las hormigas que podrían encontrarse allí quedan fuera del análisis, afectando negativamente las métricas.

En segundo lugar, influye la diferencia entre los datos de entrenamiento y los del conjunto de evaluación del sistema ARP. En este último, las hormigas tien-

den a confundirse con la tierra del fondo o a difuminarse debido al movimiento. Además, también afecta el factor obstrucción cuando las hormigas regresan cargadas con alimento o se superponen entre sí. Una situación similar se da en el conjunto de datos utilizado para el ajuste del modelo de segmentación de caminos: las imágenes de entrenamiento fueron capturadas a una distancia superior a un metro, mientras que las imágenes de evaluación del sistema ARP fueron tomadas a distancias entre 30 y 50 cm, lo que busca priorizar la visibilidad de las hormigas. Esta diferencia en escala y perspectiva también repercute en el desempeño de ambos modelos.

En base a las observaciones realizadas anteriormente, resulta útil llevar a cabo una prueba aislada que permita analizar el comportamiento de la detección de hormigas dentro del camino, sin que los resultados se vean afectados por la pérdida en las segmentaciones de caminos. Para ello, se propone ejecutar SAHI sobre la imagen completa, pero utilizando las máscaras etiquetadas en lugar de las predicciones para realizar la intersección con las cajas delimitadoras predichas para las hormigas.

En este contexto, se obtienen los resultados presentados en la tabla 6.22, donde se observa que el modelo es capaz de detectar hormigas dentro del camino con un puntaje F1 de hasta 60.94 % en el mejor de los casos. Este valor se alcanza con una nitidez neutra y sin aplicar ningún factor de expansión sobre el camino. Si bien se obtienen resultados similares cuando la nitidez es neutra, se evidencia un deterioro en el desempeño al aumentar el factor de nitidez. Esto se atribuye a que, al aumentar la nitidez, las hormigas tienden a confundirse con el fondo, lo cual afecta negativamente la sensibilidad del modelo, reduciéndola en aproximadamente 10 puntos porcentuales.

Nitidez	**Expansión**	**Puntaje F1 (%)**	**Precisión (%)**	**Sensibilidad (%)**
1	1.0	60.43	88.57	45.86
5	1.0	48.55	82.86	34.34
1	2.0	60.21	87.65	45.86
1	0.0	60.94	90.83	45.86

Tabla 6.22: Resultados de aplicar SAHI sobre imagen utilizando las mascaras de referencia de los caminos en lugar de las predicciones con una superposición de parches de 0,25.

En base al resultado obtenido en esta última prueba, se puede afirmar que la pérdida de caminos para segmentación en el pipeline ocasiona un deterioro en el puntaje F1 para la detección de hormigas dentro del camino si se compara con los resultados de la tabla 6.21. Además, se registra una caída de aproximadamente 20 puntos porcentuales en dicho puntaje en relación con los valores obtenidos durante la experimentación con modelos de detección basados en YO-

LO, entrenados sobre el conjunto de datos extraído de Roboflow (ver sección 6.2.2).

Finalmente, se realiza una última prueba con el objetivo de evaluar la velocidad de inferencia del pipeline con la mejor configuración obtenida (ver Tabla 6.20). Para ello, se mide el tiempo de inferencia del modelo sobre el conjunto de evaluación previamente mencionado, luego de una etapa de calentamiento consistente en 10 iteraciones sobre dicho conjunto. Los resultados obtenidos se presentan en la Tabla 6.23.

	Tiempo total (s)	Tiempo medio por imagen (s)
ARP	601,30	4,93

Tabla 6.23: Velocidad promedio de inferencia, calculada en el mismo equipo[1], sobre el conjunto de evaluación (122 imágenes).

Estos resultados muestran que el pipeline es capaz de procesar aproximadamente 12 imágenes por minuto. Este rendimiento se explica porque el sistema realiza las inferencias de forma secuencial. Además, la fragmentación de la imagen introducida por SAHI genera mayor latencia, ya que se infiere individualmente sobre cada parche generado. Se considera que este tiempo podría reducirse mediante la paralelización de las tareas de segmentación y detección de instancias.

En síntesis, se concluye que es necesario mejorar los conjuntos de datos utilizados para el entrenamiento, de forma que los modelos puedan adaptarse mejor al contexto y representar con mayor fidelidad las condiciones reales observadas durante la evaluación. Esto permitiría mejorar tanto la capacidad del modelo de detección para identificar hormigas en escenarios complejos, como el desempeño del modelo de segmentación de caminos en imágenes tomadas a menor altura.

[1] PC de escritorio con Windows 11 Pro, procesador Intel Core i7-14700F de 14ª generación (2.10 GHz), GPU Nvidia Gigabyte RTX 4070 con 12 GB de VRAM y 64 GB de RAM.

Capítulo 7

Conclusiones y trabajos futuro

Este capítulo presenta un resumen de los principales resultados obtenidos durante el desarrollo del sistema ARP, así como un análisis de sus limitaciones y potenciales mejoras.

7.1. Conclusiones

En primera instancia, se destaca la creación de dos conjuntos de datos en el marco de este proyecto: uno destinado al entrenamiento de los modelos de segmentación de caminos de hormigas, y otro orientado a la evaluación del sistema ARP. Ambos conjuntos están compuestos por imágenes capturadas en entornos no controlados, lo que implica un esfuerzo considerable, ya que requiere realizar las capturas, validarlas y llevar a cabo el etiquetado correspondiente. Para facilitar esta última etapa, se integran modelos en la plataforma CVAT, lo que permite automatizar parcialmente el etiquetado de hormigas y caminos.

A partir de estos conjuntos de datos, se entrenan modelos de detección y segmentación que conforman un pipeline capaz de estimar la actividad en caminos de hormigas. No obstante, los experimentos revelan que su fiabilidad está limitada por las características del conjunto de datos utilizado durante la etapa de entrenamiento.

Respecto a la segmentación de caminos (tabla 7.1), YOLOv11 funciona correctamente cuando procesa imágenes del conjunto de datos específico para los caminos de hormigas (ver sección 4.1.4.1): alcanza un puntaje F1 de 76,5 % y una precisión de 92,30 %. No obstante, su desempeño se reduce cuando se em-

plea YOLOv11 en el sistema ARP con el conjunto de evaluación descrito en la subsección 4.1.4.2, donde los caminos suelen aparecer cerca de la cámara. Se entiende que la escasez de capturas a baja o media altura en el conjunto de entrenamiento limita la capacidad de generalización del modelo, lo que genera una pérdida de la mitad de las segmentaciones de caminos (la sensibilidad es de 53,72 %).

Una posible solución consiste en ampliar dicho conjunto con imágenes obtenidas a distintas alturas, ángulos y tipos de superficie, con el objetivo de reducir los falsos negativos. Esto se vuelve especialmente necesario dado que el tamaño actual del conjunto de datos no es suficiente para que el modelo generalice correctamente. Por lo tanto, es fundamental no solo incrementar la cantidad de imágenes, sino también su diversidad, para mejorar la capacidad del modelo de adaptarse a distintas condiciones del entorno.

Métrica	**Segmentación aislada con YOLOv11 (%)**	**En conjunto con ARP (%)**
Puntaje F1	76,50	67,36
Precisión	92,30	90,28
Sensibilidad	65,30	53,72

Tabla 7.1: Comparativa de métricas para la segmentación de caminos con YOLOv11 de forma independiente y con YOLOv11 integrado en el sistema ARP. Cada evaluación se realizan con su respectivo conjunto de datos.

En cuanto a la detección de hormigas (tabla 7.2), el modelo YOLOv11 entrenado con imágenes públicas de Roboflow también logra un puntaje F1 de 81,29 % y precisión de 92,39 %. Sin embargo, cuando la detección se realiza con ARP dentro del camino, el desempeño disminuye considerablemente: el puntaje F1 desciende hasta aproximadamente 20,92 %, debido principalmente a la caída en la sensibilidad, que se reduce a 12,00 %. La falta de ejemplares de Atta y Acromyrmex, y de un contexto natural en el conjunto de entrenamiento muestra que el modelo no generaliza: hormigas cargadas de hojas o camufladas en el suelo no se detectan, y la etapa clave del pipeline, que es validar la actividad dentro del camino, se convierte en la mayor limitante. A su vez, como se menciona anteriormente, la pérdida en segmentación de caminos ocasiona que el sistema no ejecute el algoritmo de detección de hormigas dentro del camino, lo que influye negativamente en la sensibilidad y, por ende, en el puntaje F1.

Si bien los resultados obtenidos en estas evaluaciones no son óptimos, se espera una mejora en el rendimiento del sistema al operar en un contexto donde recibe imágenes de forma continua, por ejemplo, a razón de 30 capturas por segundo. Actualmente, el sistema ARP se evalúa utilizando imágenes individuales en un contexto estático, lo cual no refleja completamente las condiciones

Métrica	Detección aislada con YOLOv11 (%)	ARP – detección de hormigas (%)	ARP – detección de hormigas en el camino (%)
Puntaje F1	81,29	57,54	20,92
Precisión	92,39	77,41	81,62
Sensibilidad	72,57	45,79	12,00

Tabla 7.2: Comparativa de métricas para la detección de hormigas: con YOLOv11 de forma independiente, integrado en el sistema ARP sobre toda la imagen, e integrado en ARP considerando únicamente los caminos. Cada evaluación se realizan con su respectivo conjunto de datos.

reales de uso. Sin embargo, en condiciones operativas reales, la mayor frecuencia de observación permitiría incrementar las oportunidades de detección tanto de caminos como de hormigas.

En síntesis, la segmentación conserva un puntaje F1 aceptable (67 %), pero la detección de hormigas dentro de los caminos se deteriora, y el balance global del pipeline apenas alcanza el 20,92 % incluso con la mejor configuración. Aunque los algoritmos y los hiperparámetros ya ofrecen un punto de partida sólido, el rendimiento podría mejorar significativamente al incrementar y diversificar los conjuntos de datos, como se sugiere a continuación.

7.2. Líneas futuras de investigación

A partir de las experiencias y resultados obtenidos durante el desarrollo de este proyecto, se identifican varias líneas de trabajo que podrían explorarse para mejorar la efectividad y aplicabilidad del sistema ARP en escenarios reales. Estas posibles extensiones se orientan principalmente a la mejora de los conjuntos de datos, la optimización de los modelos, la paralelización de procesos y la integración con nuevas herramientas de anotación.

Creación de un conjunto de datos de hormigas ajustado a la realidad del problema

Uno de los principales desafíos encontrados en este trabajo fue la falta de conjuntos de datos públicos que incluyeran hormigas cortadoras de hojas, incluso en los disponibles a través de plataformas como Roboflow. La construcción de un conjunto de datos propio que refleje con precisión las condiciones reales del problema (hormigas cortadoras en ambientes no controlados dentro de caminos y con distintas alturas y perspectivas de cámara) implicaría un esfuerzo

considerable en términos de tiempo, recursos y logística. Debido a esto, se opta por utilizar el conjunto de datos de Roboflow como punto de partida, aun cuando no representara completamente la realidad concreta. Por ello, una línea de trabajo fundamental es crear un conjunto de datos específico, capturado con cámaras que minimicen el desenfoque y la difuminación, para asegurar imágenes con mayor nitidez que favorezcan tanto la anotación como el entrenamiento del modelo.

Mejora del conjunto de datos de caminos para optimizar la segmentación

En el caso de la segmentación de caminos, si bien se alcanzan resultados satisfactorios, la variación en la altura de las imágenes y el tamaño limitado del conjunto de datos constituyen una condicionante para el modelo. Por este motivo, se propone ampliar el conjunto, incorporando imágenes tomadas desde diferentes alturas (entre veinte y cincuenta centímetros) y perspectivas, así como aumentando la diversidad de escenarios. En particular, es deseable incluir terrenos con distintas texturas y materiales, como caminos de hormigas sobre arena, gravilla o tierra, para que el modelo generalice mejor en distintos ambientes.

Avances en la detección de hormigas

Para incrementar la capacidad del sistema de detectar hormigas individuales con mayor precisión, una posible mejora sería incorporar técnicas de súper resolución. Como se describe en el documento de estado del arte, donde se aplica súper resolución para mejorar la detección de automóviles en imágenes satelitales, este enfoque podría adaptarse para mejorar la calidad de las imágenes de hormigas y facilitar su identificación en condiciones de baja resolución.

Asimismo, se sugiere explorar la optimización del modelo mediante técnicas como Sliding Aided Hyper Inference and Fine-Tuning (SAHIF), para mejorar el desempeño en entornos complejos y aumentar la eficiencia computacional.

Paralelización de la etapa de detección y segmentación en ARP

Actualmente, las etapas de detección y segmentación en el pipeline ARP se ejecutan de forma secuencial, lo cual puede limitar el rendimiento en tiempo real. De hecho, las pruebas realizadas muestran que el sistema procesa aproximadamente 12 imágenes por minuto, con un tiempo medio de inferencia por imagen de 4,93 segundos. Se entiende que una forma de optimizar este tiempo es paralelizar las tareas de segmentación y detección de instancias, permitiendo que ambas se ejecuten simultáneamente. Esto podría reducir los tiempos de respuesta del sistema y mejorar su aplicabilidad en escenarios con restricciones temporales más exigentes.

Soporte a CVAT con SAM-CLIP y prompts de texto

Se propone agregar soporte a CVAT con SAM-CLIP mediante prompts de

texto, para automatizar la preanotación de caminos de hormigas a partir de descripciones. Se realizan pruebas preliminares que muestran resultados positivos que dependen de la calidad de los prompts. Para más detalles sobre las pruebas realizadas, se puede consultar el anexo A.

Integración de SAM-CLIP al sistema ARP

Se plantea experimentar con la integración de SAM-CLIP en el pipeline ARP. La incorporación de SAM puede mejorar la segmentación de los caminos al afinarlo con LoRA, mientras que CLIP, con su capacidad de relacionar imágenes y texto, podría enriquecer la identificación de hormigas y caminos mediante descripciones más precisas. Esta combinación presenta la posibilidad de mejorar significativamente la detección y segmentación del sistema, mejorando la fiabilidad del mismo.

Evaluación del sistema como clasificador de estados combinados

El sistema ARP infiere a partir de las imágenes si un camino está activo o inactivo, y si existe o no actividad de hormigas. Actualmente, en la evaluación de la segmentación multiclase basada en estos estados, solo se consideran los casos de camino activo e inactivo. Sin embargo, resultaría valioso extender el análisis para incluir el estado de actividad de hormigas en la imagen. Para ello, sería necesario construir un conjunto de datos de evaluación balanceado en cuanto a las distintas combinaciones de categorías, lo que permitiría evaluar de forma más completa y satisfactoria la tarea de clasificación del sistema.

Bibliografía

Akyon, F. C., Altinuc, S. O., y Temizel, A. (2022). Slicing aided hyper inference and fine-tuning for small object detection. *CoRR*, *abs/2202.06934*. Descargado de https://arxiv.org/abs/2202.06934

Berois, M., de Oliveira, L., y Gastelú, E. (2024). *Desarrollo de un oso hormiguero artificial* (Proyecto de grado, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay). Descargado de https://gitlab.fing.edu.uy/eduardo.gastelu/pg_robot_hormiguero/-/raw/main/Tesis/tesis.pdf?ref_type=heads&inline=false (Supervisores: Gonzalo Tejera y Guillermo Trinidad)

Bi, H., Wen, V., y Xu, Z. (2023). Comparing one-stage and two-stage learning strategy in object detection. En *Applied and computational engineering proceedings* (p. 556). Descargado de https://doi.org/10.54254/2755-2721/5/20230556 doi: 10.54254/2755-2721/5/20230556

Bochkovskiy, A., Wang, C.-Y., y Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*. Descargado de https://arxiv.org/abs/2004.10934

Boesch, G. (2022). *Everything about mask r-cnn: A beginner's guide.* Descargado de https://viso.ai/deep-learning/mask-r-cnn/ (Viso Suite blog)

Consortium, C. (s.f.). *Coco: Common objects in context.* Descargado de https://cocodataset.org/ (Accedido en julio de 2025)

de Gier, D. (2023, sep). *Ant object detection dataset* [Open Source Dataset]. Roboflow. Descargado de https://universe.roboflow.com/djay-de-gier-fopbf/ant-object-detection (visited on 2025-07-09)

Dell'Onte, H. (2023, 21 de junio). *El impacto de las hormigas en la producción agrícola y forestal.* La Mañana, sección Rurales. Descargado de https://www.xn--lamaana-7za.uy/agro/el-impacto-de-las-hormigas-en-la-produccion-agricola-y-forestal/ (Publicado el 21 de junio de 2023)

Dos Santos, A., Biesseck, B. J. G., Latte, N., de Lima Santos, I. C., dos Santos, W. P., Zanetti, R., y Zanuncio, J. C. (2022). Remote detection and measurement of leaf-cutting ant nests using deep learning and an unmanned aerial vehicle. *Computers and Electronics in Agriculture*, *198*, 107071. Descargado de https://doi.org/10.1016/j.compag.2022.107071

Elgendy, M. (2020). *Deep learning for vision systems.* Shelter Island, NY:

Manning Publications.

Erdem, B. (2020). *Understanding region of interest – part 2: Roi align.* Descargado de https://erdem.pl/2020/02/understanding-region-of-interest-part-2-ro-i-align (Accessed: 2025-07-24)

Food, y Organization, A. (2019, 8 de julio). *FAO/OECD: Latin America and the Caribbean will account for 25 % of global agricultural and fisheries exports by 2028.* Descargado de https://www.fao.org/americas/news/news-detail/FAO-OECD-Latin-America-and-the-Caribbean-will-account-for-25-of-global-agricultural-and-fisheries-exports-by-2028/en?

Food and Agriculture Organization of the United Nations. (2025). *Integrated pest management (ipm).* Descargado de https://www.fao.org/pest-and-pesticide-management/ipm/integrated-pest-management (Accedido el 24 de julio de 2025)

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning.* MIT Press. (https://www.deeplearningbook.org)

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., y Girshick, R. (2021). Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*. Descargado de http://arxiv.org/abs/2111.06377

He, K., Gkioxari, G., Dollár, P., y Girshick, R. (2017). Mask r-cnn. *arXiv preprint arXiv:1703.06870*. Descargado de https://arxiv.org/abs/1703.06870 (Versión revisada en enero de 2018) doi: 10.48550/arXiv.1703.06870

He, K., Zhang, X., Ren, S., y Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*. Descargado de https://arxiv.org/abs/1512.03385 (Submitted 10 Dec 2015; revised version also disponible) doi: 10.48550/arXiv.1512.03385

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*. Descargado de https://arxiv.org/abs/1704.04861 (Presenta los dos hiperparámetros: multiplicador de ancho y resolución) doi: 10.48550/arXiv.1704.04861

(IAEA), I. A. E. A. (2025). *Pesticide use in latin america: Trends and environmental implications* (Technical Report). IAEA / INIS. Descargado de https://inis.iaea.org/records/4d5cq-qwz88 (Record ID: 4d5cq-qwz88 at INIS)

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., y Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5mb model size. *arXiv preprint arXiv:1602.07360*. Descargado de https://arxiv.org/abs/1602.07360 doi: 10.48550/arXiv.1602.07360

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., ... Girshick, R. (2023). Segment anything. Descargado de https://arxiv.org/abs/2304.02643 doi: 10.48550/arXiv.2304.02643

LearnPyTorch.io. (2023). *Introduction to tensors - learn pytorch.* Descargado de https://www.learnpytorch.io/00_pytorch_fundamentals/#introduction-to-tensors (Accedido: 2025-07-16)

Li, R., Wang, R., Zhang, J., Xie, C., Liu, L., Wang, F., ... Liu, W. (2019). An effective data augmentation strategy for cnn-based pest localization and recognition in the field. *IEEE Access*, *7*, 160274-160283. Descargado de https://www.researchgate.net/publication/336858918_An_Effective_Data_Augmentation_Strategy_for_CNN-Based_Pest_Localization_and_Recognition_in_the_Field doi: 10.1109/ACCESS.2019.2949852

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., y Belongie, S. (2016). Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*. Descargado de https://arxiv.org/abs/1612.03144 (Versión revisada 19 de abril de 2017) doi: 10.48550/arXiv.1612.03144

Nadile, G., Marr, M., y Perera, T. (2025, Junio). *Oso hormiguero artificial: tecnología autónoma para monitoreo y control de plagas en entornos agropecuarios. estado del arte.* Descargado de https://gitlab.fing.edu.uy/tatiana.perera.iturralde/oso-hormiguero/-/blob/main/documents/Estado%20del%20Arte.pdf (Informe de Proyecto de Grado, Facultad de Ingeniería, Universidad de la República)

Pateria, S., Subagdja, B., Tan, A. H., y Quek, C. (2021, jun). Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys*, *54*(5), 1–35. Descargado de https://www.researchgate.net/publication/352160708_Hierarchical_Reinforcement_Learning_A_Comprehensive_Survey (Article 3453160) doi: 10.1145/3453160

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... others (2021). Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*. Descargado de https://arxiv.org/abs/2103.00020

Ren, S., He, K., Girshick, R., y Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *CoRR*, *abs/1506.01497*. Descargado de https://arxiv.org/abs/1506.01497

Roosjen, P. P., Kellenberger, B., Kooistra, L., Green, D. R., y Fahrentrapp, J. (2020). Deep learning for automated detection of drosophila suzukii: potential for uav-based monitoring. *Pest Management Science*, *76*(9), 2994-3002. Descargado de https://doi.org/10.1002/ps.5845

Sabattini, J. A., Sturniolo, F., Bollazzi, M., y Bugnon, L. A. (2023). Anttracker: A low-cost and efficient computer vision approach to research leaf-cutter ants behavior. *Smart Agricultural Technology*, *5*, 100252. Descargado de https://doi.org/10.1016/j.atech.2023.100252

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., y Chen, L. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. Descargado de https://arxiv.org/abs/1801.04381 doi: 10.48550/arXiv.1801.04381

Sheppard, T. (s.f.). *Detectorevaluator.py — proyecto de grado: Calidad de manzanas y navegación.* Descargado de https://gitlab.fing.edu.uy/thomas.sheppard/proyecto-de-grado-calidad-de-manzanas-y-navegacion/-/blob/main/detection/DetectorEvaluator.py?ref_type=heads (Accedido: julio 2025)

Sommerville, I. (2016). *Software engineering* (10.ª ed.). Boston: Pearson.

Stereolabs. (2023). *Performance of yolo v5, v7, and v8.* Descargado de https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8 (Accessed: 2025-07-24)

Sucar, L. E., y Gómez, G. (2020). *Visión computacional.* Puebla, México: Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE). Descargado de https://ccc.inaoep.mx/~esucar/Libros/vision-sucar-gomez.pdf

Szeliski, R. (2010). *Computer vision: Algorithms and applications.* Springer. Descargado de http://szeliski.org/Book/

Szeliski, R. (2022). *Computer vision: Algorithms and applications* (2.ª ed.). London, UK: Springer Nature.

Tafur, F. P. O. (2019). *Evaluación de la eficacia de producto natural para el control de la hormiga arriera (*atta cephalotes*)* (Informe técnico - trabajo de grado). Universidad Nacional Abierta y a Distancia (UNAD). Descargado de https://repository.unad.edu.co/bitstream/handle/10596/30193/41963752.pdf (Contiene estadísticas referenciadas a Montoya (2019))

Teixeira, A. C., Ribeiro, J., Morais, R., Sousa, J., y Cunha, A. (2023). *A systematic review on automatic insect detection using deep learning* (Vol. 13) (n.º 3). Descargado de https://www.mdpi.com/2077-0472/13/3/713 doi: 10.3390/agriculture13030713

Ultralytics. (2023). *ultralytics.data.augment.letterbox.* Descargado de https://docs.ultralytics.com/reference/data/augment/#ultralytics.data.augment.LetterBox (Versión de referencia del método LetterBox, accedido en julio de 2025)

Ultralytics. (2024). *Yolov8 architecture: Deep dive into its architecture.* YOLOv8. Descargado de https://yolov8.org/yolov8-architecture/ (Consultado el 2 de marzo de 2025)

Ultralytics. (2025). *Basepredictor.pre_transform.* Descargado de https://docs.ultralytics.com/reference/engine/predictor/#ultralytics.engine.predictor.BasePredictor.pre_transform (Accessed: 2025-03-28)

Ultralytics. (2025). *Tuner.* Descargado de https://docs.ultralytics.com/reference/engine/tuner/ (Accedido el 4 de julio de 2025)

Ultralytics Team. (2025a). *Detection model class reference.* Descargado de https://docs.ultralytics.com/reference/nn/tasks/#ultralytics.nn.tasks.DetectionModel (Ultralytics YOLO Documentation. Accessed 2025-07-08)

Ultralytics Team. (2025b). *Yolov5 vs yolov9 — performance and benchmarks.* Descargado de https://docs.ultralytics.com/compare/yolov5-vs-yolov9/#performance-and-benchmarks-yolov5-vs-yolov9 (Accedido el 8 de julio de 2025)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En *Advances in neural information processing systems* (Vol. 30). Descargado de https://arxiv.org/abs/1706.03762

Wang, H., Vasu, P. K. A., Faghri, F., Vemulapalli, R., Farajtabar, M., Mehta, S., ... Pouransari, H. (2023). *Sam-clip: Merging vision foundation models towards semantic and spatial understanding.* Descargado de `https://arxiv.org/abs/2310.15308` (arXiv preprint arXiv:2310.15308)

Wu, M., Cao, X., y Guo, S. (2020). *Accurate detection and tracking of ants in indoor and outdoor environments.* Cold Spring Harbor Laboratory. Descargado de `https://www.biorxiv.org/content/early/2020/11/30/2020.11.30.403816.full.pdf` doi: 10.1101/2020.11.30.403816

Yang, M., Chen, J., Zhang, Y., Liu, J., Zhang, J., Ma, Q., ... Ying, R. (2025). Low-rank adaptation for foundation models: A comprehensive review. *arXiv preprint arXiv:2501.00365*. Descargado de `https://arxiv.org/abs/2501.00365` (Revisión exhaustiva sobre técnicas LoRA fuera de modelos de lenguaje) doi: 10.48550/arXiv.2501.00365

Glosario

Ajuste fino El ajuste fino consiste en reentrenar las capas de extracción de características de un modelo para afinar las representaciones de características de orden superior, de modo que se adapten mejor a un nuevo conjunto de datos y sean más relevantes para la tarea objetivo. (Elgendy, 2020). 30

AP (*Average Precision*) Es una métrica, que se calcula a partir del área bajo la curva de precisión y sensibilidad (PR). (Elgendy, 2020). 32

AUC (*Area Under the Curve*) Métrica que representa el área debajo de la curva de la métrica deseada. (Elgendy, 2020). 7

Caja delimitadora Son las coordenadas que ubican el cuadro que rodea el objeto etiquetado o predicho. (Elgendy, 2020). 16

COCO (*Common Objects in Context*) Conjunto de datos de visión por computadora que contiene imágenes etiquetadas para tareas como detección de objetos, segmentación y descripción de imágenes. (Consortium, s.f.). 43

CSP (*Cross-Stage Partial*) Es una técnica utilizada en redes neuronales para mejorar la capacidad de aprendizaje de los modelos, la misma divide las características aprendidas en dos partes y las fusiona posteriormente. Esto reduce la redundancia de gradientes y mejora la eficiencia de la red.(Bochkovskiy, Wang, y Liao, 2020). 17

Hiperparámetro Es un valor de configuración que se fija antes del entrenamiento y determina tanto la arquitectura del modelo como la forma en que aprende. (Elgendy, 2020). 14

IDE Es una aplicación que integra diversas herramientas de desarrollo de software, como editor de código, compilador, depurador. (Sommerville, 2016). 43

Letterbox Es una función que re dimensiona y rellena las imágenes a una forma específica para conservar la relación de aspecto. (Ultralytics, 2023). 17

MAE (*Masked Autoencoder*) Modelo que aprende a "rellenar" las partes ocultas de una imagen usando las partes visibles. (He y cols., 2021). 24

mAP (*mean Average Precision*) Es una métrica que se calcula de la siguiente manera: primero se calcula la curva PR (calculada a partir de la Precisión y la Sensibilidad), a partir de la PR se calcula la precisión promedio (PA) calculando el área bajo la curva (AUC). Finalmente, la mPA para la detección de objetos es el promedio de la PA calculada para todas las clases. (Elgendy, 2020). 7

MLP (*Multilayer Perceptron*) Caso particular de una red neuronal feedforward. Elgendy (2020). 10

mMOTA (*mean Multi-Object Tracking Accuracy*) Es una medida para evaluar el rendimiento de sistemas de seguimiento de múltiples objetos en video. Representa la suma ponderada de las precisiones promedio de seguimiento en todos los videos del conjunto de evaluación. (Wu y cols., 2020). 9

mMOTP (*mean Multi-object Tracking Precision*) Se define como la suma ponderada de la precisión promedio de seguimiento en todos los videos del conjunto de evaluación. La precisión de seguimiento para un cuadro específico se calcula como la *intersección sobre unión (IoU)* entre los cuadros delimitadores predichos y los cuadros delimitadores reales. Es decir, mide qué tan bien se ajustan geométricamente las predicciones a las posiciones reales de los objetos. (Wu y cols., 2020). 9

MOTA (*Multi-Object Tracking Accuracy*) Esta métrica evalúa cual es el margen de error del modelo de seguimiento cuando va estimar la posición exacta de los objetos analizados. (Sabattini y cols., 2023). 9

NMS (*Non-Maximum Suppression*) Es un algoritmo de postprocesamiento utilizado en detección de objetos. Si se tienen múltiples cajas delimitadoras que detectan el mismo objeto, este conserva la que tiene mayor puntuación de confianza y descarta las cajas cuya superposición con la seleccionada (medida mediante IoU) supera un umbral predefinido. De esta forma, se reducen las detecciones redundantes. (Elgendy, 2020). 31

Normalización L2 Es una norma que mide la distancia euclidiana del vector al origen y es ampliamente usada tanto en preprocesamiento (normalización) como en regularización. (Goodfellow y cols., 2016). 23

ReLU (*Rectified Linear Unit*) Es una función que se utiliza en las capas ocultas de una red neuronal, para introducir no linealidad. Esta función mantiene los valores positivos y sustituye por cero los valores negativos.(Elgendy, 2020). 26

RoI (*Region of Interest*) Son áreas de la imagen que el sistema estima con alta probabilidad de contener un objeto, las probabilidades altas se conservan

y se envían a las siguientes etapas del proceso, mientras que las regiones con probabilidades bajas se descartan.(Elgendy, 2020). 18

RoI align Capa propuesta en Mask R-CNN para corregir la des alineación espacial causada por la cuantización de coordenadas en RoI Pooling, asegurando una mejor alineación con la imagen original y mejorando la precisión en tareas de segmentación..(He y cols., 2017). 20

RoI pooling Es un componente de las CNN que se utiliza para extraer una ventana de tamaño fijo del vector de características devuelto por las RoIs antes de alimentar las capas completamente conectadas.(Elgendy, 2020). 19

SoftMax Es una función que se utiliza habitualmente en la capa de salida de una red neuronal para obtener probabilidades de clasificación cuando existen más de dos clases mutuamente excluyentes. Esta función fuerza a que la suma de las salidas sea igual a uno, de modo que cada valor quede entre 0 y 1. (Elgendy, 2020). 25

Tensor numérico "Es un arreglo n-dimensional con n mayor o igual 2". (Elgendy, 2020, p. 3). 17

transformador Red neuronal para procesamiento de secuencias que utiliza un mecanismo de atención para modelar dependencias entre los elementos de la entrada, sin recurrencias ni convoluciones. (Vaswani y cols., 2017). 23

upsampling Capas que se utilizan para aumentar las dimensiones espaciales de los mapas de características en redes neuronales convolucionales. (Elgendy, 2020). 30

Anexo A

Experimentación con SAM-CLIP

El propósito de explorar el uso de Segment Anything Model (SAM) es facilitar la generación automática de conjuntos de datos para tareas de segmentación. Este modelo permite emplear prompts que orientan la segmentación hacia objetos específicos.

Para la exploración inicial se utiliza un prompt de tipo punto, que consiste en proporcionar una coordenada sobre la imagen para que el modelo genere únicamente la máscara del objeto ubicado en esa posición. Esta funcionalidad resulta especialmente útil para aislar la máscara de un objeto en particular, en este caso, los caminos de hormigas.

Dado que los resultados no son satisfactorios, se decide evaluar otras estrategias para adaptar el modelo a la tarea específica. Se identifica entonces la técnica de ajuste fino LoRA (Low-Rank Adaptation), que permite incorporar nueva información al modelo sin modificar directamente sus parámetros originales. En función de los recursos disponibles, esta técnica se presenta como una alternativa viable para especializar a SAM en la segmentación de caminos de hormigas.

A su vez, con el fin de automatizar la localización del prompt de tipo punto, se considera el uso de CLIP, un modelo que recibe un prompt de texto y localiza espacialmente el objeto correspondiente dentro de la imagen. De esta forma, la salida de CLIP (coordenadas del objeto deseado) puede utilizarse como entrada para SAM.

En este contexto, se solicita a CLIP el punto central del camino y se utiliza esta coordenada como entrada a SAM para simular un clic sobre el camino y obtener únicamente su máscara. Dado que CLIP funciona con prompts de texto

preexistentes, al utilizar el término "camino de hormigas" no es reconocido por el modelo. Por ello, se realiza un ajuste fino para que el modelo aprenda a asociar dicho término con la región correspondiente en la imagen.

Sin embargo, los resultados no son satisfactorios. Las métricas de evaluación, basadas en el porcentaje de aciertos según la similitud entre el prompt y la región seleccionada, resultan bajas, lo que impide su uso en el desarrollo del pipeline.

Finalmente se intenta cambiar la estrategia utilizando diferentes prompts de texto y se prueban las variantes disponibles de CLIP (ViT-B/32, ViT-L/14 y ViT-H/14) sobre una imagen de prueba, pero en esta primera instancia los resultados no mejoran de forma significativa.

En base a estas pruebas se puede observar que la elección y formulación de los prompts a utilizar es fundamental para el uso eficiente de modelos como CLIP. Por ello, se desarrolla un script que tiene como objetivo evaluar un conjunto de prompts predefinidos para cada fotografía individual en el conjunto de pruebas. Cabe destacar que este conjunto de pruebas es completamente independiente de los conjuntos utilizados para el entrenamiento y la validación, garantizando así una evaluación imparcial del rendimiento del modelo frente a nuevos datos.

El proceso ejecutado por el script consiste en:

- Aplicar a cada imagen en el conjunto de pruebas los prompts definidos.
- Para cada prompt que se aplica sobre una imagen, comparar los resultados obtenidos al analizar las métricas, que determinen el rendimiento óptimo.
- Luego se guardan los resultados del prompt que obtiene el mejor rendimiento para por imagen con el fin de obtener la configuración óptima en cada caso.
- Finalmente, en base al mejor prompt seleccionado por imagen, se procede a clasificar y determinar los prompts (del conjunto inicial) que mejor se comportan de manera general en el conjunto de datos de prueba.

A partir de esta experimentación, se identifica una oportunidad para realizar un estudio más exhaustivo sobre el diseño de prompts, específicamente enfocado en la tarea de segmentación de caminos de hormigas. El objetivo principal de esto es identificar aquellos prompts que demuestran una alta capacidad de generalización y pueden aplicarse con éxito a la mayor cantidad y diversidad de imágenes posible.

En conclusión, la meta a largo plazo es lograr la unión entre las capacidades de Segment Anything Model (SAM) y CLIP. Utilizando el conocimiento adquirido sobre los prompts y la eficiencia de estos modelos, se busca desarrollar una herramienta automática que es capaz de generar máscaras de segmentación con precisión. Esto permite utilizar estas máscaras generadas automáticamente para

entrenar otros modelos de segmentación, tales como los discutidos previamente o utilizarlos como modelo de segmentación en el sistema ARP.