UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Reconocimiento y conteo de manzanas

Informe de Proyecto de Grado presentado por

Roxana Garderes y Facundo Gutiérrez

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, 23 de noviembre de 2023

Resumen

Obtener información precisa sobre la cantidad de manzanas en plantaciones agrícolas resulta vital para potenciar las decisiones de los productores en relación con la producción, distribución y comercialización de dichos cultivos. En este contexto, la robótica y las redes neuronales emergen como herramientas óptimas, dado su notable avance en los últimos años.

A lo largo de este proyecto, se buscaron soluciones para la identificación y el conteo de manzanas en videos filmados por robots terrestres, que circulan entre hileras de árboles de manzanas en plantaciones. Con este objetivo, se propusieron combinaciones de redes neuronales de detección con algoritmos de seguimiento, tomando la cantidad de manzanas identificadas por los algoritmos como la cantidad total de manzanas en un video.

Los modelos de redes neuronales seleccionados para detección fueron Faster R-CNN, YOLOv5 y YOLOv8. Estos modelos se entrenaron utilizando múltiples conjuntos de datos, y se evaluaron utilizando sus propios datos de prueba y un conjunto adicional construido a partir de datos del ambiente real. Como algoritmos de seguimiento, se evaluaron StrongSort, Bytetrack y OCSort, en combinación con los modelos de detección entrenados YOLOv5 y YOLOv8.

Ante los resultados obtenidos en la etapa de conteo y las características de los árboles de manzanas, se plantea la utilización de dos métodos de ajuste de predicciones, uno basado en regresión lineal y otro basado en coeficientes de ajuste. Estos modelos ajustan las predicciones de cantidad de manzanas dadas por los algoritmos de seguimiento, para acercarse lo más posible a los valores reales de cantidad de manzanas.

También se destaca la implementación de un simulador de campos de manzanos utilizando Gazebo y ROS, que permite generar instancias simuladas de filas de árboles y logra generar modelos de manzanos con una morfología similar a la de los árboles reales. Esto permite la experimentación de forma independiente a la disponibilidad de datos reales.

Palabras clave: Manzanas, Redes Neuronales, Conteo, Detección, Seguimiento, ROS, YOLO, YOLOv5, YOLOv8, Faster R-CNN, SORT, OCSort, Bytetrack, StrongSort, Regresión lineal

Índice general

Capítulo 1

Introducción

1.1. Presentación del problema

El concepto de agricultura inteligente, más conocido como *Smart Farming*, ha tomado relevancia en los últimos años. El uso de tecnologías y metodologías basadas en datos ha logrado optimizar la productividad, eficiencia y sostenibilidad de la producción agrícola. Las redes neuronales, en particular las redes neuronales convolucionales (CNN), han mostrado un gran potencial en la resolución de problemas complejos, incluso en problemas vinculados a la agricultura (Kamilaris y Prenafeta-Boldú, 2018). Además, las CNN han evolucionado notablemente, convirtiéndose en sinónimo de excelencia en términos de resultados, realizando tareas de clasificación y predicción, y superando con creces a técnicas clásicas de procesamiento de imagen.

Los avances en esta área podrían tener importantes aplicaciones en la optimización de la producción agrícola, ya que un conocimiento anticipado de la cantidad de manzanas a cosechar permite una mejor planificación de ventas, así como de la mano de obra y recursos para la cosecha, traslado y distribución de las manzanas, llevando a una reducción de los costos, mayor exactitud en la planificación y fomentando una producción sustentable (van Ittersum y Rabbinge, 1997; Wang, Nuske, Bergerman, y Singh, 2013). Los resultados también podrían ser de utilidad a la hora de aplicar fertilizantes o tratamientos específicos a las manzanas (Bargoti y Underwood, 2017b).

La identificación y conteo de frutas en plantaciones representa un gran desafío debido a la variedad de tamaños, colores y formas que pueden presentar las frutas, y las oclusiones por otras manzanas, ramas y hojas. En este contexto, el presente proyecto, desarrollado en conjunto con el INIA (Instituto Nacional de Investigación Agropecuaria) y el grupo MINA (perteneciente al Instituto de Computación de Facultad de Ingeniería, Udelar), tiene como objetivo evaluar el uso de redes neuronales para la identificación y conteo de manzanas en plantaciones, con el fin de mejorar la eficiencia y precisión del proceso de recolección de la fruta.

El hecho de trabajar en ambientes no controlados, como lo es una plantación de manzanos, lleva a variables que pueden afectar notoriamente el resultado. El clima, la hora del día y las oclusiones de los frutos, condicionan el desempeño de los métodos elegidos. Además, se verá cómo el recorrido elegido para grabar los datos lleva a problemas como la visibilidad de filas contiguas, lo que altera considerablemente los resultados de conteo. Para resolver este problema, se plantea una solución basada en regresión lineal.

Otro factor importante a considerar son los datos disponibles, dado que los métodos que utilizan redes neuronales aprenden a partir de estos. Si bien existen abundantes conjuntos de datos públicos de imágenes de manzanas, que son muy útiles para abordar el problema de detección, no se dispone de grabaciones en video que contengan filas completas de manzanas, las cuales son necesarias para evaluar soluciones de conteo.

1.2. Estructura del documento

Este documento consta de una sección dedicada a la revisión de antecedentes, seguida por el marco teórico, donde se incluyen subsecciones para los modelos de detección y los algoritmos de seguimiento. Posteriormente, se encuentra la sección denominada "Decisiones tomadas", en la cual se abordan las distintas etapas del proyecto, se discuten las dificultades que surgieron en su desarrollo y se describen las estrategias empleadas para resolverlas. Luego se encuentra la sección de experimentación, donde se describen las pruebas realizadas. Finalmente, se encuentra la sección de conclusiones y trabajo futuro.

Capítulo 2

Revisión de antecedentes

En este capítulo se pretende resumir, a grandes rasgos, investigaciones previas en el ámbito de la aplicación de visión por computadora para la agricultura, en particular, la detección y el conteo de frutos. Se presentan artículos fundacionales en el área de detección de frutas, así como un análisis de los métodos destacados, resultado de la investigación inicial descrita en el documento de Estado del Arte (Garderes y Gutiérrez, 2022). Luego, se comentarán artículos relevantes, en los que se pretende resolver el problema de detección y conteo de manzanas utilizando únicamente grabaciones de los árboles.

Si bien existen diversos enfoques y tecnologías posibles, debido a que el foco de este proyecto se encuentra en la detección y el conteo de manzanas a través de visón por computadora y redes neuronales, para la investigación inicial se utilizaron cadenas de búsqueda como "*Apple detection*", "*Apple counting*", "*Apple detection using neural networks*", "*Double counting prevent in fruit counting*" y "*Apple datasets*". De los resultados obtenidos, se seleccionaron inicialmente los artículos posteriores al 2018, con la mayor cantidad de citas posibles, que utilizaran redes neuronales y trabajaran con manzanas u otros frutos con características similares, por ejemplo, el tamaño, el color o la forma. Más adelante, se encontraron y agregaron otros artículos relevantes que no aparecieron en la búsqueda inicial.

A continuación, se incluyen breves resúmenes de los artículos que tuvieron mayor influencia en este proyecto, ya sea por el enfoque, las técnicas o las tecnologías aplicadas. Inicialmente, se presenta una síntesis de las primeras publicaciones en el área de reconocimiento de frutos como artículos fundacionales. Seguidamente, se mencionan los artículos más destacados en el problema de la detección de manzanas, clasificados por las tecnologías con las que se implementan, mientras que la última sección se dedica a las publicaciones relativas al conteo de frutas en plantaciones.

2.1. Artículos fundacionales

El problema de detección y conteo de frutas no es nuevo, y existe una gran cantidad de publicaciones relacionadas con el tema. De hecho, la primera idea de robots para cultivo fue propuesta en los años 60 (Schertz y Brown, 1968), donde presentaron algunos lineamientos a tener en cuenta a la hora de automatizar el cultivo, diferenciando entre el cultivo a gran escala, donde no se toma en cuenta el estado de cada fruta, y el cultivo individual.

A fines de la década de los 70, se comienzan a ver avances sobre la detección de frutas. Concretamente, se experimenta con técnicas de reconocimiento de patrones (Parrish y Goksel, 1977), concluyendo que se podrían lograr buenos resultados sobre el problema de detección de frutas. Posteriormente, se da una primera aproximación de la utilización de la Transformada de Hough para detección de círculos en tomates parcialmente ocluidos (Whittaker, Miles, Mitchell, y Gaultney, 1984). Años más tarde, se presenta el proyecto *MAGALI* (D'Esnon, Rabatel, Pellenc, Journeau, y Aldon, 1987), que consiste en un robot con un brazo mecánico al que se le conecta en primera instancia una cámara en blanco y negro, y luego una cámara color, que detectaba frutas mediante características geométricas.

2.2. Detección de manzanas

En lo que respecta a la aplicación de redes neuronales para resolver el problema de detección de manzanas, se destacan los resultados de dos modelos, Faster R-CNN y YOLO, en sus diferentes versiones. A continuación se incluyen algunos de los artículos más relevantes con cada uno de estos modelos.

Las métricas de evaluación para detección (*Average Precision* y *Mean Average Precision*) y los conceptos necesarios para comprenderlas (precisión, recuperación, $F1$ e IoU), se definen en la sección 5.2.1.

2.2.1. Aplicación de Faster R-CNN

En el artículo *Deep Fruit Detection in Orchards* (Bargoti y Underwood, 2017a) se propone el uso del modelo Faster R-CNN como sistema de detección de frutas, concretamente almendras, mangos y manzanas. Estas tres frutas (junto con sus árboles) exhiben notables diferencias morfológicas, tales como la cantidad de frutos por árbol, el color y la textura, lo cual plantea un desafío significativo.

En dicha publicación se utiliza la técnica *transfer learning* para inicializar los pesos del modelo a la hora de entrenar, destacando que los modelos entrenados previamente con imágenes de plantaciones de otras frutas superan en desempeño a los modelos entrenados con el conjunto de datos ImageNet (Deng y cols., 2009), dataset con más de 14 millones de imágenes etiquetadas y 1000 categorías de objetos.

Los datos utilizados son imágenes capturadas durante el día en distintas plantaciones, utilizando sensores anclados a un robot terrestre. En lo que respecta a manzanas, se recolectaron 726 imágenes, complementándolas con técnicas de data augmentation para mejorar la capacidad de generalizar las detecciones y reducir el sobreajuste. En particular, se aplican las técnicas de rotación, escalado y PCA Color Augmentation.

A nivel de arquitectura, se experimenta con dos redes convolucionales distintas sobre el modelo Faster R-CNN, la red ZF y la red VGG16 (Simonyan y Zisserman, 2014), logrando resultados para la métrica $F1$ de 0.876 y 0.908 respectivamente. Además, concluye que aplicar técnicas simples de *data augmentation* lleva a lograr mejores resultados con una menor cantidad de datos de entrenamiento. Por el contrario, técnicas más complejas, como *PCA Color Augmentation*, pueden llevar al modelo a obtener resultados inferiores.

2.2.2. Aplicaciones de YOLOv5

Otra familia de redes neuronales muy utilizada para la detección de objetos en imágenes es YOLO. Si bien en la literatura se encuentran publicaciones de detección de manzanas que utilizan YOLOv3 (Kuznetsova, Maleva, y Soloviev, 2020; Tian, Yang, Wang, Li, y Liang, 2019) y YOLOv4 (L. Wu, Ma, Zhao, y Liu, 2021), YOLOv5 muestra resultados considerablemente mejores en cuanto a tamaño del modelo, tiempo y métricas de detección, por lo cual se hizo foco en este método.

En *A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5* (Yan, Fan, Lei, Liu, y Yang, 2021), se utiliza YOLOv5 para reconocer las manzanas que pueden ser recolectadas por un robot. Para esto se deben detectar las manzanas y clasificarlas entre las que puedan ser recogidas directamente por el robot y las que presentan algún obstáculo que lo impida. Por ejemplo, una manzana con oclusión de hojas puede ser recolectada, mientras que una manzana ocluida por otra manzana o rama no puede ser recolectada sin correr el riesgo de dañar el brazo robótico o las manzanas, como se ejemplifica en la figura 2.1.

En este caso, la elección del algoritmo YOLOv5 se debe a que puede ser utilizado en tiempo real (procesa hasta 140 cuadros por segundo), es rápido de entrenar y ligero, por lo que es razonable de implementar en dispositivos embebidos. Concretamente, se seleccionó el modelo YOLOv5s, por ser el más liviano en términos de almacenamiento, pero suficiente para detectar las dos clases necesarias.

YOLOv5s se compone de tres módulos principales, denominados *backbone*, *neck* y *head*. En este estudio, el *backbone* fue optimizado, reduciendo la cantidad de parámetros y capas, haciéndolo más liviano. También se modificó este módulo para que utilice las capacidades visuales de atención del robot, buscando mejorar la extracción de características. Otro cambio realizado fue el ajuste de las *anchor boxes* en el módulo de detección, para evitar el reconocimiento de manzanas muy pequeñas en el fondo. Se observa que la aplicación de este filtro es viable en este contexto, dado que las imágenes de manzanas son tomadas a una distancia

(a) Parcialmente soleado

(b) Soleado con iluminación frontal

Figura 2.1: Ejemplos de reconocimientos de manzanas utilizando YOLOv5s con diferentes iluminaciones. Las etiquetas celestes corresponden a manzanas no recolectables y las verdes a manzanas recolectables. Imagen extraída de (Yan y cols., 2021).

muy corta, por lo que existe una diferencia de tamaño considerable entre las manzanas en primer plano y las de las filas posteriores.

Las imágenes para el entrenamiento fueron obtenidas de una plantación, tanto en días soleados y nublados, como diferentes horarios (mañana, medio día y tarde) para garantizar diferentes condiciones de iluminación. Las imágenes fueron tomadas utilizando diferentes ángulos y a distancias de entre 0.5 y 1.5 metros, buscando asemejar las capacidades de detección de los robots de cosecha. En total, se tomaron 1214 imágenes de manzanas, incluyendo manzanas ocluidas por hojas, por ramas, con oclusiones mixtas y con sobreexposición de manzanas. Como opciones de *data augmentation* se seleccionaron las variaciones en brillo, inversión vertical y horizontal, rotación en varios ángulos y desenfoque gausiano de 0.02.

El entrenamiento se realizó con 300 *épocas*, aunque se encontró que los resultados mejoraban consistentemente en las primeras 100, pero se estabilizaban a partir de la 250. Las métricas utilizadas fueron precisión (P), recuperación (R), mAP (*mean average precision*) y $F1$.

Experimentalmente, mediante un análisis de 200 imágenes y variando el parámetro de confianza, determinaron que un intervalo de confianza de 0.5 permitía minimizar la detección de manzanas en el fondo, sin perder demasiadas manzanas en primer plano. Para esto se estudió la evolución de P, R y mAP variando el intervalo de confianza en el rango de 0.1 a 0.9, con incrementos de a 0.1.

El modelo entrenado alcanza valores de mAP del 86.75 %, P de 91.48 % y $F1$ de 87.49 %, con un tiempo de detección promedio de 0.013 segundos por imagen. El tamaño del modelo es muy inferior al de otros algoritmos con lo que se compara (12.7 MB), logrando de igual forma resultados superiores.

2.2.3. Conjuntos de datos

Las publicaciones anteriormente nombradas, así como los demás artículos relevados, trabajan con sus propios conjuntos de datos, que no son publicados. Esto imposibilita replicar y comparar los métodos y resultados con otros modelos. Por este motivo se destaca *MinneApple: A Benchmark Dataset for Apple Detection and Segmentation* (Häni, Roy, y Isler, 2020b). Dicho trabajo tiene por objetivo proveer un conjunto de datos diverso, extenso y conteniendo diferentes variedades de manzanas entre múltiples filas de árboles. Este conjunto de datos fue diseñado para desafiar a los algoritmos de detección de objetos, proveer a los investigadores con herramientas para evaluar sus algoritmos de forma imparcial y comparar los resultados con los de otros proyectos.

Por estos motivos, MinneApple consiste de un conjunto público de 1000 imágenes etiquetadas de árboles de manzanas, con 41000 instancias anotadas. Estas imágenes fueron tomadas a lo largo de más de un año, con diferentes especies de manzanos. La variedad de árboles se debe a que los videos fueron filmados en el *Centro de Investigación de la Universidad de Minnesota*, donde se realizan investigaciones de fenotipos. Las imágenes son cuadros de videos que fueron filmados horizontalmente, a un metro de los árboles y caminando a 1 m/s para no tener problemas de desenfoque, tomando luego 1 de cada 5 cuadros del video para el conjunto de datos. Las filmaciones fueron en el lado soleado de los árboles. Algunos ejemplos de estas imágenes se encuentran en la figura 2.2.

	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
Tiled FRCNN	0.341	0.639	0.339	0.197	0.519	0.208
Faster RCNN	**0.438**	**0.775**	**0.455**	**0.297**	**0.578**	**0.871**
Mask RCNN	0.433	0.763	0.449	0.295	0.571	0.809

Tabla 2.1: Resultados de modelos de detección aplicados a MinneApple (Häni y cols., 2020b), donde AP es el *average precision* para IoU de 50 a 95, AP_{50} es AP con IoU 50, AP_{75} es AP con IoU 75, y los restantes son AP según el tamaño de los objetos.

Con estos datos se entrenaron y probaron tres modelos de RCNN. Los resultados se incluyen en la tabla 2.1, donde se puede apreciar que los mejores resultados son los de Faster R-CNN, con AP de 0.438.

2.3. Conteo de manzanas

A la hora de contar objetos en video, el conteo de duplicados es uno de los problemas más frecuentes (W. Zhang y cols., 2022). Una manzana puede ser cubierta con otro objeto por algunos segundos y luego volver a estar visible, o ser visible desde ambos lados de la fila de árboles, llevando a ser contada más de una vez.

Para este desafío se encuentran varias propuestas de soluciones, que combinan diferentes tecnologías. Un ejemplo de esto es la construcción de modelos

Figura 2.2: Ejemplos de imágenes del conjunto de datos de MinneApple. Imagen extraída de (Häni y cols., 2020b).

3D de campos de manzanas a partir de los datos obtenidos de ambos lados de una hilera de manzanos (P. Roy, Dong, y Isler, 2018), unificando los modelos construidos con los datos de cada lado.

Un ejemplo adicional implica una nueva aplicación de Faster R-CNN (Stein, Bargoti, y Underwood, 2016), en la cual se combina la detección con el seguimiento de las frutas a lo largo de los cuadros para lograr la identificación individual y, por consiguiente, el conteo preciso. Para este propósito, se utilizan puntos proporcionados por el LiDAR para la ubicación GPS de cada árbol, y las frutas se asocian con su respectivo árbol. Sin embargo, se decidió descartar este método para utilizar técnicas enfocadas en el uso de cámaras dada su alta disponibilidad.

2.3.1. Comparación por similitud para conteo

Otra propuesta interesante se da en *Culling Double Counting in Sequence Images for Fruit Yield Estimation* (Xia y cols., 2022), donde se construye un *framework* basado en el modelo *CenterNet* y un modelo de emparejamiento de frutas para estimación de cultivo en secuencia de imágenes. Aquí se desarrolla un modelo de asociación de frutas teóricamente idénticas entre cuadros consecutivos utilizando el algoritmo húngaro.

La etapa de emparejamiento de frutas entre cuadros es una etapa de alta complejidad, dado que incluso imágenes subsecuentes pueden tener grandes diferencias, al verse afectadas por objetos como hojas, ramas, otras frutas o sombras. En la arquitectura, esto está implementado por dos módulos. En primer lugar, una serie de capas convolucionales recibe como entrada los cuadros delimitadores obtenidos en la etapa de detección, con los que se generan matrices de comparación de características, que asignan puntos de similitud a cada par de objetos. Las matrices son recibidas por la "Capa de Decisión", que implementa el algoritmo húngaro para realizar la asignación óptima entre pares de frutas detectadas.

Los resultados de este trabajo en manzanas (también se estudian naranjas)

(a) (b)

Figura 2.3: Ejemplos de emparejamientos de frutas en cuadros adyacentes del video utilizando el modelo *CenterNet* para detectar y un algoritmo basado en el algoritmo húngaro para emparejar. En la figura 2.3a se muestra para manzanas y en la figura 2.3b para naranjas. Imagen extraída de (Xia y cols., 2022).

son AP_{50} de 0.927 en detección y $F1$ de 0.816 en la comparación entre dos cuadros consecutivos, evaluado sobre 20 secuencias de imágenes de cuadros adyacentes. El tiempo promedio de cómputo es de 5.33 minutos por secuencia de imágenes, no siendo suficiente para la ejecución en tiempo real. En la figura 2.3 se puede observar un ejemplo de emparejamiento entre frutas en cuadros adyacentes.

2.3.2. Aplicaciones de seguimiento para conteo

Una mejora sobre el método de emparejamiento anterior se da en los algoritmos de seguimiento, donde se combina el emparejamiento por similitud con la posición de los objetos detectados y predicción de su desplazamiento y cambio de escala, como se realiza con el modelo Sort.

En *Deep-learning-based in-field citrus fruit detection and tracking* (W. Zhang y cols., 2022), se combinan y personalizan YOLOv3 y SORT para detectar y contar cítricos. Con esta combinación, la cantidad de identificadores asignados por el algoritmo de seguimiento es la cantidad de frutas en el recorrido, utilizando el video como única entrada de datos.

En dicho proyecto, se modifica levemente la arquitectura de YOLOv3 para detectar objetos en diferentes escalas y utilizar el análisis de las características de los fragmentos adyacentes a los objetos detectados, ya que las naranjas tienden a crecer en grupos. Con esto, logran muy bueno resultados: 0.902 de precisión (P), 0.893 de recuperación (R), 0.897 de $F1$ y 0.938 de AP.

El algoritmo de seguimiento propuesto, *OrangeSort*, está basado en SORT, pero con un algoritmo de predicción de movimiento que procura minimizar el conteo doble por oclusiones temporales de la fruta. Si se detecta que la escala de

un objeto rastreado disminuye continuamente, luego de cierto nivel se considera que el objeto no aparece más (se quita el identificador de los posibles identificadores de seguimientos a asignar). Si un objeto no es asociado a un identificador de seguimiento porque se encuentra parcialmente ocluido o no es detectado, pero aparece algunos cuadros después, *OrangeSort* recupera el identificador anterior (a diferencia de Sort, donde no se vuelve a asignar un identificador si deja de aparecer).

Otra estrategia, aplicada en lo que denominan *OrangeSort**, es realizar seguimiento en una región central de la imagen en sentido vertical, no considerando el primer y último quinto de cada cuadro, sino los 3/5 del centro. Con esto se busca minimizar la cantidad de frutas que desaparecen por oclusiones y reaparecen algunos cuadros después, generando doble conteo.

Similar a lo expuesto en la sección 2.3.1 (Xia y cols., 2022), SORT realiza una comparación de similitud entre objetos detectados, pero esto se combina con la estimación de la posición del objeto en el siguiente cuadro, de forma que el proceso de asignación de identidades es considerablemente más rápido.

Para evaluar los resultados del conteo de identificadores de seguimiento, se toman en cuenta diversas categorías, que se listan a continuación:

- total de frutas contadas
- frutas contadas correctamente
- frutas no contadas
- objetos contados incorrectamente como frutas
- frutas contadas más de una vez
- error absoluto medio sobre el total del video.

Los resultados obtenidos indican que *OrangeSort* tiene menor conteo duplicado que Sort y DeepSort, con errores absolutos entre 9 % y 40 %. Si bien *OrangeSort** tiene ligeramente menos frutas contadas correctamente, reduce el número de naranjas contadas más de una vez y logra resultados sustancialmente más cercanos al conteo manual, con un error absoluto entre 6 % y 24 %.

Capítulo 3

Marco Teórico

El propósito de esta sección es presentar el funcionamiento y arquitectura de los distintos modelos de detección y seguimiento de objetos utilizados en este proyecto. Comienza con una pequeña introducción a las redes neuronales artificiales, para luego comentar cómo funcionan las redes neuronales convolucionales. En segundo lugar, se describen los métodos de detección Faster R-CNN y YOLO, que fueron los elegidos para el desarrollo de este proyecto. También se hace una mención del modelo Fast R-CNN dado que es clave para la arquitectura del modelo Faster R-CNN.

Luego, se mencionan los algoritmos de seguimiento, comenzando por SORT, que es el algoritmo base del cual parten los modelos seleccionados. Finalmente, se desglosan los algoritmos DeepSORT, ByteTrack y StrongSORT.

3.1. Redes Neuronales

Las redes neuronales, también conocidas como redes neuronales artificiales, son un tipo de algoritmo de aprendizaje automático fuertemente inspirado en el funcionamiento del cerebro humano (Nielsen, 2015). Una red neuronal consiste en unidades interconectadas llamadas "neuronas" que están organizadas en grupos llamados "capas".

Una neurona es la unidad básica de la red. Se puede entender como una función que recibe una o más entradas y devuelve como salida un valor entre 0 y 1, conocido como "activación". Una neurona está "encendida" si su valor de activación es cercano a 1. Cada una de las entradas $x_1, ..., x_n$ de la neurona tiene asignado un peso $w_1, ..., w_n$ y un sesgo b (del inglés, *bias*) global. En el interior de la neurona, se aplica una función de activación f sobre la entrada, siendo la salida de la capa $f(w * x + b)$.

A nivel de arquitectura, las redes neuronales están compuestas por tres tipos de capas: la capa de entrada, la capa de salida y una o más capas ocultas. Como se puede notar en la figura 3.1, la capa más a la izquierda es la de entrada, que recibe datos en crudo desde afuera de la red. La cantidad de neuronas de la capa

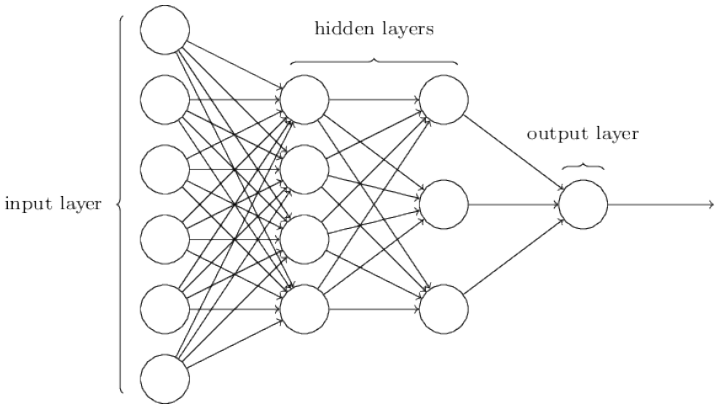


Figura 3.1: Ejemplo de red neuronal con seis nodos en la capa de entrada, dos capas ocultas de cuatro neuronas y una neurona como capa de salida. Imagen extraída de (Nielsen, 2015).

de entrada es definida por cada componente de los datos de entrada de la red. En el centro se encuentran las capas ocultas, donde la cantidad de capas y de neuronas por capa son elegidas en función del problema a resolver. Finalmente, la capa más a la derecha es la capa de salida, que produce el resultado de la red, y la cantidad de neuronas de esta depende de la tarea que se desee realizar.

Los datos de entrada de la red fluyen desde la capa de entrada hacia la capa de salida, atravesando las capas ocultas, en lo que se conoce como propagación hacia adelante. Cada neurona aplica la función de activación sobre su entrada, y propaga el resultado hacia la siguiente capa.

Dado que los pesos son inicializados de forma aleatoria, es muy poco probable que los resultados que devuelve la capa de salida se adecuen a los resultados esperados. Es por esto que el concepto de "aprendizaje" está muy ligado a las redes neuronales. El proceso de aprendizaje consiste en ajustar los pesos para disminuir el error, intentando minimizar la denominada función de pérdida.

La función de pérdida permite cuantificar la diferencia entre las predicciones del modelo y las etiquetas reales de los datos de entrenamiento. Un ejemplo es el error cuadrático medio (MSE), expresado como $\frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, donde N es el tamaño del conjunto de entrenamiento.

Según el error dado por la función de pérdida, se computa el gradiente con respecto a un peso elegido. Este proceso se conoce como propagación hacia atrás (en inglés *backpropagation*), y permite ajustar los pesos logrando que la red se ajuste a los resultados esperados.

En el contexto de este proyecto, el aprendizaje se desarrolla de forma supervisada, es decir, parte una serie de datos etiquetados. El error, y por ende la actualización de los pesos, se computa para cada uno de estos datos, comparando el valor predicho por la red con el valor real etiquetado.

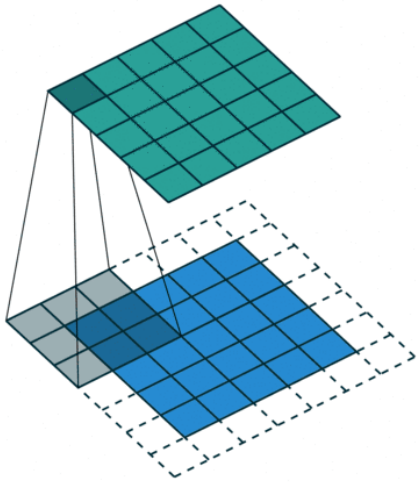


Figura 3.2: Ejemplo de aplicación de un filtro de 3×3 (en gris) sobre un canal de entrada (en azul) que genera un canal de salida (en verde). Imagen extraída de (Deeplizard, 2017)

3.1.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal artificial que en cierto modo se especializa en la detección y selección de patrones, lo que las hace útiles para el análisis de imágenes.

Las redes convolucionales tienen una serie de capas ocultas conocidas como capas *convolucionales*. Al igual que las otras capas, la capa convolucional recibe una entrada, la transforma y se la pasa a la siguiente capa. Las entradas de las capas convolucionales se conocen como canales de entrada, y las salidas como canales de salida.

En cada capa se debe definir la cantidad de filtros, que son los encargados de detectar patrones, y su cantidad determina el número de canales de salida. Entre los patrones que se pueden detectar se encuentran los bordes, formas, texturas, curvas, objetos y colores. Dependiendo del patrón que el filtro sea capaz de detectar, será el nombre que se le asigne a ese filtro. Un filtro se puede interpretar como una matriz pequeña, para la cual se decide el número de filas y columnas en función del problema a resolver, y los valores son inicializados de forma aleatoria.

Como se puede observar en la figura 3.2, el filtro (en gris) se desliza progresivamente sobre el canal de entrada (en azul), y transforma cada ventana de 3×3 en el valor correspondiente en el canal de salida. Este canal de salida es llamado mapa de características (del inglés, *feature map*).

En este caso particular, se ejemplifica con una imagen en blanco y negro, por lo que se tiene un único canal de entrada. Sin embargo, si la imagen fuera

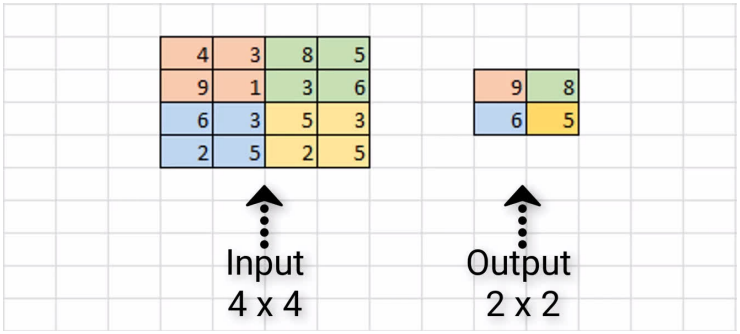


Figura 3.3: Ejemplo de aplicación de *Max Pooling* de 2×2 a una entrada de tamaño 4×4. Imagen extraída de (Deeplizard, 2017)

RGB, se tendrían tres canales de entrada. Esto implica que, suponiendo que la imagen es de 24×24 píxeles y se tienen 32 filtros, en el caso de la imagen RGB se tendría un mapa de características de tamaño $(24 \times 24 \times 3) \times 32$. A medida que aumenta el tamaño de la imagen, la cantidad de neuronas de la capa de entrada aumenta, y por ende los requerimientos de cómputo. A partir de esto se introduce el concepto de *Max Pooling*, que es un tipo de operación típicamente agregada a las redes convolucionales enseguida de cada una de las capas convolucionales.

Aplicar *Max Pooling* reduce la dimensionalidad de una imagen al disminuir la cantidad de píxeles de la salida de la última capa convolucional. En la figura 3.3 se puede observar un ejemplo de aplicación de *Max Pooling*, donde partiendo de una entrada de 4×4, se aplica esta técnica con un paso de tamaño 2, logrando bajar la dimensionalidad de la salida a 2×2. El proceso consiste en iterar en bloques de tamaño 2×2 en pasos de tamaño 2, y para cada bloque tomar como salida el valor máximo de la región. Por ejemplo, si tomamos el primer bloque naranja, como 9 es el valor máximo de la región, el valor en el canal de salida será 9.

3.2. Fast R-CNN

El modelo Fast R-CNN (Girshick, 2015) es un modelo de detección de objetos basado en regiones que utiliza redes neuronales convolucionales. Se construye a partir del modelo de estado del arte R-CNN (Girshick, Donahue, Darrell, y Malik, 2014), logrando disminuir las velocidades de entrenamiento y evaluación, al mismo tiempo que logra aumentar la precisión de las detecciones.

Ambos modelos utilizan la red VGG16 (Simonyan y Zisserman, 2014) como red pre-entrenada para la extracción de características. Sin embargo, Fast R-CNN tiene una arquitectura más simple, logrando extraer características utilizando una sola red, a diferencia de R-CNN que utiliza una red adicional. Esto se traduce en velocidades de entrenamiento 9 veces menor a los obtenidos por

R-CNN, pasando de 84 horas a 9.5 horas. Además, la velocidad de evaluación por imagen disminuye hasta 146 veces, pasando de 47 segundos a 0.3. La precisión media promedio (mAP, que se define en la sección 5.2.1) también se ve afectada positivamente, aunque no de forma sustancial, pasando de 66.0 % a 66.6 %.

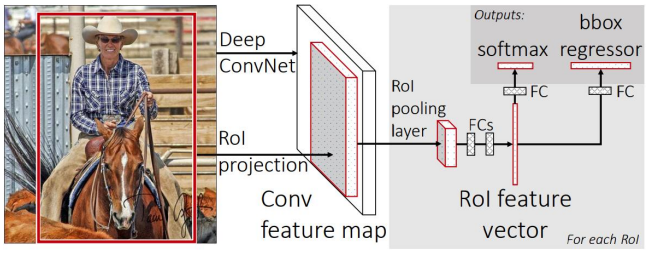


Figura 3.4: Ilustración de la arquitectura del modelo Fast R-CNN, extraída de (Girshick, 2015)

La arquitectura Fast R-CNN se puede ver representada en la figura 3.4. El modelo recibe como entrada la imagen y un conjunto de regiones de interés, que pasan por una red convolucional, generando un mapa de características de tamaño fijo. Este mapa pasa por una capa completamente conexa, donde se asocia a un vector de características. Finalmente, la salida del modelo son dos vectores por cada región de interés, las probabilidades dadas por la capa de softmax, y las *bounding boxes* por clase.

Dado que la cantidad de regiones de interés para un mismo objeto puede ser mayor a una, se aplica el mecanismo conocido como non-maximum supression, quedándose con aquella que mejor se ajuste al objeto detectado.

3.3. Faster R-CNN

El modelo Faster R-CNN (Ren, He, Girshick, y Sun, 2016) es un sistema de detección de objetos presentado en el año 2015 con el propósito de mejorar el modelo de estado del arte, Fast R-CNN (Girshick, 2015), que conseguía tasas de detección cercanas al tiempo real. El problema con el estado del arte era el cuello de botella presente en la etapa de propuesta de regiones, dado que los métodos utilizados eran ejecutados en CPU, lo que hacía que los tiempos de ejecución fueran considerablemente altos.

Este modelo está compuesto de dos módulos. El primero es un módulo de capas de convolución profundas, conocido como RPN, por sus siglas en inglés (*Region Proposal Network*), que propone regiones de interés. El segundo módulo aplica el modelo Fast R-CNN, que utiliza las regiones propuestas y las clasifica en las diferentes clases. El sistema completo, como se puede observar en la figura 3.5, es una red unificada para detección de objetos, donde la RPN tiene el rol de guiar al modelo de clasificación respecto a dónde pueden estar los objetos, y es conocida como mecanismo de atención.

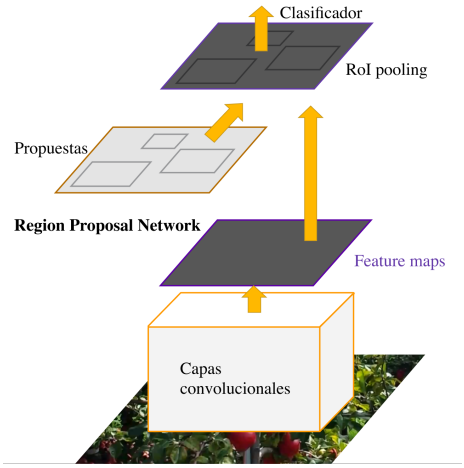


Figura 3.5: Ilustración de la arquitectura de Faster R-CNN, elaboración propia en base a (Ren y cols., 2016)

3.3.1. Region Proposal Network

El módulo RPN toma como entrada una imagen y devuelve un conjunto de regiones rectangulares que rodean los posibles objetos, y su correspondiente valor de confianza. Cada imagen pasa por una serie de capas convolucionales, compartidas entre el módulo RPN y Fast R-CNN, por ejemplo, la red *VGG-16* (Simonyan y Zisserman, 2015). Sobre el mapa de características resultado de la etapa anterior, se "desliza" una ventana de tamaño $n \times n$, que posteriormente es mapeada a una característica de menor dimensionalidad. Dichas características son la entrada de dos redes hermanas completamente conexas, una de regresión de *bounding boxes*, y otra de clasificación de *bounding boxes*.

En cada ventana deslizante se predicen simultáneamente, como máximo, k regiones, que se parametrizan relativamente a su *anchor box*. Cada *anchor box* está centrada en la ventana deslizante, y es asociada con una escala y una relación de aspecto diferente.

A partir de esto se puede decir que la capa de regresión de *anchor boxes* tiene $4k$ salidas, que codifican las coordenadas de las k *anchor boxes*, mientras que la capa de clasificación tiene $2k$ salidas, que representan las probabilidades de que haya o no un objeto. En la figura 3.6 se puede visualizar de forma más clara la arquitectura completa de la RPN.

3.4. YOLO

YOLO es una familia de frameworks para detección de objetos caracterizada por el balance entre velocidad y exactitud de detección, permitiendo la identificación de objetos de manera rápida y confiable (Terven y Cordova-Esparza,

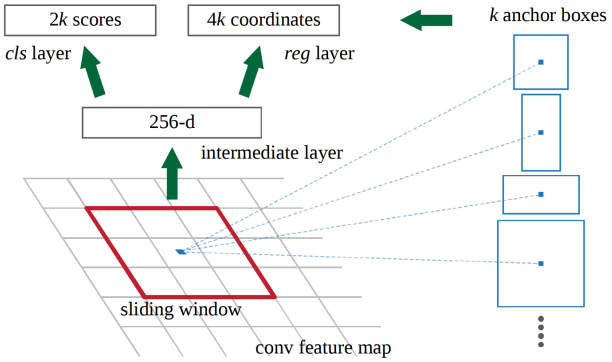


Figura 3.6: Ilustración de la arquitectura de la *Region Proposal Network*. Imagen extraída de (Ren y cols., 2016).

2023).

Su nombre es una sigla correspondiente a *"You Only Look Once"*, debido a que el algoritmo, desde su primera versión, era capaz de detectar objetos de una sola pasada y por una única red neuronal, en contraposición con las técnicas de ventana deslizante o detección en dos etapas (como Faster R-CNN). En comparación con el estado del arte, el resultado es más directo, dado que está basado únicamente en regresión. Estas características contribuyen a que la detección y el entrenamiento sean más veloces que en algoritmos similares.

Otra característica de YOLO es su variedad de versiones y particular evolución a lo largo de la historia. Por este motivo, en esta sección se hará una breve recapitulación de las versiones principales, haciendo hincapié en las versiones estudiadas en el proyecto. En la tabla 3.1 se pueden apreciar algunas diferencias entre las versiones que se tratan en este documento.

La primera versión de YOLO, YOLOv1, fue publicada en 2016 (Redmon, Divvala, Girshick, y Farhadi, 2016), y parte de sus autores fueron los que más tarde publicaron las versiones 2 y 3. YOLOv1 divide cada imagen en una grilla, luego en cada entrada se realizan las predicciones de *bounding boxes*, con su respectivo valor de confianza, para cada clase (en la primera versión soportaba 20 clases). Esto se representa en la figura 3.7.

A nivel de arquitectura, YOLOv1 consiste de 24 capas convolucionales, seguidas de dos capas que predicen las coordenadas y probabilidades de los *bounding boxes*. YOLOv2 y YOLOv3 fueron básicamente mejoras en varios aspectos de YOLOv1, llegando en YOLOv3 a un AP_{50} de 60.6 % con el dataset de COCO.

3.4.1. YOLOv4

YOLOv4 fue publicada en 2020 (Bochkovskiy, Wang, y Liao, 2020) por otros autores, mejorando los resultados pero manteniendo las características principales de las versiones anteriores. Esta versión incluyó varios cambios en su arqui-

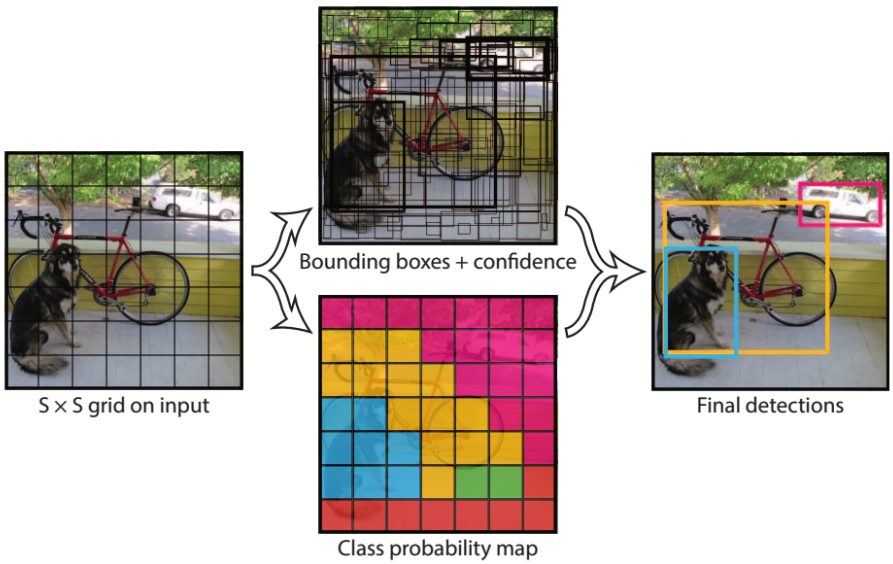


Figura 3.7: Arquitectura de YOLO

tectura, pasando a Darknet-53 como red troncal, y otros cambios como agregar mosaico como técnica de *data augmentation*. Los resultados fueron considerablemente mejores que en YOLOv3, obteniendo un AP_{50} de 65.7 %.

En un estudio aplicado a la detección de manzanas con YOLOv4 (L. Wu y cols., 2021), se obtuvo mAP de 96.85 %. Lamentablemente, el dataset utilizado no se encuentra disponible.

3.4.2. YOLOv5

YOLOv5 fue publicada tan solo meses después que YOLOv4, en 2020, por Ultralytics (Ultralytics, 2014), sin una publicación de artículo asociada. En esta versión se incorporan varias mejoras de la versión anterior y los cambios principales respecto a la arquitectura, con la diferencia de que está implementada utilizando el framework Pytorch (*PyTorch*, 2016). Por este motivo, YOLOv5 logra resultados sustancialmente mejores en cuanto a tiempos de entrenamiento y evaluación, y métricas de detección, alcanzando un AP_{50} de 50.7 % en COCO.

Además, cuenta con mantenimiento activo y gran apoyo de la comunidad. A modo de ejemplo, la versión actual es la 7.0, publicada en noviembre del 2022. En esta versión se agregó un quinto modelo, llamado nano (más pequeño que los anteriores).

En lo que respecta a la arquitectura, cuenta con un *backbone* *New CSP-Darknet53*, *neck* de *SPPF* y *head* *YOLOv3 Head* (Jocher y Waxmann, 2023), y está compuesto por las capas representadas en la figura 3.8 (Gutta, 2021), aunque en la versión 6.0, la capa “Focus” fue remplazada por “Conv2d”.

La red central o *Backbone* es una red neuronal convolucional que combina diferentes imágenes detalladas y extrae características de la imagen. La primera capa de esta red es el módulo de enfoque, diseñado para reducir los cálculos del modelo y acelerar la velocidad de entrenamiento.

En primer lugar, la imagen de entrada de tres canales se divide en cuatro fragmentos, transformando la imagen de $3 \times 640 \times 640$ (tamaño por defecto en YOLOv5s) en fragmentos de $3 \times 320 \times 320$. Posteriormente, se utiliza la operación de concatenación para conectar las cuatro secciones en profundidad, con un tamaño de mapa de características de salida de $12 \times 320 \times 320$. Luego, a través de la capa convolucional compuesta por 32 núcleos de convolución, se genera un mapa de características de salida con un tamaño de $32 \times 320 \times 320$. Finalmente, los resultados se normalizan y se envían a la siguiente capa (Yan y cols., 2021).

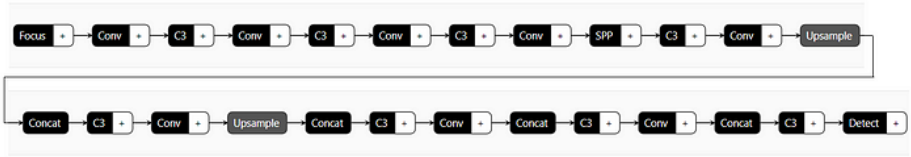


Figura 3.8: Arquitectura de YOLOv5.

3.4.3. YOLOX

En julio de 2021 se publicó la versión YOLOX (Ge, Liu, Wang, Li, y Sun, 2021), también implementada utilizando Pytorch, pero partiendo de YOLOv3, en la que se realizaron cambios importantes, que se listan a continuación:

- Utilización de *Decoupled head*, dado que comprueban experimentalmente que separando las tareas de localización y clasificación se obtienen mejores resultados.
- A nivel de *data augmentation*, agregan *Mosaic* y *MixUp*
- Es libre de anclas, quitando las anclas existentes en las versiones anteriores, argumentando que generan problemas y complejidad adicionales, y que requieren mayor cantidad de parámetros.
- Para compensar la falta de anclas, utilizan un área de 3x3 como centro del bounding boxes.
- Cambian el sistema de etiquetado.

3.4.4. YOLOv7

En julio de 2022 se publicó YOLOv7, de los mismos autores de YOLOv4. En su momento superó a todos los algoritmos de detección, con AP_{50} de 73.5 %

Versión	Fecha	Anclas	Framework	Red central	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Sí	Darknet	Darknet24	*63.4*
YOLOv3	2018	Sí	Darknet	Darknet53	36.2
YOLOv4	2020	Sí	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Sí	Pytorch	CSP v7 modificado	55.8
YOLOX	2021	No	Pytorch	CSP v5 modificado	51.2
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	RepConvN	56.8
YOLOv8	2023	No	Pytorch	YOLO v8	53.9

Tabla 3.1: Resumen de las arquitecturas de YOLO, extraído de (Terven y Cordova-Esparza, 2023), donde los AP de YOLO y YOLOv2 se evalúan sobre VOC2007, mientras que los otros se evalúan sobre COCO2017.

sobre COCO. Esto fue posible gracias a algunos cambios, como por ejemplo, en su arquitectura, que si bien no afectan el tiempo de detección, inciden en el tiempo de entrenamiento.

3.4.5. YOLOv8

Finalmente, en enero de 2023 se publicó YOLOv8 (Jocher, Chaurasia, y Qiu, 2023a), por *Ultralytics*, la misma empresa responsable de YOLOv5. Esta versión soporta no solo detección de objetos, sino segmentación de datos y seguimiento de múltiples objetos. Además, se puede instalar como librería e importar en cualquier código, pudiendo entrenarla y ejecutarla de manera extremadamente sencilla.

Siguiendo la tendencia actual, también es libre de anclas y utiliza *mosaic* como técnica de *data augmentation* durante el entrenamiento (excepto en las últimas *épocas*, donde se demostró experimentalmente que es perjudicial).

En lo que respecta a su arquitectura, la información disponible en este momento proviene de análisis directos del código (Solawetz, 2023), pero en términos generales es similar a la de YOLOv5, con diferencias como el uso de *decoupled head* (King, 2023).

Esta versión cuenta con cinco modelos: YOLOv8n (nano), YOLOv8s (chico), YOLOv8m (mediano), YOLOv8l (grande) y YOLOv8x (extra grande). El modelo YOLOv8x alcanza un AP de 53.9 % con imágenes de 640 píxeles, mientras que YOLOv5 alcanzaba únicamente 50.7 %. En la figura 3.9 se incluye una comparación entre YOLOv8 y algunas de sus versiones anteriores.

Cabe mencionar que, si bien YOLOv8 fue publicado cuando la etapa de investigación ya había concluido, de la misma manera que YOLOv6 (publicado en setiembre de 2022), se incluyó YOLOv8 por contar con rastreo nativo y por sus diferencias a nivel de arquitectura con YOLOv5, lo cual despertaba interés en su estudio.

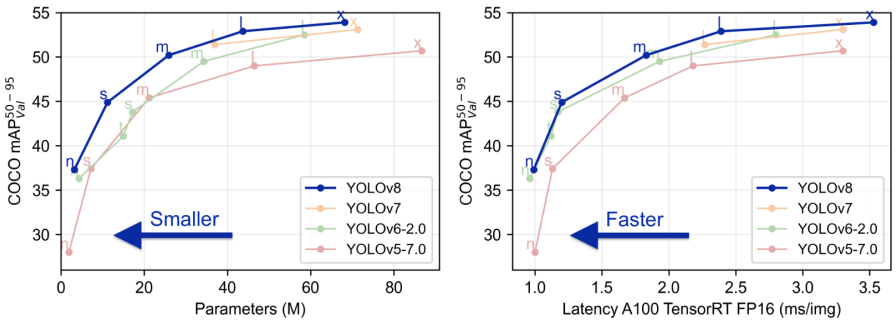


Figura 3.9: Comparación entre YOLOv5 a YOLOv8, extraída de (Jocher y cols., 2023b)

3.4.6. Algoritmos de seguimiento

El seguimiento de objetos es el proceso de localización y rastreo de objetos o personas en un video o secuencia de imágenes. Algunos de los usos más frecuentes de estos algoritmos se dan en videovigilancia y vehículos autónomos, pero también se utilizan en otros ámbitos de la robótica y visión por computadoras. Para esta última, el seguimiento implica analizar la variación de movimiento de cada detección y predecir dónde se va a encontrar cada objeto detectado en el siguiente cuadro de video (Leal-Taixé, Milan, Reid, Roth, y Schindler, 2015). A nivel de producción agrícola, los algoritmos de seguimiento de múltiples objetos pueden ser claves para solucionar el problema de conteo (W. Zhang y cols., 2022), si bien no están específicamente diseñados ni optimizados con ese fin.

A la hora de realizar seguimiento de frutas en plantaciones, la mayor dificultad para un algoritmo de seguimiento se encuentra en la alta oclusión de las frutas, por lo que interesaba buscar algoritmos que solucionen ese problema de la mejor manera posible. También conformaba un aliciente que el seguimiento pudiera hacerse en tiempo real, aunque no era prioritario, y poder cambiar el algoritmo de detección para poder realizar pruebas con diferentes redes neuronales.

En esta sección se hace un breve recuento de las principales diferencias entre los algoritmos seleccionados de acuerdo a sus respectivas publicaciones. Se incluyen a su vez SORT y DeepSORT para dar contexto histórico y facilitar la comprensión de la construcción de los algoritmos de seguimiento más recientes.

SORT

Al investigar algoritmos de seguimiento, se encuentra que la mayoría de los algoritmos actuales están basados en *The Simple Online and Realtime Tracking (SORT)* (Bewley, Ge, Ott, Ramos, y Upcroft, 2016). En el momento de su publicación, SORT era un algoritmo más simple que los algoritmos de seguimiento del momento. Utilizaba filtro Kalman (Kalman, 1960) para predecir la siguiente

posición del objeto y el algoritmo húngaro (Kuhn, 1955) para la asociación entre cuadros, con lo que logró convertirse en el estado del arte por ser 20 veces más rápido que los otros y obteniendo el mejor resultado con la métrica MOTA sobre el conjunto de datos *MOTChallenge 2015* (Leal-Taixé y cols., 2015).

Los algoritmos de seguimiento en general, y SORT en particular, funcionan en etapas, las cuales se listan a continuación:

1. Detección: se aplica un algoritmo de detección para identificar los objetos de interés. La elección del algoritmo de detección tiene un impacto significativo en el desempeño global.

2. Modelo de estimación: se aproxima el desplazamiento entre cuadros de cada objeto con una constante de velocidad lineal, tanto en su desplazamiento vertical y horizontal como en el tamaño que ocupa en la imagen, y se proyecta al siguiente cuadro.

3. Asociación de datos: se crea una matriz de costos entre los objetos detectados y los estimados, utilizando la distancia entre el IoU de cada detección con cada predicción de los cuadros anteriores. La asignación es la optimización de menor costo para las asociaciones entre las predicciones realizadas en el cuadro anterior y los objetos detectados en el cuadro actual, resuelto mediante el algoritmo Húngaro. Además, se impone un mínimo IoU para considerar una asignación.

4. Creación y detección de identidades de seguidores (*track identities*): en esta etapa se determina si se debe crear o destruir identificadores. Se crea un identificador si la superposición es menor que el IoU mínimo, ya que indica que el objeto no está siendo seguido. Los identificadores se consideran finalizados si no son detectados por cierta cantidad de cuadros (T_{Lost}), para prevenir errores de localización. En SORT, T_{Lost} es 1, por lo que si un objeto desaparece y reaparece, se le asigna un identificador nuevo.

Se observa que, para simplificar el problema, se asume que los objetos raramente tienen oclusiones, lo cual evidentemente no aplica a los árboles de manzanas. También se descartan las características de apariencia fuera de los bounding boxes detectados, y solo la posición y tamaño de los bounding boxes se usa en la estimación de movimiento y en la asociación de datos.

DeepSORT

Por lo anterior, SORT presenta problemas para seguir el movimiento de objetos con oclusiones, cambiando la identidad de los objetos si en algún momento quedan ocluidos y luego vuelven a aparecer. Por esto surge DeepSORT (Wojke, Bewley, y Paulus, 2017), donde reemplazan la métrica de asociación con otra métrica que combina la información del movimiento con la de aparición del objeto. Para esto, utiliza una red neuronal convolucional (CNN), entrenada para

volver a identificar objetos (en inglés *re-identification*), ReID, logrando así aumentar la robustez y disminuir errores, pero manteniendo el algoritmo simple y eficiente. Este algoritmo será la base o punto a comparación para todos los algoritmos que se describen a continuación.

ByteTrack

En ByteTrack (Y. Zhang y cols., 2022) se plantea que es un error descartar las detecciones con baja confianza, por lo que se propone un método de asociación denominado BYTE, que permite realizar asociaciones teniendo en cuenta las detecciones con baja confianza. Para esto se dividen las detecciones en puntuaciones de confianza altas o bajas, y se comienza por asociar detecciones de puntuaciones altas a los seguimientos. Algunos seguimientos no se asocian a ninguna detección, normalmente porque los objetos se ven ocluidos, son desenfocados o cambian de tamaño, por lo que luego se busca una asociación entre los seguimientos no asociados y las detecciones con bajo puntaje, aumentando los objetos seguidos a la vez que se filtra mejor el fondo.

Para la primera asociación, con las detecciones de alta confianza (D_{high}), se utiliza la asociación por IoU. Luego se utiliza el algoritmo húngaro para terminar la asociación por similitud. Las detecciones de (D_{high}) sin asociar y los seguimientos sin asociar se mantienen en conjuntos separados, D_{remain} y T_{remain} respectivamente. Para las detecciones en D_{remain} se crean nuevos seguidores.

Para la segunda asociación se utilizan las detecciones de baja confianza (D_{low}) y T_{remain} y se utiliza únicamente IoU como función de similitud, ya que probablemente existan oclusiones o desenfoque que haga que la comparación por apariencia no sea confiable. Los seguimientos que no se asocien luego de esta segunda instancia, se colocan en otro conjunto T_{lost}, mientras que las detecciones de D_{low} que no se asocien son descartadas. Los seguimientos en T_{lost} se mantienen hasta por 30 cuadros, luego de lo cual son descartados de los seguimientos a asociar.

StrongSORT

En 2022 surge StrongSORT (Du y cols., 2023), implementación basada en DeepSORT que busca mejorarla. Los cambios principales aplicados se listan a continuación.

- Se le agregan módulos avanzados como YOLO-X para la detección y BoT para la extracción de características de apariencia, remplazando la CNN original y obteniendo mejores resultados.

- Se reemplaza el banco de características, donde se reservan la información a largo plazo, con una estrategia menos sensible al ruido de la detección. Dicha estrategia actualiza el estado de la apariencia con un movimiento EMA (*exponential moving average*), incorporando la información de los

cambios entre cuadros y minimizando el impacto del ruido de las detecciones en el seguimiento.

- Se adopta el modelo ECC (*enhanced correlation coefficient maximization*) para compensar los movimientos de la cámara. Esta técnica estima el movimiento global de rotación y traslación entre cuadros.
- Los autores de StrongSORT plantean que el filtro Kalman es vulnerable a las detecciones en baja calidad y que ignora la información de la escala y ruido en la detección, por lo que se aplica el algoritmo NSA Kalman, que calcula la covarianza del ruido y utiliza el puntaje de confianza en la detección, mejorando la exactitud de los resultados.
- Para calcular la distancia entre detecciones, se tiene en cuenta tanto la distancia en apariencia como la de movimiento, aunque se pondera para darle mayor relevancia a la apariencia. DeepSORT, en cambio, solo tiene en cuenta la apariencia.

Con estos cambios, StrongSORT logra superar los resultados de DeepSORT en todas las métricas, obteniendo los mejores resultados con MOT20.

OC-SORT

Otro algoritmo de seguimiento que surge como propuesta de mejora sobre SORT es *Observation-Centric SORT* u OC-SORT (Cao, Weng, Khirodkar, Pang, y Kitani, 2022). En esta propuesta se identifican algunos errores de SORT que se busca solucionar: la sensibilidad al ruido en las detecciones, la acumulación de errores en el tiempo y estar centrado en las estimaciones.

Para reducir estos problemas, se propone un módulo de observación del estado de los objetos, para volver a actualizar las estimaciones de los cuadros y corregir las asignaciones de identificadores. Esta etapa se dispara cuando el seguimiento de un elemento se vuelve a activar, al ser asociado a una detección luego de un periodo de inactividad. Para esto se utilizan observaciones virtuales, que provienen de la trayectoria generada, utilizando los datos de cuando el objeto fue visto por última vez. A esto le denominan *Observation-centric Re-Update* (ORU).

También proponen una mirada "centrada en la observación" para incorporar la dirección en la matriz de costos utilizada en la asociación, a la que llaman Observation-Centric Momentum (OCM).

Finalmente, agregan un módulo, *Observation-Centric Recovery* (OCR), dedicado a verificar la última detección de un elemento seguido, para recuperar en caso de que se haya perdido. Esto se ejecuta luego de la etapa de asociación de identificadores, para tratar de asociar los objetos no asociados al cuadro anterior con otros identificadores que quedaron ocultos previamente, durante un corto intervalo de tiempo.

En la figura 3.10 puede verse la aplicación de los tres módulos que se agregan en OC-SORT. Los rectángulos rojos indican detecciones; los anaranjados, elementos seguidos activamente; los azules, elementos que todavía no están siendo

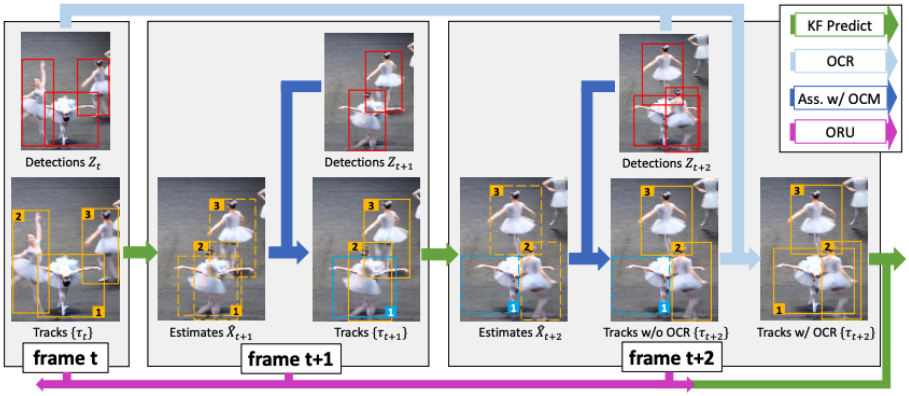


Figura 3.10: Ilustración del flujo de OC-SORT, extraída de (Cao y cols., 2022)

seguido, y las líneas punteadas son estimaciones realizadas por el filtro Kalman. El elemento 1 es detectado y seguido en el cuadro t, pero se pierde en el $t+1$ porque queda ocluido. En el siguiente cuadro se recupera, basado en la observación del cuadro t, gracias al módulo OCR. Esta asignación dispara el módulo ORU, que revisa los cuadros anteriores y reasigna el identificador a 1 en $t+1$, por lo que termina siendo seguido desde el cuadro t al $t+2$.

Capítulo 4

Decisiones tomadas

En este capítulo se abordarán las principales decisiones tomadas a lo largo del proyecto. En primer lugar, se comentan brevemente las diferentes etapas de la investigación. Luego, se presenta una concisa descripción de la información relevada respecto a la morfología y distribución de los árboles de manzanas en INIA Las Brujas, lugar en el cual se llevarán a cabo los experimentos. Seguidamente, se describe el proceso de implementación del simulador de campos de manzanas, haciendo énfasis en su funcionamiento y las tecnologías utilizadas. Por último, se detalla el proceso de selección de los modelos de detección y seguimiento utilizados.

4.1. Etapas del proyecto

El proyecto se dividió en tres etapas. En la etapa inicial, se llevó a cabo un proceso investigativo que involucró la búsqueda de publicaciones pertinentes. En este contexto, se generó el documento de estado del arte y se alcanzó un profundo nivel de comprensión sobre la materia en cuestión. Dicha comprensión abarcó aspectos cruciales, como las principales problemáticas, diferentes soluciones según las tecnologías empleadas en cada proyecto y cómo estas influenciaban en sus requerimientos y resultados. Como parte del proceso de inmersión en el problema, fue necesario estudiar características de las plantaciones de manzanas, siendo preciso consultar publicaciones y expertos en agronomía como orientadores.

A partir de la información recabada, se identificó la necesidad de disponer de un generador de datos para realizar experimentación, ya que no se encontraron videos de campos de manzanas sobre los que se pudieran realizar pruebas, ni conjuntos de datos etiquetados para video. Por este motivo, se adaptó un generador de simulaciones de plantaciones de tomate para que produzca campos con árboles de manzanas, como se detalla en la sección 4.3.

Paralelamente, se encontraron diversos conjuntos de datos etiquetados de imágenes de manzanas, parte de los cuales se describen en la sección 5.3. Aunque

no necesariamente correspondían a las condiciones específicas de las plantaciones de INIA Las Brujas ni estaban asociadas a proyectos con publicaciones previas, estas fuentes de datos demostraron ser un buen punto de partida trabajar con los algoritmos de detección.

Asimismo, fue necesario explorar alternativas a las redes neuronales utilizadas en las publicaciones encontradas, ya que algunas de ellas habían dejado de ser el estado del arte en detección. Este proceso requirió de investigar y probar tanto redes de detección como de seguimiento, y combinaciones de estas, buscando modelos relativamente simples de entrenar y suficientemente livianos para ejecutar con los recursos disponibles. Esto se detalla en profundidad en la sección 4.4.

Finalmente, se llevó a cabo la etapa de experimentación del proyecto, que consistió a su vez de cuatro instancias:

1. El entrenamiento de los modelos de redes neuronales para la detección de manzanas, con la respectiva comparación de resultados y selección de los mejores modelos entrenados de acuerdo a su desempeño.

2. La evaluación de los modelos de seguimiento, utilizando los modelos de detección seleccionados previamente y una serie de métricas específicas de seguimiento con el fin de comparar los resultados.

3. La aplicación de los modelos de seguimiento para el conteo de manzanas, limitado al estudio de la cantidad de manzanas recolectadas en una fila particular en INIA Las Brujas o generados con el simulador.

4. El análisis de los datos obtenidos y pruebas de ajuste aplicando regresión lineal y un modelo basado en coeficientes.

4.2. Morfología y distribución de los manzanos

Tomando en cuenta que los objetos a detectar y contar son manzanas, resultó necesario conocer en detalle las características morfológicas y de desarrollo de los manzanos. Para esto, se realizaron algunas reuniones con expertos del INIA para recabar información acerca de las plantaciones de manzanas y su comportamiento.

En general, un árbol se caracteriza por sus dimensiones y la cantidad de ejes que componen su tronco (algunos ejemplos se incluyen en la figura 4.1). En el caso de los manzanos, se relevó que los árboles de eje central son los más comunes, aunque los que se encuentran en las plantaciones del INIA Las Brujas son árboles de dos ejes.

En cuanto a las dimensiones, un árbol de manzanas puede alcanzar alturas de hasta 3.5 m, o 2.3 m en Muros Bajos (Cabrera y Rodríguez, 2021). El diámetro del tronco principal varía según el desarrollo del árbol, desde 15 mm cuando recién se planta, hasta 14 cm. Las frutas suelen aparecer a partir de los 50 cm o 60 cm desde el suelo.

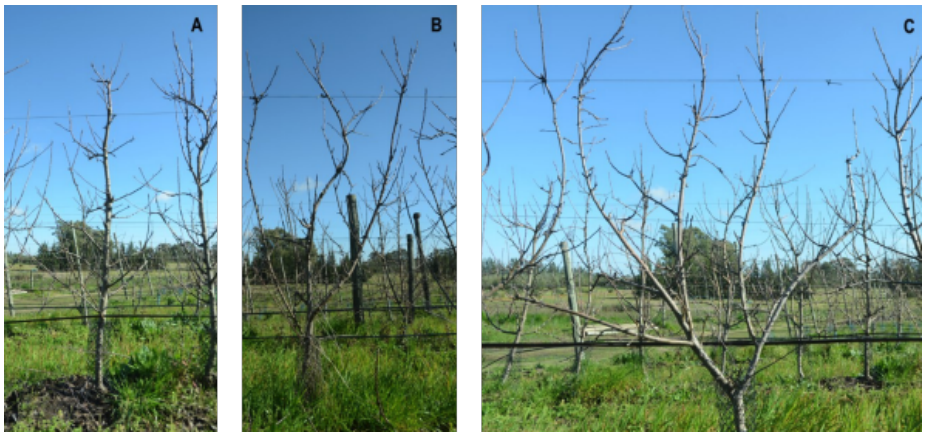


Figura 4.1: Ilustración de distintas cantidades de ejes en árboles, siendo A de un eje, B de tres ejes y C de 4 ejes. Imágenes extraídas de (Cabrera y cols., 2021)

A partir de los ejes centrales surgen las ramas, brindillas y espolones, siendo las segundas particularmente relevantes para el proyecto, ya que de estas surgen las manzanas y las hojas. Las brindillas son ramas de entre 15 cm y 40 cm de largo, que cuentan con un ancho de entre 6 mm y 20 mm, según sean más nuevas o viejas y tengan o no frutos.

Las manzanas varían de tamaño según la variedad, pero se consideran frutas comerciales a partir de los 65 mm. Dicho tamaño no presenta una distribución específica en cada rama, por lo que puede haber manzanas más grandes o pequeñas tanto hacia afuera como adentro del árbol. Tampoco se da un patrón fijo de distancia entre las manzanas de una misma rama, pudiendo separarse entre 5 cm y 15 cm.

Dada la aplicación de técnicas de raleo, las manzanas de producción agrícola no crecen en grupos tan apretados como las silvestres, pero aun así tienden a juntarse en grupos de hasta 5 manzanas, con hojas entre medio. Esto también depende de la variedad de manzanas, ya que las que son genéticamente muy grandes se dejan de a dos, para que no crezca demasiado la fruta, pero en otras variedades se busca que no quede más de una manzana por grupo.

El color y textura de las manzanas varía de acuerdo a la especie, aunque también puede variar un poco según la incidencia de la luz sobre ellas. Las hojas de los manzanos son bastante parecidas entre ellas, contando con unos 2 cm al momento del brote y hasta 8 cm en su máximo desarrollo.

Respecto a la distribución de los árboles en las plantaciones, estas se dan en filas de árboles, donde las filas tienen entre 2.5 m y 4.0 m de distancia entre sí, mientras que las plantas en cada fila se encuentran a una distancia de entre 1.0 m y 1.2 m.

4.3. Simulador

En la etapa de investigación del proyecto se identificó la necesidad de disponer de grabaciones de recorridos en campos de manzanas, que siguieran una trayectoria panorámica que cubra ambos lados de la fila de árboles. Sin embargo, no existían a la fecha datos públicos con estas características para campos de manzanas, y no era posible la construcción de un conjunto de datos, dado que en ese momento las manzanas aún no estaban maduras.

Por estos motivos, se decidió llevar a cabo el desarrollo de un simulador de campos de manzanas que permita resolver los problemas mencionados previamente. En esta sección se comentan los requisitos relevados, las herramientas utilizadas, detalles de implementación y forma de uso de la solución.

4.3.1. Requisitos relevados

En base a las características mencionadas en la sección 4.2 y requisitos funcionales para la aplicación, se presentan a continuación los principales requisitos relevados.

Requisitos de funcionalidad

- El simulador debe consistir en un ambiente simulado con un robot que circule en un campo de árboles de manzanas distribuidos de forma consistente a la realidad.
- El robot debe integrar sensores que publiquen su información en tópicos de ROS, debiéndose priorizar la integración de sensores de cámara y de posicionamiento.
- Los recorridos podrán ser guardados para su posterior utilización.

Requisitos sobre los árboles

- Los árboles pueden tener varios ejes, los más comunes son los de eje central.
- Los ejes principales tienen un diámetro máximo de 14 cm.
- La altura de los árboles va desde 2.3 m hasta 3.5 m.
- Las brindilla pueden tener un diámetro máximo de 2 cm.
- Los frutos se recolectan a partir de 40 cm de altura.

Requisitos sobre las frutas y hojas

- Cada brindilla puede tener hojas y de 3 a 5 frutas.
- Los frutos miden en promedio entre 6.5 y 7.0 cm.

- La textura de las manzanas depende de múltiples factores, como la luz solar incidente.
- Las hojas tienen todas la misma forma y textura, midiendo como máximo 15 cm.

Requisitos sobre el campo

- Los árboles se distribuyen en filas, a una distancia de 1.0 m a 1.2 m entre sí.
- Las filas están a una distancia entre 2.5 m y 4.0 m.
- El suelo del campo es pasto.

4.3.2. Herramientas utilizadas

A continuación se presentan algunos de los aspectos clave de las herramientas requeridas para el desarrollo del simulador de campos de manzanos. Esto se considera necesario ya que, dada la complejidad del proyecto, fue de gran utilidad comprenderlas para poder desarrollar el simulador.

ROS

ROS (Stanford Artificial Intelligence Laboratory, 2007), del inglés *Robot Operating System*, es una colección de herramientas y librerías de código abierto para el desarrollo de sistemas robóticos complejos. Provee servicios que se esperarían de un sistema operativo (de ahí su nombre), como por ejemplo, abstracción de hardware, control de dispositivos a bajo nivel, pasaje de mensaje entre procesos y manejo de paquetes. Además, tiene a disposición herramientas para obtener, construir, escribir y ejecutar código tanto de forma local como distribuida entre múltiples equipos.

Está basado en una arquitectura de grafos, conocida como grafo de ejecución, que consiste en una red punto a punto de procesos poco acoplados. Dichos procesos, o nodos, se comunican mediante distintos mecanismos, incluyendo la comunicación asíncrona a través de tópicos.

La comunicación por tópicos sigue el patrón de publicador y suscriptor, representada en la figura 4.2, donde cada tópico permite el desacoplamiento entre la producción de información y su consumo. En general, los nodos no están al tanto de con quién se están comunicando. Los nodos que generan los datos los publican en un tópico, mientras que los nodos que están interesados en determinados datos se suscriben al tópico correspondiente, por lo que cada tópico puede tener múltiples publicadores y suscriptores.

Ignition Gazebo

Gazebo (Koenig y Howard, 2004) es una colección de librerías de código abierto diseñadas con el fin de simplificar el desarrollo de aplicaciones de al-

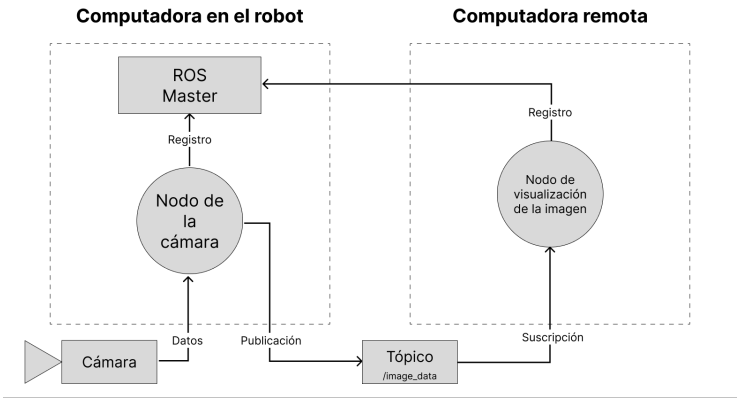


Figura 4.2: Ejemplo de arquitectura en ROS, un nodo de cámara recibe información de un sensor y la publica en un tópico, mientras un nodo de visualización se suscribe a ese tópico para recibir y mostrar la imagen. Ambos nodos se registran en el nodo maestro de ROS. Elaboración propia a partir de (Clearpath Robotics, 2015)

ta performance. El público objetivo de Gazebo son desarrolladores de robots, diseñadores y educadores.

A pesar de que su arquitectura está diseñada para ajustarse a múltiples casos de uso, el caso de uso principal de esta herramienta es el desarrollo de simulaciones realistas en escenarios robóticos.

En lo que respecta a simulación, Gazebo permite lograr excelentes resultados por múltiples factores, entre los que destacan su alta precisión en cuanto a física, gráficos 3D avanzados y simulación de sensores, incluyendo cámaras monoculares, de profundidad, LiDAR e IMU.

Otro factor muy importante es su integración con ROS, a partir de un puente que traduce automáticamente mensajes en el formato de Gazebo a mensajes de ROS. También provee una interfaz gráfica utilizada para visualizar las simulaciones y acceder a un conjunto de tópicos auxiliares, incluyendo tópicos de visualización, entrega de mensaje, control del mundo simulado y estadísticas.

Cabe notar que la versión de Gazebo utilizada en este proyecto lleva el nombre Ignition Fortress, respondiendo a un cambio en la implementación del producto. Sin embargo, el producto volvió a llamarse Gazebo en versiones posteriores.

Blender

Blender (Blender Foundation, 2022) es un software gratuito y de código abierto para la creación de contenido en 3D. Soporta todas las etapas para la generación de contenido en 3D, incluyendo el modelado, animación, simulación, renderizado, composición y seguimiento de movimiento. Además, provee

una API para Python que permite interactuar con la aplicación e implementar herramientas personalizadas.

Es una aplicación multiplataforma basada en OpenGL (Woo, Neider, Davis, y Shreiner, 1999), por lo que se ejecuta correctamente en dispositivos Linux, Windows y Macintosh.

Blender provee un intérprete de Python embebido que ejecuta en paralelo a la ejecución del proceso principal (Blender Foundation, 2014). Dicho intérprete permite ejecutar *scripts* para dibujar en la interfaz de usuario, y es utilizado por algunas herramientas internas de Blender. Además, provee módulos propios, como `bpy` y `mathutils`, que pueden ser utilizados para acceder a datos, clases y funciones de Blender.

4.3.3. Código fuente de referencia

La solución presentada parte del proyecto *Fields Ignition* (Polgár, 2020), que se desarrolla con el objetivo de generar ambientes aleatorios de pruebas para robots agrícolas. En particular, se pretende generar campos de árboles de tomate aleatorios, utilizando Ignition Gazebo y la *API* de Python provista por Blender. En la figura 4.3 se puede observar un ejemplo de mundo simulado, generado a partir de este proyecto.

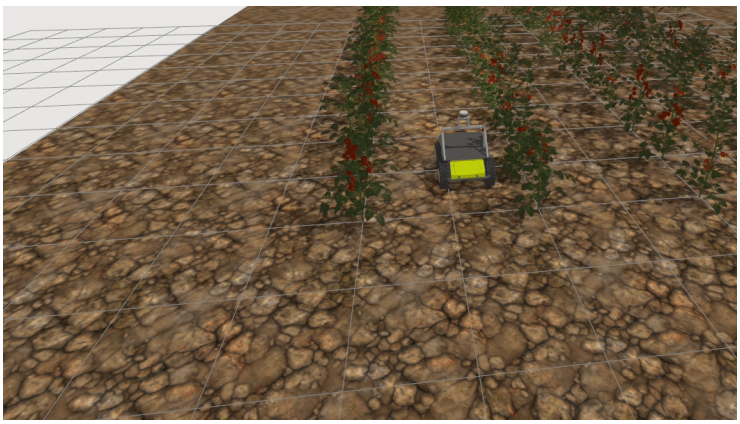


Figura 4.3: Ejemplo de instancia generada con "Fields Ignition". Imagen extraída de (Polgár, 2020).

A nivel de implementación, cada instancia generada parte de una estructura base (archivos y directorios), que incluye elementos estáticos (configuración del entorno, iluminación, modelo del robot y del suelo), y árboles que son agregados de forma dinámica a partir de los parámetros. Para simplificar la generación de instancias a partir de esta estructura, utiliza de la librería `CookieCutter` (A. Roy, 2013).

Entre los parámetros que se permiten definir, se encuentran la cantidad de árboles y de filas. A partir de dichos parámetros, se calcula la posición de cada

árbol, y se incluye en el archivo de configuración el modelo generado en Blender.

Tanto el mundo simulado como los modelos utilizados por este, son definidos en el formato SDF (OSRF, 2012), que puede ser interpretado por Ignition para su visualización. Además, los sensores del robot son completamente funcionales, publicando en tópicos de Ignition, que luego son asociados a tópicos de ROS.

A su vez, el proyecto provee una implementación de un detector simple de tomates, que incluye un nodo de ROS que publica la posición de los tomates de la instancia, y otro que identifica aquellas que son visibles. No se entrará en detalle sobre esta solución, dado que no fue utilizada en el presente proyecto.

4.3.4. Solución propuesta

El desarrollo de la solución se puede dividir en dos etapas, la etapa de generación de los árboles y la etapa de generación del mundo simulado. A continuación se detalla en qué consiste cada una de estas etapas.

Generación de los árboles

Los árboles de manzanas y los de tomates tienen diferencias morfológicas considerables. En el proyecto original (Polgár, 2020), los árboles generados consisten en una rama principal perpendicular al suelo, y pequeñas ramas que son colocadas a distintas alturas, de las cuales crecen las hojas y frutos. En el caso de los árboles de manzanas, se tienen distintas morfologías posibles (Cabrera y cols., 2021). Los manzanos pueden tener una o más ramas principales, donde nacen ramas secundarias. En las ramas secundarias crecen brindillas, en las que brotan las hojas y las manzanas.

El proceso de construir el árbol también puede dividirse en dos etapas, la generación del esqueleto, y la agregación de frutas y hojas. Este proceso se desarrolla utilizando la *API* de Python de Blender, y da como resultado un árbol, como el de la figura 4.4a. El código que se encarga de esto interpreta una serie de parámetros, que determinarán la morfología del árbol, entre los que destacan los tamaños y cantidades (definida como un coeficiente) de manzanas y hojas.

La etapa de generación del esqueleto comienza con las ramas principales y el tronco. Se asume que el tronco es una rama principal, por lo cual, si el parámetro es 1, el árbol tiene una única rama principal (el tronco). Para cada rama principal se generan ramas secundarias, de las cuales nacen las frutas y hojas. En la figura 4.4b se puede observar un ejemplo de esqueleto de un árbol generado.

Cada rama es un conjunto de tramos con forma cilíndrica de 10 cm de alto (denominados nodos), que están conectados entre sí, dando como resultado la forma de la rama. El diámetro de cada tramo está dado por la posición respecto a la base de la rama y el coeficiente de decrecimiento del radio de la rama. El último tramo está cerrado en su extremo superior. La cantidad de ramas secundarias se calcula a partir de la altura de la principal, la distancia entre

(a) (b)

Figura 4.4: Ejemplo de un árbol (4.4a) y esqueleto del árbol (4.4b) generado por el simulador

cada una de estas y la distancia desde la base de la rama principal y la primera rama secundaria.

En cada uno de los tramos se agregan las manzanas y las hojas con probabilidades uniformes $p1$ y $p2$ dadas por los parámetros. Cada manzana puede tomar de forma aleatoria una de las tres texturas expuestas en la figura 4.5, y tiene una forma fija, dada por el modelo visible en la figura 4.6a, aunque el tamaño es aleatorio en un rango dado. Análogamente, las hojas son agregadas a los nodos, con un tamaño aleatorio dentro de cierto rango, pero con una textura y forma fijas, exhibidas en la figura 4.6b.

Generación del mundo simulado

Para generar cada instancia se necesitan cuatro parámetros clave: la cantidad de filas (n), la cantidad de árboles por fila (m), la distancia entre árboles y la

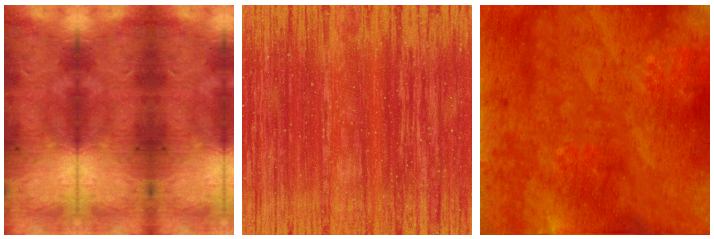
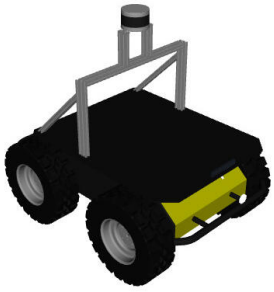


Figura 4.5: Texturas para las manzanas

(a) Modelo de manzana

(b) Modelo de hoja



(c) Modelo del robot Husky. Imagen extraída de (OpenRobotics, 2021)

Figura 4.6: Modelos utilizados en la generación de árboles de manzanas.

distancia entre filas. A partir de estos parámetros, se generan los $n \times m$ árboles siguiendo el proceso detallado en la sección anterior, con los parámetros definidos en el archivo indicado en el parámetro `-file`.

La instancia se genera partiendo de una plantilla utilizando `CookieCutter`, tal como se describió en la sección 4.3.3. El archivo principal, que define la estructura del mundo, es el llamado `apple_field.sdf`, donde se incluyen los modelos de los árboles generados, el suelo y el robot, utilizando el formato SDF.

El modelo de robot elegido es el robot Husky (OpenRobotics, 2021), que incluye sensores de navegación, LiDAR y cámara RGB. El modelo original se puede visualizar en la figura 4.6c, sin embargo, este se modificó de forma que la posición de la cámara sea tomada como parámetro de la generación de la instancia, y se elimina el modelo 3D que representa el soporte de los sensores, a modo de evitar posibles interferencias con la cámara. Además, los parámetros de configuración de la cámara se modifican para simular uno de los lentes de las cámaras ZED 2 (StereoLabs, 2020), con el fin de replicar con mayor precisión el ambiente real.

El modelo de suelo consiste de una superficie cuadrada modelada en un objeto *DAE*, al cual se le asignan físicas de colisión para evitar que el modelo pase a través de él. Respecto al modelo original, cambia únicamente la escala

Argumento	Descripción
world_name	Nombre del mundo generado.
field_id	Nombre identificador de la instancia del simulador.
row_length	Cantidad de árboles por fila.
row_count	Cantidad de filas.
row_dist	Distancia entre cada fila.
crop_dist	Distancia entre árboles por fila.
origin_coordinates	Coordinadas de origen para la posición de los árboles.
tree_properties	Propiedades de los árboles generados.
camera_position	Posición de la cámara.
husky_initial_position	Posición inicial del robot.

Tabla 4.1: Parámetros para la generación de las instancias del simulador.

y la textura, pasando de simular un suelo de tierra a pasto. Los parámetros configurables para la generación de la instancia se detallan en la tabla 4.1.

Ejecución de la simulación e interfaz gráfica

El simulador se instala como un paquete de ROS llamado *fields_ignition*. De esta forma, el lanzamiento de las instancias simuladas se puede realizar con la herramienta *roslaunch*, ya que se provee un archivo de lanzamiento de ROS (`field.launch`).

La ejecución de una instancia del simulador se puede traducir al esquema representado en la figura 4.7. Ignition interpreta el esquema definido en el archivo SDF que define la instancia y publica los tópicos de Ignition correspondientes, los cuales son traducidos a mensajes de ROS, que se publican en los tópicos de ROS. Entre los mensajes publicados se encuentra mensajes de navegación, de la cámara y de otros sensores como el LiDAR.

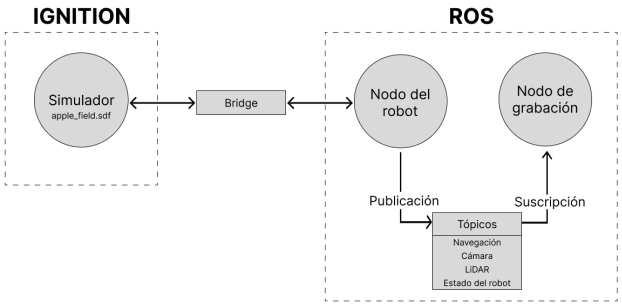


Figura 4.7: Arquitectura del simulador.

Al lanzar una instancia, se despliega una ventana de Ignition Gazebo, con controles de navegación y ventanas de visualización del tópico de la cámara y de las coordenadas de GPS. Esta ventana se puede observar en la figura 4.8.

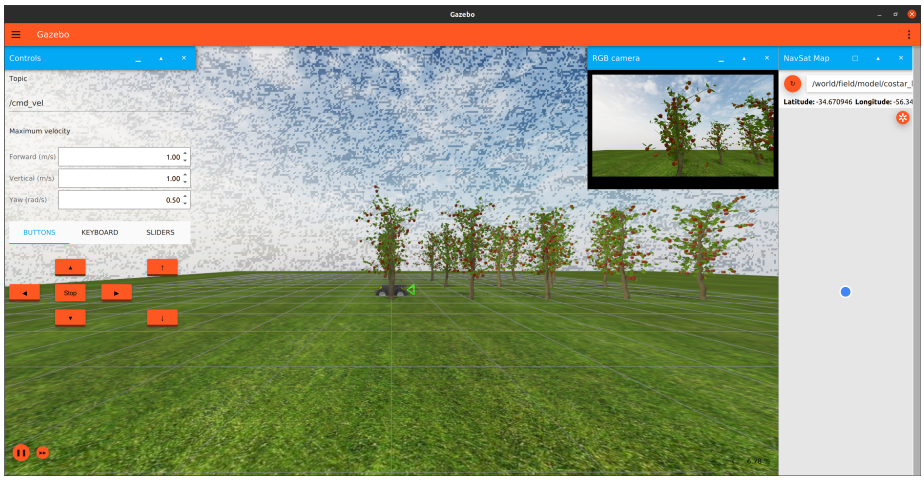


Figura 4.8: Ventana de Ignition con una instancia de simulador ejecutando.

Se provee además una herramienta auxiliar desarrollada para simplificar la grabación de la ejecución. Esta herramienta consiste en un nodo de ROS que ejecuta un *script* en Python, que se suscribe al tópico de la cámara RGB y acumula cada mensaje en un video con formato MP4. Para ejecutar esta herramienta se utiliza un archivo de lanzamiento, que recibe como parámetro el nombre de la instancia que se está ejecutando y un nombre con el que se guarda.

Para simplificar el proceso de generación, lanzamiento y grabación de instancias, se provee una interfaz gráfica, exhibida en la figura 4.9. Dicha interfaz permite el lanzamiento de las instancias generadas, listando en un selector las instancias disponibles. Permite además la grabación de un video o un bag del recorrido, que se almacenarán dentro el directorio de la instancia generada.

Finalmente, la interfaz permite la generación de instancias a partir de la configuración disponible en el campo de texto, completamente modificable por el usuario.

4.3.5. Posibles mejoras

Si bien la solución desarrollada se considera suficiente para el contexto de este proyecto, existen múltiples mejoras que pueden llevarse adelante. La primera y principal es la de generación de distintas morfologías de árboles, dado que, en la solución actual, esto no está optimizado cuando se tienen múltiples ramas principales.

Como trabajo futuro, se podría agregar la posibilidad de modificar las condiciones ambientales como la iluminación, condiciones climáticas, irregularidades del terreno, entre otras. En lo que respecta al robot, se podrían agregar modelos alternativos al robot Husky, además de agregar sensores que permitan capturar otros aspectos de la simulación, como por ejemplo, una cámara estéreo.

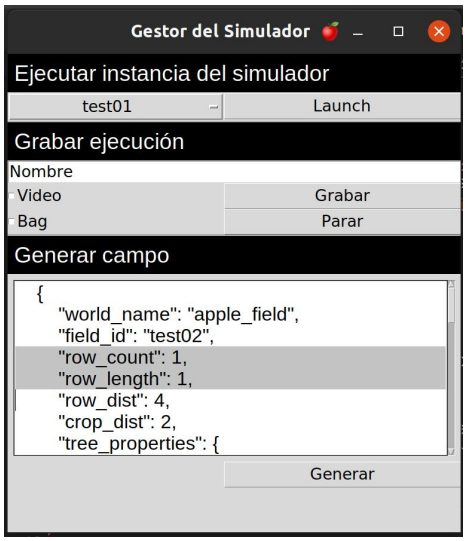


Figura 4.9: Interfaz de usuario del simulador.

4.4. Selección de modelos de redes neuronales

En esta sección se detalla el proceso de selección de los modelos de detección y seguimiento de objetos para resolver el problema de detección y conteo de manzanas. Dicho proceso comenzó con la revisión de distintas implementaciones de los modelos del estado del arte, descartando proyectos inactivos o incompatibles con los recursos disponibles. Esto requirió de numerosas pruebas, ya que varias de las ejecuciones de las redes neuronales (y en particular su entrenamiento) requirieron adaptaciones de código y actualizaciones de dependencias.

Una vez elegidas las arquitecturas, los modelos utilizados se seleccionaron a partir de un análisis de métricas como la precisión media (mAP), el tiempo de evaluación y el tamaño del modelo.

4.4.1. Modelos de detección

A nivel de redes neuronales para detección, se seleccionaron modelos de YOLO y de Faster-RCNN, con el fin de comparar arquitecturas diferentes. Dichas selecciones se detallan a continuación.

YOLO

Como se mencionó en la sección 3.4, las implementaciones seleccionadas para la parte experimental fueron YOLOv5 y YOLOv8. La elección de YOLOv5 fue sencilla, ya que el algoritmo era el estado del arte en el momento de la búsqueda inicial, con respaldo de la comunidad y múltiples aplicaciones documentadas,

incluyendo casos de detección de manzanas. En particular, se escogió el modelo *yolov5s.yaml* (*small*), por ser el más liviano de los disponibles en el 2022.

La selección de YOLOv8 fue más compleja, siendo que este fue publicado luego de la etapa de investigación, pero se llevó a cabo debido al buen desempeño de este algoritmo y porque incluía la posibilidad de ejecutar seguimiento nativo. Para esta implementación se seleccionó el modelo $YOLOv8n$ (*nano*), por ser el más liviano y rápido de entrenar y ejecutar.

Si bien se realizaron pruebas con YOLOX para la detección y conteo de manzanas, y se consideró fuertemente esta versión para incluir en la experimentación, eventualmente fue descartada, ya que no era sencilla de entrenar. Al intentar entrenarla y ejecutarla se obtienen errores, principalmente debido a que solo funciona con versiones específicas de algunas librerías, varias de las cuales quedaron obsoletas.

De manera similar, se barajó la posibilidad de utilizar YOLOv7, pero también fue descartado por presentar dificultad en el entrenamiento. En los intentos que se hicieron de entrenar YOLOv7 con conjuntos de datos de manzanas, se produjeron errores poco explícitos que impidieron finalizar cualquier entrenamiento, tanto al intentarlo localmente, como con Google Colab.

Faster R-CNN

En la etapa de investigación, se observó que la librería Detectron2 (Y. Wu, Kirillov, Massa, Lo, y Girshick, 2019), desarrollada por Facebook, ofrece una forma más simple e intuitiva de utilizar Faster R-CNN que en las implementaciones oficiales. Detectron2, implementado en Python y utilizando la librería Pytorch, proporciona una interfaz sencilla que facilita la implementación, entrenamiento y evaluación de algoritmos de detección y segmentación, entre los que destaca Faster R-CNN.

En lo que respecta a Faster R-CNN, Detectron2 provee una serie de modelos preentrenados (Y. Wu, 2019), que combinan distintas redes *backbone* y cantidad de épocas de entrenamiento (sobre COCO). Para cada combinación, se encuentran publicados resultados de referencia que incluyen parámetros como el tiempo de entrenamiento e inferencia, el espacio de almacenamiento, y la medida mAP sobre el conjunto de datos COCO. A partir de estos resultados, los modelos seleccionados fueron aquellos con mejor desempeño en cuanto a tiempo de inferencia y medida mAP, que se listan a continuación:

1. faster_rcnn_R_50_FPN_3x: con mAP de 40.2 % y un tiempo de inferencia de 0.038 segundos.

2. faster_rcnn_X_101_32x8d_FPN_3x: con mAP de 43.0 % y un tiempo de inferencia de 0.098 segundos.

Donde R_50_FPN y X_101_32x8d_FPN son las redes *backbone* y $3x$ es la cantidad de épocas sobre el conjunto COCO. En adelante, a estos modelos se les nombrará como Faster-50 y Faster-101 respectivamente.

4.4.2. Algoritmos de seguimiento

Dados los modelos de seguimiento que interesaban probar, se buscaron implementaciones que permitieran ejecutar algoritmos de detección con diferentes algoritmos de seguimiento. Si bien se encontraron varias implementaciones de este tipo, se terminó optando por una que además ofrecía la posibilidad de exportar métricas en el formato de MOT Challenge (Doerflinger, Miller, Nock, y Watkins, 2015), ya que facilitaba la evaluación de los resultados de seguimiento.

Dicho proyecto (Broström, 2022), ofrece como modelos de detección tanto a YOLOv5 como a YOLOv8, que en el momento de esta selección eran los candidatos principales para ser probados en con seguimiento.

Respecto a los algoritmos de seguimiento, el proyecto incluye como opciones a OCSort, ByteTrack y StrongSort. Si bien se incluyen otras opciones, estos algoritmos presentan numerosas diferencias y se consideraron suficientes para la etapa de experimentación.

Capítulo 5

Experimentación

En este capítulo se describe la etapa de experimentación del proyecto, comenzando por el análisis de la infraestructura para la ejecución de los experimentos. Seguidamente, se abordan las métricas de evaluación empleadas con el propósito de medir el rendimiento de a la detección y seguimiento de objetos, detallando el propósito y cálculo de cada métrica seleccionada.

A continuación, se presentan los conjuntos de datos que fueron utilizados, abarcando tanto la detección de objetos como el conteo de los mismos. Además, se establecerán directrices fundamentales que servirán como referencia para analizar los resultados obtenidos.

En última instancia, se expondrán los experimentos realizados y los resultados obtenidos, interpretando dichos resultados y analizando las causas de los errores obtenidos.

5.1. Infraestructura

Tomando en cuenta que parte de lo que se busca evaluar son los tiempos de entrenamiento y ejecución de los modelos, era necesario realizarlos en un ambiente de ejecución estable. Es por esto que se decidió utilizar la plataforma Google Colab (*Google Colab*, 2017), dado que provee un ambiente de ejecución Python con disponibilidad de GPUs. Para entrenar los modelos de detección, considerando que los tiempos de ejecución son considerablemente más altos, siendo insuficiente la utilización del plan gratuito de Colab, fue necesaria la utilización del plan Pro+ de la plataforma. Sin embargo, para la evaluación de los modelos se utilizó el plan gratuito. Los detalles de infraestructura para cada etapa se detallan en la tabla 5.1.

5.2. Métricas

En esta sección se abordan las métricas utilizadas para evaluar la detección y el seguimiento. En ambos casos, se presentan los conceptos fundamentales para

Hardware	
Entrenamiento	**Evaluación**
RAM 16 GB	RAM 13 GB
Intel(R) Xeon(R) CPU @ 2.20GHz	Intel(R) Xeon(R) CPU @ 2.20GHz
NVIDIA A100, P100 o T4	NVIDIA K80 o T4
Software	
Python 3.10	
Cuda 11.1	

Tabla 5.1: Infraestructura de experimentación

su comprensión y se describe en el procedimiento de cálculo.

5.2.1. Métricas de detección

La métrica comúnmente utilizada para evaluar modelos de detección de objetos es la llamada *Average Precision*, tradicionalmente utilizada como *Mean Average Precision* (*mAP*) (Terven y Cordova-Esparza, 2023). Sin embargo, antes de entrar en detalle sobre dicha métrica, es necesario comprender algunos conceptos y otras métricas que forman parte de este cálculo.

Conceptos base

La fórmula de *mAP* se basa en los conceptos de matriz de confusión, intersección sobre unión, recuperación y precisión, más conocidos por sus nombres en inglés: *confusion matrix*, *intersection over union (IoU)*, *recall* y *precision* respectivamente (Shah, 2022).

Para calcular la matriz de confusión se necesitan cuatro atributos:

- Verdaderos positivos (TP, *True Positives*), que indica cuántos objetos el modelo identificó correctamente.
- Verdaderos negativos (TN, *True Negatives*), que indica cuántos objetos el modelo no identificó y no pertenecen al *ground truth*.
- Falsos Positivos (FP, *False Positives*), que indica los objetos que el modelo identificó y no pertenecen al *ground truth*;
- Falsos Negativos (FN, *False Negatives*), que indica los objetos que el modelo no identificó y pertenecen al *ground truth*.

Estos valores se agrupan en una matriz de 2×2, tal como se puede observar en la figura 5.1, donde las filas representan los valores predichos, y las columnas los valores reales en el *ground truth*.

La métrica *IoU* indica el grado de solapamiento de la *bounding box* predicha por el modelo, en comparación con la *bounding box* perteneciente al *ground*

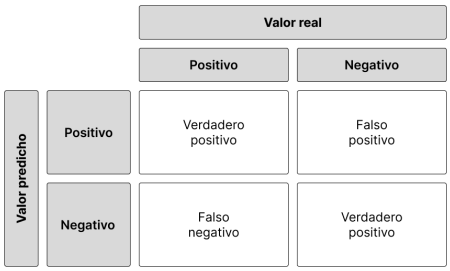


Figura 5.1: Matriz de confusión. Elaboración en base a (Shah, 2022)

truth. Como se puede observar en la figura 5.2, esta medida se calcula como el cociente entre el área de solapamiento y el área de unión de ambas *bounding boxes*.

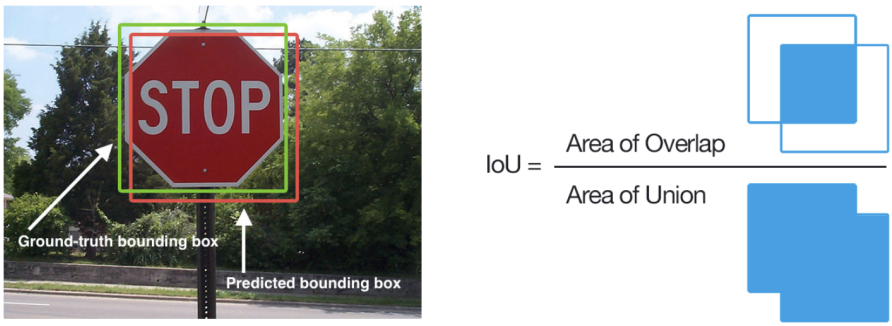


Figura 5.2: A la izquierda, objeto delimitado por la *bounding box* etiquetada y la predicha. A la derecha, la fórmula para calcular la IoU. Imagen extraída de (Shah, 2022).

Finalmente, la precisión mide la proporción de predicciones positivas correctas del modelo respecto al total de predicciones positivas, mientras que la recuperación mide la capacidad del modelo para detectar todos los ejemplos positivos en el conjunto de datos, en relación con el *ground truth*. Las fórmulas para estas métricas se detallan en las ecuaciones 5.1 y 5.2 respectivamente.

$$precisión = \frac{TP}{TP + FP} \qquad (5.1) \qquad recuperación = \frac{TP}{TP + FN} \qquad (5.2)$$

Dado que el objetivo es predecir *bounding boxes*, para calcular la precisión y la recuperación se define el concepto de umbral de IoU, del inglés, IoU *Threshold*. El IoU *Threshold* es un valor umbral utilizado para determinar si la detección de objetos es correcta o no. Si el IoU entre la *bounding box* predicha y la real supera

el valor del umbral, se considera que la predicción es correcta y se obtiene un verdadero positivo. Si, por el contrario, el IoU no supera dicho umbral, entonces la predicción no es correcta y se obtiene un falso positivo. En la figura 5.3 se puede comprender mejor este concepto; tomando un umbral de 0.5, se puede notar que la imagen a la izquierda es un falso positivo (dado que el IoU es 0.3, menor a 0.5), mientras que la imagen de la derecha es un verdadero positivo (dado que el IoU es 0.7, mayor a 0.5).

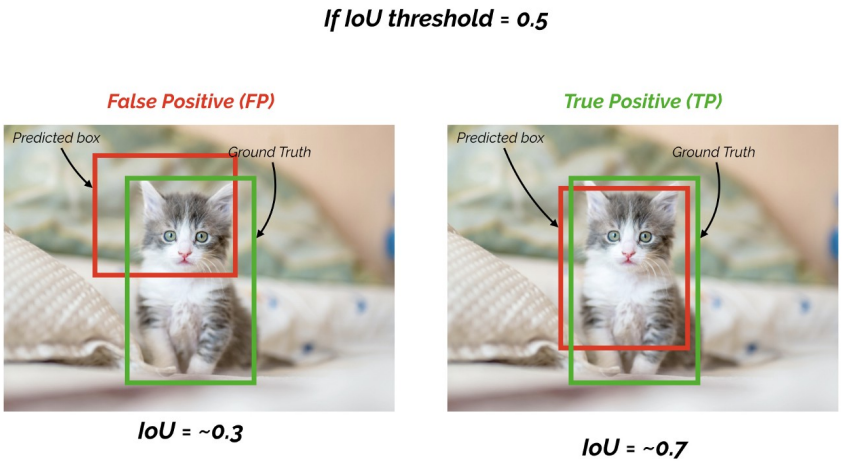


Figura 5.3: Cálculo del *IoU threshold*. Imagen extraída de (Shah, 2022)

Cuando un modelo tiene alta recuperación, pero baja precisión, el modelo clasifica la mayoría de las entradas positivas correctamente, pero tiene una gran cantidad de falsos positivos, es decir, clasifica múltiples entradas negativas como positivas. Por otro lado, si el modelo tiene alta precisión, pero baja recuperación, tiene un buen rendimiento a la hora de calificar una entrada como positiva, pero únicamente califica una parte de las entradas positivas (Shah, 2022).

Dicho esto, a la hora de evaluar un modelo, no es suficiente observar únicamente la precisión o la recuperación, sino que se deben considerar métricas que las combinen. Una de estas métricas es la medida $F1$, que es la media armónica entre la precisión y la recuperación, y se calcula como se describe en la ecuación 5.3.

$$F1 = \frac{2 * Precisión * Recuperación}{Precisión + Recuperación} \tag{5.3}$$

Otro concepto importante es el de curva precisión-recuperación (curva PR). La curva PR representa gráficamente la relación entre la precisión y la recuperación de un modelo, para distintos umbrales de confianza (Anwar, 2022). Dicha curva se calcula variando en pequeños incrementos el umbral de confianza, y calculando la precisión y recuperación en cada uno de estos. Como resultado, se

representan los valores en una gráfica que tiene en su eje horizontal los valores de recuperación, y en su eje vertical los valores de precisión. En la figura 5.4 se puede ver un ejemplo de curva precisión-recuperación, y sus correspondientes valores de confianza.

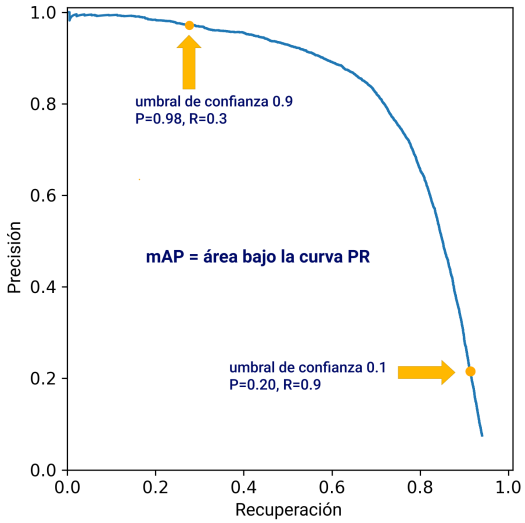


Figura 5.4: Ejemplo de curva de Precisión-Recuperación. Elaboración en base a (Jocher, 2020)

A partir de la curva PR se define la métrica precisión media (*Average Precision*), que se calcula como el área bajo la curva PR, y su fórmula está dada por la ecuación 5.4. Es importante notar que el *IoU threshold* elegido impacta directamente en el valor de esta métrica.

$$AveragePrecision(AP) = \int_{r=0}^{1} p(r)\, dr \tag{5.4}$$

Mean Average Precision

Como fue mencionado previamente, la métrica *Mean Average Precision* (mAP) se utiliza de forma estándar para evaluar modelos de detección de objetos.

El cálculo de esta métrica se realiza partiendo de un conjunto definido de *IoU thresholds*. Para cada *IoU* y para cada clase k que el modelo debe identificar, se calcula el valor de AP. Esto elimina la necesidad de elegir un *IoU* óptimo, ya que se utiliza un conjunto de *IoU thresholds* que cubren de principio a fin los valores de precisión y recuperación, promediando los resultados de los *IoU* (AP por clase) y luego promediando los AP de todas las clases. Para mayor claridad, la ecuación 5.5 representa la fórmula de mAP.

$$mAP = \underbrace{\frac{1}{n}}_{\substack{\text{n = número} \\ \text{de clases}}} \sum_{k=1}^{n} \overbrace{AP_k}^{\substack{AP_k = \text{AP} \\ \text{por clase}}} \tag{5.5}$$

En la figura 5.5 se muestra un ejemplo de cálculo de mAP para un modelo que debe predecir tres clases. Para esto se promedian los resultados de AP de todas las clases, donde cada AP toma a su vez cuatro valores de *IoU thresholds* distintos.

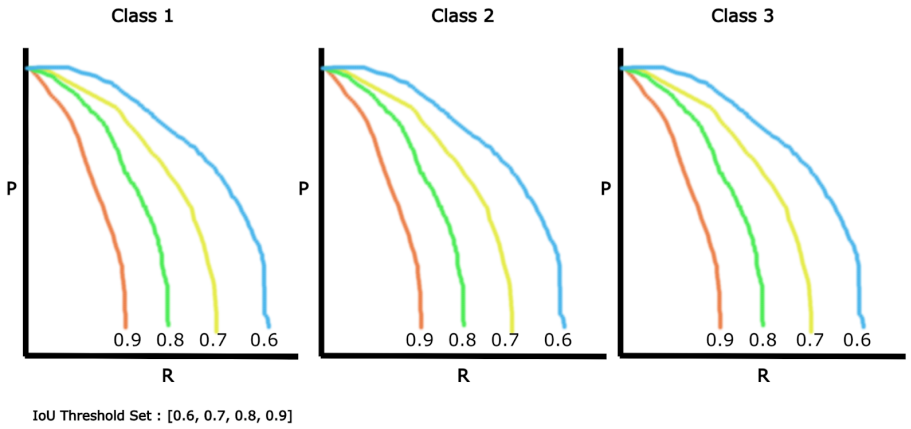


Figura 5.5: Cálculo de mAP para cada clase en el conjunto de datos, tomando como IoU para calcular el AP por clase los valores: 0.6, 0.7, 0.8 y 0.9. Imagen extraída de (Shah, 2022).

Es claro que, en el caso de detección de una única clase, la mAP corresponde con la AP para dicha clase.

Métricas de COCO

El llamado *COCO Challenge* es un desafío en el área de detección de objetos basado en el conjunto de datos COCO (*Common Objects in Context*) (Lin y cols., 2015), que permite evaluar distintos modelos de detección sobre las 80 clases disponibles en dicho conjunto.

Además, provee un conjunto estandarizado de métricas para evaluar distintos modelos de detección. Estas métricas consisten en distintas variaciones de *Average Precision*, descritas en la tabla 5.2. Dichas variaciones incluyen evaluar en distintos umbrales de IoU, además de considerar diferentes escalas de objetos.

Otra métrica utilizada en *COCO Challenge* es la recuperación media (*Average Recall, AR*), que se calcula como el promedio de la recuperación en diferentes

Métrica	Descripción
AP	Precisión media promediada sobre un rango de umbrales de IoU, entre 0.50 a 0.95. También es conocida como $AP_{50:95}$.
AP_{50}	Precisión media cuando el umbral de IoU es 0.50.
AP_{75}	Precisión media cuando el umbral de IoU es 0.75.
AP_{small}	Evaluando únicamente objetos pequeños (área $< 32^2$ píxeles).
AP_{medium}	Evaluando únicamente objetos medianos ($32^2 <$ área $< 96^2$ píxeles).
AP_{large}	Evaluando únicamente objetos grandes (área $> 96^2$ píxeles).

Tabla 5.2: Variaciones de AP utilizadas por COCO. Elaboración propia a partir de los datos de (Padilla y cols., 2021).

umbrales de IoU. En particular, para COCO es el promedio de la máxima recuperación obtenida en diferentes umbrales IoU, en general, entre 0.5 y 1.

AR es una métrica útil en escenarios donde detectar todas las instancias de una clase es más importante que detectarlas con un mayor valor de confianza (Padilla y cols., 2021). Dado que no considera dicho valor en su cálculo, la métrica puede no ser útil en todos los casos, además de que nunca se debe utilizar como única métrica de evaluación, sino como complemento de métricas como el AP o $F1$.

Tomando en cuenta que el objetivo del proyecto es el conteo de manazas, y que es importante detectar únicamente manzanas (por lo que el valor de confianza juega un rol importante), se decide no tomar en cuenta el AR a la hora de evaluar los modelos, y limitarse a las diferentes variaciones sobre AP presentadas por COCO.

5.2.2. Métricas de seguimiento

El seguimiento de múltiples objetos, en inglés, *Multi-Object Tracking* (MOT) es la tarea de detectar la presencia de múltiples objetos en video, asociando entre sí las detecciones en cada cuadro del video. En definitiva, consiste en buscar asignar un identificador único a las detecciones que correspondan al mismo objeto, y, por lo tanto, la cantidad de identificadores asignados corresponde a la cantidad de objetos rastreados. Dicha tarea es compleja, ya que requiere de una correcta detección, localización y asociación de identificadores.

Para la selección de métricas a utilizar en esta etapa se tomaron algunas de las utilizadas en MOTChallenge (Dendorfer y cols., 2021), que se presenta a sí mismo como una referencia para MOT de una sola cámara. Dentro de lo que proponen se encuentra un conjunto de métricas para video y desafíos de videos etiquetados para la evaluación de algoritmos de seguimiento, para tener un punto de evaluación común que facilite las comparaciones entre los algoritmos. De las métricas que proponen, se seleccionaron $MOTA$, $IDF1$ y $HOTA$ para este proyecto, por razones que se explicarán a continuación.

Antes de plantear las métricas de seguimiento, es necesario comprender qué

tipo de errores pueden ocurrir (Bernardin y Stiefelhagen, 2008) y cómo estos influyen en el resultado final. Dichos errores son:

- Fallos (*misses*): objetos que no son detectados, equivalentes a los falsos negativos (FN).
- Falsos positivos (FP): seguimientos que no están asociados a un objeto real.
- Errores de desajuste (*mismatch errors*): la identificación de seguimiento de un mismo objeto cambia entre un cuadro y el siguiente. Por ejemplo, en el caso de que se intercambie el identificador de dos objetos cuando se encuentran cerca (como se representa en la figura 5.6), o cuando se reinicializa un identificador de seguimiento porque el objeto quedó cubierto.

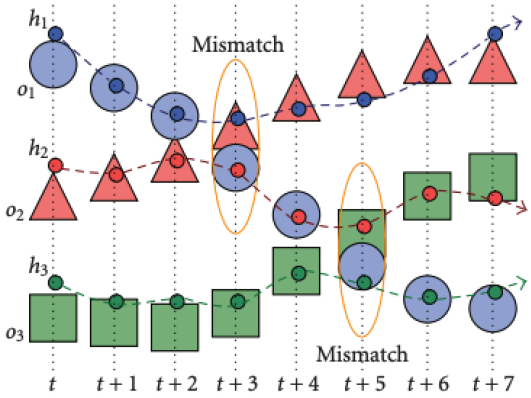


Figura 5.6: Error de desajuste: el seguimiento h_2 se asocia primero al objeto o_2, pero luego los objetos o_1 y o_2 se cruzan, lo que hace que h_1 y h_2 sigan a los objetos equivocados. Luego se repite el error con el objeto o_3. Imagen extraída de (Bernardin y Stiefelhagen, 2008).

MOTA

Según los tipos de errores posibles, el desempeño del seguimiento puede verse intuitivamente como la combinación de la precisión del seguimiento, que expresa qué tan bien se estiman las posiciones de los objetos, y la exactitud del seguimiento, que muestra los errores del *tracker*, ya sean falsos negativos, falsos positivos, errores de desajuste, entre otros (Bernardin y Stiefelhagen, 2008). Para esto se proponen dos métricas de desempeño complementarias: $MOTP$ y $MOTA$.

1. MOTP (*multiple object tracking precision*): representa el error en la posición estimada para los pares coincidentes de objetos y predicciones, promediado por el número total de coincidencias realizadas. Esto muestra

la capacidad del rastreador para estimar posiciones precisas de objetos, independientemente de su habilidad para reconocer características de los objetos y mantener trayectorias consistentes.

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}$$

Siendo d_t^i la distancia entre el objeto o_i y su predicción h_i en cada cuadro t, y c_t la suma de asociaciones correctas en cada cuadro t.

2. MOTA (*multiple object tracking accuracy*): representa el inverso a la cantidad de errores sobre el *ground truth*.

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + MM_t)}{\sum_t G_t}$$

Siendo FN_t los fallos (*misses*), FP_t los falsos positivos, MM_t los errores de desajuste y G_t el *ground truth* equivalente a la cantidad total de objetos etiquetados, para cada cuadro t del video.

Se explica el cálculo de ambas métricas porque normalmente se utilizan conjuntamente y fueron concebidas de esa manera. Sin embargo, $MOTA$ resulta mucho más relevante para este proyecto, ya que la dificultad de seguimiento de manzanas en los videos de campos de manzanas se encuentra más en la detección y en el evitar realizar errores de seguimiento, que en la predicción de la próxima posición de un objeto.

IDF1

Otra métrica que se consideró relevante para el caso es IDF1 (Ristani, Solera, Zou, Cucchiara, y Tomasi, 2016). Esta métrica se focaliza en evaluar la conservación de las identidades asignadas a los objetos seguidos a lo largo de todo el video, por lo que en este caso, el *ground truth* para la comparación es global, no por cuadro como en $MOTA$.

Luego de que se establecen las asociaciones, se computan los verdaderos positivos de las identidades ($IDTP$), falsos negativos de las identidades ($IDFN$) y falsos positivos de las identidades ($IDFP$), generalizando el concepto de TP, FN y FP por cuadro a las identidades asignadas en el video completo. Con esto se calculan la precisión en identificación IDP (ecuación 5.6), la recuperación en identificación IDR (ecuación 5.7) y el IDF_1 (ecuación 5.8), que es el equivalente al $F1$ en identificación, o sea un ratio entre las identificaciones correctas sobre el promedio del *ground truth* y las detecciones, balanceando precisión y recuperación.

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{5.6}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{5.7}$$

$$IDF_1 = \frac{2IDTP}{2IDTP + IDFP + IDFN} \tag{5.8}$$

HOTA

Finalmente, se seleccionó la métrica $HOTA$ (*Higher Order Tracking Accuracy*) (Luiten y cols., 2021), que combina las métricas anteriores con $Track - mAP$ (métrica que compara las predicciones de trayectorias con su *ground truth*), procurando combinar todos los aspectos del seguimiento equilibradamente. De esta manera, $HOTA$ mide qué tan bien se alinean las trayectorias de las detecciones coincidentes y promedia esto sobre todas las detecciones, a la vez que penaliza las detecciones que no coinciden.

Para evaluar la detección, $HOTA$ utiliza TP (verdadero positivo), FN (falso negativo) y FP (falso positivo) respecto a un umbral α. Esto es, se considera positivo si la superposición entre el área predicha y el área etiquetada superan dicho α, y falso en otro caso. También se definen medidas para evaluar la asociación, como se listan a continuación, considerando que cada detección ($gtDet$) tiene asignado un identificador que identifica al objeto a lo largo del video ($gtID$) y, de la misma forma, cada predicción ($prDet$) tiene asociado un identificador ($prID$).

- TPA (*True Positive Associations*): verdadero positivo en asociación, por lo que su predicción coincide con el GT. Es definido como:

$$\begin{aligned} &TPA(c) = \{k\}, \\ &\quad k \in \{TP | prID(k) = prID(c) \wedge gtID(k) = gtID(c)\}, c \in TP \end{aligned} \tag{5.9}$$

- FNA (*False Negative Associations*): falso negativo en asociación. Es el conjunto de $gtDets$ con el mismo $gtID$ que c, pero con diferente $prID$ o sin $prID$:

$$\begin{aligned} &FNA(c) = \{k\}, \\ &\quad k \in \{TP | prID(k) \neq prID(c) \& gtID(k) = gtID(c)\} \\ &\qquad \cup \{FN | gtID(k) = gtID(c)\}, c \in TP \end{aligned} \tag{5.10}$$

- FPA (*False Positive Associations*): falso positivo en asociación. Es el conjunto de $prDets$ con el mismo $prID$ que c, pero sin $gtID$ o con diferente $gtID$:

$$\begin{aligned} &FPA(c) = \{k\}, \\ &\quad k \in \{TP | prID(k) = prID(c) \wedge gtID(k) \neq gtID(c)\} \\ &\qquad \cup \{FP | prID(k) = prID(c)\}, c \in TP \end{aligned} \tag{5.11}$$

Luego, se define la siguiente función de puntuación:

$$HOTA_\alpha = \sqrt{\frac{\sum_{c\in\{TP\}} A(c)}{|TP| + |FN| + |FP|}} \tag{5.12}$$

$$A(c) = \frac{|TPA(c)|}{|TPA(c)| + |FNA(c)| + |FPA(c)|} \tag{5.13}$$

Finalmente, se calcula $HOTA$ como el área bajo la curva de los valores $HOTA_\alpha$ variando el valor de α, de manera de agregarle la localización. Esto se aproxima tomando α en el intervalo 0.05 a 0.95, con incrementos de 0.05, y promediando.

$$HOTA = \int_0^1 HOTA_\alpha d\alpha \cong \frac{1}{19} \sum_{\alpha\in\{0{,}05,0{,}1,..,0{,}95\}} HOTA_\alpha \tag{5.14}$$

5.3. Conjuntos de datos

En esta sección se describen los conjuntos de datos utilizados en las diferentes etapas de entrenamiento y evaluación de los modelos seleccionados. Se comienza detallando los conjuntos de datos utilizados en detección de manzanas, mencionando también las elecciones tomadas respecto a la aplicación de *data augmentation*. Luego, se mencionan los datos etiquetados utilizados para evaluar seguimiento y los videos disponibles para evaluar conteo. Finalmente, se incluyen algunos resultados de otros trabajos para utilizar a modo de lineamiento en la evaluación de los resultados experimentales.

5.3.1. Datos para detección

Para entrenar y evaluar los algoritmos de detección se utilizaron 5 conjuntos de datos, seleccionados por diferentes motivos.

- **Dataset 1 (DS1):** este conjunto de datos fue obtenido de la plataforma *Universe* de Roboflow. Consta de 232 imágenes de árboles de manzanas, en general tomadas de cerca. Este *dataset* presenta algunas diferencias en el etiquetado respecto a las recomendaciones para su uso en detección de objetos (Nelson, 2020), siendo que el área etiquetada es mayor que la que estrictamente ocupan las manzanas, y algunas frutas ocluidas no son etiquetadas. Esto coincide con la manera en la que se etiquetan los conjuntos de datos para la recolección de manzanas, ya que interesa que se detecten únicamente aquellas que se pueden cosechar. A pesar de esta diferencia, se consideró interesante evaluar los resultados a partir del entrenamiento con este tipo de etiquetas, siendo que existen múltiples *datasets* etiquetados de esta forma, y además acelera el proceso de etiquetado en nuevos *datasets*. Lamentablemente, la fuente original no se encuentra disponible actualmente.

- **Dataset 2 (DS2):** este *dataset* también fue obtenido de Roboflow (wangwang, 2022), y está compuesto de 1789 imágenes de árboles de manzanas. Se trata de una recopilación de *datasets*, incluyendo MinneApple (Häni y cols., 2020b), imágenes desde una vista aérea por encima de los árboles, y primeros planos de manzanas. Este *dataset* se seleccionó bajo la hipótesis de que una mayor variedad y cantidad de imágenes conducirían a buenos resultados.

- **Dataset 3 (DS3):** al igual que los anteriores, este conjunto de datos fue obtenido de Roboflow (Girling, 2023). Está compuesto de 331 imágenes de árboles de manzanas, donde las manzanas en el piso y las pertenecientes a otras filas de árboles no están etiquetadas. Se observa que, al haber sido tomadas de un *dataset* con *data augmentation*, algunas de estas imágenes no tienen las proporciones originales, lo que es frecuente en el entrenamiento de redes neuronales. Este *dataset* se eligió por tener un criterio de etiquetado que favorece al objetivo del proyecto (detectar manzanas en una sola fila de árboles y no detectar las manzanas caídas del árbol).

- **Dataset 4 (DS4):** este *dataset* consta de 80 imágenes, tomadas en una salida de campo a INIA Las Brujas, en una fila de árboles con red. Si bien no todas las manzanas están cubiertas por la red, sino aproximadamente la mitad superior del árbol, la existencia de la red dificulta la identificación de las manzanas, incluso para la vista humana. Sin embargo, se pensó que podía tener un resultado similar a un filtro de data augmentation, por lo cual se etiquetaron 80 de estas imágenes, capturadas con un celular Xiaomi Redmi Note 11 Pro 5G.

  El criterio de etiquetado fue análogo al del dataset 3: se etiquetaron las manzanas pertenecientes a los árboles de la fila fotografiada expresamente, dejando fuera las manzanas visibles de árboles en filas posteriores y manzanas en el suelo. Se etiqueta la superficie justa de la manzana, ya sea porque la manzana es visible o porque es evidente, a criterio del etiquetador, que la manzana se encuentra allí. Un ejemplo de este último caso, es una manzana parcialmente cubierta con hojas. En este caso, se etiqueta toda la superficie de la manzana, y no solamente la superficie visible.

- **Dataset 5 (DS5):** este dataset está compuesto por imágenes de árboles de manzanas tomadas en INIA Las Brujas, pero a diferencia del dataset 4, se trata de una fila de manzanas sin red. De estas imágenes, 44 son en realidad fotogramas aleatorios de un video de dicha fila, el cual se utiliza para la validación de seguimiento. A esto se sumaron cuatro imágenes adicionales de la misma fila, pero tomadas como fotos y no del video.

  Para la selección de fotogramas del video se utilizó una herramienta de Roboflow, buscando tener una cantidad de imágenes suficientemente representativa, pero posible de etiquetar en el tiempo que se disponía. El criterio de etiquetado fue el mismo que para el dataset 4.

Cabe notar que, si bien las imágenes del dataset 4 y 5 fueron tomadas en el mismo lugar y el mismo día, las primeras fueron en la mañana, con un clima soleado y cielo prácticamente despejado, y las de la fila sin red fueron tomadas en la tarde, con un cielo nublado y luego de una tormenta, por lo que la iluminación difiere considerablemente. Se observa a su vez que el propósito de este dataset es para probar la detección en los algoritmos, no para entrenarlos, por lo que la cantidad de imágenes se considera suficiente.

- **Dataset 7 (DS7):** este dataset corresponde al dataset MinneApple (Häni y cols., 2020b), que contiene 1000 imágenes con 41.000 manzanas etiquetadas, con imágenes tomadas a lo largo de dos años y de diferentes variedades de manzanas. Este dataset se incluyó con el propósito de comparar los resultados obtenidos en este proyecto con otros proyectos que lo utilizan.

A modo de resumen, los primeros tres datasets fueron obtenidos de Roboflow *Universe*, ya etiquetados. Los dos siguientes datasets fueron etiquetados como parte del proyecto, y se componen de imágenes tomadas en una visita a INIA Las Brujas. El último dataset fue obtenido de una referencia bibliográfica (Häni y cols., 2020b), siendo utilizado en más de una proyecto con resultados publicados. En la figura 5.7 se incluye una imagen de cada dataset con sus etiquetas.

Se considera oportuno aclarar que la razón por la que los conjuntos de datos no tienen una numeración secuencial es que el "Dataset 6" fue combinado con el "Dataset 5" previo a la experimentación.

Data augmentation

Respecto a los parámetros de *data augmentation*, se optó por utilizar los mismos parámetros para todos los conjuntos de datos, y realizar un único entrenamiento con cada par de red neuronal y conjuntos de datos. Los tipos elegidos fueron los más similares posibles a los indicados por los tutores, que a su vez fueron usados en otro proyecto similar. A continuación se incluye una descripción de cada uno, con su respectivo nombre en la plataforma (Roboflow, 2023).

- Corte (*crop*): se selecciona una parte de la imagen aleatoriamente, resultando un equivalente a entre 0 % y 20 % de zoom.
- Rotación (*rotation*): rotación aleatoria de las imágenes entre -15° y +15°.
- Saturación (*saturation*): ajuste aleatorio de la saturación de una imagen entre -40 % y +40 %, para hacerla más brillante o más oscura.
- Brillo (*brightness*): ajuste del brillo de una imagen de entre -50 % y +50 %.
- Exposición: (*exposure*): ajuste aleatorio de la exposición gama de una imagen entre -15 % y +15 %, para hacerla más brillante o más oscura.
- Desenfoque (*blur*): agregado de *gaussian blur* de hasta 1px.

(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

(d) Dataset 4 (e) Dataset 5 (f) Dataset 7

Figura 5.7: Imágenes representativas de cada *dataset*.

De acuerdo a la bibliografía (L. Wu y cols., 2021), diferentes tipos de *data augmentation* agregan robustez a la detección de diversas maneras. Los cambios como el corte o rotación ayudan dado que las manzanas en plantaciones se encuentran en diferentes posiciones y son vistas desde diferentes ángulos. Los cambios en brillo, saturación y exposición facilitan la detección, ya que la iluminación natural varía con el clima, hora del día, momento del año y ubicación geográfica. Los cambios de tipo desenfoque ayudan en los casos en los que el video sea borroso o poco nítido.

Por lo tanto, si bien se podrían haber incluido más parámetros de *data augmentation*, se considera que los seleccionados son suficientes y con un aporte a los resultados finales ya validado en otros proyectos.

5.3.2. Datos para seguimiento

Para evaluar los algoritmos de seguimiento, se disponía de dos videos: uno generado con el simulador y otro filmado con un celular *Samsung Galaxy S21 FE* sujeto a un robot Jackal, de la fila 22 de manzanos, en INIA Las Brujas, recorriendo la fila en un sentido de un lado de la fila y en el sentido opuesto del otro lado. De acuerdo a los datos proporcionados por lo expertos del INIA, la fila correspondía a manzanas Red Chief, con un sistema de muro bajo y sistema de doble eje; contaba con 47 plantas, de 2.2 metros en promedio, 1.2 metros entre cada planta y 2.5 metros entre cada fila, de la cual se extrajeron 2133 frutos. La fila de manzanas era además una de las utilizadas en el DS4 anteriormente descrito, pero sin la red. Cabe mencionar que, si bien se contaba con videos grabados con una cámara ZED 2 sujeta a un robot Jackal, la calidad de la imagen y la iluminación no eran suficientemente buenas para realizar pruebas.

El video filmado en INIA Las Brujas se dividió en dos partes, una por cada lado de la fila de manzanas, denominadas "ida" y "vuelta". Un fragmento de cada recorrido (que se procuró que fuera a la misma altura de la fila, pero en cada uno de los lados) se etiquetó utilizando la plataforma *Supervisely* (Supervisely, 2017), llegando a etiquetar más de 200 cuadros en cada uno. También se etiquetó un fragmento del video generado con el simulador, alcanzando 65 cuadros etiquetados. En cada caso, la decisión de no etiquetar más cuadros de los videos se debió al tiempo y dedicación que requiere el proceso, siendo que cada manzana debe estar etiquetada en todos los cuadros que aparezca con el mismo identificador, requiriendo de mucho tiempo y esfuerzo para ser correctamente etiquetadas.

Si bien la plataforma provee una herramienta que anticipa la ubicación y tamaño de un objeto dada una etiqueta, por cierta cantidad de cuadros, esta no cuenta con gran precisión, por lo que se requería de correcciones en prácticamente todos los cuadros para que los *bounding boxes* fueran realmente del tamaño correcto. Como no se disponía del tiempo suficiente para realizar todas las correcciones (que hubiera tardado varias semanas), se optó por dejar las etiquetas cubriendo los objetos, pero en algunos casos superando el tamaño de estos.

En resumen, los conjuntos de datos de videos son:

- DS-SIM: correspondiente al video "Simulado_recorrida_1.mp4", generado con el simulador y etiquetado. Tiene una duración de 6 segundos y 61 manzanas.

- DS-INIA-1: correspondiente al video "Recorrido_jackal_f22.mp4", fragmento del video de "ida" de la filmación de la fila 22, etiquetado, con una duración de 20 segundos y 139 manzanas.

- DS-INIA-2: correspondiente al video "Recorrido_jackal_f22_back.mp4", fragmento del video de "vuelta" de la filmación de la fila 22, etiquetado, con una duración de 25 segundos y 200 manzanas.

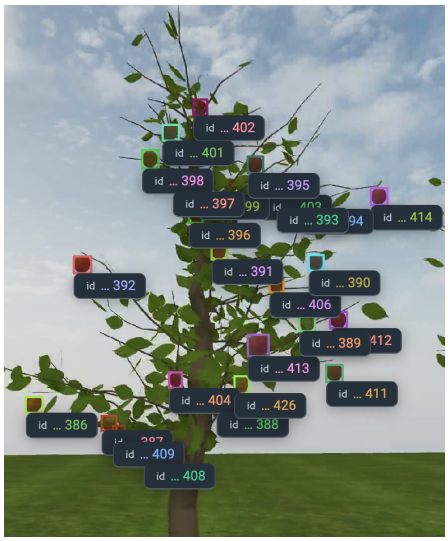


Figura 5.8: Cuadro etiquetado del video generado con el simulador

Figura 5.9: Cuadro etiquetado del video filmado en INIA Las Brujas

- DS-INIA-IDA: correspondiente a "rcec_jackal_cel_ida.mp4", primera mitad del recorrido de la fila 22 del INIA (de ida), sin etiquetar, de 2 minutos 38 segundos, donde se sabe que la fila completa tiene 2133 manzanas, aunque no se sabe exactamente cuántas aparecen en el video.

- DS-INIA-VUELTA: correspondiente a "rcec_jackal_cel_vuelta.mp4", segunda mitad del recorrido de la fila 22 del INIA (de vuelta), sin etiquetar, de 2 minutos 39 segundos, donde la fila completa tiene 2133 manzanas, pero no se sabe exactamente cuántas aparecen en el video.

- DS-INIA-COMPLETO: correspondiente a "rcec_jackal_cel_completo.mp4", recorrido completo de la fila 22 del INIA (ida y vuelta, con el giro intermedio), sin etiquetar, de 5 minutos 45 segundos, donde la fila completa tiene 2133 manzanas, pero no se sabe exactamente cuántas aparecen en el video.

5.3.3. Lineamientos

Antes de evaluar los resultados obtenidos, se consideró necesario establecer una línea base en términos de detección. Dado que el conjunto de datos de COCO (Lin y cols., 2015) es el estándar con el que se comparan las redes neuronales en sus publicaciones, resultó natural tomarlo como base. En la bibliografía (Terven y Cordova-Esparza, 2023; Y. Wu, 2019), se exhibe que se alcanzan los siguientes AP sobre COCO (evaluando todas las clases), como se detalla en la figura 5.3

Modelo	AP
YOLOv5x	50.7 %
YOLOv8x	53.9 %
faster_rcnn_X_101_32x8d_FPN_3x	43.0 %
faster_rcnn_R_50_FPN_3x	40.2 %

Tabla 5.3: Mejores resultados de AP por modelo de detección sobre COCO.

Otra posibilidad sería tomar los resultados de COCO limitándose a la clase manzanas (Consortium, 2015), del mejor modelo entrenado con el *dataset* COCO a la fecha (tabla 5.4).

AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
0.384	0.494	0.425	0.187	0.42	0.666

Tabla 5.4: Resultados de COCO para la clase “apple”

Se considera relevante mencionar que las imágenes de manzanas en COCO son muy variadas, incluyendo manzanas cortadas, cocidas, dibujos de manzanas o manzanas en primer plano, como las de la figura 5.10. Además, los grupos de manzanas pequeñas se etiquetan como grupo, no buscando etiquetar cada manzana por separado. Por lo tanto, la tarea de detección es diferente a la de este proyecto, y las métricas no pueden ser comparadas directamente, sino como una guía.

A su vez, se tiene que *Faster R-CNN* alcanza AP_{50} de 0.775 sobre el conjunto de datos de *MinneApple* (DS7) (Häni y cols., 2020b).

5.4. Experimentación de detección

Los cuatro modelos de detección seleccionados (YOLOv5, YOLOv8, Faster-101 y Faster-50) fueron entrenados con los conjuntos de datos descritos en la sección 5.3.1 con una distribución de 70 % para entrenamiento, 20 % para va-

Figura 5.10: Imágenes con manzanas de COCO

lidación y 10 % para pruebas. Para garantizar consistencia y disponibilidad de recursos, los entrenamientos se realizaron en la plataforma *Google Colab Pro* (*Google Colab*, 2017).

5.4.1. Entrenamientos

Inicialmente, se seleccionaron los hiperparámetros para la ejecución de los entrenamientos. En líneas generales, se optó por mantener los valores por defecto, considerando que su modificación no habría aportado un beneficio significativo o habría incrementado la complejidad del proyecto. No obstante, en ciertos casos, esta opción no fue factible.

Uno de los parámetros necesario a definir es la cantidad de épocas (ciclos de entrenamiento). Tomando como referencia un proyecto donde se utiliza YOLOv5 para la detección de cítricos verdes en árboles (Lyu y cols., 2022), para lo que se entrena el algoritmo con 50 y 100 épocas, se optó por entrenar con estas opciones.

Sin embargo, entrenar Faster-101 con 50 épocas supuso un gran uso de recursos que *Google Colab* no soportaba. Si bien se cambiaron los parámetros por defecto para permitir la ejecución, como el tamaño de *batch* y la cantidad máxima de iteraciones, aun así el tiempo que llevaba el entrenamiento no era sostenible, por lo que se agregó el caso de 20 épocas.

El modelo de Faster-50 es más liviano, por lo que lograba terminar las ejecuciones en menos tiempo y fue posible completar las de 20 y hasta 50 épocas. Más que esto no fue posible, ya que los entrenamientos fácilmente hubieran tomado más de un día.

Por el contrario, con YOLOv8 se ejecutaron los entrenamientos con 50 y 100 épocas en tiempos aceptables (en el entorno de horas en el peor caso), y los resultados en algunos casos mostraban mejoras a mayor cantidad de épocas. Por esto, se optó por correr entrenamientos con 200 épocas. En particular, las gráficas de precisión, recuperación y AP_{50} por ciclos mostraban crecimiento, mientras que las funciones de pérdida decrecían. En dos casos, con el DS1 y DS2, se llegó a probar con 500 épocas, pero los resultados no mejoraron desde las ejecuciones 49 y 151 respectivamente, por lo cual se descartó la idea y se conservó el límite de 200 épocas. A modo de referencia, se incluyen las gráficas resultantes del entrenamiento de YOLOv8 con el DS2 para 200 épocas en la figura 5.11. Cabe mencionar que estos resultados son similares a los mencionados en la sección 2.3.2, donde se encontró que los resultados mejoraban consistentemente en las primeras 100 épocas, pero se estabilizaban a partir de la 250 (Yan y cols., 2021).

5.4.2. Evaluación de los modelos entrenados

En primera instancia, para evaluar los modelos entrenados según la cantidad de épocas, se tomó como referencia la métrica AP_{50} sobre el conjunto de datos de validación, como se suele hacer en la optimización de hiperparámetros. Los mejores resultados por modelo de detección y por *dataset* se representan en

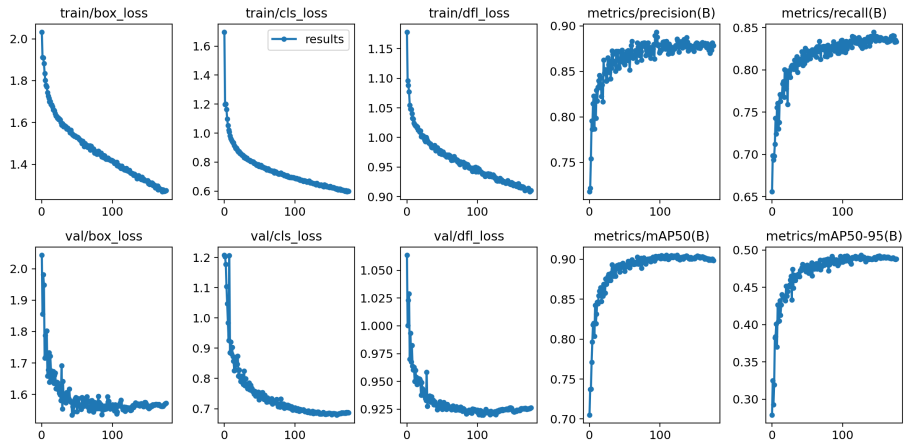


Figura 5.11: Resultados del entrenamiento de YOLOv8 con DS2 para 200 épocas.

la tabla 5.5, donde se puede apreciar que los conjuntos de datos DS1 y DS2 obtienen los mejores resultados respecto a su propio conjunto de validación.

Modelo	**Mejor resultado**			**Segundo mejor resultado**		
	Dataset	Épocas	AP_{50}	Dataset	Épocas	AP_{50}
YOLOv5	DS2	50	0.906	DS1	20	0.803
YOLOv8	DS2	200	0.904	DS1	100	0.814
Faster-101	DS2	20	0.869	DS1	20	0.801
Faster-50	DS2	50	0.878	DS1	20	0.799

Tabla 5.5: Mejores resultados de AP_{50} por modelo de detección y por conjunto de datos.

Sorprenden los resultados obtenidos con el DS1, en comparación con el DS3 que tiene más imágenes y mejor etiquetadas. Se considera posible que sea porque, al no tener etiquetadas muchas manzanas con oclusiones, la comparación con el conjunto de validación dé mejores resultados aunque haya manzanas sin etiquetar, disminuyendo la dificultad de la detección.

A su vez, se observa que Faster-101 logra resultados similares a los de YOLOv5 y YOLOv8 en 20 épocas, pero el tiempo requerido para entrenar el modelo es considerablemente mayor.

Por otro lado, analizando los resultados por conjunto de datos, como se representa en la tabla 5.6, puede apreciarse que para los conjuntos de datos grandes (a partir de 1000 imágenes), los entrenamientos con 50 épocas dan resultados ligeramente mejores que en 100 o 200 épocas, indicando que no existen mejoras a partir de las 50 épocas. En cambio, en los *datasets* más pequeños se obtienen mejores resultados con 100 y 200 épocas, al menos para YOLOv5 y YOLOv8, con los que se pudieron entrenar con esas ejecuciones.

Datasets		Mejor resultado			Segundo mejor resultado		
Nombre	Imágenes	Modelo	Épocas	AP_{50}	Dataset	Épocas	AP_{50}
DS1	232	YOLOv8	100	0.814	YOLOv5	20	0.803
DS2	1790	YOLOv5	50	0.906	YOLOv8	200	0.904
DS3	331	YOLOv5	200	0.799	YOLOv5	100	0.797
DS4	80	YOLOv5	100	0.738	YOLOv8	100	0.724
DS7	1000	YOLOv5	50	0.733	YOLOv8	50	0.717

Tabla 5.6: Mejores resultados por conjunto de datos, por modelo y *épocas*.

Continuando el análisis por conjunto de datos, se observa que DS4 obtiene resultados meramente aceptables, probablemente por contar con muy pocas imágenes. También se observa que DS7, correspondiente a *MinneApple*, no logra tan buenos resultados como los mostrados en su repositorio, y que, si bien el DS2 se compone principalmente del DS7, las imágenes adicionales hacen que se logren resultados considerablemente mejores.

A partir de este análisis se seleccionaron los mejores modelos de pesos para cada par de red neuronal y conjunto de datos de entrenamiento.

Evaluación sobre datos de prueba

Los modelos seleccionados en la etapa anterior se evaluaron sobre su propio conjunto de datos de prueba (normalmente denominado conjunto de *test*), correspondiente al 10 % del *dataset*. Los resultados se incluyen en la tabla 5.7.

Nuevamente, se observa que DS1 con YOLOv8 obtiene los mejores resultados sobre su propio conjunto de prueba en AP, y supera a los demás *datasets* en los demás modelos. Así como sobre su conjunto de validación, se considera que puede ser porque varias de las manzanas parcialmente ocluidas no se encuentran etiquetadas, haciendo más fácil la detección y obteniendo mejores resultados de AP que los demás conjuntos de datos. Aun teniendo esto en cuenta, es también el único modelo que logra un AP superior al 50 %, mientras que, con los otros *datasets*, no se alcanzan los resultados de los modelos más poderosos de los mismos algoritmos, sin entrenar, evaluados sobre COCO (aunque se recuerda que existen diferencias entre los *datasets*, y que para los resultados disponibles se evalúa en todas las clases).

En el caso de la métrica AP_{50}, los mejores resultados por modelo se dan siempre con el DS2. Además, se destaca que, para esta métrica, el mejor resultado se logra con DS2 en el modelo YOLOv5, logrando un valor de 0.911.

En comparación con el AP de la clase de manzanas en COCO, se supera su métrica con DS1, DS2 y DS3 con YOLOv5 y YOLOv8, DS1 en Faster-101 y DS1 y DS2 en Faster-50. En cambio, ni DS4 ni DS7 llegan al índice. Para AP_{50}, todos los pares de modelos y *datasets* superan el 0.494 de COCO, siendo que DS4 es el de peor desempeño, con el cual se logran valores entre 0.499 y 0.626 de AP_{50}, superando apenas el índice de COCO.

En tiempos de detección por cuadro, YOLOv8 es considerablemente más rápido que el resto, con un promedio de 11.10 ms, seguido por YOLOv5 con

18.85 ms. En comparación con YOLOv8, Faster R-CNN, tarda unas 16 veces más con el modelo Faster-101 y 7 veces más con el modelo Faster-50. Esto confirma que YOLOv8 y YOLOv5 pueden ser utilizados en tiempo real, siendo cualquiera de ellos sustancialmente más rápido que Faster R-CNN.

Modelo	Dataset	Imágenes	AP	AP_{50}	Tiempo (ms)
YOLOv5	DS1	232	**0.483**	0.902	22.7
	DS2	1790	0.484	**0.911**	13.6
	DS3	331	0.418	0.801	20.8
	DS4	80	0.251	0.626	33.9
	DS7	1000	0.338	0.692	18.3
YOLOv8	DS1	232	**0.551**	0.896	10.4
	DS2	1790	0.488	**0.904**	10.7
	DS3	331	0.428	0.791	13.1
	DS4	80	0.270	0.615	11.2
	DS7	1000	0.330	0.664	10.1
Faster-101	DS1	232	**0.419**	0.856	195.0
	DS2	1790	0.393	**0.878**	196.0
	DS3	331	0.360	0.725	197.0
	DS4	80	0.168	0.501	172.0
	DS7	1000	0.248	0.593	167.0
Faster-50	DS1	232	**0.424**	0.879	82.0
	DS2	1790	0.421	**0.882**	75.5
	DS3	331	0.373	0.734	88.0
	DS4	80	0.168	0.499	72.0
	DS7	1000	0.261	0.606	76.0

Tabla 5.7: Evaluación de modelos entrenados sobre su conjunto de prueba. En los *datasets* se incluye la cantidad de imágenes y se resaltan los mejores resultados por red neuronal y conjunto de datos para facilitar el análisis.

Evaluación sobre DS5

Como se comentó en la sección 5.3.1, DS5 es un conjunto de datos etiquetados a partir de imágenes de manzanos tomadas en INIA Las Brujas, muchas de las cuales son cuadros de los videos utilizados para probar en la sección de seguimiento y conteo. Por este motivo, la validación más relevante de los modelos entrenados es sobre este conjunto de datos, ya que cuenta con características ambientales y espaciales más similares a las que van a ser utilizadas en las pruebas de video.

Los resultados se incluyen en la tabla 5.8, y de ellos se extraen las siguientes observaciones:

- El mejor AP_{50} se da en el DS2 con YOLOv5, mientras que el más bajo se da en MinneApple con Faster-50.

Red neuronal	**Dataset**	AP	AP_{50}	**Tiempo** (ms)
YOLOv5	DS1	0.164	0.599	19.3
	DS2	**0.443**	**0.865**	**14.3**
	DS3	0.320	0.707	19.7
	DS4	0.438	**0.854**	21.7
	DS7	0.437	0.836	15.9
YOLOv8	DS1	0.285	0.720	13.8
	DS2	**0.503**	**0.853**	**4.9**
	DS3	0.274	0.653	11.2
	DS4	0.474	**0.850**	10.9
	DS7	0.465	0.823	12.6
Faster-101	DS1	0.227	0.715	275.0
	DS2	0.352	0.783	282.0
	DS3	0.350	0.650	283.0
	DS4	0.279	0.622	295.0
	DS7	0.255	0.558	278.0
Faster-50	DS1	0.192	0.699	135.0
	DS2	0.356	0.773	139.0
	DS3	0.265	0.544	137.0
	DS4	0.323	0.708	145.0
	DS7	0.248	0.524	139.0

Tabla 5.8: Resultados exportados por cada modelo, evaluando sobre DS5. Se destacan los resultados del DS2 en las mejores redes neuronales y el AP_{50} de DS4.

- El mejor AP se da con el DS2 en YOLOv8, mientras que el peor se da por lejos en el DS1 con YOLOv5 (por el tamaño de las *bounding boxes* y que hay muchas manzanas sin etiquetar, como se mencionó anteriormente).
- El DS4 da mejores resultados sobre el DS5 que sobre su propio conjunto de *test* en todos los casos. Esto se explica con que el DS4 y DS5 son ambos tomados en el mismo lugar, pero el DS4 tiene imágenes con redes, mientras que el DS5 no.
- Sorprendentemente, MinneApple se desempeña muy por debajo del DS2 a pesar de ser un subconjunto de este.
- Una de las razones por las que el AP no es muy bueno, es porque se detectan manzanas en árboles de la fila siguiente, que no están etiquetadas, aun con el DS4, que está entrenado con manzanas sin etiquetar en segunda fila. Esto baja la recuperación (*recall*), ya que agrega falsos positivos, disminuyendo el AP.

Respecto a tiempos de inferencia, YOLOv8 vuelve a ser más veloz que los demás modelos, seguido por YOLOv5, al que supera por menos de 10 ms en promedio. En el caso de Faster R-CNN, hay una diferencia sustancial respecto

ambas versiones de YOLO, dado que Faster-50 demora al menos 10 veces más y Faster-101 20 veces más.

Evaluación con Pycocotools

Pycocotools es una librería de Python utilizada por COCO para cargar, interpretar y visualizar las anotaciones (Cocodataset, 2020). En este proyecto se utilizó como herramienta para comparar los resultados de los algoritmos, garantizando uniformidad de criterios en la evaluación y unificando las métricas que cada algoritmo utiliza.

Si bien se esperaba que las métricas que YOLOv5 exporta difieran de las de Pycocotools entre un 2 % y 3 % (Jocher, 2021), los resultados que se obtuvieron reflejan que las diferencias pueden llegar al 6 %. Sin embargo, esta diferencia no es suficientemente significativa y los análisis anteriores continúan siendo válidos.

En la tabla 5.9 se incluyen los resultados relativos al AP de la evaluación sobre el DS5. Cabe mencionar que, aunque la métrica AR pertenece al conjunto de métricas que exporta Pycocotools, se opta por no incluirla, dado que no es relevante para los objetivos del proyecto.

Modelo	**Dataset**	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
YOLOv5	DS1	0.153	0.571	0.009	0.016	0.091	0.177
	DS2	0.424	**0.838**	0.370	0.130	0.310	0.469
	DS3	0.303	0.675	0.233	0.107	0.278	0.317
	DS4	0.420	0.828	0.376	0.197	0.339	0.452
	DS7	0.418	0.812	0.393	0.224	0.352	0.445
YOLOv8	DS1	0.271	0.693	0.118	0.066	0.174	0.309
	DS2	**0.480**	0.825	**0.502**	0.196	**0.367**	**0.523**
	DS3	0.259	0.626	0.173	0.077	0.222	0.277
	DS4	0.456	0.823	0.483	**0.228**	0.361	0.493
	DS7	0.444	0.796	0.470	0.201	0.350	0.480
Faster-101	DS1	0.227	0.715	0.012	0.055	0.103	0.269
	DS2	0.352	0.783	0.217	0.092	0.229	0.398
	DS3	0.350	0.650	0.373	0.032	0.270	0.382
	DS4	0.279	0.622	0.193	0.066	0.132	0.365
	DS7	0.255	0.558	0.211	0.077	0.204	0.292
Faster-50	DS1	0.192	0.699	0.007	0.000	0.079	0.233
	DS2	0.356	0.773	0.239	0.110	0.283	0.387
	DS3	0.265	0.544	0.253	0.076	0.213	0.288
	DS4	0.323	0.708	0.241	0.086	0.174	0.379
	DS7	0.248	0.524	0.219	0.099	0.220	0.297

Tabla 5.9: Resultados evaluando sobre DS5 con Pycocotools, donde se resaltan los mejores resultados de cada métrica.

La mayoría de los *dataset* no devuelven valores para AP_{large} ni AR_{large}, por no tener objetos a detectar suficientemente grandes. Los resultados para AP_{medium} son superiores a AP_{small} en todos los casos, así como los de AR_{medium} son superiores a AR_{small}, por lo que se puede concluir que las detecciones medianas son mejores que las de objetos muy pequeños.

Para el AP_{75} se obtienen resultados muy inferiores al promedio de COCO. Los únicos casos en que se supera son con YOLOv8 entrenados con el DS2 y con

el DS4. En cualquier caso, no es una métrica muy relevante para el proyecto, ya que para contar los frutos no es realmente necesario detectar el área completa, para lo que basta con AP_{50}.

A modo de resumen, los mejores resultados fueron obtenidos con YOLOv5 y YOLOv8, ambos entrenados con el DS2, por lo que son los modelos que se utilizan para la experimentación con los algoritmos de seguimiento.

5.5. Experimentación de seguimiento

Con el propósito de evaluar los algoritmos de seguimiento, se presentan en esta sección los resultados principales de los experimentos realizados. Para esto, se utilizaron como algoritmos de detección los modelos YOLOv5 y YOLOv8, entrenados con el conjunto de datos DS2 debido a que obtuvo los mejores resultados sobre el conjunto DS5. Los algoritmos de seguimiento evaluados fueron OCSort, StrongSort y Bytetrack.

En esta etapa se realizaron tres tipos de pruebas. La primera fue con fragmentos de videos etiquetados, donde se analizan las métricas de seguimiento para cada combinación de algoritmos de detección y seguimiento. El segundo tipo de prueba fue utilizando los videos completos, sin etiquetar, por lo que solo se conoce la cantidad de manzanas con las que cuenta cada conjunto de árboles (no necesariamente las manzanas visibles). Aquí se analiza la cantidad de manzanas contadas y el tiempo de ejecución de los algoritmos. Finalmente, se realizan pruebas de regresión lineal con videos generados con el simulador, donde se evalúa la viabilidad del uso de un coeficiente de ajuste para los resultados obtenidos a partir del seguimiento.

5.5.1. Experimentación con videos etiquetados

Las métricas a utilizar son las explicadas en la sección 5.2.2 ($HOTA$, $MOTA$, $IDF1$) y la métrica IDs que indica la cantidad de identificadores. Los conjuntos de datos etiquetados, detallados en la sección 5.3.2, se reiteran a continuación:

- DS-SIM: generado con el simulador, contiene 61 manzanas.
- DS-INIA-1: filmado en el INIA, contiene 139 manzanas.
- DS-INIA-2: filmado en el INIA, contiene 200 manzanas.

En la tabla 5.10 se registran los resultados obtenidos al evaluar los conjuntos de datos con los algoritmos antes mencionados. Se recuerda que las métricas influidas por la superposición de *bounding boxes* ($HOTA$ y $MOTA$), pueden no ser tan precisas, ya que los bounding boxes de las etiquetas no siempre están del todo ajustados a los objetos.

Como se puede observar, los mejores resultados en $HOTA$ y $MOTA$ se dan con OCSort en todos los casos, mientras que los mejores resultados de EA_IDs (error absoluto en la cantidad de IDs) son siempre menores con StrongSort, y los de $IDF1$ varían entre StrongSort y OCSort. Esto induce a pensar que, si

RN	Dataset	Seguim.	HOTA	MOTA	IDF1	EA IDs	Tiempo (ms)
YOLOv5 (DS2)	DS-SIM	OCSort	50.406	58.372	76.864	11	47.9
		Bytetrack	43.729	43.627	65.062	56	34.1
		Strongsort	47.471	55.052	72.471	**1**	177.8
	DS-INIA-1	OCSort	55.310	43.424	71.141	205	40.9
		Bytetrack	45.986	30.007	61.713	211	39.2
		Strongsort	54.39	28.676	69.612	151	170.9
	DS-INIA-2	OCSort	52.981	51.114	70.341	172	39.0
		Bytetrack	46.451	39.353	61.836	183	29.1
		Strongsort	50.762	37.248	68.020	96	154.6
YOLOv8 (DS2)	DS-SIM	OCSort	55.418	**64.441**	**79.196**	26	43.3
		Bytetrack	45.923	57.194	68.166	67	30.0
		Strongsort	52.069	61.728	74.927	**1**	175.0
	DS-INIA-1	OCSort	**60.240**	54.424	73.473	184	36.2
		Bytetrack	50.081	49.257	67.453	95	**26.7**
		Strongsort	59.301	53.316	73.832	69	134.2
	DS-INIA-2	OCSort	57.109	58.684	70.326	164	32.0
		Bytetrack	47.072	47.771	63.797	83	29.7
		Strongsort	56.086	55.377	71.031	26	118.9

Tabla 5.10: Resultados de algoritmos de seguimiento con métricas, donde RN son las redes neuronales entrenadas, $EA\ IDs$ corresponde al error absoluto de cantidad de identificadores asignados.

bien el seguimiento es mejor con OCSort, la asociación de los identificadores con los objetos rastreados es más exacta con StrongSort.

En los casos donde el mejor $IDF1$ no coincide con el mejor EA_IDs, se deduce que los errores son mayoritariamente causados por errores de desajuste. La ocurrencia de estos casos se da con OCSort con mejor $IDF1$ y StrongSort con menor EA_IDs, aunque la diferencia entre los $IDF1$ del OCSort y de StrongSort es escasa.

Respecto al tiempo de ejecución, los resultados parecen indicar que, aun en los peores casos, los tiempos son muy rápidos, pudiendo ser utilizados en tiempo real. Este análisis se profundiza en la siguiente etapa, donde los videos tienen mayor duración.

Analizando según los conjuntos de datos, se encuentra que la cantidad de objetos detectados y contados varía considerablemente entre estos. El *dataset* generado con el simulador es el que logra mejores resultados, obteniendo los

menores errores absolutos. Se considera que esto puede deberse a que el entorno del simulador es más simple que el entorno real, siendo que no hay otros objetos en el entorno ni filas de manzanos posteriores. Además, se producen menos oclusiones entre las manzanas y entre hojas y manzanas.

5.5.2. Experimentación con videos completos

En segundo lugar, se realizaron pruebas buscando validar únicamente la cantidad de manzanas contadas en los videos filmados en INIA Las Brujas. Dentro de este objetivo se encuentra analizar la relación entre las frutas visibles desde cada lado de la fila de manzanas y las contadas en el recorrido completo, esto es, de ida y vuelta a lo largo de la fila.

Los resultados se incluyen en la tabla 5.11, donde se registran los modelos de detección utilizados (nuevamente YOLOv5 y YOLOv8 entrenados con el DS2), los algoritmos de seguimiento con los que se combinan y el *ground truth* (GT) correspondiente a la cantidad de manzanas recolectadas en la fila de manzanas, el mismo día que se filmó el video. Los resultados obtenidos con los videos DS-INIA-IDA, DS-INIA-VUELTA y DS-INIA-COMPLETO se registran en las columnas "Ida", "Vuelta" y "Recorrido completo" respectivamente, mientras que la columna "Suma ida y vuelta" es la suma de los resultados de las columnas "Ida" y "Vuelta" en cada fila. Además, se dejan disponibles públicamente algunos videos correspondientes a resultados de seguimiento (Garderes y Gutiérrez, 2023a).

Analizando los resultados, se observa que, en todos los casos, la suma de la cantidad de manzanas identificadas en la ida y en la vuelta es inferior a la del recorrido completo. Observando los videos generados por los rastreadores, se constata que la diferencia se encuentra en el giro que realiza el robot para cambiar de un lado de la fila al otro, fragmento del video que no se incluye en los videos de "ida" y "vuelta", pero donde se registran manzanas adicionales de otras filas.

En general, los resultados de cantidad total de manzanas son considerablemente superiores a la cantidad real de manzanas. Si bien parte de estos errores pueden darse por errores en la identificación a nivel de seguimiento, a simple vista los identificadores asignados a las manzanas no parecen tener gran variación. En cambio, se observa que se detectan muchas manzanas de la fila de árboles posterior a la que está siendo estudiada.

La detección en segunda fila es problemática, ya que las manzanas detectadas tienen *confidence* muy alta, por lo que aumentar el umbral de confianza no mejoraría los resultados. Incluso podría ser problemático, ya que no consideraría aquellas manzanas de la fila con baja *confidence*, como las manzanas parcialmente ocluidas. Tampoco sería una solución filtrar *bounding boxes* por tamaño, dado que muchas de las manzanas de la segunda fila son etiquetadas con *bounding boxes* de tamaños similares a la de la fila estudiada, además de que algunas manzanas etiquetadas con *bounding boxes* pequeñas en la fila estudiada serían descartadas.

La solución a esto es eliminar el fondo del video, quedando únicamente la fila

que se está recorriendo. Se evaluaron distintas alternativas para lograrlo, siendo la más prometedora la utilización de la cámara estéreo ZED 2. Con la imagen estéreo, se puede generar mapas de disparidad, con los cuales filtrar el fondo. Sin embargo, la aplicación de esta alternativa no fue útil dada la calidad de la grabación, siendo que las manzanas apenas eran distinguibles.

Otra alternativa que se manejó fue la utilización de herramientas específicas para la eliminación de fondo, tanto en video como en imagen. No obstante, no se encontraron herramientas que logren eliminar de forma precisa el fondo, sin eliminar parcial o totalmente manzanas de la fila recorrida. La última alternativa manejada fue la de generar una imagen panorámica a partir del recorrido, aunque no se lograron resultados de calidad por los movimientos inestables de la cámara.

Por otro lado, resulta interesante comparar los resultados de la suma de ambos lados de la fila de manzanos con los obtenidos por un proyecto similar (Häni, Roy, y Isler, 2020a). Si bien los conjuntos de datos y las técnicas utilizados en dicho proyecto son diferentes, en la suma de ambos lados de las filas obtienen entre 102 % y 150 % de la cantidad real, dependiendo del conjunto de datos. Los detalles y resultados del proyecto, así como la comparación los resultados obtenidos en este, se detallan en el anexo B.1.

Detección	Seguimiento	GT	Ida	Vuelta	Suma ida y vuelta	Recorrido completo
YOLOv5 (DS2)	OCSort	2133	2932	2789	5721	5782
	Bytetrack	2133	2587	2503	5090	5121
	StrongSort	2133	2659	2385	5044	5092
YOLOv8 (DS2)	OCSort	2133	2621	2569	5190	5259
	Bytetrack	2133	1877	1818	3695	3730
	StrongSort	2133	1959	1808	3767	3810

Tabla 5.11: Resultados de algoritmos de seguimiento sin métricas, donde “GT” es la cantidad real de manzanas en la fila, “Ida”, “Vuelta” y “Recorrido completo” con resultados de cada video y “Suma ida y vuelta” la suma de “Ida” y “Vuelta”.

En cuanto a tiempos de ejecución, en la tabla 5.12 se muestran los tiempos de ejecución promedio por cuadro para los distintos algoritmos de seguimiento y detección. Se puede apreciar que el método más lento es StrongSort. Esto no era tan evidente empíricamente en los primeros experimentos con seguimiento, pero al utilizar videos más largos la diferencia fue notoria. Como puede apreciarse, Bytetrack es el algoritmo más veloz, con 22.0 ms en promedio, seguido por OCSort con 27.8 ms en promedio. StrongSort, en cambio, tarda al menos 5 veces más.

En el caso de Bytetrack y OCSort, con los tiempos obtenidos se podrían procesar hasta 45 y 35 cuadros por segundo respectivamente, por lo que pueden ser utilizados en tiempo real (siempre y cuando se cuente con el hardware adecuado, ya que de no utilizar GPU los tiempos aumentan). En el caso de StrongSort, con las mismas condiciones se podrían procesar hasta 8 cuadros por segundo.

Para esto, sería recomendable que el desplazamiento sea más lento, minimizando el movimiento de las manzanas entre cuadro y cuadro y facilitando así la predicción del movimiento del algoritmo de seguimiento.

Detección	Seguimiento	Ida	Vuelta	Completo	Promedio
YOLOv5 (DS2)	OCSort	22.3	32.2	21.6	25.3
	Bytetrack	20.0	19.7	20.6	20.1
	StrongSort	104.7	95.9	90.5	97.0
YOLOv8 (DS2)	OCSort	31.3	30.0	30.6	30.7
	Bytetrack	24.5	23.7	23.5	23.9
	StrongSort	169.1	156.0	134.1	153.1

Tabla 5.12: Tiempos en milisegundos por cuadro de las ejecuciones de algoritmos de seguimiento sobre los videos filmados en el INIA.

5.5.3. Ajuste mediante regresión lineal

A la hora de contar la cantidad de manzanas en un recorrido completo, se pueden clasificar las manzanas en tres categorías (Apolo-Apolo, Martínez-Guanter, Egea, Raja, y Pérez-Ruiz, 2020):

1. Manzanas que son visibles desde un único lado.
2. Manzanas que son visibles de ambos lados.
3. Manzanas que no son visibles desde afuera del árbol.

En lo que respecta a las categorías 2 y 3, en el contexto de conteo mediante mecanismos de seguimiento, resultan en errores en los valores finales devueltos por el algoritmo. Es por esto que se plantea el uso de un método simple de regresión lineal para calcular el error y predecir el valor real a partir del valor estimado.

Para construir métodos de ajuste, se generaron 20 instancias del simulador con parámetros constantes, que se listan a continuación:

- Cinco árboles de manzanas.
- Misma proporción de hojas y frutas.
- Recorrido a la misma velocidad (0.2 m/s) y con el mismo camino.
- Mismas condiciones ambientales.
- Misma posición e inclinación de la cámara.

Además, se generan cinco instancias de prueba, las primeras tres con exactamente los mismos parámetros de las instancias de entrenamiento. En la cuarta instancia, se cambia únicamente la cantidad de árboles a 10, mientras que en la última se utilizan 5 árboles, pero aumentando la cantidad de filas a 3, de forma

de que aparezcan manzanas de fondo. En adelante, las instancias con 1 fila y 5 árboles se referencian como V1_5, las de 1 fila y 10 árboles V1_10, y V3_5 la instancia de 3 filas y 5 árboles en cada una.

Para la realización de este experimento fue necesaria la grabación de los recorridos del robot en las instancias del simulador generadas con los parámetros definidos anteriormente. Sobre los videos, se ejecutaron los algoritmos de seguimiento y detección definidos previamente (YOLOv5 y YOLOv8 combinados con StrongSort, Bytetrack y OCSort). Algunas de las ejecuciones se encuentran disponibles en una lista de reproducción pública del proyecto (Garderes y Gutiérrez, 2023b).

A partir de estas ejecuciones, se obtiene una predicción de la cantidad de manzanas por instancia. Estos datos, combinados con la cantidad real de manzanas de cada video, son utilizados como entrada de un modelo de regresión, utilizando el modelo `LinearRegression` provisto por la librería *sklearn* (Pedregosa y cols., 2011), que incluye en el eje x el valor predicho, y en el eje y el valor real. La recta de regresión permite ajustar nuevos resultados de predicciones, minimizando la diferencia con el valor real de la instancia.

A modo ilustrativo, en la figura 5.12 se incluyen los gráficos generados para cada modelo de detección y seguimiento, donde los puntos azules corresponden al par formado por la cantidad de identificadores contados y *ground truth*, mientras que la línea naranja es la dada por el modelo de regresión.

Como método alternativo, se registraron los coeficientes de ajuste por cada combinación de modelos. La idea detrás de este método es encontrar un factor que permita ajustar las predicciones de manera más simple, aplicando únicamente una multiplicación.

El coeficiente de ajuste es definido como el promedio de los cocientes de cada resultado esperado (*ground truth*) sobre su predicción. Esto se representa en la ecuación 5.15, donde n es la cantidad total de entradas del modelo.

$$\text{coeficiente} = \frac{1}{n}\sum_{1}^{n}\frac{cantidad\_real}{cantidad\_predicha} \tag{5.15}$$

En la tabla 5.13 se muestran los coeficientes promedios y la desviación estándar para cada algoritmo de detección y método de seguimiento. Como puede verse, los coeficientes se mantienen bastante constantes entre los videos de entrenamiento y por algoritmo, presentando una desviación estándar menor a 0.1. La desviación estándar de los resultados de YOLOv8 es ligeramente menor que los de YOLOv5 en todos los casos.

Tanto para YOLOv5 como para YOLOv8, los coeficientes resultantes de las predicciones de StrongSort son mayores que los coeficientes de los otros algoritmos. Esto era esperable, ya que los resultados de StrongSort son los más cercanos al valor real de manzanas en todos los casos, por lo tanto, sus coeficientes están más cercanos a 1, indicando una menor necesidad de ajuste.

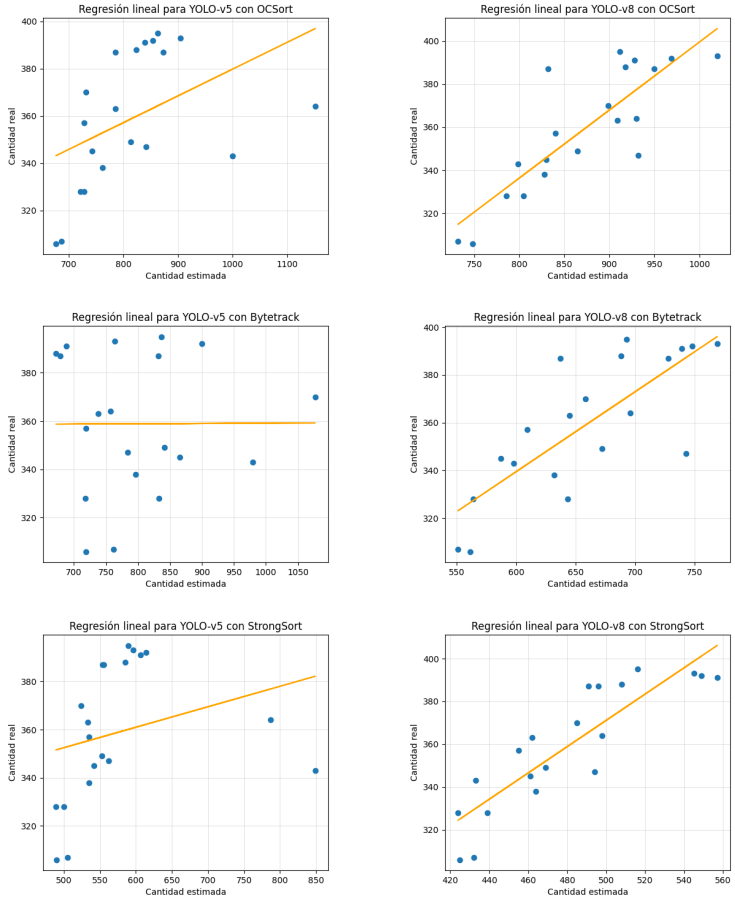


Figura 5.12: Gráficas de regresión lineal por modelo

RN Detección	Rastreo	Promedio de coeficientes	Desviación estándar
YOLOv5	OCSort	0.445	0.045
	Bytetrack	0.456	0.067
	StrongSort	0.633	0.074
YOLOv8	OCSort	0.412	0.019
	Bytetrack	0.547	0.034
	StrongSort	0.748	0.031

Tabla 5.13: Promedio de coeficientes y desviación estándar, por algoritmo de rastreo y detección, calculado a partir de los 20 videos de entrenamiento.

Evaluación de métodos de ajuste

Para evaluar los modelos de ajuste por regresión lineal y por coeficiente se utilizaron las cinco instancias de prueba mencionadas anteriormente. El conjunto de instancias V1_5 permite evaluar el comportamiento de los modelos en una instancia con las mismas condiciones de las de entrenamiento. Con la instancia V1_10 se evalúa el comportamiento al intentar predecir sobre una cantidad de árboles mayor a la que fueron entrenados. Finalmente, la instancia V3_5, permite evaluar cómo se comportan los modelos en un ambiente con condiciones más realistas, donde las manzanas de las filas posteriores, al ser visibles, inducen errores a la hora de realizar predicciones

Sobre dichas instancias, sumando el video del recorrido completo filmado en INIA Las Brujas (DS-INIA-COMPLETO), se ejecutaron los algoritmos mencionados en la sección anterior. Los resultados para cada tipo de video de evaluación son presentados en la tabla 5.14 (en el caso de V1_5 se presenta el promedio de los resultados de estas instancias). Para cada video y para cada combinación de modelos de detección y seguimiento, se incluye la cantidad real de manzanas, la predicción dada por los modelos, los resultados de aplicar ambos métodos de ajuste y sus respectivos errores relativos.

Para mayor claridad, en la tabla 5.15 se incluyen los mismos datos, pero agrupando los resultados de los modelos de seguimiento y detección.

Como se puede notar, los resultados difieren según el tipo de instancia evaluada. En el caso de V1_5 y V3_5, los errores relativos de ambos modelos se mantienen por debajo del 10 %, con una desviación estándar inferior al 5 %. En estos casos, si bien los errores relativos son muy similares, el error cometido por el modelo de regresión es menor que el modelo de coeficiente, indicando un mejor desempeño del primero.

Por otro lado, si observamos los resultados para la instancia V1_10, los resultados se invierten notablemente. Mientras el modelo de ajuste por coeficiente mantiene un error promedio cercano al 10 %, el modelo de regresión comete un error relativo promedio del 32 %. Esto se repite también en el caso de DS-INIA-COMPLETO, donde el error relativo del modelo de regresión asciende al 45 % y el del modelo de coeficiente asciende a 20 %.

Instancia	RN	GT	R	R*CA	ER_{RCA}	Reg.	ER_{Reg}
V1_5	Y5 OCSort	375	831	370	0.064	361	0.038
	Y5 Bytetrack	375	841	384	0.053	359	0.042
	Y5 StrongSort	375	544	345	0.081	356	0.050
	Y8 OCSort	375	962	397	0.089	387	0.058
	Y8 Bytetrack	375	697	381	0.070	372	0.027
	Y8 StrongSort	375	484	362	0.034	361	0.036
V1_10	Y5 OCSort	722	1562	695	0.037	444	0.385
	Y5 Bytetrack	722	1509	689	0.046	360	0.501
	Y5 StrongSort	722	1029	652	0.097	397	0.450
	Y8 OCSort	722	1678	692	0.041	613	0.151
	Y8 Bytetrack	722	1202	658	0.089	541	0.251
	Y8 StrongSort	722	860	644	0.109	592	0.180
V3_5	Y5 OCSort	362	766	341	0.058	353	0.024
	Y5 Bytetrack	362	747	341	0.058	359	0.009
	Y5 StrongSort	362	560	355	0.020	358	0.012
	Y8 OCSort	362	843	348	0.040	350	0.033
	Y8 Bytetrack	362	640	350	0.032	353	0.025
	Y8 StrongSort	362	472	353	0.024	354	0.022
DS_INIA_ COMPLETO	Y5 OCSort	2133	5782	2574	0.207	921	0.568
	Y5 Bytetrack	2133	5121	2337	0.095	364	0.829
	Y5 StrongSort	2133	5092	3226	0.512	743	0.652
	Y8 OCSort	2133	5259	2169	0.017	1743	0.183
	Y8 Bytetrack	2133	3730	2042	0.043	1387	0.350
	Y8 StrongSort	2133	3810	2851	0.337	2407	0.128

Tabla 5.14: Resultados de aplicar los modelos entrenados de regresión lineal a los resultados obtenidos por modelo de detección y seguimiento, donde "GT" es la cantidad de manzanas total del video, "R" es el resultado de los modelos, "Reg." la predicción con regresión lineal, "R*CA" los resultados de multiplicar el resultado por el coeficiente de ajuste y de aplicar regresión lineal, y "ER" los errores relativos respectivos. En la columna "RN", Y5 y Y8 corresponden a YOLOv5 y YOLOv8 respectivamente.

En el caso de DS-INIA-COMPLETO, se observan desviaciones estándar de 19 % y 28 % respecto a los errores relativos de los modelos de coeficiente y regresión respectivamente. Esta diferencia induce a pensar que los resultados no son tan homogéneos como en el caso de V1_5 y V3_5, probablemente por las diferencias entre las instancias de entrenamiento y el entorno real. Si bien ambos modelos tienen un error relativo mayor en esta instancia que en las instancias simuladas, el modelo de ajuste por coeficiente logra buenos resultados en el caso de YOLOv8 con OCSort y con Bytetrack, por lo que la idea de aplicar modelos de ajuste parece prometedora, pero con modelos entrenados con datos más similares a los reales.

Cabe observar que, si bien en el video V3_5 hay tres filas de árboles, las manzanas de las filas posteriores a la fila grabada se ven muy pequeñas en comparación con las manzanas de la fila principal. Por este motivo, prácticamente no se detectan manzanas de otras filas, a diferencia del ambiente real. Esto se da por varios factores, como la iluminación, la distancia entre las filas de árboles y la distancia entre el robot y la fila de árboles filmada, llevando a que

Instancia	GT	R*CA	ER_{RCA}	Reg.	ER_{Reg}
V1_5	375	373	$0{,}06 \pm 0{,}04$	366	$0{,}04 \pm 0{,}03$
V1_10	722	672	$0{,}07 \pm 0{,}03$	491	$0{,}32 \pm 0{,}15$
V3_5	362	348	$0{,}04 \pm 0{,}02$	354	$0{,}02 \pm 0{,}09$
DS_INIA_COMPLETO	2133	2533	$0{,}20 \pm 0{,}19$	1261	$0{,}45 \pm 0{,}28$

Tabla 5.15: Resultados promedio de aplicar los modelos entrenados de ajuste a los resultados obtenidos, donde "GT" es la cantidad de manzanas total del video, "R" es el resultado de los modelos, "Reg." la predicción con regresión lineal, "R*CA" los resultados de multiplicar el resultado por el coeficiente de ajuste y de aplicar regresión lineal, y "ER" los errores relativos promedio y sus desviaciones correspondientes.

las manzanas de diferentes filas del ambiente real se vean más similares. Dicha diferencia puede apreciarse en la figura 5.13, donde se incluyen dos recortes de detecciones de videos del simulador y de DS-INIA-COMPLETO.

A partir de estos resultados, se puede decir que, en los casos que los datos con los que se evalúa el modelo mantienen las mismas características que los datos de entrenamiento, ambos modelos logran aproximar el valor real de forma satisfactoria. Si las características del ambiente se mantienen, pero aumenta la cantidad de árboles, el desempeño del modelo de regresión disminuye considerablemente, dado que las rectas de regresión, como se muestra en la figura 5.12, son crecientes.

Es importante considerar que, al tratarse de instancias simuladas, donde las condiciones se mantienen relativamente constantes y las manzanas tienen mayor visibilidad que en el campo real, puede haber diferencias en los resultados obtenidos de un ambiente no simulado. Teniendo esto en cuenta, los resultados parecen indicar que la idea de aplicar regresión lineal es válida para ajustar los resultados de la detección y el rastreo como métodos de conteo, siempre y cuando los datos con los que se construya el modelo incluyan instancias con condiciones similares a las que se buscan estimar.

(a) Video simulado (b) Video de entorno real

Figura 5.13: Capturas de manzanas en filas posteriores de videos.

Capítulo 6

Conclusiones y Trabajo Futuro

A lo largo de este proyecto, se buscaron soluciones para la identificación y el conteo de manzanas en videos filmados por robots terrestres, que circulan entre hileras de árboles de manzanas en plantaciones. Con este objetivo, se propusieron combinaciones de redes neuronales de detección con algoritmos de seguimiento, entrenando las primeras con diferentes conjuntos de datos y evaluando los resultados de acuerdo a métricas de detección, seguimiento y tiempo de ejecución total.

Se entrenaron y evaluaron cuatro modelos de redes neuronales para la detección de manzanas en plantaciones. De estos, los mejores fueron YOLOv5 con AP_{50} 0.865 y YOLOv8 con AP_{50} 0.853, ambos entrenados con el DS2. Estos modelos fueron seleccionados para ser combinados con tres algoritmos de seguimiento.

Se destaca que uno de los conjuntos de datos de entrenamiento, así como otros tres conjuntos utilizados para la evaluación, fueron generados y etiquetados como parte del proyecto. También se destaca la implementación de un generador de simulaciones de campos de manzanas, parametrizable y con una interfaz sencilla. Este simulador posibilita la creación de conjuntos de datos bajo diversas condiciones de filmación, variaciones en las características de los manzanos y disposición de los árboles en los campos. Además, garantiza la independencia con respecto al grado de madurez de las manzanas y las condiciones climáticas del entorno real.

6.1. Conclusiones

Durante la etapa de entrenamiento, se encontró que la cantidad de imágenes de los conjuntos de datos impacta directamente en los ciclos de entrenamiento necesarios, ya que se requiere una menor cantidad de épocas en conjuntos de datos de mayor tamaño para llegar al resultado más óptimo del entrenamiento.

También se confirmó que la similitud de los datos de entrenamiento con los de evaluación influyen positivamente en los resultados. Por este motivo, el DS4, con apenas 80 imágenes, pero tomadas en INIA Las Brujas, obtiene resultados mejores que los de MinneApple (con 1000 imágenes) al evaluarlo sobre el DS5.

Otros factores determinantes son el tamaño y la variedad de los datos, lo que se visualiza en los mejores resultados de DS2 en comparación con DS7, siendo que el primero incluye al segundo y, por tanto, tiene mayor cantidad y variedad de imágenes.

A su vez, se concluyó que los conjuntos de datos etiquetados para recolección de manzanas son inadecuados para entrenar algoritmos de detección y conteo, como se puede ver en los resultados del DS1. Esto es relevante dada la mayor disponibilidad y facilidad de construcción de *datasets* para recolección.

Respecto a las redes neuronales para detección de objetos evaluadas, se concluye que YOLOv5 y YOLOv8 son ampliamente superiores a Faster-101 y Faster-50, tanto en los resultados obtenidos como en los tiempos de entrenamiento y de detección. Concretamente, los modelos de Faster R-CNN evaluados tardan al menos 10 veces más que los modelos de YOLO, llevando a tiempos de ejecución que no son suficientemente buenos para la ejecución en tiempo real.

Al comparar ambas versiones de YOLO, se obtienen resultados bastante similares, aunque YOLOv5 tiende a obtener mejores resultados de AP_{50}, mientras que YOLOv8 obtiene mejores resultados en AP y con menores tiempos de ejecución. Se observa que esto puede estar afectado por el modelo de YOLOv8 seleccionado, siendo la versión *nano*, mientras que la versión de YOLOv5 seleccionada es la *small*. Aun teniendo esto en cuenta, se considera que YOLOv8 es la red neuronal con mejores resultados y que convendría seguir estudiando.

Los modelos de seguimiento evaluados fueron StrongSort, Bytetrack y OCSort. De estos, OCSort es el que obtiene mejores resultados en general (logrando los mejores resultados de MOTA y HOTA, y destacando en IDF1), pero StrongSort obtiene la menor cantidad de errores absolutos en detección. En cuanto a tiempos, Bytetrack es el más veloz, seguido de cerca por OCSort, y superando ampliamente a StrongSort, que tarda al menos 3 veces más.

Por otro lado, a la hora de evaluar sobre el recorrido completo, el modelo que se acerca más al valor real de cantidad de manzanas es StrongSort. Si bien esto es consistente con sus valores de error absoluto de identificadores, los resultados de todos los algoritmos sobre este video se ven afectados por las manzanas detectadas en segunda fila y las condiciones de iluminación (problemas que no afectan los fragmentos de video etiquetados con los que se obtienen las métricas antes mencionadas).

Por los motivos anteriores, se concluye que el mejor algoritmo de seguimiento es OCSort, siendo el más equilibrado en resultados obtenidos y tiempo de ejecución.

En lo que respecta a los modelos de ajuste, se destacan los buenos resultados de los modelos elegidos, logrando ajustar la estimación para tomar en cuenta las frutas que son visibles de ambos lados de la fila y las que no son visibles.

Sin embargo, estos resultados no parecen ser consistentes cuando las características de los datos evaluados difieren de las de los datos de entrenamiento.

Particularmente, el modelo de regresión lineal se destaca cuando la cantidad de árboles se mantiene, con un error relativo menor al 10 %, pero aumenta al 30 % al tener el doble de árboles en el video evaluado. Esto no aplica para el modelo del coeficiente, que logra mantener el error relativo en el orden del 10 % incluso cuando se duplica la cantidad de árboles.

Por lo tanto, para evaluar predicciones sobre videos con una cantidad de árboles similar a la de los datos de entrenamiento, se recomienda el uso del modelo de regresión. Por el contrario, si se quiere variar la cantidad de árboles, se recomienda el uso del modelo del coeficiente.

6.2. Trabajo Futuro

En cuanto al trabajo futuro, se encuentran varios puntos que, por limitaciones de tiempo y alcance, quedaron fuera de este proyecto. Una de ellos es la combinación de conjuntos de datos para el entrenamiento de redes neuronales de detección de objetos. En particular, la combinación del DS2 (de disponibilidad pública y 1780 imágenes) con DS4 (de elaboración propia pero con apenas 80 imágenes). Si bien el primero obtuvo mejores resultados, DS4 obtuvo resultados bastante buenos para el tamaño reducido del conjunto de datos, por lo que se considera que el entrenamiento de los modelos de detección con una combinación de ambos conjuntos puede ser prometedora. Adicionalmente, dada la cantidad de imágenes totales y resultados obtenidos, se puede afirmar que bastaría con un entrenamiento de 100 épocas para obtener resultados óptimos.

Otro aspecto en el que expandir el proyecto es en las redes neuronales seleccionadas. Dado que el área está en constante desarrollo, continúan surgiendo modelos nuevos con variaciones en sus enfoques y arquitecturas, que presentan posibilidades de mejoras. Un ejemplo de esto es YOLO-NAS (Deci, 2023), que por su fecha de publicación no pudo ser incluido en este proyecto. También resultaría interesante experimentar con los modelos más pesados de YOLOv5 y YOLOv8, replicando los entrenamientos que tuvieron mejores resultados en cada uno.

En lo que respecta a conteo, el problema de la detección de manzanas en segunda fila afecta considerablemente los resultados finales. Por esto, se podría profundizar en el estudio de técnicas de eliminación del fondo y de iluminación de la fila.

Se podrían evaluar los modelos de ajuste utilizando datos reales para entrenamiento, y evaluar sobre datos reales (sin fondo) con datos de entrenamiento simulados.

También se podría investigar en profundidad el método de conteo a lo largo de la fila presentado en el artículo (Häni y cols., 2020a), que utilizando técnicas de reconstrucción 3D estima la posición de las manzanas visibles, lo que permite estimar con gran precisión la cantidad de manzanas.

Referencias

Anwar, A. (2022, Mayo). *What is average precision in object detection localization algorithms and how to calculate it?* Towards Data Science. Descargado de https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b

Apolo-Apolo, O., Martínez-Guanter, J., Egea, G., Raja, P., y Pérez-Ruiz, M. (2020). Deep learning techniques for estimation of the yield and size of citrus fruits using a uav. *European Journal of Agronomy*, *115*, 126030. Descargado de https://www.sciencedirect.com/science/article/pii/S1161030120300381

Bargoti, S., y Underwood, J. (2017a). Deep fruit detection in orchards. En *2017 ieee international conference on robotics and automation (icra)* (pp. 3626–3633).

Bargoti, S., y Underwood, J. P. (2017b). Image segmentation for fruit detection and yield estimation in apple orchards. *Journal of Field Robotics*, *34*(6), 1039–1060.

Bernardin, K., y Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, *2008*, 1–10.

Bewley, A., Ge, Z., Ott, L., Ramos, F., y Upcroft, B. (2016). Simple online and realtime tracking. En *2016 ieee international conference on image processing (icip)* (pp. 3464–3468).

Blender Foundation. (2014). *Api overview 2014; blender python api.* https://docs.blender.org/api/current/info_overview.html. (Accedido: 2023-06-11)

Blender Foundation. (2022). *Blender - a 3D modelling and rendering package.* Stichting Blender Foundation, Amsterdam. Descargado de http://www.blender.org

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

Broström, M. (2022). *BoxMOT: A collection of SOTA real-time, multi-object trackers for object detectors* . Descargado de https://github.com/mikel-brostrom/yolo_tracking doi: https://zenodo.org/record/7629840

Cabrera, D., Rodríguez, P., y Carra, B. (2021). Performance of different training systems in a pedestrian orchard for an efficient apple production. En

Xii international symposium on integrating canopy, rootstock and environmental physiology in orchard systems 1346 (pp. 319–326).

Cabrera, D., y Rodríguez, P. (2021). Muro bajo - cultivo peatonal. un sistema eficiente para trabajadores y productores. *Revista INIA*(64), 68–72.

Cao, J., Weng, X., Khirodkar, R., Pang, J., y Kitani, K. (2022). Observation-centric sort: Rethinking sort for robust multi-object tracking. *arXiv preprint arXiv:2203.14360*.

Clearpath Robotics. (2015). *Introduction to the robot operating system.* https://www.clearpathrobotics.com/assets/guides/melodic/ros/Intro%20to%20the%20Robot%20Operating%20System.html. (Accedido: 2023-05-04)

Cocodataset. (2020, Feb). *pycocotools/coco.py.* COCO Consortium. Descargado de https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/coco.py (Accedido: 2023-06-20)

Consortium, C. (2015). *Coco.* Descargado de https://cocodataset.org/#detection-leaderboard (Accedido: 2023-05-16)

Deci. (2023, Jun). *Yolo-nas by deci achieves state-of-the-art performance on object detection using neural architecture search.* Descargado de https://deci.ai/blog/yolo-nas-object-detection-foundation-model/ (Accedido: 2023-08-10)

Deeplizard. (2017). https://deeplizard.com/learn/video/YRhxdVk_sIs. Descargado de https://deeplizard.com/learn/video/YRhxdVk_sIs (Accedido: 2023-02-07)

Dendorfer, P., Osep, A., Milan, A., Schindler, K., Cremers, D., Reid, I., ... Leal-Taixé, L. (2021). Motchallenge: A benchmark for single-camera multiple target tracking. *International Journal of Computer Vision*, *129*, 845–881.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. En *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

D'Esnon, A. G., Rabatel, G., Pellenc, R., Journeau, A., y Aldon, M.-J. (1987). Magali: a self-propelled robot to pick apples..

Doerflinger, F. C., Miller, W. B., Nock, J. F., y Watkins, C. B. (2015). Relationships between starch pattern indices and starch concentrations in four apple cultivars. *Postharvest Biology and Technology*, *110*, 86–95.

Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T., y Meng, H. (2023). Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*.

Garderes, R., y Gutiérrez, F. (2022). *Estado del arte.* https://gitlab.fing.edu.uy/facundo.gutierrez/tesis-2022/-/blob/main/estado_del_arte/Estado_del_arte_V2.pdf.

Garderes, R., y Gutiérrez, F. (2023a). *Videos: Algoritmos de seguimiento sobre la fila 22 de inia las brujas.* https://www.youtube.com/watch?v=7Jf0U9UgeUw&list=PLUL58q01TzjuJRTfec98MCn75ZKruyiPO.

Garderes, R., y Gutiérrez, F. (2023b). *Videos: Construcción y evaluación de métodos de ajuste.* https://www.youtube.com/watch?v=DpCvVA3c_qU&list=PLUL58q01TzjuGC-vsQkdnunfs36MG9vCt.

Ge, Z., Liu, S., Wang, F., Li, Z., y Sun, J. (2021). Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*.

Girling, O. (2023, Enero). *Apple recognition dataset* [Open Source Dataset]. https://universe.roboflow.com/oliver-girling0-gmail-com/apple-recognition. Roboflow. Descargado de https://universe.roboflow.com/oliver-girling0-gmail-com/apple-recognition (Accedido: 2023-05-03)

Girshick, R. (2015). Fast r-cnn. En *Proceedings of the ieee international conference on computer vision* (pp. 1440–1448).

Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation.*

Google colab. (2017). Google. Descargado de https://colab.research.google.com/ (Accedido: 2023-05-31)

Gutta, S. (2021, Ago). *Object detection algorithm - yolo v5 architecture.* Analytics Vidhya. Descargado de https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef

Häni, N., Roy, P., y Isler, V. (2020a). A comparative study of fruit detection and counting methods for yield mapping in apple orchards. *Journal of Field Robotics*, *37*(2), 263–282.

Häni, N., Roy, P., y Isler, V. (2020b). Minneapple: a benchmark dataset for apple detection and segmentation. *IEEE Robotics and Automation Letters*, *5*(2), 852–858.

Jocher, G. (2020). *Precision-recall curve · issue #898 · ultralytics/yolov3. github.* Ultralytics. Descargado de https://github.com/ultralytics/yolov3/issues/898 (Accedido: 2023-07-26)

Jocher, G. (2021). *Competition: Pycocotools map alignment · issue #2258 · ultralytics/yolov5.* Descargado de https://github.com/ultralytics/yolov5/issues/2258

Jocher, G., Chaurasia, A., y Qiu, J. (2023a). *Yolo by ultralytics.* Descargado de https://github.com/ultralytics/ultralytics (Accedido: 2023-06-18)

Jocher, G., Chaurasia, A., y Qiu, J. (2023b, enero). *YOLO by Ultralytics.* Descargado de https://github.com/ultralytics/ultralytics

Jocher, G., y Waxmann, S. (2023). *Architecture summary.* Descargado de https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *ASME–Journal of Basic Engineering*.

Kamilaris, A., y Prenafeta-Boldú, F. X. (2018). A review of the use of convolutional neural networks in agriculture. *The Journal of Agricultural Science*, *156*(3), 312–322.

King, R. (2023, Enero). *Brief summary of yolov8 model structure · issue #189 · ultralytics/ultralytics.* Descargado de https://github.com/ultralytics/ultralytics/issues/189

Koenig, N., y Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. En *2004 ieee/rsj international conference*

on intelligent robots and systems (iros)(ieee cat. no. 04ch37566) (Vol. 3, pp. 2149–2154).

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, *2*(1-2), 83–97.

Kuznetsova, A., Maleva, T., y Soloviev, V. (2020). Using yolov3 algorithm with pre-and post-processing for apple detection in fruit-harvesting robot. *Agronomy*, *10*(7), 1016.

Leal-Taixé, L., Milan, A., Reid, I., Roth, S., y Schindler, K. (2015). Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942*.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., ... Dollár, P. (2015). *Microsoft coco: Common objects in context.*

Luiten, J., Osep, A., Dendorfer, P., Torr, P., Geiger, A., Leal-Taixé, L., y Leibe, B. (2021). Hota: A higher order metric for evaluating multi-object tracking. *International journal of computer vision*, *129*, 548–578.

Lyu, S., Li, R., Zhao, Y., Li, Z., Fan, R., y Liu, S. (2022). Green citrus detection and counting in orchards based on yolov5-cs and ai edge system. *Sensors*, *22*(2). Descargado de https://www.mdpi.com/1424-8220/22/2/576 doi: 10.3390/s22020576

Nelson, J. (2020, Nov). *How to label images for computer vision models.* Descargado de https://blog.roboflow.com/tips-for-how-to-label-images/

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). Determination press San Francisco, CA, USA.

OpenRobotics. (2021, 2 de October). *Costar husky sensor config 1.* Descargado de https://fuel.gazebosim.org/1.0/OpenRobotics/models/COSTAR_HUSKY_SENSOR_CONFIG_1

OSRF. (2012). *What is sdformat.* Descargado de http://sdformat.org/

Padilla, R., Passos, W. L., Dias, T. L., Netto, S. L., y Da Silva, E. A. (2021). A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, *10*(3), 279.

Parrish, E., y Goksel, A. (1977). Pictorial pattern recognition applied to fruit harvesting. *Transactions of the ASAE*, *20*(5), 822–0827.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Polgár, A. (2020). *GitHub - azazdeaz/fields-ignition: Generate random crop fields for Ignition Gazebo — github.com.* Descargado de https://github.com/azazdeaz/fields-ignition

Pytorch. (2016). Descargado de https://pytorch.org/ (Accedido: 2023-06-18)

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016). *You only look once: Unified, real-time object detection.*

Ren, S., He, K., Girshick, R., y Sun, J. (2016). *Faster r-cnn: Towards real-time object detection with region proposal networks.*

Ristani, E., Solera, F., Zou, R., Cucchiara, R., y Tomasi, C. (2016). Perfor-

mance measures and a data set for multi-target, multi-camera tracking. En *Computer vision–eccv 2016 workshops: Amsterdam, the netherlands, october 8-10 and 15-16, 2016, proceedings, part ii* (pp. 17–35).

Roboflow. (2023). *Generate augmented images to improve model performance.* Descargado de https://docs.roboflow.com/image-transformations/image-augmentation (Accedido: 2023-06-07)

Roy, A. (2013). *Better project templates.* Descargado de https://cookiecutter.readthedocs.io/en/stable/

Roy, P., Dong, W., y Isler, V. (2018). Registering reconstructions of the two sides of fruit tree rows. En *2018 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 1–9).

Schertz, C. E., y Brown, G. K. (1968). Basic considerations in mechanizing citrus harvest. *Transactions of the ASABE*, *11*, 343-0346.

Shah, D. (2022, Mar). *Mean average precision (map) explained: Everything you need to know.* Descargado de https://www.v7labs.com/blog/mean-average-precision#:~:text=What%20is%20Mean%20Average%20Precision%20(mAP)%3F,values%20from%200%20to%201.

Simonyan, K., y Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Simonyan, K., y Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition.*

Solawetz, J. (2023, Enero). *What is yolov8? the ultimate guide.* Roboflow Blog. Descargado de https://blog.roboflow.com/whats-new-in-yolov8/

Stanford Artificial Intelligence Laboratory. (2007). *Robotic operating system.* Descargado de https://www.ros.org

Stein, M., Bargoti, S., y Underwood, J. (2016). Image based mango fruit detection, localisation and yield estimation using multiple view geometry. *Sensors*, *16*(11), 1915.

StereoLabs. (2020). *Zed2 stereo camera.* Product information. (URL: https://www.stereolabs.com/zed2/)

Supervisely. (2017). *Supervisely, unified os for computer vision.* Descargado de https://supervisely.com/ (Accedido: 2023-06-07)

Terven, J., y Cordova-Esparza, D. (2023). A comprehensive review of yolo: From yolov1 to yolov8 and beyond. *arXiv preprint arXiv:2304.00501*.

Tian, Y., Yang, G., Wang, Z., Li, E., y Liang, Z. (2019). Detection of apple lesions in orchards based on deep learning methods of cyclegan and yolov3-dense. *Journal of Sensors*, *2019*.

Ultralytics. (2014). *About.* Descargado de https://ultralytics.com/about (Accedido: 2023-06-18)

van Ittersum, M. K., y Rabbinge, R. (1997). Concepts in production ecology for analysis and quantification of agricultural input-output combinations. *Field crops research*, *52*(3), 197–208.

Wang, Q., Nuske, S., Bergerman, M., y Singh, S. (2013). Automated crop yield estimation for apple orchards. En *Experimental robotics: The 13th international symposium on experimental robotics* (pp. 745–758).

wangwang. (2022, Mayo). *Apple dataset* [Open Source Dataset].

https://universe.roboflow.com/wangwang/apple-shybf. Roboflow. Descargado de https://universe.roboflow.com/wangwang/apple-shybf (Accedido: 2023-05-04)

Whittaker, A. D., Miles, G. E., Mitchell, O. R., y Gaultney, L. D. (1984). Fruit location in a partially occluded image. *Transactions of the ASABE*, *30*, 591-0596.

Wojke, N., Bewley, A., y Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. En *2017 ieee international conference on image processing (icip)* (pp. 3645–3649).

Woo, M., Neider, J., Davis, T., y Shreiner, D. (1999). *Opengl programming guide: the official guide to learning opengl, version 1.2*. Addison-Wesley Longman Publishing Co., Inc.

Wu, L., Ma, J., Zhao, Y., y Liu, H. (2021). Apple detection in complex scene using the improved yolov4 model. *Agronomy*, *11*(3), 476.

Wu, Y. (2019). *Detectron2 model zoo and baselines.* Detectron2. Descargado de https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., y Girshick, R. (2019). *Detectron2.* https://github.com/facebookresearch/detectron2.

Xia, X., Chai, X., Zhang, N., Zhang, Z., Sun, Q., y Sun, T. (2022). Culling double counting in sequence images for fruit yield estimation. *Agronomy*, *12*(2), 440.

Yan, B., Fan, P., Lei, X., Liu, Z., y Yang, F. (2021). A real-time apple targets detection method for picking robot based on improved yolov5. *Remote Sensing*, *13*(9), 1619.

Zhang, W., Wang, J., Liu, Y., Chen, K., Li, H., Duan, Y., ... Guo, W. (2022). Deep-learning-based in-field citrus fruit detection and tracking. *Horticulture Research*, *9*.

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., ... Wang, X. (2022). Bytetrack: Multi-object tracking by associating every detection box. En *Computer vision–eccv 2022: 17th european conference, tel aviv, israel, october 23–27, 2022, proceedings, part xxii* (pp. 1–21).

Glosario

anchor box Se refiere a un cuadro delimitador predefinido utilizado como referencia para estimar la ubicación y clasificación de objetos en una imagen. Ayuda a generar propuestas de regiones que contengan objetos y mejora la precisión del proceso de detección. 5, 16

ancla Punto de referencia utilizado para guiar o estabilizar el proceso de entrenamiento de la red neuronal. 19, 20

backbone Parte central de las redes neuronales, que en general realiza la extracción de características y calcula los mapas de características a partir de los datos de entrada. Pueden ser redes convolucionales pre-entrenadas, como VGG16 o CSPDarkNet53. 5, 18, 40

bag En el contexto de ROS, formato de archivo para almacenar datos de mensajes generados por los nodos. Se utiliza para grabar y reproducir datos, permitiendo análisis y desarrollo fuera del tiempo real. 38

batch Grupo de ejemplos de entrenamiento que se procesan juntos para mejorar la eficiencia del entrenamiento. 60

bounding box Rectángulo que rodea un objeto indicando su posición, clase y nivel de confianza (en predicciones). 15–17, 19, 22, 44, 45, 57, 64, 66, 68

brindilla Pequeñas ramas que se encuentran en los árboles o plantas, más pequeñas, flexibles y delgadas en comparación con las ramas principales. 29, 30

confidence Refiere a la medida de certeza o confianza que tiene el modelo de detección sobre la precisión de sus predicciones. Es un valor numérico que indica la probabilidad estimada de que una detección o clasificación sea correcta. Cuanto mayor sea el valor de confidence, mayor será la confianza del modelo en la precisión de la detección. Se utiliza para establecer umbrales de confianza y tomar decisiones sobre la validez de las detecciones realizadas por el modelo. 68

data augmentation Conjunto de tecnicas utilizadas para incrementar la cantidad de datos mediante la adición de copias de algunas imágenes ligereamente modificadas o creación de imágenes nuevas mediante la combiación de imágenes existentes. Sirve para mejorar las predicciones en algoritmos de aprendizaje automático y reducir su overfitting. 5, 6, 18–20, 53, 54, 56

dataset Conjunto de datos estructurados utilizados para entrenar y evaluar modelos. 4, 53–56, 59–63, 65, 67, 78, 91–93, 96

espolón Brotes o ramas cortas que crecen en el tronco o en las ramas principales de un árbol o planta. Estas protuberancias suelen ser más pequeñas y menos desarrolladas que las ramas normales. En los manzanos, los espolones son importantes para la formación de nuevos racimos de frutas. En la poda de árboles frutales, a menudo se eliminan algunos espolones para promover un crecimiento más equilibrado y una mejor producción de frutas. 29

framework Estructura de software que proporciona una base de trabajo para el desarrollo de aplicaciones. Ofrece reglas, convenciones y funcionalidades predefinidas que permiten a los desarrolladores crear software de manera más eficiente y consistente. 16, 18

ground truth En el área de aprendizaje automático, representa el conjunto de datos etiquetado contra el cual se evaluarán los modelos en las etapas de entrenamiento e inferencia. 44, 45, 51, 52, 68, 71

head Parte final de las redes neuronales que se encarga de realizar tareas específicas, como clasificación, detección de objetos o segmentación semántica, utilizando las características procesadas por el *neck*. Generalmente se compone por una o varias capas de neuronas que realizan cálculos más especializados y específicos para la tarea en particular que se está abordando. Es aquí donde se toman las decisiones finales o se generan las predicciones basadas en las características extraídas y procesadas por el resto de la red neuronal. 5, 18

mecanismo de atención En aprendizaje profundo, un mecanismo de atención, del inglés *Attention Mechanism*, es un método que permite al modelo enforcase en las partes más relevantes de los datos de entrada al hacer una predicción. 15

neck Parte intermedia de las redes neuronales, que se utiliza para fusionar y transformar las características extraídas por el *backbone* antes de ser utilizadas por la cabeza (*head*) de la red neuronal. Su función principal es realizar operaciones de procesamiento intermedio y mejorar la representación de las características para tareas específicas, como la detección de objetos o la segmentación semántica. 5, 18

non-maximum supression Algoritmo que permite seleccionar una única entidad sobre un conjunto de entidades superpuestas, por ejemplo, bounding boxes. Funciona eliminando entidades redundates, al seleccionar aquella con mayor confianza y suprimir las entidades con un alto grado de solapamiento. Las entidades son ordenadas según su confianza y se evalúa su solapamiento utilizando la medida IoU, lo que garantiza que solo se mantenga la entidad más confiable y evita que se detecten objetos duplicados. 15

PCA Color Augmentation técnica de Data Augmentation que funciona alternando las intensidades de los canales RGB a lo largo de variaciones naturales de las imágenes. Realiza el análisis en componentes principales de los canales de color. 5

raleo Práctica que consiste en la eliminación o corte selectivo de árboles, plantas o flores de una población para mejorar el crecimiento y desarrollo de los individuos restantes. Esta técnica se utiliza en plantaciones con el objetivo de reducir la competencia entre los frutos, proporcionar espacio para un crecimiento más saludable y promover una distribución adecuada. 29

softmax Función que convierte un vector de N dimensiones a una distribución de probabilidades con N posibles salidas. En general se usa como la última función de activación de una red neuronal para normalizar la salida de una red a una distribución de probabilidades sobre las clases de salida. 15

transfer learning Técnica de aprendizaje automático que implica utilizar un modelo previamente desarrollado para una tarea específica como punto de partida para crear otro modelo destinado a abordar una tarea diferente a la original. 4

épocas Número de veces que un algoritmo de entrenamiento recorre completamente todo el conjunto de datos de entrenamiento. Durante cada época, el algoritmo de entrenamiento ajusta los pesos y los sesgos de la red neuronal con el objetivo de minimizar la función de pérdida. 6, 20, 40, 60–62, 77, 79, 91, 92

Apéndice A

Resultados completos de detección

A.1. Entrenamientos de los modelos por cantidad de épocas

En la tabla A.1 se indican los resultados obtenidos durante el entrenamiento de los modelos. Para cada modelo y dataset se indica el valor de AP_{50} en cada una de las cantidades de épocas elegidas, mostrando además el mejor valor obtenido en una época del entrenamiento, y el valor obtenido en la última época.

Modelo	Dataset	Número de épocas							
		20		50		100		200	
		Mejor	Última	Mejor	Última	Mejor	Última	Mejor	Última
YOLOv5	DS1	0.803	0.786	0.770	0.770	0.774	0.766	0.782	0.772
	DS2	0.903	0.903	0.906	0.904	0.896	0.895	0.904	0.891
	DS3	0.769	0.769	0.788	0.788	0.797	0.793	0.796	0.793
	DS4	0.486	0.486	0.689	0.677	0.726	0.738	0.729	0.715
	DS7	0.720	0.706	0.733	0.732	0.729	0.720	0.727	0.705
YOLOv8	DS1	0.800	0.800	0.794	0.773	0.814	0.777	0.801	0.782
	DS2	0.884	0.884	0.903	0.903	0.902	0.902	0.904	0.899
	DS3	0.756	0.756	0.778	0.779	0.783	0.783	0.798	0.799
	DS4	0.711	0.701	0.711	0.701	0.722	0.724	0.720	0.721
	DS7	0.703	0.703	0.717	0.709	0.712	0.710	0.712	0.709
Faster-101	DS1	0.801	0.722	-	-	-	-	-	-
	DS2	0.869	0.869	-	-	-	-	-	-
	DS3	0.717	0.703	-	-	-	-	-	-
	DS4	0.603	0.587	-	-	-	-	-	-
	DS7	0.632	0.632	-	-	-	-	-	-
Faster-50	DS1	0.799	0.778	0.796	0.735	-	-	-	-
	DS2	0.867	0.867	0.878	0.861	-	-	-	-
	DS3	0.700	0.700	0.724	0.724	-	-	-	-
	DS4	0.557	0.557	0.633	0.620	-	-	-	-
	DS7	0.624	0.624	0.642	0.629	-	-	-	-

Tabla A.1: Valores de AP_{50} obtenidos por cada modelo según el número de épocas, indicando el valor de la mejor época y de la última época del entrenamiento. Si no se incluye el resultado, no se realizó el entrenamiento del modelo para esa cantidad de épocas.

A.2. Pycocotools sobre conjunto de test

En la tabla A.2 se presentan los resultados obtenidos por cada modelo sobre cada datasets sobre su conjunto de test.

Modelo	Dataset	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
YOLOv5	DS1	0.471	0.893	0.456	0.36	0.502	-1
	DS2	0.443	0.895	0.375	0.437	0.539	0.649
	DS3	0.392	0.774	0.364	0.366	0.634	-1
	DS4	0.215	0.579	0.099	0.215	-1	-1
	DS7	0.314	0.672	0.256	0.309	0.560	-1
YOLOv8	DS1	0.544	0.885	0.578	0.386	0.582	-1
	DS2	0.438	0.885	0.352	0.430	0.546	0.696
	DS3	0.401	0.764	0.389	0.374	0.649	-1
	DS4	0.234	0.557	0.151	0.234	-1	-1
	DS7	0.306	0.644	0.261	0.300	0.510	-1
Faster-101	DS1	0.419	0.856	0.350	0.247	0.463	-1
	DS2	0.393	0.878	0.244	0.388	0.480	0.562
	DS3	0.360	0.725	0.317	0.338	0.572	-1
	DS4	0.168	0.501	0.068	0.168	-1	-1
	DS7	0.248	0.593	0.171	0.245	0.447	-1
Faster-50	DS1	0.424	0.879	0.355	0.302	0.458	-1
	DS2	0.421	0.882	0.335	0.417	0.465	0.553
	DS3	0.373	0.734	0.329	0.348	0.605	-1
	DS4	0.168	0.499	0.056	0.168	-1	-1
	DS7	0.261	0.606	0.183	0.257	0.471	-1

Tabla A.2: Resultados de la evaluación de los datasets sobre su conjunto de Test utilizando Pycocotools.

A.3. Experimentación con MinneApple

En esta sección se incluyen los resultados obtenidos durante la experimentación entrenando y evaluando cada modelo con el dataset MinneApple (Häni y cols., 2020b), comparándolo con los resultados descritos en su artículo.

Como se puede notar en la tabla A.3, los resultados obtenidos son considerablemente peores que los oficiales. Esto llama la atención dado que, si bien el tamaño del conjunto de *test* puede variar, los datos de entrenamiento son los mismos.

Se observa que para la métrica AP_{large} no se obtienen resultados dado que las imágenes fueron reescaladas a para ser imágenes cuadradas de 640×640 con márgenes negros para completar las imágenes, por lo que las manzanas del dataset original que cumplían esta condición dejaron de cumplirla.

Modelo	AP	AP_{50}	AP_{75}	AP_{small}	AP_{medium}	AP_{large}
Resultados oficiales						
Faster RCNN	0.438	0.775	0.455	0.297	0.578	0.871
Resultados obtenidos						
YOLOv5	0.314	0.672	0.256	0.309	0.560	-1
YOLOv8	0.306	0.644	0.261	0.300	0.510	-1
Faster-101	0.248	0.593	0.171	0.245	0.447	-1
Faster-50	0.261	0.606	0.183	0.257	0.471	-1

Tabla A.3: Resultados obtenidos sobre MinneApple sobre su conjunto de test en comparación con los resultados oficiales (Häni y cols., 2020b).

Apéndice B

Resultados completos de conteo

B.1. Comparación de resultados

En este anexo se describen los resultados obtenidos en la publicación *"A comparative study of fruit detection and counting methods for yield mapping in apple orchards"* (Häni y cols., 2020a). En dicho artículo se presenta un sistema para la estimación de la cosecha, partiendo de la evaluación modelos para la detección de manzanas y llegando al conteo de manzanas en toda la fila.

Al igual que en el presente proyecto, se ven enfrentados a varios problemas ya descritos en este documento. Entre estos se destacan las oclusiones y el conteo de frutas repetidas, visibles desde ambos lados de la fila.

En lo que respecta a la detección de manzanas, demuestran cómo la red de segmentación semántica U-Net puede ser utilizada para esto, superando los resultados de distintos modelos del estado del arte.

Lo novedoso de este artículo es su aproximación para resolver el problema de conteo en toda la fila. Para esto, a partir del video, utiliza el algoritmo SfM (*Structure from Motion*), que estima puntos 3D y la posición de la cámara para recrear la escena en 3D. A partir de esto, utiliza técnicas de proyección para asociar puntos 3D a las detecciones de la manzana en cada cuadro del video. Finalmente, las reconstrucciones 3D de ambos lados son unidas para eliminar el conteo de frutas repetidas. De esta forma, la cantidad de objetos en el modelo 3D representa la cantidad estimada de manzanas de la plantación.

Al comparar los resultados de las sumas de ambos lados de la fila para los resultados del proyecto en cuestión (tabla B.1) con los resultados obtenidos por nuestro proyecto (tabla B.2), se puede apreciar que en ambos casos los resultados son mayores de lo esperado. Esto se da por numerosas razones, pero principalmente por la existencia de manzanas visibles desde ambos lados de la fila.

También se observa que los resultados obtenidos mediante la reconstrucción

3D son, a simple vista, mejores que los de seguimiento. Sin embargo, esta comparación no es adecuada, ya que las diferencias entre los conjuntos de datos no permiten una comparación válida. Un ejemplo de esto, es que en los dataset 1 a 3 no se enfrentan al problema de la detección de manzanas de filas posteriores, mientras en el video de INIA Las Brujas las manzanas en segunda fila son bastante visibles.

Datasets	**GT**	**Lados unidos**		**Suma de lados**	
		GMM	ResNet50	GMM	ResNet50
Dataset-1	270	256 (94.81 %)	258 (95.56 %)	348 (129 %)	347 (129 %)
Dataset-2	274	252 (91.98 %)	268 (97.81 %)	411 (150 %)	405 (148 %)
Dataset-3	414	392 (94.68 %)	405 (97.83 %)	422 (102 %)	430 (104 %)

Tabla B.1: Resultados de (Häni y cols., 2020a) para el conteo de manzanas en la fila para cada uno de sus conjuntos de datos.

RN	**Seguimiento**	**GT**	**Suma ida y vuelta**
YOLOv5	OCSort	2133	5721 (268 %)
	Bytetrack	2133	5090 (239 %)
	StrongSort	2133	5044 (236 %)
YOLOv8	OCSort	2133	5190 (243 %)
	Bytetrack	2133	3695 (173 %)
	StrongSort	2133	3767 (177 %)

Tabla B.2: Resultados de conteo para la fila 22 de INIA Las Brujas, donde se suman los resultados de ambos lados.

B.2. Instancias de entrenamiento de modelos de ajuste

En las tablas B.3 y B.4 se incluyen las instancias de video generadas con el simulador para el entrenamiento de los modelos de regresión, con sus respectivas cantidades de manzanas totales y las predicciones y ajustes de cada modelo de seguimiento. En particular, las cantidades de manzanas identificadas por cada algoritmo se indican en la columna “Valor”, y las columnas “Coef.” son los coeficientes, calculados como el cociente entre la cantidad total (GT) y el valor predicho.

$$\text{Coef.} = \frac{\text{GT}}{\text{Valor}} \tag{B.1}$$

Instancia	GT	OCSort		Bytetrack		StrongSort	
		Valor	Coef.	Valor	Coef.	Valor	Coef.
regression_ap_0	328	721	0.455	833	0.394	489	0.671
regression_ap_1	349	813	0.429	841	0.415	553	0.631
regression_ap_2	364	1151	0.316	757	0.481	787	0.463
regression_ap_3	347	841	0.413	784	0.443	562	0.617
regression_ap_4	363	785	0.462	738	0.492	533	0.681
regression_ap_5	395	862	0.458	837	0.472	589	0.671
regression_ap_6	343	999	0.343	979	0.350	849	0.404
regression_ap_7	338	761	0.444	796	0.425	535	0.632
regression_ap_8	393	904	0.435	764	0.514	596	0.659
regression_ap_9	370	731	0.506	1077	0.344	524	0.706
regression_ap_10	387	785	0.493	679	0.570	555	0.697
regression_ap_11	328	727	0.451	718	0.457	500	0.656
regression_ap_12	392	853	0.460	900	0.436	614	0.638
regression_ap_13	388	823	0.471	672	0.577	585	0.663
regression_ap_14	307	686	0.448	762	0.403	505	0.608
regression_ap_15	306	676	0.453	719	0.426	490	0.624
regression_ap_16	357	728	0.490	719	0.497	535	0.667
regression_ap_17	387	873	0.443	832	0.465	554	0.699
regression_ap_18	391	839	0.466	688	0.568	606	0.645
regression_ap_19	345	742	0.465	865	0.399	542	0.637

Tabla B.3: Resultados de YOLOv5 de conteo por instancia de regresión para cálculo de coeficientes promedios.

Instancia	GT	OCSort		Bytetrack		StrongSort	
		Valor	Coef.	Valor	Coef.	Valor	Coef.
regression_ap_0	328	786	0.417	564	0.582	424	0.774
regression_ap_1	349	865	0.403	672	0.519	469	0.744
regression_ap_2	364	930	0.391	696	0.523	498	0.731
regression_ap_3	347	932	0.372	743	0.467	494	0.702
regression_ap_4	363	909	0.399	645	0.563	462	0.786
regression_ap_5	395	912	0.433	693	0.570	516	0.766
regression_ap_6	343	799	0.429	598	0.574	433	0.792
regression_ap_7	338	828	0.408	632	0.535	464	0.728
regression_ap_8	393	1020	0.385	769	0.511	545	0.721
regression_ap_9	370	899	0.412	658	0.562	485	0.763
regression_ap_10	387	832	0.465	637	0.608	496	0.780
regression_ap_11	328	805	0.407	643	0.510	439	0.747
regression_ap_12	392	969	0.405	748	0.524	549	0.714
regression_ap_13	388	918	0.423	688	0.564	508	0.764
regression_ap_14	307	732	0.419	551	0.557	432	0.711
regression_ap_15	306	748	0.409	561	0.545	425	0.720
regression_ap_16	357	840	0.425	609	0.586	455	0.785
regression_ap_17	387	950	0.407	728	0.532	491	0.788
regression_ap_18	391	928	0.421	739	0.529	557	0.702
regression_ap_19	345	830	0.416	587	0.588	461	0.748

Tabla B.4: Resultados de YOLOv8 de conteo por instancia de regresión para cálculo de coeficientes promedios.

B.3. Resultados completos de métodos de ajuste

En la tabla B.5 se encuentran los resultados de aplicar cada par de modelos de detección y de seguimiento a cada una de las instancias de videos de regresión de tipo V1_5 (de una única fila de árboles con 5 árboles). Para cada instancia se incluye la cantidad total de manzanas como GT (*ground truth*). Los resultados de conteo se incluyen en la columna resultados (R), y en las columnas $R * CA$ y Regresión se incluyen los resultados de aplicar cada modelo de ajuste, así como su error relativo.

Seguimiento	Instancia	GT	R	R * CA		Regresión	
				Valor	ER	Valor	ER
YOLOv5 OCSort	*regression_val_0*	390	940	418	0.073	373	0.044
	regression_val_1	360	767	341	0.052	353	0.019
	regression_val_2	375	786	350	0.067	356	0.051
	Promedio	**375**	**831**	**370**	**0.064**	**361**	**0.038**
YOLOv5 Bytetrack	*regression_val_0*	390	950	433	0.111	359	0.079
	regression_val_1	360	761	347	0.035	359	0.003
	regression_val_2	375	812	371	0.012	359	0.043
	Promedio	**375**	**841**	**384**	**0.053**	**359**	**0.042**
YOLOv5 StrongSort	*regression_val_0*	390	564	357	0.084	358	0.082
	regression_val_1	360	535	339	0.059	355	0.014
	regression_val_2	375	533	338	0.100	355	0.053
	Promedio	**375**	**544**	**345**	**0.081**	**356**	**0.050**
YOLOv8 OCSort	*regression_val_0*	390	1117	461	0.181	436	0.119
	regression_val_1	360	828	342	0.051	345	0.041
	regression_val_2	375	940	388	0.034	380	0.015
	Promedio	**375**	**962**	**397**	**0.089**	**387**	**0.058**
YOLOv8 Bytetrack	*regression_val_0*	390	773	423	0.085	397	0.019
	regression_val_1	360	603	330	0.083	340	0.054
	regression_val_2	375	714	391	0.042	378	0.007
	Promedio	**375**	**697**	**381**	**0.070**	**372**	**0.027**
YOLOv8 StrongSort	*regression_val_0*	390	508	380	0.025	376	0.036
	regression_val_1	360	467	349	0.029	351	0.026
	regression_val_2	375	478	358	0.046	358	0.046
	Promedio	**375**	**484**	**362**	**0.034**	**361**	**0.036**

Tabla B.5: Resultados de aplicar los modelos entrenados de regresión lineal a los resultados obtenidos por modelo de detección y seguimiento, donde "GT" es la cantidad de manzanas total del video, "R" es el resultado de los modelos, "CA" el coeficiente de ajuste, "Regresión" la predicción con regresión lineal, "Valor" los resultados de multiplicar el resultado por el coeficiente de ajuste y de aplicar regresión lineal, y "ER" los errores relativos respectivos.