UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Pipeline de detección y seguimiento de manzanas para geolocalización de anomalías

Informe de Proyecto de Grado presentado por

Alexei Guchin, Thomas Sheppard

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, 15 de abril de 2024

Resumen

El objetivo principal de este trabajo es desarrollar una solución que permita automatizar y optimizar el proceso de análisis de frutas en entornos agrícolas. Para lograr este propósito, se diseñó un sistema que combina técnicas de visión por computadora y aprendizaje profundo.

En este trabajo, presentamos la implementación de un sistema que clasifica manzanas en sanas o enfermas y las asocia con una posición geográfica para identificar dónde se encuentran. Este sistema, es un pipeline que integra algoritmos de detección, tracking (seguimiento) en video y clasificación de manzanas, vinculándolas a una ubicación geográfica para generar un mapa de calor que identifica áreas con una mayor presencia de manzanas enfermas. Durante el procesamiento de cada frame de un video, se lleva a cabo la detección de las manzanas presentes, su seguimiento en el tiempo y el conteo de las manzanas vistas. Al finalizar de procesar el video se clasifican las manzanas detectadas utilizando todas las imágenes que se obtuvieron de cada una. También puede clasificar las manzanas en tiempo real, es decir, a medida que se procesan los frames. El pipeline es muy flexible al uso de otros algoritmos ya que fue implementado de forma genérica, teniendo que solamente implementar interfaces que usen el algoritmo.

Además, está integrado con ROS (Robot Operating System) para que pueda ejecutarse en robots.

Para la detección de manzanas se utilizó YOLOv8, Faster R-CNN y SAM. Para el seguimiento en video se emplearon StrongSORT, HybridSORT, DeepOCSORT, BoT-SORT y Norfair. Para la clasificación de manzanas se utilizó CLIP y VisionTransformer.

Palabras clave: Detección de objetos, Tracking, Seguimiento en video, Clasificación de imágenes, Manzanas, Redes Neuronales, Inteligencia Artificial, Visión por computadora, Robótica, Agricultura de precisión

Índice general

Capítulo 1

Introducción

En la industria agrícola y alimentaria, la evaluación de la calidad de los productos desempeña un papel fundamental en la toma de decisiones relacionadas con la producción, la distribución y la comercialización. En particular, la calidad de las frutas, como las manzanas, es un aspecto crucial que influye significativamente en su valor de mercado y satisfacción del consumidor. Una de las razones importantes que afectan a la exportación de manzanas es que la calidad de las manzanas es bastante irregular. Con un aumento en la atención a la calidad y los estándares de seguridad de las frutas, la demanda de identificación de calidad automática, precisa y rápida continúa creciendo. Además, el crecimiento exponencial de la población amenaza con reducir los niveles de seguridad alimentaria a medida que avanza el tiempo. (F. Li, 2021). Por lo tanto, la detección y clasificación precisa de la calidad de las manzanas, que incluye aspectos como su madurez, firmeza, color, presencia de defectos y deterioro, es un desafío complejo e indispensable de abordar, que ha llevado a la búsqueda de soluciones innovadoras.

Históricamente, la evaluación de la calidad de las manzanas ha estado en gran medida a cargo de inspecciones manuales realizadas por personas, lo que puede ser subjetivo, costoso y propenso a errores. Además, con la creciente demanda de una producción eficiente y de alta calidad, y con el decrecimiento de la cantidad de trabajadores agrícolas, se ha vuelto necesario desarrollar sistemas automatizados que puedan realizar esta tarea de manera precisa y eficiente (Shaohua Wan, 2019).

Por estas razones es que han surgido numerosas investigaciones en las que se intenta automatizar el proceso de detección de frutas y estimación de calidad. Este enfoque se encuentra enmarcado dentro del concepto de agricultura de precisión (PA), que refiere a la ciencia de mejorar los rendimientos de los cultivos y ayudar en las decisiones de gestión mediante el uso de sensores de alta tecnología y herramientas de análisis (*Precision agriculture*, s.f.). La agricultura de precisión tiene una amplia gama de utilidades, entre las que se encuentran optimizar la cosecha, determinando el momento óptimo para la recolección y evitando así la cosecha prematura o tardía; estimar los recursos necesarios pa-

ra la cosecha; controlar la calidad en tiempo real, lo que permite el monitoreo continuo para evitar que una fruta enferma pueda contaminar a las vecinas. Además, reducen costos de personal que diariamente monitorizan huertas. En este trabajo nos centraremos en las manzanas, que es una fruta que ha sido muy estudiada, además de ser muy consumida y producida.

El objetivo principal de este proyecto fue estudiar el estado del arte en visión por computadora y redes neuronales aplicado a detección de calidad de manzanas, para elaborar un sistema que utilice los algoritmos del estado del arte para analizar la calidad a partir de un video. Este trabajo está enmarcado en un proyecto del grupo MINA (Network Management / Artificial Intelligence) del InCo (Instituto de Computación, Fing, Udelar) en conjunto con el Instituto Nacional de Investigación Agropecuaria (INIA).

Este trabajo consistió en primera instancia en investigar cómo se ha abordado en los últimos años la automatización del control de calidad, de la detección, conteo y seguimiento de manzanas desde el punto de vista del software más que de la parte robótica y operativa (principalmente estudiando técnicas de visión por computadora con redes neuronales).

Luego de realizar este relevamiento, se decidió integrar todas las técnicas estudiadas para hacer un sistema que realice todo el ciclo (pipeline) desde la detección, el seguimiento y la evaluación de calidad, a partir de múltiples imágenes de cada manzana. Esto último puede aumentar la eficacia del método de clasificación, ya que se analizan imágenes desde distintos ángulos de cada manzana, lo cual contribuye a detectar enfermedades debido a que se estaría inspeccionando una mayor superficie de cada manzana y desde distintos puntos de vista.

El sistema también procesa datos de información geográfica y asocia a cada manzana una ubicación para poder realizar un mapa de calor de manzanas enfermas. El mapa de calor es útil para poder identificar plagas prematuramente, si se detectan zonas donde hay mucha cantidad de manzanas enfermas.

En definitiva, en este trabajo se diseñó e implementó un sistema que integra diversas técnicas de visión por computadora, junto con información geográfica para ubicar las zonas con manzanas enfermas. El sistema está integrado a ROS para que pueda ser embebido y utilizado fácilmente en robots. Además, está diseñado para que la adaptación a nuevos algoritmos sea una tarea sencilla, es decir, no está acoplado a ningún algoritmo, lo cual asegura su relevancia y utilidad a futuro.

La estructura del presente trabajo se compone de los siguientes capítulos: Marco teórico, Revisión de antecedentes, Parte Central, Experimentación y Conclusiones y Trabajo Futuro.

Capítulo 2

Marco teórico

Hasta hace unos años, para la detección de objetos y clasificación de imágenes se empleaban principalmente técnicas de extracción de características en imágenes. A partir de estas características, se entrenaba un modelo de aprendizaje automático que evaluaba la imagen según el valor de cada característica. Con el boom de las redes neuronales, que se dio alrededor del año 2010, este tipo de métodos, que se conocen como métodos clásicos (sección 2.1) fueron dejados a un segundo plano por las redes convolucionales (sección 2.2), las cuales a día de hoy conforman el estado del arte. Día a día salen nuevos modelos de redes convolucionales con mejores resultados, ya sea por el uso de una arquitectura más compleja y disruptiva o por ser entrenadas con mayor cantidad de datos.

2.1. Métodos clásicos

Los métodos clásicos de detección de objetos se basan en el uso de características de cada imagen, tales como bordes, texturas, colores y tamaño, para detectar objetos en una imagen. Estos métodos se presentan a título informativo, ya que no son el foco del trabajo ni fueron utilizados durante el mismo. Para profundizar en el tema, ver el proyecto de grado titulado *Reconocimiento y cuantificación de manzanas* (Tanco, 2015).

Los métodos clásicos pueden clasificarse en tres tipos según su enfoque:

- **Enfoque basado en características:** utiliza algoritmos para extraer características específicas de las imágenes y luego utiliza algoritmos de clasificación para detectar objetos. Ejemplos incluyen el método de HOG (Priya, Jyoshna, Amaraneni, y Swamy, 2020), (Y. Li, Feng, Liu, y Han, 2021), la técnica GLCM (Y. Li y cols., 2021) y el algoritmo de Viola-Jones (Winarno, Hadikurniawati, Nirwanto, y Abdullah, 2018).

- **Enfoque basado en regiones:** divide la imagen en regiones y luego se examina cada región para detectar objetos. Algunos ejemplos son el

método de búsqueda exhaustiva (por fuerza bruta) y el método de Sliding Window (Wan y Goudos, 2019).

- **Enfoque basado en segmentación**[1]**:** primero segmenta la imagen en regiones que contienen objetos y luego utiliza técnicas de clasificación para detectar objetos en cada región. Entre los ejemplos se encuentran el método Contour Segmentation, Clustering Segmentation y Watershed Segmentation (Bargoti y Underwood, 2017).

Una desventaja que poseen los métodos clásicos frente a las redes convolucionales es que requieren la definición manual de las características y algoritmos específicos de extracción de características, lo que puede ser un proceso complejo, costoso y propenso a errores. Las redes convolucionales de todas formas requieren el ajuste de metaparámetros como cantidad de capas y tamaño de filtros. Otra desventaja, es que los métodos clásicos tienen menos capacidad de generalización y comúnmente son más sensibles a la escala y orientación de los objetos en las imágenes (Wan y Goudos, 2019). Por otro lado, las redes convolucionales (sección 2.2) son capaces de aprender características y patrones de grandes conjuntos de datos, logrando de forma precisa y eficiente la segmentación y clasificación de imágenes. Las características a analizar las aprende la propia red durante el entrenamiento, no las define el programador. Sin embargo, a diferencia de los métodos clásicos, las redes convolucionales requieren de una cantidad de imágenes de entrenamiento mucho mayor para que funcionen razonablemente bien (Fu, 2020).

2.2. Redes convolucionales

Las redes neuronales convolucionales (CNN por sus siglas en inglés) son un tipo de red neuronal ampliamente utilizado hoy en día para tareas de visión por computadora como clasificación de imágenes.

Las redes neuronales, de forma general, conforman una familia de algoritmos de aprendizaje automático. Su nombre y estructura están inspirados en el cerebro humano, imitando la forma en que las neuronas biológicas se comunican entre sí (*What are neural networks?*, s.f.). Una red neuronal está compuesta por un conjunto de neuronas, las cuales están agrupadas en capas y conectadas con neuronas de otras capas (ver figura 2.1). Cada neurona tiene un conjunto de entradas, que surgen de la salida de las neuronas con las que está conectada. A cada entrada se la multiplica por un peso y se suman todas las entradas ponderadas. Al resultado de esta suma, se le aplica una función no lineal (llamada función de activación) y el resultado de aplicar esta función es la salida de la neurona. Los mencionados pesos, que se multiplican por las salidas de las neuronas, son los parámetros que la red neuronal aprende durante el entrenamiento.

Volviendo a las redes convolucionales, éstas se empezaron a consolidar para el reconocimiento y clasificación de imágenes a partir del año 2012, y al día

[1] La segmentación es un proceso de clasificación por píxel que asigna una categoría a cada píxel de la imagen analizada (*Segmentación (procesamiento de imágenes)*, 2023).

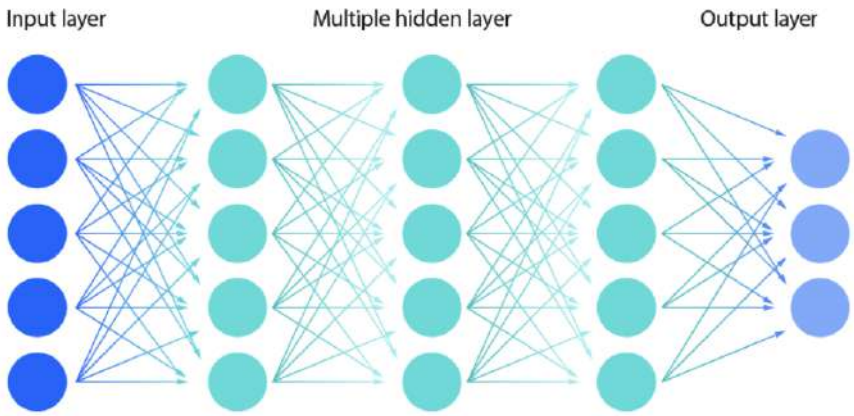

Figura 2.1: Arquitectura básica de una red neuronal feedforward (es feedforward ya que la salida de cada neurona en una capa fluye directamente a la capa siguiente). Se tiene una capa con la entrada de la red, un conjunto de capas ocultas y una capa de salida. Extraída de (*What are neural networks?*, s.f.).

de hoy es el método más usado, ya que supera categóricamente a los métodos clásicos en aspectos como el tiempo y la efectividad. El uso de redes neuronales se remonta a los años 90s, cuando se logró entrenar la primera red neuronal con backpropagation[2]. En esos años se pudo entrenar una red para reconocimiento de dígitos en imágenes. Sin embargo, se seguía optando por los métodos de extracción de características manuales, ya que se confiaba más en la elección de las características que en los métodos de caja negra como las redes neuronales que tienen millones de parámetros que aprender, incomprensibles para un humano. Con la llegada del deep learning (aprendizaje profundo) hubo un cambio de paradigma ya que los métodos clásicos fueron rotundamente superados. Se le llama aprendizaje profundo ya que este paradigma se basa en aumentar considerablemente la cantidad de capas (redes muy profundas), de neuronas y entrenar con muchísimos más datos. Las CNN fueron desplazando a los métodos tradicionales, hasta convertirse en el estado del arte (*Curso de Aprendizaje Profundo para Visión Artificial - Clase 1*, 2022).

Las CNN aprenden por sí solas las características de las imágenes sin que el programador deba especificar qué características quiere extraer. Esta es la primera gran diferencia con los métodos clásicos, en los que se debía elegir previamente las características y entrenar algoritmos específicos para cada característica. Las redes convolucionales extraen características de muy alto nivel

[2]Es un algoritmo que se utiliza para entrenar una red neuronal usando la regla de la cadena proveniente del cálculo. Después de cada pasada hacia adelante a través de una red, backpropagation realiza una pasada hacia atrás ajustando los parámetros del modelo (pesos y sesgos) para minimizar el error entre la salida esperada y la obtenida de la red (*Understanding Backpropagation Algorithm*, 2019).

ya que en las capas ocultas de las redes profundas se aprenden representaciones complejas de los datos de entrada de distintos niveles de abstracción (*Curso de Aprendizaje Profundo para Visión Artificial - Clase 1*, 2022).

La arquitectura básica de una CNN consta de:

- **capas convolucionales:** cada una de ellas tiene un conjunto de filtros que son aplicados a la salida de la capa anterior. A la salida de una capa convolucional se le llama conjunto de mapas de características, ya que en estas capas es donde ocurre la extracción de características. Los filtros son matrices de dimensión pequeña (como de 3x3 píxeles) que se aplican a las imágenes filtradas de la capa anterior mediante la operación convolucional. Para entender la operación convolucional, supongamos que en una imagen de 300x300 queremos aplicar un filtro de 5x5. Por cada píxel de la imagen[3], se toma un cuadrado de 5x5 con el píxel en cuestión como centro del cuadrado, se hace el producto de matrices entre el cuadrado y el filtro, se suman todas las entradas de la matriz resultante y ese resultado es el que va en el píxel en cuestión de la matriz de salida. Luego de aplicar los filtros, se obtienen un conjunto de mapas de activación, los cuales pasan por una función de activación (como ReLU) para obtener los mapas de características (salida de la capa).

- **capas de pooling:** se encargan de reducir la resolución de las imágenes filtradas en una capa de convolución. Lo más común es colocar una capa de pooling luego de una convolucional. Hay varios tipos de pooling como max pooling y average pooling. La utilidad que tienen estas capas es que aceleran el procesamiento de las imágenes porque reducen la cantidad de parámetros de la red. En la imagen 2.2 se muestra un ejemplo de max pooling con un filtro de tamaño 2x2. Se toman submatrices de 2x2, luego se toma el máximo de la submatriz y ese es el valor que se agrega a la matriz resultante. Stride 2 refiere a que el filtro de 2x2 se aplica con desplazamiento 2.

- **red completamente conectada:** Luego de un conjunto de capas convolucionales y de pooling, los mapas de características de la última capa se pasan a un vector (se "aplastan", flatten en inglés) y ese vector se lo manda a una red completamente conectada que es la encargada de emitir la salida de la red, que puede ser una clasificación de la imagen en varias categorías, dando una cierta probabilidad para cada categoría.

Las CNN tienen muchos menos parámetros que las redes neuronales completamente conectadas ya que los parámetros a aprender son los filtros que son de dimensiones muy chicas. Además, al aplicar el mismo filtro en toda la imagen, hace que las CNN sean invariantes a la traslación, es decir, no importa en qué lugar de la imagen se encuentran los objetos para poder extraer las características correctamente. Con el uso de filtros, se aprovecha la localidad que tienen las

[3]Esto es así en el caso que se trabaje con stride igual a 1. El stride es el desplazamiento del filtro durante la operación de convolución.

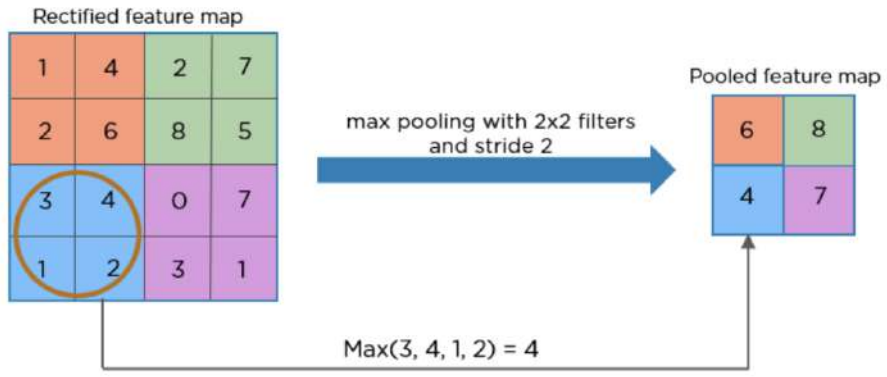

Figura 2.2: Ejemplo visual de funcionamiento de max pooling con filtro de 2x2 y stride 2. Extraída de (*Convolutional Neural Network Tutorial*, 2023).

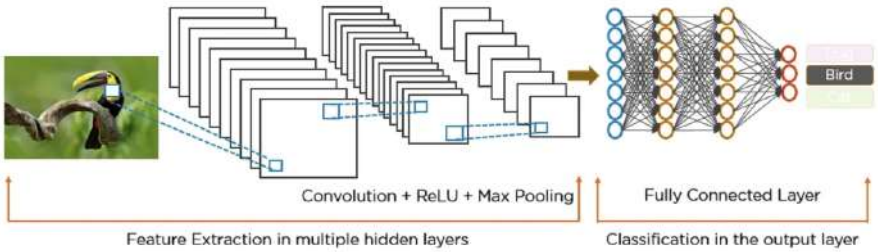

Figura 2.3: Arquitectura básica de una red convolucional. Extraída de (*Convolutional Neural Network Tutorial*, 2023) (*Convolutional Neural Network Tutorial*, 2023).

imágenes: los píxeles cercanos entre sí están relacionados. En una red completamente conectada, píxeles muy lejanos entre sí estarían conectados agregando muchísimos parámetros a la red.

En el trabajo *Intelligent fruit yield estimation for orchards using deep learning based semantic segmentation techniques—a review* (Maheswari, 2021), mencionan que hay tres familias de métodos de CNNs para segmentación de imágenes (figura 2.4):

- **CNN con predicción a nivel de píxel (pixel-wise):** Son útiles para tener una localización más precisa de los objetos ya que además no usan capas de pooling, es decir, no pierden resolución las imágenes.
- **Predicción completamente convolucional:** se usan solamente capas convolucionales. No hay capas de pooling para no bajar la resolución de las imágenes, tampoco hay redes completamente conectadas. Usan arquitec-

turas encoder-decoder[4], donde las primeras capas convolucionales hacen downscaling y las últimas hacen upscaling para devolver una imagen del mismo tamaño que la entrada pero segmentada.

- **Predicción basada en regiones:** se predicen regiones de interés (en inglés ROIs, Region of Interest) y se clasifica cada región. En esta familia se encuentran métodos como Faster R-CNN y YOLO, los cuales son los más usados para detectar manzanas.

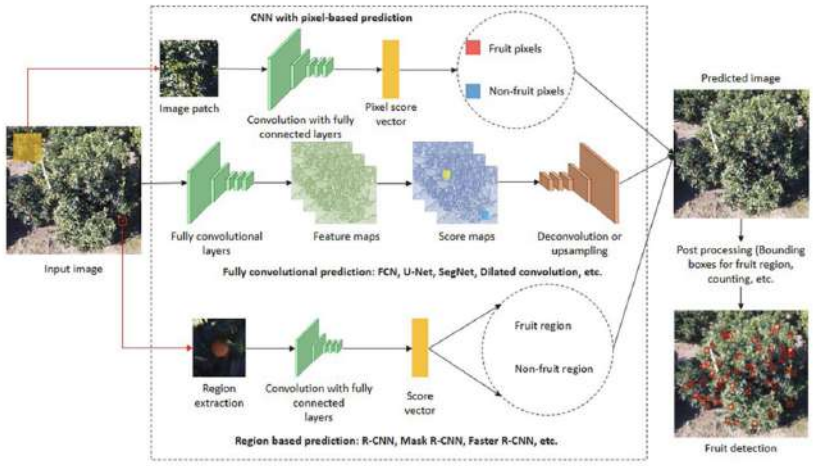

Figura 2.4: Familias de métodos de segmentación de imágenes. Extraída de *Intelligent fruit yield estimation for orchards using deep learning based semantic segmentation techniques—a review* (Maheswari, 2021).

2.3. Tracking

El tracking (seguimiento en video) es la tarea de tomar un conjunto de detecciones de objetos, crear una identificación única para cada una de las detecciones y luego seguir cada uno de los objetos a medida que se mueven a lo largo de los frames en un video, manteniendo la asignación de identificaciones (*Object Tracking*, s.f.).

Hay dos grandes paradigmas dentro del tracking (Ricardo Pereira, 2022):

[4]Es un tipo de arquitectura de red neuronal que se utiliza para el aprendizaje de secuencia a secuencia. Está compuesto por dos partes, el codificador y el decodificador. El codificador procesa una secuencia de entrada para producir un conjunto de vectores de contexto, que luego son utilizados por el decodificador para generar una secuencia de salida. Esta arquitectura permite realizar tareas como la traducción automática, resúmenes de texto, descripción de imágenes, entre otras (*A Perfect guide to Understand Encoder Decoders in Depth with Visuals*, 2023).

- Tracking by detection: Consiste en hacer tracking en dos etapas: primero se detectan los objetos de un frame con un detector como puede ser YOLO y luego se realiza una asociación de objetos vistos en frames anteriores con los objetos vistos en el frame actual.

  El avance de los algoritmos de detección en los últimos años y la simplicidad de esta técnica, hace que sea la más usada (Yunhao Du, 2022).

  Este paradigma reduce el problema de tracking a una tarea de asociación de objetos entre frames consecutivos.

  Se utilizan dos enfoques que en muchos casos se combinan:

  - usar predicción de movimiento para determinar los lugares donde es posible que cada objeto pueda estar en el próximo frame. La desventaja que tiene este enfoque, es que una vez se pierde un objeto ya sea por oclusión o por una falencia del detector, es altamente probable que se asigne otro identificador distinto a objetos ya vistos una vez se vuelven a detectar. Este problema se llama re-identificación.
  - almacenar features de cada objeto para comparar con features de objetos del próximo frame y así matchear objetos en base a su apariencia visual. Se computa la similaridad de mapas de features para determinar si dos objetos son el mismo o no. Este enfoque mejora el problema de re-identificación, permitiendo reconocer objetos que hace varios frames no aparecen.

  Esta estrategia se basa entonces en extraer embeddings de movimiento y/o apariencia y luego hacer matcheo entre dos conjuntos: el de los objetos vistos en el frame actual y el de los objetos vistos en frames anteriores.

- Joint tracking (tracking en conjunto): Consiste en tener un solo modelo que realiza las tareas de detección y tracking al mismo tiempo. El problema más importante de este tipo de modelos se da en la competencia que termina ocurriendo entre la detección y la re-identificación en el aprendizaje de la representación de los objetos. Como la red comparte los mismos vectores de features para ambas tareas, la detección requiere que objetos de una misma clase tengan vectores similares o cercanos, mientras que la tarea de tracking debe poder distinguir entre dos objetos de una misma clase, lo que las hace tareas contradictorias.

  Es más eficiente computacionalmente ya que en una pasada se realizan ambas tareas.

2.4. ROS

ROS (Robot Operating System) es una colección de herramientas y librerías de código abierto para el desarrollo de sistemas robóticos (*ROS - Robot Operating System*, s.f.). Proporciona servicios que se esperan de un sistema operativo, como por ejemplo abstracción de hardware, control de dispositivos a bajo nivel,

comunicación mediante el paso de mensajes entre procesos y gestión de paquetes (*ROS Introduction*, s.f.).

A los procesos que corren en ROS se les llama nodos. Estos nodos pueden, por ejemplo, realizar tareas como controlar un robot, leer datos de sensores, analizar información sensada. Los nodos se pueden comunicar mediante los mecanismos que provee ROS para comunicación entre nodos. En este proyecto, se utilizaron los tópicos, los cuales constituyen una implementación del patrón Observer. En este patrón, los nodos emiten mensajes a los tópicos y, a su vez, tienen la capacidad de suscribirse a tópicos para acceder a la información compartida por otros nodos.

ROS tiene la capacidad de grabar los datos de los sensores en archivos con extensión bag, los cuales pueden ser reproducidos y ejecutados posteriormente. Esta funcionalidad es fundamental para el desarrollo y depuración de aplicaciones robóticas, ya que permite analizar y verificar el comportamiento del robot sin necesidad de que los sensores estén activos o el robot esté en movimiento.

Capítulo 3

Revisión de antecedentes

En este capítulo se presenta el relevamiento llevado a cabo acerca de los antecedentes y el estado del arte tanto en detección, tracking (seguimiento) y clasificación. Se comentan métodos, algoritmos, técnicas y las contribuciones más relevantes acontecidas en los últimos años en las mencionadas áreas, tanto de manera general como aplicados a agricultura de precisión. El capítulo está dividido en Detección, Calidad y Tracking. Cada sección contiene contiene algoritmos del estado del arte y el resumen de los artículos más interesantes que fueron utilizados como referencia ya sea por sus resultados o por las técnicas utilizadas. Para ver más detalles sobre este relevamiento, consultar el documento elaborado acerca del estado del arte (*Estado del arte*, 2023).

3.1. Detección

La detección de objetos es una tarea en visión por computadora que implica identificar y localizar la presencia de objetos en una imagen. El objetivo principal es proporcionar información sobre la ubicación de cada objeto mediante la delimitación de un área rectangular alrededor de cada objeto, conocida como "bounding box".

Las familias de algoritmos más usadas en los últimos años son Faster R-CNN (Ren, He, Girshick, y Sun, 2015) y YOLO (Redmon, Divvala, Girshick, y Farhadi, 2015). Durante el relevamiento, se observó que son las CNN más usadas para detección de manzanas.

3.1.1. Faster R-CNN

Faster R-CNN es una arquitectura de red neuronal convolucional para la detección de objetos en base a regiones. Es una evolución de R-CNN (que fue la arquitectura original) y de Fast R-CNN. Es un método en dos pasos porque primero se procesa la imagen para extraer características y después se analizan los mapas de características para buscar regiones con objetos. Fue publicado en

2015 en el artículo *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* (Ren y cols., 2015).

Faster R-CNN toma como entrada una imagen y como primer paso la procesa con una red convolucional para extracción de características (esta red se puede elegir). Luego, los mapas de características generados son enviados a una Region Proposal Network (RPN), la cual retorna regiones rectangulares (bounding boxes) de la imagen donde la RPN determinó que hay objetos junto con la probabilidad de que haya un objeto en la región. Para generar estas regiones, por cada mapa de características emitido por la red convolucional mencionada anteriormente, se va deslizando una ventana por la que se evalúan k cajas de tamaños predefinidos (anchors lo llaman en el artículo) dentro de la ventana. A cada uno de estos anchors (para cada posición de la ventana) se lo pasa por una capa de pooling que los lleva a las mismas dimensiones para ser procesados por dos redes completamente conectadas: una clasifica cada región (box-classification layer) y la otra refina el rectángulo que la delimita (box-regression layer). En el artículo trabajaron con k=9, lo que significa que para cada píxel se evaluaron 9 cajas de tamaño predefinido. Se utiliza también Non-Maximum Suppression (NMS) para descartar regiones que se superponen y detectan al mismo objeto (*A Step-by-Step Introduction to the Basic Object Detection Algorithms*, 2018).

3.1.2. YOLO

YOLO (You Only Look Once) es un modelo de detección de objetos en imágenes. YOLO utiliza aprendizaje profundo y redes neuronales convolucionales para detectar objetos, y se distingue de sus competidores porque requiere de procesar la imagen una sola vez, lo que le permite ser de los modelos más rápidos. La primera versión de YOLO fue presentada en 2015 en el artículo *You Only Look Once: Unified, Real-Time Object Detection* (Redmon y cols., 2015).

El modelo divide la imagen en grillas y predice simultáneamente la probabilidad de que en cada grilla haya un objeto, la probabilidad del objeto (si lo hay) de pertenecer a cada clase y su bounding box. Desde que se introdujo la primera arquitectura, se han desarrollado varias arquitecturas basadas en YOLO, todas conocidas por su precisión, rendimiento en tiempo real y capacidad para permitir la detección de objetos en dispositivos (edge-devices) y en la nube. YOLOv6, YOLOv7, YOLOv8 y YOLO-NAS son los modelos de última generación de la familia YOLO, que forman parte del estado del arte en la detección y clasificación de objetos en imágenes. Además de realizar detección de objetos, también se ha expandido para hacer segmentación, clasificación, seguimiento y estimación de pose[1].

[1]Se define la estimación de pose como la tarea que implica identificar la ubicación de puntos específicos en una imagen. Los puntos clave pueden representar diversas partes del objeto, como articulaciones, puntos de referencia u otras características distintivas.

3.1.3. SAM

Durante el relevamiento del estado del arte, se encontró que Meta (la empresa que engloba a Facebook) acababa de lanzar un modelo llamado SAM (Segment Anything Model), el cual se afirma estar capacitado para poder segmentar e identificar cualquier objeto, incluso aunque no haya visto imágenes ni objetos similares durante su entrenamiento con resultados sorprendentes, debido a que había sido entrenado con el dataset más grande de segmentación existente. Al descubrir este modelo, se decidió indagar más sobre él y probarlo para evaluar si se podían lograr resultados superiores al estado del arte con este modelo disruptivo.

El 5 de abril de 2023 se presentó el artículo *Segment Anything* (Kirillov, Mintun, y Ravi, 2023) en el que detallan el modelo. En el mismo, afirman que tiene la capacidad de zero-shot-generalization, lo que quiere decir, que SAM tiene una noción general de lo que es un objeto y no necesita entrenamiento extra para poder reconocer un objeto nuevo para el modelo. Esta es una de sus cualidades disruptivas, ya que hasta la fecha lo más común era entrenar modelos para que sean capaces de determinar cierto tipo de objetos que era con los que se entrenaba, obteniendo buenos resultados con imágenes similares a las del entrenamiento. El caso de SAM es distinto ya que es un modelo generalista, si se quiere reconocer manzanas no hay que entrenarla con imágenes de manzanas.

Los objetivos del proyecto son: reducir la necesidad de modelado específico de tareas, de tener que generar datos con anotaciones (muchas veces se hace de manera manual), y de invertir grandes cantidades de poder de cómputo en entrenamiento.

Fue entrenado con más de 11 millones de imágenes, generando más de mil millones de máscaras, lo cual según afirman, disponen del dataset público más grande existente en segmentación de imágenes (*Introducing Segment Anything: Working toward the first foundation model for image segmentation*, 2023).

Tiene como novedoso que permite segmentación automática e interactiva. Segmentación interactiva refiere a que puede segmentar según prompts (instrucciones o comandos que se le da como entrada al modelo) como un punto (devuelve la máscara del objeto que contiene al punto), cuadrados (devuelve las máscaras de objetos dentro del cuadrado) o texto (devuelve las máscaras de objetos que correspondan al texto) aunque este último tipo de prompt no fue liberado pero sí explican en el artículo que lo implementaron usando otro modelo llamado CLIP (sección 3.2.1).

El funcionamiento de este modelo consiste de un encoder de imágenes, un encoder de prompts el cual es ligero, y un decoder que también es ligero que combina la información de ambos encoders que predicen las máscaras de segmentación. El encoding de la imagen se genera una única vez y se reutiliza para todos los prompts que sean ingresados por el usuario. Luego de generado este encoding, dado un prompt, la red demora 50 milisegundos en producir las máscaras, lo cual permite que funcione en tiempo real.

3.1.4. Trabajos relacionados

El artículo *Deep Fruit Detection in Orchards* (Bargoti y Underwood, 2017) utiliza y adapta Faster R-CNN en el contexto de detección de frutas en huertas. Una de las razones por las que eligieron esta arquitectura es por ser open source y de fácil implementación. Mencionan que para lograr un funcionamiento eficiente a gran escala, es necesario poder capturar a todo un árbol con una imagen y a la vez, que la imagen sea de alta resolución ya que el tamaño de las frutas es relativamente pequeño respecto al del árbol. Otra complejidad agregada es el trabajar en entornos exteriores, que puede provocar variaciones de iluminación, distancia a la fruta (lo que varía el tamaño de las frutas en las imágenes), amontonamiento de frutas, ángulo de la cámara que generan variabilidad en las imágenes recolectadas.

Hacen un análisis de la diferencia de trabajar en una huerta y en un invernadero. En promedio las frutas les ocuparon a ellos 32 +/- 9 píxeles en imágenes en huertas comparado con 112 +/- 29 píxeles en invernadero, lo cual indica la información acotada que se obtiene de cada fruta. Por eso fue que optaron por usar un sensor con resolución de 1616×1232 píxeles. Una resolución tan grande hace que se necesite gran capacidad de procesamiento. Por eso fue que a cada imagen la subdividieron en imágenes de 202×308 píxeles. Esta es una alternativa a bajarle la resolución a una imagen sin que pierda calidad.

El trabajo lo hicieron para manzanas, mangos y almendras usando la misma red pero entrenando un modelo para cada fruta. Un objetivo que se propusieron fue determinar la importancia del transfer learning[2] y con qué imágenes se entrenó el modelo del que se transfieren sus parámetros. Para esto compararon el rendimiento de hacer transfer learning entrenando la red con ImageNet[3] respecto a hacer transfer learning después de entrenar el modelo con las imágenes de almendras o mangos para las manzanas. En este último caso, el modelo estaría pre-entrenado con imágenes similares a las de las manzanas ya que todas las imágenes se obtuvieron de la misma huerta. Obtuvieron resultados muy similares, lo que da cuenta que hacer transfer learning con ImageNet es igual de efectivo que hacerlo con imágenes similares.

Hicieron una comparativa entre hacer data augmentation[4] y no hacerlo. Las técnicas que utilizaron fueron hacer flip, reescalar las imágenes y hacer ambas en la misma imagen (flip-scale). Llegan a la conclusión que cuando no se usa data augmentation, el rendimiento logrado es el mismo que cuando se usa esta técnica con la mitad de datos. Esta propiedad se les cumplió fuera de la asínto-

[2] Consiste en entrenar un modelo en una tarea y luego utilizar ese conocimiento previo para mejorar el rendimiento en otra tarea relacionada. En lugar de entrenar un modelo desde cero, el transfer learning permite reutilizar los conocimientos adquiridos en tareas previas y aplicarlos a nuevas tareas.

[3] Base de datos que contiene 1000 categorías de objetos y más de un millón de imágenes.

[4] Es una técnica utilizada para mejorar la capacidad de generalización de un modelo y aumentar la cantidad de datos de entrenamiento, generando datos adicionales a partir de un dataset. Consiste en realizar transformaciones a las imágenes como cambios en la orientación, tamaño, brillo, contraste, color de las imágenes, rotación, cambio de escala, entre otros.

ta de rendimiento de la red[5]. Esta asíntota la lograron con alrededor de 730 imágenes en el caso de las manzanas. Por lo tanto, el uso de data augmentation es realmente importante cuando se tienen pocos datos de entrenamiento.

	Faster R-CNN (VGG)	Faster R-CNN (ZF)	Red pixel-wise
Entrenamiento	90min	Sin datos	3hs
Detección	0,13s	0,04s	2-3s
F1 score (manzanas)	0,908	0,892	0,861

Tabla 3.1: Comparativa de tiempo de entrenamiento, de detección y F1 score de las redes utilizadas en el artículo *Deep Fruit Detection in Orchards* (Bargoti y Underwood, 2017). Elaboración propia en base a los resultados presentados en el artículo.

En el artículo presentan resultados de usar Faster R-CNN con dos redes diferentes de extracción de features (VGG y ZF) y además comparan con una red pixel-wise (clasificación a nivel de pixel). Observando la tabla 3.1, se puede notar que Faster R-CNN es considerablemente más rápida de entrenar que una red pixel-wise. Respecto a los tiempos de detección, Faster R-CNN es casi 10 veces más rápida en el caso de VGG y casi 100 veces más rápida con ZF. La métrica F1-score también supera a la red pixel-wise. Para las otras frutas también fue Faster R-CNN la mejor y pixel-wise la peor. Comparando entre las redes Faster R-CNN, VGG dio mejores resultados que ZF aunque con mayor tiempo de detección.

En el trabajo *A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5* (Yan, 2021) adaptan y mejoran YOLOv5 para reconocer manzanas y clasificarlas según si pueden ser recolectadas o no. Por ejemplo, una manzana que está ocluida por una hoja puede ser recolectada, no así con una manzana obstaculizada por una rama. Mencionan que no hay trabajos previos que traten esta problemática. De todas formas, hacen un relevamiento de los resultados de otros trabajos en detección de manzanas en árboles que se encuentra en la tabla 3.2, que servirá como referencia del estado del arte.

En el grupo MINA del InCo, se destaca el trabajo de *Reconocimiento y conteo de manzanas* (Garderes-Gutiérrez, 2023), que tiene un objeto de estudio similar

[5]Es el punto a partir del cual aumentar la cantidad de datos no mejora la performance.

Modelo	Precisión	Recall	F1-score	mAP
YOLOv3 mejorado	0.97	0.90	-	87.71
LedNet	0.85	0.82	0.83	82.60
Faster R-CNN mejorado	0.897	0.899	0.898	94.8
Mask R-CNN	0.85	0.90	0.88	-
Faster R-CNN (VGG)	-	-	-	89.3

Tabla 3.2: Relevamiento realizado en (Yan, 2021) de rendimiento de modelos en detección de manzanas.

al presente proyecto, pero enfocado hacia el conteo de manzanas, analizando diferentes métodos de ajuste para mejorar la precisión del número de manzanas en un video. En dicho trabajo, se analizó el funcionamiento de YOLO y de Faster R-CNN para la detección de manzanas. Se evaluó YOLOv8 frente a YOLOv5, encontrando mejores resultados con la versión más reciente. Por lo tanto, se optó por trabajar con YOLOv8 en el presente trabajo.

3.1.5. Datasets utilizados

Los conjuntos de datos para la detección de objetos se componen de imágenes junto con metadatos asociados. Estos metadatos contienen información sobre cada objeto presente en las imágenes, incluyendo las coordenadas del cuadro delimitador que encierra el objeto (llamado bounding box). También puede contener la clase del objeto (qué tipo de objeto es). Para representar el bounding box, se pueden emplear las coordenadas de los dos vértices opuestos del rectángulo. Otra forma de representarlo consiste en utilizar las coordenadas del vértice superior izquierdo, junto con el ancho y la altura del rectángulo. Por lo general en los datasets hay un archivo de metadatos por imagen y este archivo suele estar almacenado en formato txt o xml.

Los datasets utilizados para detección fueron los siguientes:

- **Dataset1**: Ujjalfinal (*ujjalfinal Dataset*, 2023). Fue obtenido del sitio Roboflow (*Roboflow*, 2023), que es una página donde se encuentran datasets opensource. Consiste de 4199 imágenes en el conjunto de entrenamiento de manzanas en contextos muy variados (ver figura 3.1). Los conjuntos de desarrollo y test tienen 2399 y 1211 imágenes respectivamente. Este dataset al ser de imágenes variadas puede generar que los modelos entrenados con el mismo, funcionen mejor en ambientes complejos y que se genere una mejor representación de lo que es una manzana, al haberlas observado en entornos muy diversos. Además, las manzanas del dataset están clasificadas como sanas y enfermas, por lo que también puede ser usado como dataset de calidad. Roboflow permite al descargar un dataset, elegir el formato deseado de metadatos. Se eligió usar el formato YOLOv8, el cual consiste de archivos txt en el que en cada línea se especifica la clase del objeto y las coordenadas normalizadas (entre 0 y 1) de dos puntos opuestos que delimitan el bounding box.

- **Dataset2**: Deteksi Buah (*deteksi-buah Dataset*, 2023). Este dataset también fue extraído de Roboflow. El dataset está compuesto por 807 imágenes de manzanas en el conjunto entrenamiento, 75 en el conjunto de desarrollo y 40 en test. Las imágenes ya fueron pasadas por data augmentation, ya que de cada una hay varias versiones con aumento del ruido, giro, cambio de iluminación. Las manzanas se encuentran situadas en una bandeja, por lo que no se trata de escenarios complejos ni en entornos exteriores, sino que el fondo es invariante entre las imágenes. En la figura 3.2 se pueden ver un par de ejemplos de imágenes del dataset. El dataset tiene además imágenes de peras, bananas y carambolas.

Figura 3.1: Ejemplos de imágenes del Dataset1 (*ujjalfinal Dataset*, 2023)

Si bien se encontraron más datasets, los anteriormente mencionados fueron los que se terminaron utilizando. A la hora de seleccionarlos, se tuvo en cuenta el tamaño (se intentó utilizar datasets con más de 1000 imágenes) y el tipo de imágenes que tuviera (cada uno tiene alguna particularidad por la cual le encontramos utilidad). Ambos datasets tienen el mismo formato de metadatos (YOLOv8), lo cual simplifica el uso de los mismos ya que no se debió desarrollar código específico para utilizar cada dataset.

3.2. Calidad

La tarea de determinar la calidad de una manzana también se puede lograr mediante redes convolucionales ya que se trata esencialmente de una tarea de clasificación de imágenes (puede ser en sanas o enfermas, clasificar según una puntuación de calidad, o determinando qué enfermedad tienen). Esta clasificación puede ser en la enfermedad que tenga la manzana, o simplemente una clasificación binaria según si la manzana está sana o enferma, lo cual es el foco de este trabajo ya que desde el INIA la clasificación binaria era algo requerido y suficiente.

En los trabajos relevados que se presentan a continuación, distintas arquitecturas son propuestas junto con los resultados obtenidos. Además, en varios de ellos se comentan técnicas interesantes para aumentar los datos de entrenamiento, teniendo en cuenta que es una tarea difícil encontrar datasets grandes de manzanas sanas y enfermas.

De forma paralela a estos métodos mencionados con redes convolucionales,

Figura 3.2: Ejemplos de imágenes del Dataset2 (*deteksi-buah Dataset*, 2023)

han existido y se siguen desarrollando métodos que utilizan imágenes hiper o multiespectrales (sección 3.2.3). Este tipo de imágenes contienen información de luz en diferentes longitudes de onda y se capturan con cámaras específicas para este propósito. Esta información adicional a la que proporcionan las cámaras RGB, que son las más comunes, puede servir para detectar defectos más tempranamente, si es que se detectan en longitudes de onda que el ser humano no puede ver.

3.2.1. CLIP

CLIP (Contrastive Language-Image Pre-Training) es un modelo multimodal[6] presentado por OpenAI en enero de 2021 en *Learning Transferable Visual Models From Natural Language Supervision* (Radford, Kim, y Hallacy, 2021), que combina conocimientos de conceptos del lenguaje inglés y conocimientos semánticos sobre imágenes para clasificar imágenes en texto.

Como fue explicado en la sección 3.1.3, el problema que presenta SAM es que nada más segmenta objetos sin clasificar qué objeto es cada uno. En el artículo *Segment Anything* (Kirillov y cols., 2023) se comenta que utilizaron CLIP para realizar la clasificación de los objetos usando texto. En este trabajo se integraron SAM y CLIP para evaluar estos modelos en la detección de manzanas. SAM utilizado para detectar objetos y CLIP para filtrar cuáles de ellos son manzanas. También se testeó CLIP para clasificación de calidad de manzanas.

CLIP fue entrenado utilizando una arquitectura de red neuronal que combina un modelo de lenguaje natural (similar a GPT) con un modelo de visión por computadora (similar a ResNet). Ambos componentes fueron pre-entrenados con 400M de pares (imagen, texto) para aprender representaciones generales de ambas modalidades. Luego, estos componentes se unieron para formar un modelo conjunto capaz de realizar tareas específicas, como clasificación de imágenes o recuperación de imágenes basadas en consultas de texto. La mayoría de modelos de aprendizaje automático aprenden una tarea específica, sin embargo, CLIP tiene la capacidad de ser zero-shot al igual que SAM.

El aprendizaje por lenguaje natural tiene una gran ventaja sobre la mayoría de enfoques de aprendizaje no-supervisado o auto-supervisado y es que no solo aprende la representación sino que también conecta esa representación con lenguaje, lo que favorece la capacidad de zero-shot (Radford y cols., 2021). También se menciona en el artículo de CLIP que los modelos zero-shot son mucho más robustos que los correspondientes modelos ImageNet supervisados ya que su rendimiento no decae al cambiar de dataset, lo que sugiere que la evaluación de zero-shot en modelos task-agnostics (agnósticos de las tareas) es mucho más representativa de la capacidad de un modelo.

Para que las imágenes y los textos puedan estar conectados unos con otros, estos deben estar encajados (embedded). El modelo está compuesto por dos submodelos llamados encoders, uno para los textos y otro para las imágenes. Luego

[6] Un modelo multimodal puede procesar e interpretar múltiples modalidades, como texto e imágenes.

de convertidos los textos e imágenes a vectores o matrices, estas se comparan a través de la medida similitud coseno, para obtener qué tan parecidos son los dos vectores (ver figura 3.3).

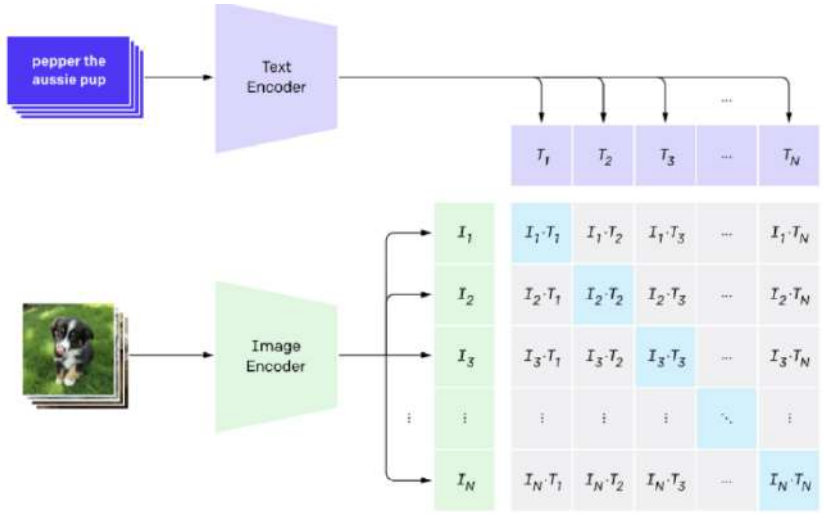

Figura 3.3: Esquema de funcionamiento de CLIP. La idea es buscar el texto que maximice la similitud coseno en los cuadrados azules (N pares), que representan las combinaciones donde el texto y la imagen coinciden, y que minimice la medida en los cuadrados grises ($N^2 - N$), que es donde el texto e imagen no coinciden. Extraída del artículo *Learning Transferable Visual Models From Natural Language Supervision* (Radford y cols., 2021).

CLIP tiene dos usos principales: dado una imagen como entrada, predice el pie de foto (caption) o el resumen (summary) más probable de la imagen; y el segundo, dado un texto (prompt) o una lista de textos predice la probabilidad de que el/los texto(s) matcheen con la imagen.

El artículo también expresa las limitaciones o puntos débiles de CLIP, los cuales son:

- A pesar de que CLIP se desempeña bien en reconocer objetos comunes, falla en tareas sistemáticas o más abstractas tales como contar objetos en una imagen, y en tareas más complejas como por ejemplo, predecir la distancia a la que está el auto más cercano en la imagen.

- Además, a CLIP le complica la clasificación de granularidad fina como reconocer las diferencias entre modelos de autos o aviones, o entre especies de flores.

En la figura 3.4 se puede ver cómo CLIP sostiene la exactitud por encima del 60 % llegando casi al 90 % con datasets muy variados, que si bien todos tienen

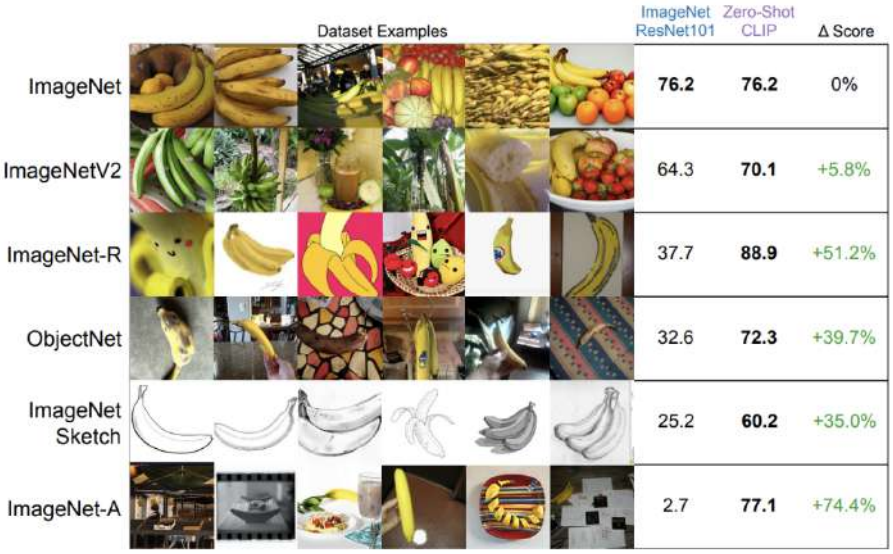

Figura 3.4: Comparativa de exactitud entre CLIP y ResNet101 (entrenado con ImageNet) para distintos datasets. Extraído de (Radford y cols., 2021)

bananas, las muestran de distintas formas (verdes, cortadas, caricaturizadas, dibujadas). Por otro lado, el rendimiento de ResNet101 decae estrepitosamente a medida que el dataset contiene imágenes más diferentes respecto a ImageNet, llegando incluso al 2.7 %. Cabe destacar que CLIP iguala a ResNet101 en ImageNet, que es el dataset de entrenamiento de ResNet101 y para el que debería funcionar mejor. Esta comparativa muestra la robustez de CLIP y que está a la altura de redes entrenadas específicamente para un tipo de imagen.

3.2.2. Vision Transformer

En el artículo *An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale* (Alexey Dosovitskiy, 2021) se introdujo la arquitectura Vision Transformer, que es la adaptación de la arquitectura Transformer (Ashish Vaswani, 2017) a visión por computadora. Transformer conforma el estado del arte en Procesamiento de Lenguaje Natural (PLN) (Alexey Dosovitskiy, 2021).

Transformer sigue la arquitectura encoder-decoder, tomando como entrada todas las palabras de un texto. Como novedoso para PLN, con Transformer se puede procesar todo un texto en paralelo. Previamente, con el uso de redes recurrentes, para procesar una palabra de un texto, se precisaba el vector de contexto resultante de procesar todas las palabras anteriores. En Transformer, al procesar cada palabra con el encoder, se utiliza la propia palabra junto con el resto de palabras del texto, asignándoles determinada atención a cada una en función de un score que se calcula entre palabras. El mecanismo atencional refiere a que ciertas palabras tendrán más peso e incidencia que otras en el

procesamiento. Como este cálculo de scores no es más que productos de matrices y no se tiene un vector de contexto que se va propagando con el procesamiento de palabras, se pueden procesar todas las entradas en paralelo. La arquitectura del Transformer es mucho más compleja que esto, ya que este mecanismo atencional junto con capas de normalización y una red completamente conectada componen un bloque de Transformer. Tanto el encoder como el decoder están conformados por varios de estos bloques apilados. Esta arquitectura está diagramada en la figura 3.5.

Si se abstrae la idea básica detrás del Transformer, ésta se basa en procesar una secuencia de entradas ordenadas. En el caso de PLN son palabras, y para visión de computadora, son regiones de la imagen. En Vision Transformer, las imágenes son divididas en regiones de 16x16 píxeles, las cuales se linealizan a vectores para luego ser la secuencia de entrada del encoder del Transformer (ver figura 3.6). Estas regiones se tratan de igual manera que las palabras en Transformer aplicado a PLN.

Cuando se preentrena Vision Transformer con grandes cantidades de datos y se transfiere a múltiples benchmarks como ImageNet, CIFAR-100, VTAB, se logran excelentes resultados en comparación con las redes convolucionales de última generación, además de que requiere sustancialmente menos recursos computacionales para el entrenamiento (Alexey Dosovitskiy, 2021). En la figura 3.7 se puede ver que Vision Transformer supera el rendimiento de los modelos convolucionales del estado del arte en esos datasets al momento de la realización del artículo.

3.2.3. Imágenes hiperespectrales y multiespectrales

Hasta ahora veníamos analizando métodos de visión por computadora, los cuales se basan en el uso de imágenes RGB. Este tipo de imágenes son capturadas por cámaras que tienen tres filtros centrados en las longitudes de onda del color rojo, del verde y del azul para obtener imágenes similares a cómo ve el ojo humano. Estas longitudes de onda no son las mejores para detectar aspectos de calidad en manzanas como moretones o descomposición de manera temprana (Zhang, Gu, y cols., 2018). Para obtener información de diferentes longitudes de onda se utilizan las cámaras hiperespectrales, las cuales pueden adquirir un conjunto de imágenes monocromáticas de casi cientos de miles de longitudes de onda continuas. Una imagen hiperespectral se puede ver como un stack de imágenes de dos dimensiones de longitudes de onda casi continuas (Zhang, Liu, y cols., 2018).

La principal ventaja de las imágenes hiperespectrales es la cantidad de información que contienen. Los defectos de las frutas pueden ser muy claros en una sola imagen monocromática o ser más fáciles de reconocer en varias imágenes monocromáticas o combinando imágenes. Sin embargo, tienen una clara desventaja que es el tiempo empleado en procesar esa cantidad de imágenes (Zhang, Liu, y cols., 2018). Esto hace que sea imposible usar imágenes hiperespectrales en aplicaciones de tiempo real. Por eso es que para este tipo de uso las imágenes multiespectrales son ideales, debido a que estas extraen información de una

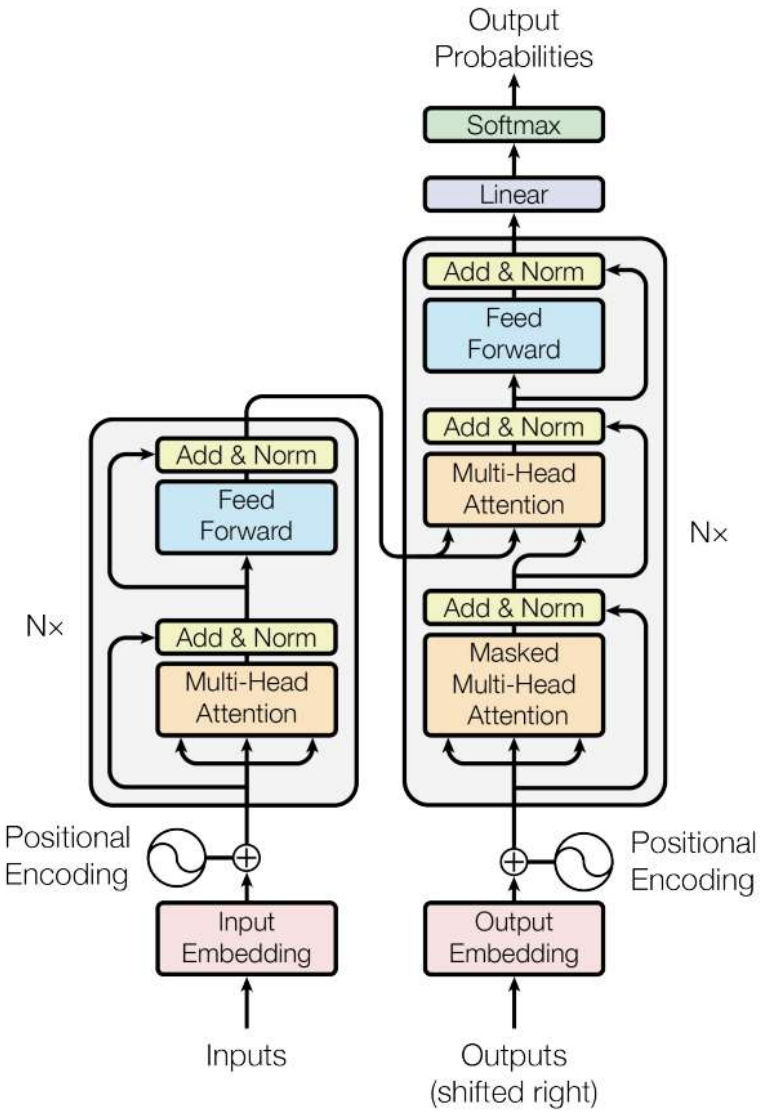

Figura 3.5: Arquitectura del modelo Transformer. Extraído de (Ashish Vaswani, 2017)

cantidad seleccionada de longitudes de onda.

En la introducción del artículo *From hyperspectral imaging to multispectral imaging: Portability and stability of HIS-MIS algorithms for common defect detection* (Zhang, Liu, y cols., 2018) mencionan las diferentes técnicas que hay para trabajar con imágenes hiper y multiespectrales. PCA es uno de los algo-

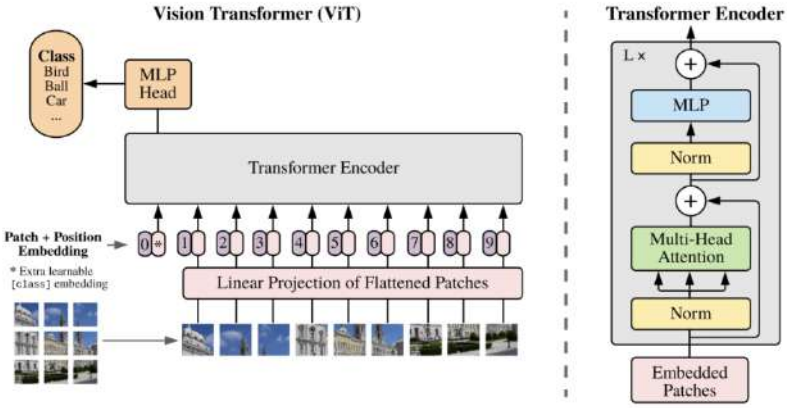

Figura 3.6: Arquitectura del modelo Vision Transformer. Extraído de (Ashish Vaswani, 2017)

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Figura 3.7: Comparativa de exactitud entre tres modelos de Vision Transformer preentrenados con el datset JFT y dos modelos de redes convolucionales del estado del arte: Big Transfer (basado en ResNet) y Noisy Student (basado en EfficientNet). En el artículo de Vision Transformer, afirman que Noisy Student es el estado del arte en ImageNet y BiT-L en los otros conjuntos de datos utilizados en la comparativa. Extraído de (Ashish Vaswani, 2017).

ritmos más utilizados. Se encarga de reducir el conjunto de imágenes de tal forma que el subconjunto generado sea el más representativo para determinar la clasificación que se quiera hacer. Por lo general se utiliza para encontrar las longitudes de onda más representativas y usar esas para clasificar. A veces se acompaña PCA con MNF (Minimum Noise Fraction), que es una transformación más compleja que contiene dos transformaciones de PCA en cascada de la imagen hiperespectral cruda. Para el análisis de las imágenes hiper y multiespectrales se utilizan muchos métodos matemáticos de banda como la adición, multiplicación de bandas entre dos imágenes de diferentes longitudes de onda.

Estos métodos ayudan a reducir fluctuaciones por condiciones de luz desparejas, por superficies curvas o variación de color.

En el artículo *On line detection of defective apples using computer vision system combined with deep learning methods* (Fan y cols., 2020) se mencionan peores resultados de estos métodos en frutas con cáscara de dos colores como ocurre con algunas variedades de manzanas. También mencionan que los sistemas de imágenes multiespectrales son muy costosos, complicados y voluminosos. La calidad de imágenes obtenidas depende del sistema beam splitting[7] utilizado. Imágenes de mala calidad pueden hacer complicada la tarea de distinguir defectos de tallos y cálices (stem y calyx). Por estas razones, es que hay pocos sistemas de este tipo comercializados. Sin embargo, mencionan que este método tiene sus méritos en la identificación temprana de contusiones y deterioro en la fruta.

En *Challenges and solutions of optical-based nondestructive quality inspection for robotic fruit and vegetable grading systems: A technical review* (Zhang, Gu, y cols., 2018) se analizan los retos a los que se enfrentan los métodos que hacen uso de imágenes hiper o multiespectrales. Los retos identificados son:

- influencia de la variabilidad física y biológica. Esto refiere a que la luz reflectada no depende solamente de la luz ambiente sino de la forma del objeto. En superficies curvas, la luz no se refleja de forma uniforme y esto incide en las imágenes capturadas. Particularidades del ambiente donde se hacen las plantaciones, la estación del año, la madurez de la fruta por mencionar algunos, son factores biológicos que alteran la espectroscopía de las frutas y hacen menos robustos a este tipo de métodos.
- detección de toda la superficie
- diferenciación entre defecto y tallo o cáliz
- detección de defectos no obvios (se refiere a no visibles al ojo humano como la fase temprana de podredumbre o de un moretón)
- robustez de los algoritmos y características
- rapidez de los sistemas.

3.2.4. Trabajos relacionados

En el artículo *Diagnosis of Typical Apple Diseases: A Deep Learning Method Based on Multi-Scale Dense Classification Network* (Tian y Li, 2021) se proponen dos arquitecturas llamadas Multi-scale Dense Inception-V4 y Multi-scale Dense Inception-Resnet-V2 para clasificar manzanas. En estas arquitecturas se usan módulos de Inception v4 e Inception Resnet v2. Los módulos de Inception son arquitecturas ya definidas de capas convolucionales que se diferencian por poner varios filtros convolucionales de distinto tamaño en el mismo nivel

[7] Proceso de dividir un haz de luz en dos o más haces separados. Puede lograrse mediante el uso de diferentes técnicas, como prismas, placas de vidrio o películas delgadas.

de profundidad y combinar los resultados en vez de apilar filtros. Esto hace que sean capas más anchas y menos profundas (*Deep Learning: Understanding The Inception Module.*, 2020). La primera versión fue publicada en el artículo *Going deeper with convolutions* (Christian Szegedy, 2014), elaborado en su mayoría por empleados de Google. La única diferencia entre las dos arquitecturas que propone el artículo está en que una usa módulos residuales[8] de Inception mientras que la otra no.

En las arquitecturas definidas se usan las salidas de varias de las capas anteriores como entrada de algunas capas. La idea con esto es facilitar el reuso de las features de capas anteriores para que no queden "olvidadas" en una red profunda. A esto se le llama multiescala porque algunas capas tienen como entrada features de diferentes dimensiones ya que provienen de capas distintas.

La red propuesta clasifica hojas y manzanas con un total de 11 clasificaciones incluyendo matices de gravedad para el mismo defecto o enfermedad. Algunas de estas clasificaciones son: antracnosis general, antracnosis severa, sarna general de manzana, sarna severa de manzana, mancha gris en manzana.

Afirman que las características de las imágenes con enfermedades y sin enfermedades son muy parecidas y eso hace que la distinción entre diferentes grados de la misma enfermedad sea una tarea complicada. Por esta razón vieron necesario mejorar el rendimiento de las redes existentes creando redes nuevas.

Para las imágenes de manzanas, utilizaron un dataset creado por ellos mismos. Usaron la técnica de Cycle-GAN para aumentar los datos. Cycle-GAN sigue la arquitectura encoder-decoder explicada en la sección 2.2. Esta red aprende las características de manzanas sanas y enfermas para generar nuevas imágenes de manzanas enfermas y luego con un discriminador define si la imagen generada constituye una manzana sana o defectuosa. De esta forma se obtienen nuevas imágenes creadas y clasificadas artificialmente.

Los resultados obtenidos se pueden ver en la tabla 3.2.4. Se puede ver que el rendimiento en cuanto a precisión, exactitud y F1 score es levemente mejor en la implementación residual, incluso con tiempo de entrenamiento menor, aunque con un tamaño mayor del modelo.

Además, compararon la arquitectura con otras redes. Los resultados se pueden ver en la figura 3.8. Las implementaciones de la arquitectura mejoran el rendimiento de las redes Inception. También superan a AlexNet y a VGG.

En el artículo *Apple quality identification and classification by image processing based on convolutional neural networks* (Y. Li y cols., 2021), se plantean como principal objetivo estudiar modelos para la clasificación rápida y precisa de la calidad de las manzanas. En este artículo se proponen tres modelos para la clasificación de la calidad de las manzanas en tres clases: premium, middle y poor.

Los modelos propuestos son: un método tradicional clásico con un clasificador SVM basado en extracción de características con GLCM y HOG, el segundo,

[8] Las redes neuronales residuales contienen como entrada de algunas capas la suma entre la salida de la capa anterior y la salida de capas anteriores. Estas redes logran mejores resultados en redes profundas ya abordan el problema del desvanecimiento del gradiente que se da en redes muy profundas durante el entrenamiento. (*ResNet: Residual Neural Networks*, 2023)

Model	Accuracy (%)	Precision (%)
Inception-V4	92.72	92.61
Inception-ResNet-V2	93.37	92.65
Multi-scale Dense Inception-V4	94.31	94.36
Multi-scale Dense Inception-ResNet-V2	94.74	94.71
LeNet	85.21	83.66
AlexNet	92.04	91.75
VGG	92.45	92.32
Densenet-121	93.68	93.66

Figura 3.8: Comparativa de las arquitecturas propuestas en *Diagnosis of Typical Apple Diseases: A Deep Learning Method Based on Multi-Scale Dense Classification Network* (Tian y Li, 2021) con otras redes. Extraída del artículo.

	Arquitectura residual	Arquitectura no residual
Exactitud	94.74 %	94.31 %
Precisión	94.71 %	94.36 %
F1-score	94.86 %	94.32 %
Tamaño del modelo	255M	201M
Tiempo de entrenamiento	2.11h	2.58h

Tabla 3.3: Resultados obtenidos en *Diagnosis of Typical Apple Diseases: A Deep Learning Method Based on Multi-Scale Dense Classification Network* (Tian y Li, 2021). Elaboración propia en base a los resultados presentados en el artículo.

el modelo Google Inception v3 pre-entrenado para clasificación y detección de objetos y el último, un modelo propio basado en CNN.

El problema del tamaño insuficiente del conjunto de datos de entrenamiento lo resolvieron utilizando técnicas de data augmentation y con el fin de reducir el tiempo de las técnicas, re-escalaron las imágenes al 25 % de su tamaño original. Las técnicas aplicadas fueron: el aumento de Salt Noise y Pepper Noise[9], flips, rotaciones, brillo y oscuridad.

En su experimento, capturan cuatro imágenes de cada manzana desde diferentes ángulos para poder ver los diferentes lados de las manzanas. Luego, estas imágenes son ingresadas en el modelo entrenado que predice y devuelve una puntuación para cada categoría de calidad. La categoría con mayor puntuación es considerada como el resultado predicho. Capturaron 3600 imágenes con resolución 3120 x 4160. Luego de hacer data augmentation, llegaron a las 36.000

[9]Se refiere a una amplia variedad de procesos que resultan en la misma degradación básica de la imagen: solo unos pocos píxeles son ruidosos, pero son muy ruidosos. El efecto es similar a espolvorear puntos blancos y negros, sal y pimienta, sobre la imagen(*Pepper Noise*, 2012).

imágenes. En la figura 3.10 se pueden ver ejemplos de las imágenes capturadas de todas las categorías.

La red propuesta en el artículo tiene 6 capas convolucionales y 6 de pooling y luego le siguen 2 capas completamente conectadas y la salida que es una función softmax. La red está presentada en la figura 3.9. Es una red convolucional bastante simple, sin elementos arquitectónicos o módulos complejos.

Layer	Operation	Kernel shape	Number of Kernel	Stride	Number of neurons	Number of training parameters	Number of connections
Conv1	Convolution	5×5	8	1	–	208	8,998,912
	Pooling	3×3	–	2	–	24	865,280
Conv2	Convolution	3×3	16	1	–	1168	12,633,088
	Pooling	3×3	–	2	–	48	432,640
Conv3	Convolution	1×1	32	1	–	1168	1,470,976
Conv4	Convolution	3×3	64	1	–	18,496	50,013,184
	Pooling	3×3	–	2	–	192	432,640
Conv5	Convolution	3×3	64	1	–	46,460	31,204,160
	Pooling	3×3	–	2	–	240	115,200
Conv6	Convolution	3×3	64	1	–	466,144	6,644,736
	Pooling	3×3	–	2	–	192	23,040
Fc1	Full connection	–	–	–	256	590,080	590,080
Fc2	Full connection	–	–	–	256	65,792	65,792
Output	Softmax	–	–	–	3	–	–

Figura 3.9: Red convolucional propuesta en *Apple quality identification and classification by image processing based on convolutional neural networks* (Y. Li y cols., 2021). Extraída del artículo.

Modelos	**Clásico (SVM + HOG/GLCM)**	**Google Inception v3**	**CNN propuesto**
Entrenamiento	287 min	51 min	27 min
Exactitud (entrenamiento)	-	92 %	99 %
Exactitud (validación)	78.14 %	91.2 %	98.98 %
Exactitud (testing)	77.67 %	91.33 %	95.33 %

Tabla 3.4: Comparativa del rendimiento del método clásico usado, la red Google Inception v3 y la red propuesta según tiempo de entrenamiento y exactitud en conjunto de entrenamiento, validación y testing. Elaboración propia en base a los resultados presentados en el artículo *Apple quality identification and classification by image processing based on convolutional neural networks* (Y. Li y cols., 2021).

Se puede ver en la tabla 3.4 que el modelo CNN propuesto es considerablemente mejor en todos los aspectos, en tiempo de entrenamiento y en exactitud

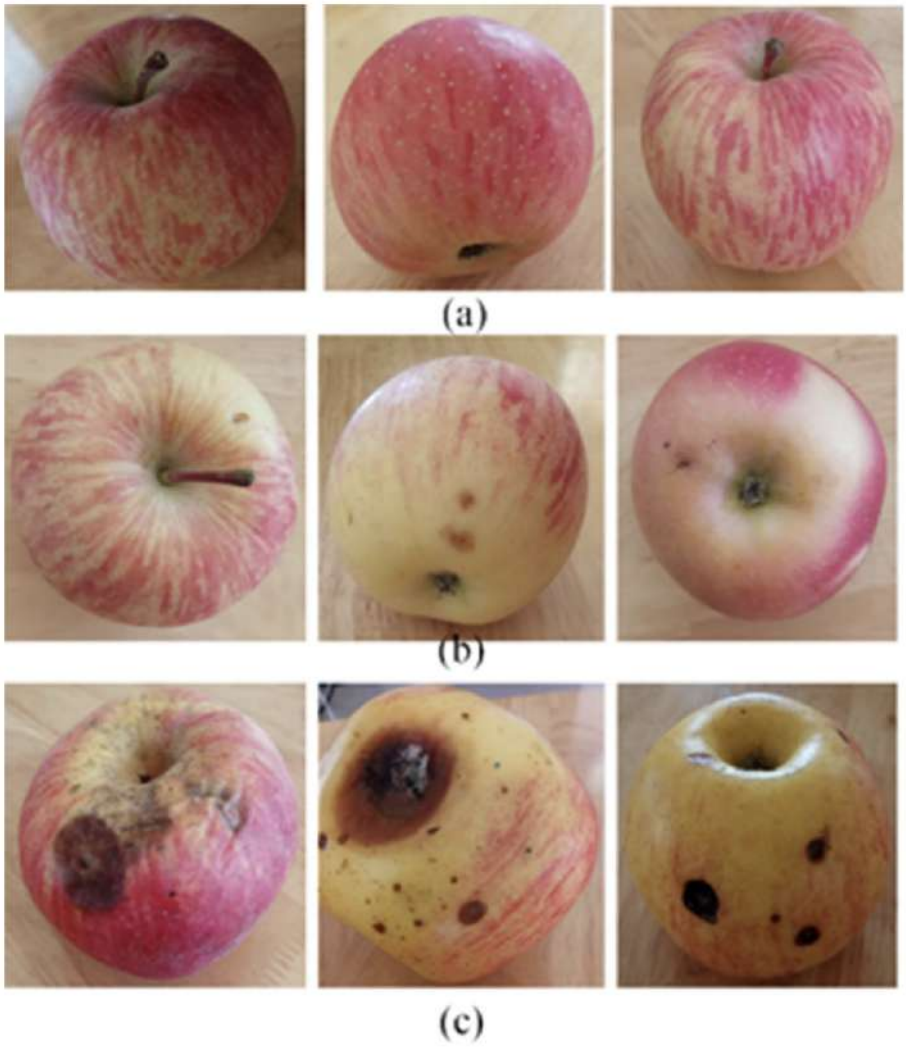

Figura 3.10: Ejemplos de imágenes de manzana de las categorías premium (a), middle (b) y poor (c). Extraída del artículo *Apple quality identification and classification by image processing based on convolutional neural networks* (Y. Li y cols., 2021).

en todos los conjuntos de datos. La exactitud del modelo propuesto es realmente muy alta. En el artículo analizado antes, los resultados eran algo inferiores aunque la tarea del otro artículo era mucho más compleja porque los modelos propuestos clasifican la enfermedad de la manzana y de las hojas, incluso a la enfermedad la clasifica como general o severa. El hecho de tener muchas más clasificaciones posibles hace que sea más difícil obtener métricas elevadas.

Otro elemento a tener en cuenta de los resultados de este artículo, es que las imágenes de las manzanas con las que trabajaron son en primer plano, con la manzana ocupando casi la totalidad de la imagen (ver figura 3.10). Este rendimiento sin dudas no se trasladará a imágenes en campo, donde hay muchas manzanas por imagen y de cada manzana se disponen de pocos píxeles de información.

En la imagen 3.11 se presenta una gráfica donde se comparan distintos modelos de redes convolucionales existentes según la exactitud (eje y), cantidad de operaciones (eje x) y memoria que ocupa el modelo (tamaño del círculo). Se puede ver que la familia de modelos VGG consumen mucha memoria, tienen baja eficiencia y exactitud media. La red NASNet-A-Large es la que mejor exactitud posee pero la que más operaciones requiere, es decir que tiene muy baja eficiencia. Por otro lado, la red AlexNet es la que tiene menor exactitud pero es la que requiere menos cantidad de operaciones para ser entrenada. Con una exactitud bastante alta, eficiencia y uso de memoria promedio se ubican las redes Inception y ResNet que ya fueron mencionadas.

3.2.5. Datasets utilizados

Los datasets de calidad de manzanas están compuestos por imágenes de manzanas etiquetadas. La clasificación es a nivel de imagen (ya que se presenta una sola manzana por imagen) y por lo general están clasificadas como manzana sana o enferma. También puede haber otro tipo de clasificaciones como en distintos grados de calidad, o según la enfermedad de la manzana. Se buscaron datasets que etiqueten a las manzanas como sanas o enfermas, ya que así se definió que se iba a clasificar la calidad en el alcance del proyecto. Estos datasets no tienen metadatos asociados a cada imagen, sino que las imágenes de cada categoría están agrupadas en carpetas.

Se utilizaron dos datasets de calidad, que son los siguientes:

- **Dataset3**: Fruits fresh and rotten for classification (Reddy, 2018). Fue obtenido del sitio Kaggle (*Kaggle*, 2023), que es una comunidad de ciencias de datos donde se pueden encontrar datasets. El conjunto de datos tiene 1694 imágenes de entrenamiento de manzanas sanas y 2343 manzanas enfermas. Para el conjunto de test hay 396 sanas y 602 enfermas. Además, el dataset contiene imágenes de bananas y naranjas tanto sanas como enfermas. Las imágenes son de una manzana en primer plano y con fondo blanco (ver figura 3.12). El dataset ya tiene data augmentation, ya que hay imágenes rotadas, trasladadas y con aumento de salt y pepper noise.

- **Dataset4**: Apple Detection Dataset (Olafenwa, 2019). Fue obtenido de un repositorio de Github. Si bien también es un dataset de detección como indica el nombre, las imágenes tienen clasificación de calidad también. El conjunto de datos tiene 294 imágenes de entrenamiento de manzanas sanas y 269 manzanas enfermas. En el conjunto de test hay 74 sanas y 74 enfermas. Las imágenes son de manzanas en diferentes contextos (ver

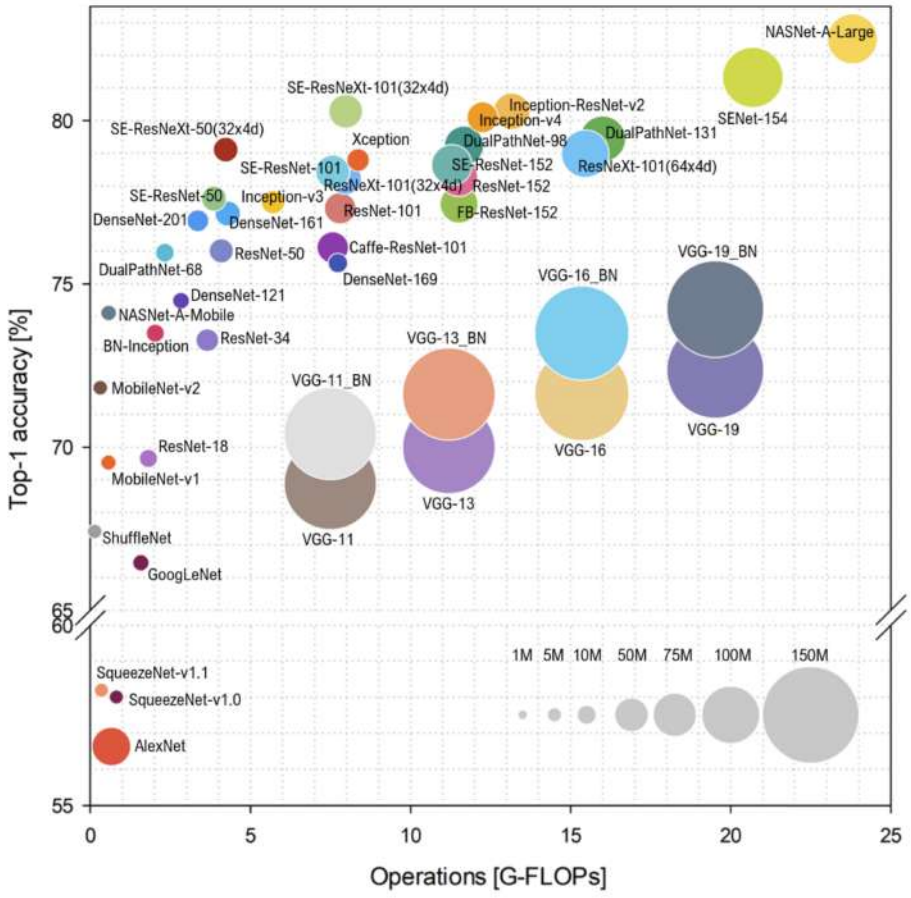

Figura 3.11: Comparativa de modelos de redes convolucionales según cantidad de operaciones, memoria utilizada y exactitud. Extraída de (Ghimire y cols., 2022).

figura 3.13). Hay varias imágenes con más de una manzana pero son todas del mismo tipo.

3.3. Tracking

Los algoritmos de tracking más utilizados, son los que siguen el paradigma de tracking by detection. En este contexto, se introducirá a continuación el algoritmo SORT como una primera aproximación dentro de esta familia de algoritmos, seguido por versiones sucesivas que han demostrado obtener mejores resultados y representan el estado del arte en la materia.

Figura 3.12: Ejemplos de imágenes del Dataset3 (Reddy, 2018)

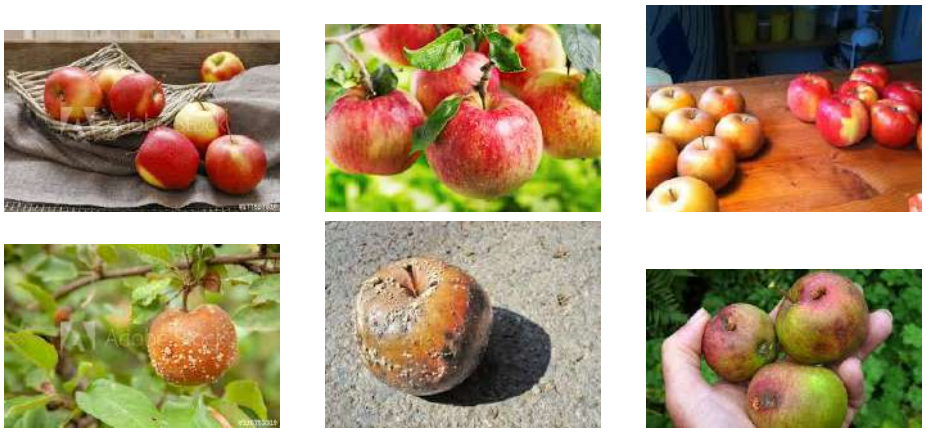

Figura 3.13: Ejemplos de imágenes del Dataset4 (Olafenwa, 2019)

3.3.1. SORT

Simple Online And Realtime Tracking (SORT) (Alex Bewley, 2016) es una implementación simple y eficiente de un algoritmo con el enfoque tracking by detection. No tiene en cuenta características de apariencia más allá del componente de detección. SORT utiliza la posición y el tamaño de los bounding boxes tanto para la estimación del movimiento como para la asociación de datos a lo largo de los frames (*Top 5 Object Tracking Methods*, 2021). Es un algoritmo publicado en el año 2016 y ha servido como punto de referencia para los que le siguieron.

Este algoritmo utiliza filtros de Kalman para predecir movimientos. Los filtros de Kalman son una herramienta poderosa para estimar y predecir estados de un sistema (en este caso posición y velocidad de objetos) en presencia de incertidumbre (detecciones inexactas), y se utiliza ampliamente como componente fundamental en aplicaciones como el seguimiento de objetivos, la navegación y el control (*KalmanFilter.net*, s.f.).

Luego de realizar predicción de movimientos, SORT computa la intersección dividido la unión (IoU) entre los bounding boxes predichos por los filtros de Kalman (de los objetos conocidos previamente) y los bounding boxes observados en el frame actual. Usando el algoritmo húngaro se hace la asignación de detecciones a objetos. El algoritmo húngaro se utiliza para encontrar la asignación más eficiente en términos de costos entre dos conjuntos de elementos, donde cada elemento de un conjunto se debe asignar a exactamente un elemento del otro conjunto, y viceversa (*Algoritmo húngaro*, 2023).

3.3.2. DeepSORT

DeepSORT (Nicolai Wojke, 2016) es una extensión del algoritmo SORT que integra redes neuronales para mejorar la precisión del seguimiento. Se extraen características visuales de los objetos detectados con una red convolucional. Es uno de los primeros algoritmos en utilizar deep learning para tracking (Yunhao Du, 2022). En DeepSORT, la función de similitud utilizada para la asociación (en los costos del algoritmo húngaro) se calcula utilizando tanto la información de las características de apariencia como las características de movimiento. La información de movimiento sirve además para filtrar asociaciones improbables por la posición de los objetos entre un frame y el anterior. Usa un banco de features que consta de almacenar los features de los últimos 100 frames de cada objeto. DeepSORT al utilizar la apariencia visual de los objetos reduce significativamente el problema de re-identificación.

3.3.3. BoT-SORT

En el artículo *BoT-SORT: Robust Associations Multi-Pedestrian Tracking* (Nir Aharon, 2022) presentan el algoritmo BoT-SORT como una extensión con mejoras en las asociaciones de los objetos trackeados. En él, establecen que las técnicas basadas en IoU tales como SORT dependen unívocamente de la calidad

de los bounding boxes de las predicciones. En muchos escenarios complejos las predicciones no son precisas debido al movimiento de la cámara, lo que provoca bajo solapamiento entre bounding boxes del mismo objeto y por lo tanto, baja performance del tracker. En el artículo resuelven este problema utilizando Image Registration (Malek, 2019) para estimar el movimiento de la cámara y así ajustar el filtro de Kalman. A esa técnica la llamaron Compensación del Movimiento de la Cámara (en inglés CMC, Camera Motion Compensation).

En definitiva, las mayores contribuciones del artículo son: las mejoras basadas en la compensación de los movimientos de la cámara y el ajuste de los filtros de Kalman para mejorar la localización de los bounding boxes; y un método para lograr asociaciones más robustas entre detecciones y tracklets fusionando el IoU y la re-identificación por distancia coseno.

3.3.4. StrongSORT

StrongSORT es presentado en el artículo del 2022 titulado *StrongSORT: Make DeepSORT Great Again* (Yunhao Du, 2022) y, como sugiere el título, es un trabajo en el que actualizan DeepSORT, mejorándolo desde las perspectivas de la detección de objetos, las features extraídas y la asociación de trayectorias.

Los autores seleccionaron al algoritmo DeepSORT para actualizar debido a su simplicidad, capacidad de expansión y eficacia. Afirman que DeepSORT tiene un rendimiento inferior en comparación con los métodos del estado del arte debido a sus técnicas desactualizadas, y no por el paradigma de seguimiento. Además, se comenta que en los últimos años, la tarea de tracking está siendo dominada por el paradigma tracking by detection.

En StrongSORT, se reemplaza Faster R-CNN (usado en DeepSORT) por YOLO-X (un algoritmo de la familia de YOLO) para la detección. El extractor de features en DeepSORT era una red convolucional simple, la cual es reemplazada por BoT, una red que extrae características más discriminativas. Otra diferencia está en que en StrongSORT se elimina el banco de features y se sutituye por el uso de un único vector de features por objeto que se va actualizando mediante un mecanismo definido en el artículo.

Proponen dos técnicas que afirman que son fáciles de agregar a cualquier otro tracker, siendo ambas ligeras en procesamiento y mejoran considerablemente los resultados. Estas técnicas se ejecutan de manera offline, es decir, luego de procesar toda la secuencia de imágenes.

Una técnica está diseñada para resolver el problema de re-identificación. Muchos trackers de manera offline intentan asociar objetos diferentes entre sí con información global y usando información de apariencia. El método que proponen en el artículo se llama AFLink (Appearance Free Link) que solo usa información espacio temporal para determinar si dos objetos en realidad deberían tener el mismo identificador. Con esto se logra un buen equilibrio entre exactitud y eficiencia computacional. Esta técnica consiste de una red convolucional que recibe dos conjuntos de detecciones y determina si corresponden al mismo objeto o no.

La otra técnica propuesta se llama Gaussian-Smoothed Interpolation (GSI) y se emplea para resolver el problema llamado missing detection: pensar que algo es parte del fondo cuando en realidad es objeto.

A StrongSort con GSI y AFLink le llaman StrongSort++. StrongSort++ logra resultados del estado del arte en varios benchmarks como MOT-Challenge[10] como se puede ver en la figura 3.14.

Method	Ref.	HOTA(↑)	IDF1(↑)	MOTA(↑)	AssA(↑)	DetA(↑)	IDs(↓)	FPS(↑)
SORT [3]	ICIP2016	34.0	39.8	43.1	31.8	37.0	4,852	**143.3**
DAN [51]	TPAMI2019	39.3	49.5	52.4	36.3	43.1	8,431	6.3
TPM [39]	PR2020	41.5	52.6	54.2	40.9	42.5	1,824	0.8
DeepMOT [65]	CVPR2020	42.4	53.8	53.7	42.7	42.5	1,947	4.9
Tracktor++ [1]	ICCV2019	44.8	55.1	56.3	45.1	44.9	1,987	1.5
TubeTK [37]	CVPR2020	48.0	58.6	63.0	45.1	51.4	4,137	3.0
ArTIST [45]	CVPR2021	48.9	59.7	62.3	48.3	50.0	2,062	4.5
MPNTrack [6]	CVPR2020	49.0	61.7	58.8	51.1	47.3	**1,185**	6.5
CenterTrack [77]	ECCV2020	52.2	64.7	67.8	51.0	53.8	3,039	3.8
TransTrack [50]	arxiv2021	54.1	63.5	75.2	47.9	61.6	3,603	59.2
TransCenter [64]	arxiv2021	54.5	62.2	73.2	49.7	60.1	4,614	1.0
GSDT [59]	ICRA2021	55.5	68.7	66.2	54.8	56.4	3,318	4.9
PermaTrack [54]	ICCV2021	55.5	68.9	73.8	53.1	58.5	3,699	11.9
MAT [19]	NC2022	56.0	69.2	67.1	57.2	55.1	1,279	11.5
CSTrack [30]	arxiv2020	59.3	72.6	74.9	57.9	61.1	3,567	15.8
FairMOT [74]	IJCV2021	59.3	72.3	73.7	58.0	60.9	3,303	25.9
ReMOT [67]	IVC2021	59.7	72.0	77.0	57.1	62.8	2,853	1.8
CrowdTrack [48]	AVSS2021	60.3	73.6	75.6	59.3	61.5	2,544	140.8
CorrTracker [57]	CVPR2021	60.7	73.6	76.5	58.9	62.9	3,369	15.6
RelationTrack [68]	arxiv2021	61.0	74.7	73.8	61.5	60.6	1,374	8.5
TransMOT [9]	arxiv2021	61.7	75.1	76.7	59.9	63.7	2,346	1.1
GRTU [58]	ICCV2021	62.0	75.0	74.9	62.1	62.1	1,812	3.6
MAATrack [49]	WACVw2022	62.0	75.9	79.4	60.2	64.2	1,452	**189.1**
ByteTrack [73]	arxiv2021	63.1	77.3	**80.3**	62.0	**64.5**	2,196	29.6
DeepSORT* [62]	ICIP2017	61.2	74.5	78.0	59.7	63.1	1,821	13.8
StrongSORT	ours	63.5	78.5	78.3	63.7	63.6	1,446	7.5
StrongSORT+	ours	**63.7**	**79.0**	78.3	**64.1**	63.6	1,401	7.4
StrongSORT++	ours	**64.4**	**79.5**	**79.6**	**64.4**	**64.6**	**1,194**	7.1

Figura 3.14: Comparativa de algoritmos del estado del arte de tracking. Para cada métrica se indica con azul y rojo los mejores resultados. Extraída de (Yunhao Du, 2022). En el Anexo A se encuentran las definiciones de las métricas de tracking.

3.3.5. DeepOCSORT

El artículo *Deep OC-SORT: Multi-Pedestrian Tracking by Adaptive Re-Identification* (Maggiolino, 2023) publicado en febrero del 2023, presenta mejoras con respecto a los algoritmos OCSORT y DeepSORT, basados plenamente en el movimiento de los objetos. Entre las mejoras se encuentra el aprovechamiento de las aparien-

[10] MOT-Challenge es un benchmark para tracking, el cual estandariza el formato de datos para almacenar datasets de tracking. Hay una gran cantidad de datatsets que siguen este estándar e incluso herramientas que calculan todas las métricas de tracking si se sigue este formato(*Multiple Object Tracking Benchmark*, 2014).

cias visuales de los objetos para ser utilizadas junto al análisis del movimiento de los objetos a seguir.

DeepOC-SORT fue creado a partir del algoritmo basado en filtros de Kalman, OC-SORT, y alcanzó el primer y segundo puesto en los benchmarks MOT17 y MOT20 respectivamente, convirtiéndose en un método del estado del arte. Las mejoras las logran a partir de la implementación de tres módulos: CMC (al igual que en el algoritmo BoT-SORT), Dynamic Appearance (DA) y Adaptive Weighting (AW). La técnica de apariencia dinámica consiste en modificar los embeddings visuales de los elementos calculados frame a frame solo en situaciones de "alta calidad", en donde la confianza de detección de los objetos en cuestión es alta (mayor que un cierto umbral). La técnica Adaptive Weighting se basa en ajustar los pesos de los features de apariencias de los objetos, utilizando la similaridad coseno entre los objetos detectados y los trackeados anteriormente.

3.3.6. HybridSORT

En agosto del 2023 se presentó el artículo *Hybrid-SORT: Weak Cues Matter for Online Multi-Object Tracking* (Yang, 2023) en donde exponen el algoritmo del estado del arte, HybridSORT. Este algoritmo sigue el paradigma de la familia SORT que utiliza los filtros de Kalman para la estimación del movimiento de los tracklets y la re-identificación. La tarea de asociación también es resuelta por el algoritmo húngaro, pero en el artículo agregan tres nuevas características que mejoran considerablemente la performance en los benchmarks MOT17, MOT20 y DanceTrack.

La mayoría de los métodos de tracking utilizan pistas (cues) fuertes como información espacial y de apariencia para resolver la asociación entre tracklets. Sin embargo, cuando los objetos son ocluidos o se forman clusters de objetos (muchos objetos pegados) ese tipo de información es ambigua debido a la alta superposición de los objetos. En el artículo se demuestra que este desafío puede ser resuelto incorporando pistas débiles como lo son la confianza de las detecciones, el estado de la altura y la dirección de la velocidad de los objetos. Estas pistas débiles compensan y complementan a las pistas fuertes en los casos de oclusión y clustering. Esto lo hacen extendiendo el filtro de Kalman para que utilice la confianza, la altura y la componente de velocidad para cada objeto.

El estado de altura refleja en cierta medida información de profundidad de los objetos. En muchos escenarios de tracking, la altura de los objetos depende solamente de qué tan cerca o lejos están de la cámara, lo que hace que el estado de altura sea una medida efectiva para distinguir objetos muy solapados. Además, el estado de altura es invariante a las distintas posiciones de los objetos y los giros que puedan realizar, por lo que conforma un estado de estimación preciso para trackear objetos.

Este algoritmo fue testeado con muy buenos resultados, superando a trackers del estado del arte (OC-SORT, DeepSORT, ByteTrack) por 10 unidades en la métrica HOTA y 4 unidades en IDF1, en el benchmark DanceTrack, donde ocurren muchas interacciones y oclusiones severas en movimientos de los objetos

muy complejos.

3.3.7. Norfair

Norfair (*Norfair*, 2020) es una librería liviana de Python que agrega seguimiento de objetos en tiempo real a cualquier detector. El método de seguimiento utiliza filtros de Kalman, vectores de apariencias, soporta movimiento de la cámara y re-identificación basada en embeddings de apariencia. La librería puede integrarse fácilmente con ROS ya que cuenta con un paquete ROS y ambiente para ejecutarlo en Docker.

3.3.8. Datasets utilizados

Los datasets de tracking están compuestos por un conjunto de imágenes secuenciales (posiblemente obtenidas de un video) y un archivo de texto que en cada línea tiene: número de imagen, identificador de objeto, bounding box del objeto. El identificador del objeto es lo que permite asociar objetos en imágenes distintas.

Los dos datasets utilizados de tracking fueron:

- **Dataset5**: (*Apple orchard production estimation using deep learning strategies: a comparison of tracking-by-detection algorithms*, 2022). Este dataset continene 1802 imágenes de manzanas en árboles (ver figura 3.15).
- **Dataset6**: Reconocimiento y conteo de manzanas (Garderes-Gutiérrez, 2023). Este dataset fue elaborado en otro proyecto de grado de la Facultad de Ingeniería. Las imágenes fueron obtenidas del campo del INIA, llamado Las Brujas. Son en realidad dos datasets etiquetados a partir de dos videos distintos de alrededor de 25 segundos de duración cada uno (ver figura 3.16). Se los distinguirán como **Dataset6.1** y **Dataset6.2**.

Figura 3.15: Ejemplos de imágenes del Dataset5 (*Apple orchard production estimation using deep learning strategies: a comparison of tracking-by-detection algorithms*, 2022).

Figura 3.16: Ejemplos de imágenes del Dataset6 (Garderes-Gutiérrez, 2023).

Capítulo 4

Parte Central

El proyecto se puede descomponer en dos etapas. La primera consistió en hacer una búsqueda de algoritmos de detección, seguimiento y clasificación de código abierto y aplicarlos al área de estudio para evaluar su funcionamiento. En lugar de utilizar algoritmos diseñados específicamente para manzanas, se optó por probar algoritmos que habían demostrado un buen desempeño en otras tareas. Esta decisión fue tomada debido a que los artículos consultados que abordan este problema, no disponibilizaron el código ni los modelos entrenados. Los algoritmos utilizados, fueron testeados para evaluar si su rendimiento en manzanas tiene margen de mejora o si están cerca del estado del arte en agricultura de precisión. Se puede decir que los resultados que se fueron obteniendo iban marcando el rumbo a seguir.

En el caso de los algoritmos que alegan ser zero-shot como SAM y CLIP, se utilizaron los modelos tal cual se encuentran publicados, de igual forma que para los algoritmos de tracking, mientras que para el resto de algoritmos no generalistas (como YOLO, Faster R-CNN, VisionTransformer), se efectuó un fine tuning[1] para perfeccionar su funcionamiento en manzanas. El proceso de fine tuning consistió de entrenar, usando los datasets de manzanas, modelos ya preentrenados con millones de datos de diverso tipo para ajustar el modelo al área de estudio.

Se decidió utilizar YOLOv8 y Faster R-CNN en sus versiones base y entrenados por nosotros, ya que se observó en el relevamiento que es lo más usado y lo que brinda mejores resultados. Además se quiso testear SAM y CLIP, dos modelos recientemente surgidos muy prometedores que alegaban ser zero-shot, para ver si estos nuevos modelos que empezaron a surgir de grandes empresas cambiaban el estado del arte. Para el análisis de calidad se optó por utilizar CLIP y VisionTransformer, siendo este último parte del estado del arte en clasificación de imágenes y además no se encontró su uso en este contexto. También se quiso probar el rendimiento de modelos de tracking para ser usados en video, que es lo que un robot captura.

[1] Cuando a un modelo preentrenado se lo ajusta o adapta a un conjunto de datos específico o a una tarea particular

La segunda etapa del proyecto consistió del diseño e implementación de un sistema que integre de manera genérica a los algoritmos de detección, clasificación de calidad y de seguimiento, de tal forma que no esté acoplado a ningún algoritmo en particular, y pueda ser extendible al uso de nuevos algoritmos que sean implementados. El código del pipeline se encuentra disponible en el repositorio (*Proyecto de grado - Pipeline de detección y seguimiento de manzanas para geolocalización de anomalías*, 2023), donde se explica cómo se configura y se pone en funcionamiento. Además, el repositorio contiene los modelos entrenados y los scripts utilizados para el entrenamiento.

4.1. Diseño del sistema

Para llevar a cabo el pipeline que integra los diferentes tipos de algoritmos, se diseñaron tres módulos abstractos: uno para integrar un detector al sistema, uno para integrar un clasificador y otro para integrar un tracker (que realiza el seguimiento de los objetos). Los módulos no son más que interfaces. Para hacer uso de un algoritmo nuevo, no se necesita más que hacer una implementación de la interfaz correspondiente al tipo de algoritmo que se quiera integrar, que dicha implementación utilice al algoritmo que se está integrando al sistema y luego hacer que el programa principal del pipeline use esa implementación nueva.

Cada interfaz tiene un conjunto de funciones a implementar, las cuales tienen parámetros de entrada y salida tipados, para que no haya acoplamiento con el tipo de datos que devuelva un algoritmo en particular. En la figura 4.1, se pueden ver las interfaces y los tipos de datos definidos.

El sistema cada vez que recibe una imagen, invoca al módulo de detección para obtener los bounding boxes de las manzanas de la imagen para luego pasárselas al módulo de tracking, el cual va almacenando la información de cada manzana, junto con todas las imágenes de las mismas en el sistema de archivos. Esto se hace para que se puedan posprocesar las imágenes de las manzanas para clasificarlas de manera offline en otro momento. Si se quiere procesar la calidad en tiempo real y no de forma offline, el sistema clasifica cada manzana detectada en el frame y almacena los resultados. Además, el programa registra el número de frame que se procesó junto con la marca de tiempo en la que se recibió, para que se realice el mapeo de ubicación geográfica que se explicará en la sección 4.2. En la figura 4.2 se puede visualizar cómo funciona el pipeline y cómo interactúan los distintos componentes.

De lo anterior se desprende que el sistema tiene dos modos de funcionamiento:

- Clasificación de manzanas en tiempo real: En cada frame, se clasifican todas las manzanas detectadas y se almacena el resultado de la clasificación. Cada manzana tiene entonces una lista de clasificaciones correspondiente a todas las veces que la manzana apareció en un frame del video y por lo tanto fue clasificada. A partir del conjunto de clasificaciones de cada manzana se determina si cada una de ellas está sana o no. Luego se entrará en detalle sobre cómo se determina si una manzana está sana o no en función

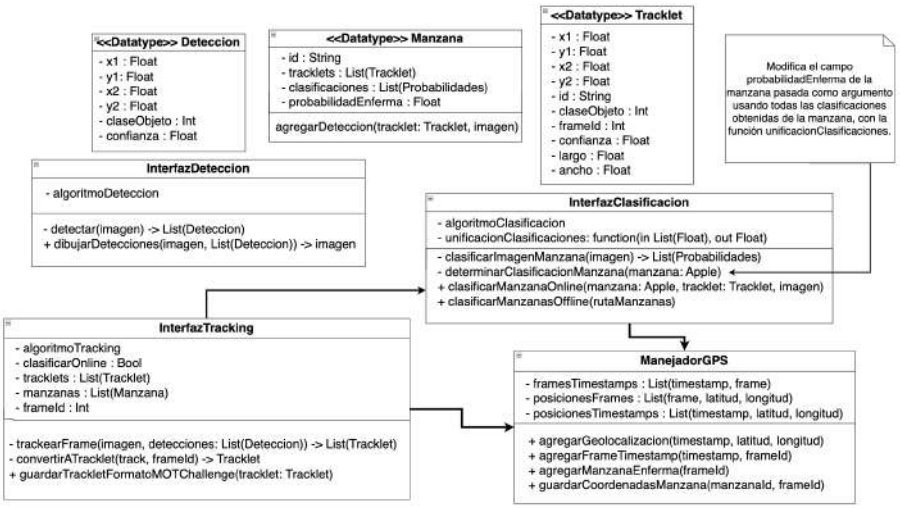

Figura 4.1: Diagrama de interfaces del sistema. Los métodos marcados con (-) son métodos abstractos, es decir, requieren una implementación específica en cada instancia de la interfaz. Por otro lado, los métodos marcados con (+) ya están implementados en la interfaz, lo que significa que no dependen del algoritmo utilizado.

de un conjunto de clasificaciones. Esta modalidad hace que no se requiera de posprocesamiento de las manzanas detectadas pero reduce la performance del sistema ya que se deben clasificar manzanas al procesar cada frame en vez de hacerse en otro momento. Como ventaja, este enfoque no requiere tener todas las imágenes de todas las manzanas simultáneamente almacenadas en disco.

- Clasificación de manzanas offline: Durante el procesamiento del video se almacenan en el sistema de archivos todas las imágenes capturadas de cada manzana y no se realiza ninguna clasificación. Luego de procesado el video, se debe invocar un programa desarrollado que ejecuta un método definido en la interfaz de los clasificadores, el cual toma de cada manzana todas las imágenes almacenadas en el sistema de archivos y las clasifica con el método del clasificador que lleva a cabo esto. El problema que tiene este enfoque, es que el consumo de memoria del disco es alto, ya que se tienen almacenadas todas las imágenes vistas de cada manzana. Dependiendo del hardware utilizado y las características del campo, el sistema podría quedarse sin memoria muy rápidamente.

Para clasificar una manzana a partir de un conjunto de imágenes de la misma se tienen dos enfoques:

- Clasificar una a una las imágenes y a las probabilidades resultantes de ser sanas o no, aplicarles una operación, como puede ser calcular el promedio

o el máximo. El uso del máximo sirve para quedarse con la máxima probabilidad de que esté enferma, ya que puede pasar que solamente desde un ángulo se pueda observar un defecto de la manzana. También puede ocurrir que tomando el máximo se obtengan clasificaciones ruidosas, dejándose de lado todo el resto de clasificaciones. Otra opción es usar algún sistema de votación en el que se ponderen de alguna forma cada clasificación de la manzana (por ejemplo, según la confianza que se tenga de estar viendo a la manzana en cada frame).

- Utilizar una red neuronal secuencial. Este tipo de redes neuronales recibe un conjunto de entradas de tamaño arbitrario y va almacenando un estado interno que condensa la información vista en las entradas anteriores y luego clasifica en base al estado interno. Un ejemplo de este tipo de red es LSTM (Hochreiter, 1997). También las arquitecturas encoder-decoder (como la red Transformer, vista en la sección 3.2.2) son ideales para este caso ya que generan un vector de contexto al aplicarle el encoder a cada imagen y luego al aplicar el decoder usan estos vectores donde se tiene la información de cada imagen. Utilizando este enfoque, en vez de almacenar un conjunto de probabilidades para cada manzana, se almacenaría un conjunto de vectores o estados internos. La desventaja que tiene este enfoque es que requiere entrenar una red neuronal utilizando como datos de entrenamiento, un dataset que contenga varias imágenes distintas para cada manzana. Un dataset de este tipo no se tiene a disposición aunque con el pipeline desarrollado en este trabajo se puede generar, ya que el pipeline a medida que procesa imágenes almacena en el sistema de archivos todas las imágenes de cada manzana trackeada.

En este trabajo se optó por implementar el primer enfoque, ya que es el más sencillo y no requiere entrenar una red con un dataset que habría que generar. Los métodos de combinación de clasificaciones que se seleccionaron fueron el promedio de clasificaciones y la moda. La moda funciona como un sistema de votación, donde la clasificación más frecuente determina la clasificación final para la manzana. Se descartó el uso del máximo por ser muy susceptible a clasificaciones ruidosas.

4.2. Integración con información geográfica

Este sistema también ubica la posición de cada manzana detectada geográficamente. El sistema realiza un mapeo entre posición geográfica y manzana, para que, al terminar de clasificar todas las manzanas, se genere un mapa de calor similar al de la figura 4.3. El mapa de calor tiene zonas de diferentes colores, como verde, amarillo, naranja y rojo, para representar diferentes niveles de densidad de manzanas enfermas detectadas. Las áreas coloreadas en tonos más intensos, como el rojo, indican una mayor concentración de manzanas enfermas, mientras que las áreas en tonos más suaves, como el verde, sugieren una menor presencia de manzanas enfermas.

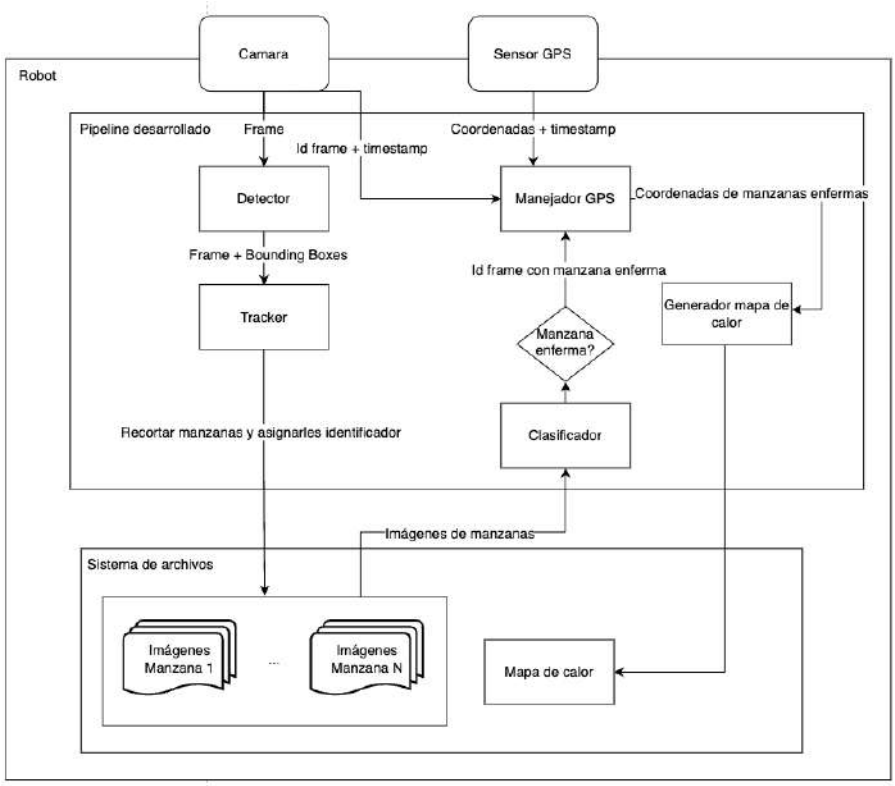

Figura 4.2: Diagrama de funcionamiento y componentes del sistema

Para llevar a cabo esto, se implementó una clase cuya función es manejar la información geográfica, de tal forma que esto no quede acoplado a los manejadores de algoritmos de aprendizaje automático y este manejo sea transparente a ellos.

La vinculación entre posición geográfica y manzana se hace a partir de un timestamp, es decir, de una marca de tiempo. Al utilizar el entorno ROS, los mensajes publicados tienen un timestamp en el cabezal. Por lo tanto, se dispone del timestamp de cada posición geográfica del robot así como del de cada imagen capturada por el robot. Es fundamental sincronizar ambas fuentes de información a través de una marca de tiempo, dado que el sistema no procesa las imágenes de manera instantánea (esto depende de los algoritmos empleados y su tiempo de procesamiento). Por lo tanto, utilizar la última posición obtenida no resulta conveniente ya que en ese caso, habría una baja precisión en la ubicación de las manzanas.

Para realizar esta sincronización, el manejador de información geográfica expone una función para recibir datos geográficos junto con el timestamp en el que se obtuvieron y otra función en el que recibe el número de imagen y el

timestamp en el que se obtuvo. Esta última función se encarga de buscar el timestamp más cercano de los datos geográficos para vincular la imagen a una ubicación.

Este manejador también dispone de una función para almacenar la ubicación geográfica de una manzana, indicando el identificador de la manzana y el primer número de imagen en el que aparece. Una alternativa a esto es utilizar el promedio de los números de imágenes donde aparece cada manzana en vez del primero, ya que este número tendería a ser el número de imagen en cual el robot ve a la manzana en el medio de la imagen, es decir, que está frente a ella. Esto es una posible mejora al sistema para que las posiciones estimadas sean más precisas. Se optó por utilizar la primera imagen por simplicidad.

La información de la ubicación de todas las manzanas se almacena en el sistema de archivos. Esto permite que si se realiza la clasificación offline, se pueda saber la ubicación de cada manzana para generar el mapa de calor.

Además, el manejador tiene una función para indicar en qué frame se observó una manzana enferma. Esta función guarda en un archivo CSV[2] las ubicaciones geográficas correspondientes a todas las manzanas enfermas detectadas. Para generar el mapa de calor a partir de estas coordenadas, se utilizó la librería gmplot (*gmplot*, s.f.) que permite, entre otras cosas, hacer mapas de calor utilizando los mapas de Google Maps.

Notar que la posición de cada manzana se estima usando la posición del robot en el momento que la comenzó a ver (alternativamente, en la posición media en la que la vio) sin hacer una transformación de dicha posición. Para obtener una mejor aproximación, se debería usar información de profundidad ya sea estimando la distancia del robot a cada manzana detectada o utilizando algún sensor que permita saberla. Con la distancia del robot a cada manzana y la orientación del robot, se puede hacer una transformación de la posición geográfica del robot para tener mejores aproximaciones en la ubicación de las manzanas.

4.3. Algoritmos utilizados

Para todos los algoritmos que se presentan en esta sección, se desarrolló una implementación de la interfaz correspondiente (según el tipo de algoritmo) para poder integrarlo al pipeline.

4.3.1. Detección

Los algoritmos de detección utilizados fueron **YOLOv8**, **Faster R-CNN** y **SAM junto con CLIP**. Se decidió utilizar YOLO y Faster R-CNN por ser los algoritmos más utilizados en el área de estudio y para poder compararlos entre

[2]Un archivo csv (Comma-Separated Values) es un tipo de archivo de texto plano que se utiliza para almacenar datos tabulares en forma de texto, donde cada línea corresponde a una fila de datos y los valores de cada columna están separados por un delimitador, comúnmente una coma.

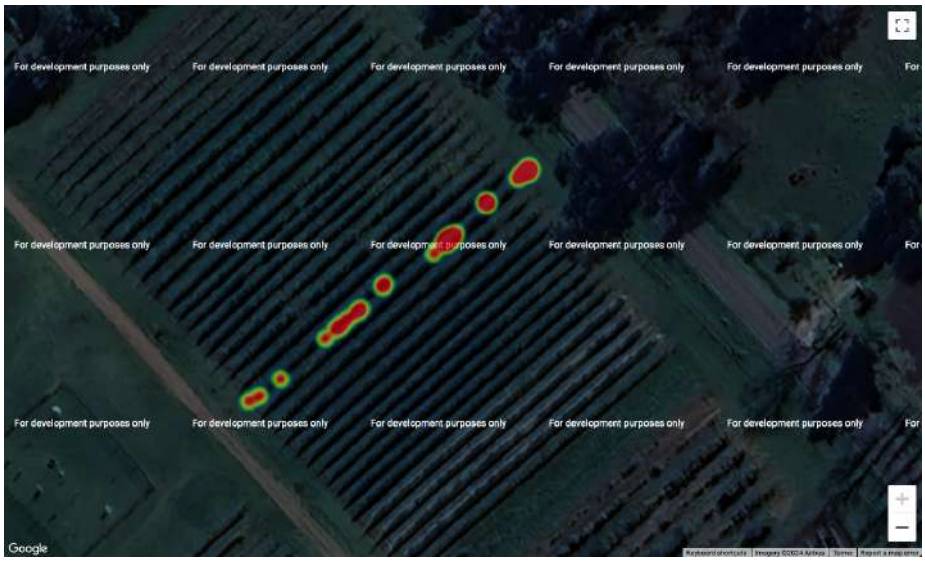

Figura 4.3: Ejemplo de mapa de calor generado por el sistema.

sí. SAM fue utilizado por ser un algoritmo novedoso, prometedor y recientemente publicado al momento de la investigación. Tanto YOLOv8 como Faster R-CNN se testearon sin entrenar (con las versiones base que vienen con los algoritmos ya preentrenados con imágenes de todo tipo) y también luego de entrenar con datasets de manzanas. A SAM y a CLIP no se los entrenó ya que estos algoritmos alegan ser zero-shot y no ofrecer la posibilidad de ser entrenados.

Para entrenar YOLO se utilizó la funcionalidad de la librería del algoritmo (*Ultralytics*, 2024) que permite entrenar utilizando una versión como base (para hacer finetuning sin entrenar de cero), e indicando el dataset a utilizar como entrenamiento y metaparámetros mediante un archivo de configuración. En general, procuramos mantener los metaparámetros por defecto en su mayoría, dado su número y la cantidad de las combinaciones posibles, evitando así posibles efectos inesperados que podrían surgir de ajustarlos simultáneamente. En YOLO, nos enfocamos en modificar únicamente el número de épocas y el umbral de IoU, ya que los consideramos como los más significativos. Se testearon distintas configuraciones para ambos datasets de detección. Los mejores resultados se lograron con tres épocas[3], ya que con más notamos que sobreajustaba al conjunto de entrenamiento y los resultados decaían en el resto de datasets. Empleamos el modelo base yolov8n, que tiene poca cantidad de parámetros en comparación con otros modelos de YOLO, pero esto permitió un entrenamiento más rápido, evaluación de imágenes más rápida y logrando buenos resultados. En el repositorio del proyecto (*Proyecto de grado - Pipeline de detección y seguimiento de manzanas para geolocalización de anomalías*, 2023) se encuentran

[3]Cantidad de iteraciones sobre el dataset de entrenamiento

los scripts de entrenamiento con todos los metaparámetros utilizados.

Para entrenar Faster R-CNN se utilizó la librería Detectron2 (*Detectron2*, 2024) de Meta, junto con el modelo base "faster_rcnn_X_101_32x8d_FPN_3x". Se optó por ajustar la tasa de aprendizaje (utilizamos 0.001) y el número de imágenes por batch (utilizamos 4), dejando el resto de metaparámetros por defecto.

Se realizó una integración entre SAM y CLIP siguiendo la idea presentada en el artículo de SAM (Kirillov y cols., 2023), en el que mencionan que para poder clasificar las máscaras generadas por SAM utilizaron CLIP, que dada una imagen y un conjunto de clasificaciones en forma de texto, devuelve la probabilidad para cada clasificación de que la imagen sea mejor descrita por esa clasificación. Esta integración de SAM y CLIP no fue liberada por los desarrolladores de SAM y fue lo que se implementó y testeó. Para llevar a cabo esto, a cada imagen a analizar se la pasa por SAM para que este algoritmo la segmente y detecte objetos. Se utilizaron los bounding boxes en vez de la segmentación realizada por SAM para mejorar la performance y utilizar menos memoria. Luego se recorta la imagen con los bounding boxes y a cada subimagen se la pasa por CLIP, el cual clasifica si el objeto es una manzana o no. Los labels utilizados en CLIP para que determine si se trata de una manzana o no fueron: "apple", "no apple". Se testearon variantes de estos labels y se observó que no tenía mayor impacto en el resultado final.

4.3.2. Clasificación

Los algoritmos de clasificación de manzanas utilizados fueron **CLIP** y **Vision Transformer**. La elección de estos modelos se debió a que no se constató su previo uso en esta área. CLIP es un algoritmo muy prometedor y se quiso indagar su rendimiento en manzanas. En el caso de Vision Transformer, es un algoritmo del estado del arte capaz de lograr muy buenos resultados (ver en ranking (*State of the Art - Image Classification on ImageNet*, 2024) que Vision Transformer aparece varias veces en diversas variantes) y por eso se seleccionó.

Los labels utilizados en CLIP para que determine si se trata de una manzana sana o enferma fueron: "a healthy apple", "a rotten apple". Como CLIP es zero-shot no se lo entrenó.

Por otro lado, se entrenó a Vision Transformer con un dataset de calidad de manzanas utilizando la librería transformers de Python (*Pip - Transformers*, 2024). Realizamos el fine-tuning siguiendo el ejemplo que se encuentra en el sitio web de HuggingFace (*Fine-Tune ViT for Image Classification with Transformers*, 2024). Los metaparámetros ajustados fueron las épocas (se utilizaron 4) y la tasa de aprendizaje (se utilizó 0.0002).

4.3.3. Seguimiento

Los algoritmos de seguimiento de objetos utilizados fueron **StrongSORT**, **HybridSORT**, **DeepOCSORT**, **BoT-SORT** y **Norfair**. Los primeros cuatro trackers fueron utilizados mediante la librería BoxMot (*BoxMOT: pluggable*

SOTA tracking modules for segmentation, object detection and pose estimation models, 2020) que los tiene integrados. Se seleccionaron por ser algoritmos del estado del arte en tracking y además por su fácil instalación y uso que ofrece la librería BoxMOT.

4.4. Testeo del sistema

Existen varias opciones para utilizar y evaluar el sistema propuesto. La manera ideal, dado que el sistema se diseñó con este propósito, consiste en ejecutarlo en un robot integrado con ROS, en el que se publica información geográfica y de la cámara que dispone.

Otra alternativa que siempre es posible al utilizar ROS, es utilizar archivos bags grabados a partir de un robot. Este tipo de archivos se pueden generar mientras el robot está operativo, almacenando todos los mensajes publicados para luego, de manera offline y sin necesidad de utilizar el robot, poder reproducir el archivo simulando que se está ejecutando en el robot.

En caso de no contar con un robot disponible o no tener la posibilidad de ir a un campo con manzanas, se desarrolló un nodo de ROS que publica al tópico que se desee, un video en formato mp4. Además, se elaboró un nodo, el cual a partir de un archivo de texto con datos geográficos donde cada línea tiene timestamp, latitud y logitud separados por una coma, los publica en un tópico. De esta forma se puede simular que esta información la están publicando los sensores del robot.

Si no se dispone de ROS o se desea ejecutar el sistema fuera de este entorno para llevar a cabo un posprocesamiento de datos, se puede ejecutar el sistema utilizando otro programa que hace las mismas funciones que el mencionado anteriormente solo que sin utilizar ROS. Este programa lee directamente el video en formato mp4 y un archivo de texto con los datos de geolocalización. Dado que no se utiliza ROS, los frames del video carecen de marcas de tiempo, por lo que se implementó una clase que asigna marcas de tiempo ficticias a cada frame. Para asignar timestamps ficticios a frames, se le asigna al primer frame, el primer timestamp del archivo de geolocalizaciones y luego al resto de frames, se les asigna la suma entre el primer timestamp más el tiempo transcurrido entre el primer frame y el frame que se está procesando. El tiempo transcurrido entre el primer frame y un frame cualquiera se calcula multiplicando el número de frame por el tiempo entre dos frames (es el inverso de los frames por segundo). Esta asignación ficticia de timestamps a frames permite que luego el mapeo de marcas de tiempo a manzanas se pueda realizar.

Capítulo 5

Experimentación

En este capítulo se presentan los resultados obtenidos, tanto para los algoritmos utilizados individualmente, como para el sistema en su conjunto.

Las pruebas fueron llevadas a cabo en una MacBook Pro con procesador M1 y 16GB de memoria RAM.

5.1. Detección

Para testear algoritmos de detección, se desarrolló una clase llamada DetectorEvaluator, que implementa la funcionalidad de recibir una lista de bounding boxes predichas (por el algoritmo de detección a evaluar) y la lista de bounding boxes reales de una imagen (obtenida de los metadatos del dataset) y calcula los verdaderos positivos, falsos positivos y falsos negativos según la metodología descrita en el Anexo B. Esta clase también calcula el IoU (Intersección dividido Unión) entre dos bounding boxes.

Al finalizar de evaluar todas las imágenes del dataset, en la clase DetectorEvaluator hay una función para generar el reporte de las métricas de detección logradas. Las métricas que se extrajeron fueron: tiempo promedio de detección por imagen, precisión, recall, F1-score y Average Precision (AP). En el Anexo A se definen cada una de ellas.

El uso de esta clase sumado al hecho que el sistema devuelve tipos de datos predefinidos independientemente del algoritmo de detección utilizado, permiten que sea muy fácil testear un algoritmo nuevo. No conlleva más que cambiar el módulo de detección utilizado. Por lo tanto, el sistema desarrollado también tiene la funcionalidad de testeo muy simple de algoritmos.

Para calcular precisión, recall y F1-score, se utilizó la librería scikit-learn (*scikit-learn*, 2007) de Python que las calcula a partir de los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Para el cálculo de AP, se utilizó la librería (*Object Detection Metrics*, 2018), a la cual hay que proveerle dos archivos (uno con los datos reales y otro con los predichos) en cierto formato, que la clase DetectorEvaluator los genera.

Se utilizaron los datasets **Dataset1** y **Dataset2** (ver sección 3.1.5) de detección y además se transformaron los datasets de tracking (**Dataset5** y **Dataset6**, sección 3.3.8) a datasets de detección. Esto último se hizo para poder testear la detección con imágenes de manzanas en árboles, que es el tipo de imágenes que el sistema tiene que analizar. Los algoritmos testeados fueron YOLO (con y sin entrenamiento), Faster R-CNN (con y sin entrenamiento) y SAM junto con CLIP (sin entrenar).

Una decisión que hubo que tomar es el umbral a utilizar para determinar si una detección es correcta o no. En caso que el IoU entre una detección y el objeto real supere dicho umbral, se considera la detección como un verdadero positivo, como se explica en el Anexo B. Tomando como referencia el artículo *ImageNet Large Scale Visual Recognition Challenge* (Russakovsky, 2015) donde se describe el benchmark ImageNet, que es uno de los más usados para evaluar y comparar algoritmos de reconocimiento de imágenes y detección de objetos, establecen como umbral 0.5. Sin embargo, mencionan que para objetos de tamaño muy reducido, este umbral puede resultar inadecuado. De hecho, el artículo proporciona una fórmula para calcular umbrales adaptativos en función del tamaño del objeto. Aplicar un umbral estático de 0.5 a objetos pequeños puede conducir a la clasificación errónea de numerosas detecciones como falsos positivos, aunque en realidad puedan ser detecciones válidas. En el artículo *Deep Fruit Detection in Orchards* (Bargoti y Underwood, 2017), que se presentó en la sección 3.1.4 y en el que utilizan imágenes con pocos píxeles por manzana, se utilizó como umbral 0.2. Se menciona que este umbral equivale a una superposición de 58 % en cada eje del objeto, lo cual consideran suficiente para una aplicación de mapeo de frutas. Se decidió entonces utilizar 0.5 como umbral en los datasets con manzanas en primer plano como sugiere el artículo de ImageNet y 0.2 para los datasets con manzanas en campo como sugiere el artículo *Deep Fruit Detection in Orchards* (Bargoti y Underwood, 2017), para poder comparar las métricas obtenidas con las del artículo.

En las tablas 5.1, 5.2, 5.3 y 5.4, se pueden ver los resultados obtenidos por los diferentes algoritmos utilizados y entrenados en cada dataset. En detección, una precisión alta significa que cuando el modelo predice una manzana hay probabilidades altas de que realmente lo sea. Si la precisión es baja, significa que detecta como manzanas muchas cosas que no lo son. Por otro lado, un alto recall indica que el modelo logra detectar la mayoría de las manzanas de manera precisa, mientras que un recall bajo sugiere que pasa por alto y no detecta muchas manzanas que están presentes en las imágenes. Dicho de otra forma, la precisión se enfoca en medir la calidad de las detecciones que hace el modelo, mientras que el recall se enfoca en medir qué tan bien detecta los objetos reales. Como es necesario asociar el recall con la precisión para evaluar su equilibrio, se utilizaron las métricas F1-score y AP.

En lo que refiere a tiempo de detección de los modelos utilizados (tabla 5.5), se puede ver que YOLO es el modelo más rápido por bastante diferencia, tal como se esperaba según lo visto en la revisión de antecedentes ya que este algoritmo en una pasada por una imagen la analiza completamente. Faster R-CNN demora cerca de un segundo por imagen ya que realiza la detección en

varios pasos. El algoritmo desarrollado combinando SAM y CLIP tiene tiempos similares a Faster R-CNN. Se esperaba que SAM y CLIP fuera aún más lento que Faster R-CNN ya que por cada objeto detectado por SAM se clasifica con CLIP y además siendo Faster R-CNN una red ya optimizada y pensada para detectar y clasificar objetos.

Observando la tabla 5.1, donde se pueden ver las métricas de los distintos modelos de detección con el **Dataset1**, podemos ver que YOLO en su versión base tiene una alta precisión aunque recall muy bajo. Percibimos al analizar los resultados que para muchas manzanas, YOLO las confundía de clase con peras o naranjas. Por esto es que intentamos observar el rendimiento de YOLO sin utilizar la clasificación que el propio algoritmo hace de cada objeto, es decir, sin hacer un filtrado de los objetos que no son clasificados como manzanas, sino que tomar todos los objetos detectados (a esto le llamamos YOLO sin filtro en las tablas de métricas). Con YOLO sin filtro, el recall se incrementó considerablemente respecto a YOLO base, lo que indica que el modelo dejó de ignorar manzanas que antes ignoraba por clasificarlas mal. Sin embargo, bajó la precisión, lo cual quiere decir que comenzó también a detectar más cosas que no son manzanas. Esto tiene sentido ya que el filtrado por tipo de objeto no se usó en YOLO sin filtro. De todas formas, el recall subió en mucho mayor medida de lo que cayó la precisión. Esta mejoría en la detección también se ve reflejada que YOLO sin filtro mejora considerablemente los resultados en el F1-score y en el AP. De esto se concluye que YOLO es un buen algoritmo para detectar objetos pero, por lo menos en el caso de las manzanas que fue lo testeado, no es tan bueno clasificándolas, teniendo muchos fallos en determinar qué objeto es. Recordar que este dataset tiene manzanas en contextos muy diversos, por lo tanto, se puede sacar esta conclusión ya que no son imágenes en un contexto fijo, lo cual podría ser el contexto el que no favorecía en la clasificación de manzanas.

El detector YOLO entrenado con el propio **Dataset1**, logró mucho mejores métricas que la versión base, pasando de 33 de AP a 85, que es equiparable al estado del arte presentado en la sección 3.1.4. Esto implica que aprendió una mejor representación de las manzanas al verlas durante el entrenamiento en contextos diversos. Por otro lado, el modelo entrenado con el **Dataset2** logró resultados inferiores que con YOLO base. Esto podría deberse a que al entrenar con imágenes con entorno fijo y monótono como lo es el de las manzanas en bandejas, luego tiene dificultades de generalizar y detectarlas en otro tipo de contextos que tiene **Dataset1**. En AP logró la mitad de rendimiento que la versión base de YOLO.

Entrenando con el **Dataset5**, no fue capaz de detectar ninguna manzana, lo mismo ocurrió con este modelo al evaluarlo para el resto de datasets. Lo que puede estar influyendo es que al ser de tracking contiene manzanas en tamaño muy pequeño, por lo que YOLO al entrenar con este dataset no fue capaz de aprender características de las manzanas, al ver muy poca información de ellas. Sin embargo, el modelo de YOLO entrenado con el **Dataset6.1** logró resultados aceptables a pesar de ser también un dataset de tracking. Logró precisión alta aunque recall muy bajo. Más adelante en esta sección se analizará esta diferencia de rendimiento que se logra con estos datasets.

El algoritmo de SAM y CLIP logró en el **Dataset1** recall elevado, precisión intermedia y en lo que refiere al AP, es la misma que YOLO en su versión base.

En el caso de Faster R-CNN, la versión base logró la mitad de la precisión que la lograda por YOLO, aunque con un recall altísimo, cercano a 1. Es decir, casi todas las manzanas reales las detecta aunque también detecta muchas otras cosas que no son manzanas. Entrenando Faster R-CNN con el **Dataset1**, el problema con la precisión se corrige y se logra un AP de 79.5, que si bien es alto, es superado por YOLO entrenado con el propio dataset. Para el resto de modelos de Faster-RCNN (entrenados con el resto de datasets) los resultados son deficientes y el entrenamiento fue contraproducente.

En la tabla 5.2 se tienen las métricas para los mismos modelos que en la tabla ya analizada pero testeando con el **Dataset2**. La dificultad que tiene este dataset es que las imágenes tienen data augmentation, como aumento del ruido, distorsión de la escala y rotaciones en las imágenes. La versión base de YOLO logró resultados muy similares respecto al anterior dataset. El modelo entrendado con el **Dataset1** no bajó en gran medida su buen rendimiento, por lo que fue capaz de extrapolar de buena manera lo aprendido durante el entrenamiento. SAM y CLIP mejoró sus resultados respecto al otro dataset logrando 44 de AP y superando a YOLO en su versión base, equiparando a su versión sin filtro y también superando a Faster R-CNN base. Esto puede dar cuenta de una mejor adaptación de SAM y CLIP a la perturbación de los datos y también el contexto estático de las imágenes lo pudo beneficiar. En los modelos de Faster R-CNN, el que destaca es el modelo entrenado con el **Dataset1**, mientras que el resto de modelos entrenados logran resultados deficientes.

La tabla 5.3 muestra los resultados de aplicar los algoritmos de detección mencionados con un dataset de tracking. Este dataset tiene imágenes con muchas manzanas en árboles y cada manzana ocupa un tamaño muy reducido en la imagen. Esto sin dudas dificulta la tarea de detección. Recordar también que hay una diferencia en el IoU usado. Como ya se explicó y justificó para los datasets con manzanas grandes se utilizó 0.5 mientras que los de tracking 0.2. De todas formas YOLO en su versión sin entrenar logra resultados muy similares a los que logra con los otros datasets. Sin embargo, YOLO entrenado con datasets de detección logra resultados pésimos, sin lograr detectar manzanas. Esto debe estar causado por el tamaño de las manzanas y que los modelos no fueron capaces de generalizar a manzanas muy pequeñas porque no vieron ninguna con esas dimensiones durante el entrenamiento. Sorprendentemente YOLO entrenado con **Dataset6.1** que tiene manzanas en tamaño similar a **Dataset5** tuvo resultados inferiores que YOLO base, principalmente causado por el recall. Es decir, que tuvo problemas identificando manzanas y detectó pocas. Quitando el filtro por clase para este modelo, se pudo duplicar en AP. En Faster R-CNN los mejores resultados fueron logrados por el modelo entrenado con **Dataset1**, el modelo base y el entrenado con **Dataset6.1**.

En la tabla 5.4 se tienen las métricas con el otro dataset de tracking. Los resultados fueron superiores que con el anterior dataset, lo que da cuenta que el **Dataset5** es un dataset complicado para todos los modelos. Tanto el entrenamiento como la evaluación con este dataset dio pésimos resultados. Observando

Dataset1	Precisión	Recall	F1-score	AP
YOLO base	0.83	0.40	0.54	33.17
YOLO sin filtro	0.70	0.76	**0.73**	**53.46**
YOLO (**Dataset1**)	0.89	0.96	0.92	85.65
YOLO (**Dataset2**)	0.55	0.28	0.37	18.97
YOLO (**Dataset5**)	0	0	0	0
YOLO (**Dataset6.1**)	**0.84**	0.27	0.41	23.20
SAMCLIP	0.57	0.81	0.67	32.00
Faster R-CNN	0.41	0.95	0.57	39.86
Faster R-CNN (**Dataset1**)	0.80	0.99	0.89	79.50
Faster R-CNN (**Dataset2**)	0.02	**0.99**	0.04	2.41
Faster R-CNN (**Dataset5**)	0.02	0.15	0.04	0.35
Faster R-CNN (**Dataset6.1**)	0	0.07	0	0

Tabla 5.1: Métricas de detección obtenidas evaluando con el **Dataset1**. Entre paréntesis se indica con qué dataset fue entrenado el modelo. YOLO sin filtro refiere a no filtrar los objetos detectados utilizando la clasificación de objetos que hace YOLO, es decir, se utilizan todos los objetos detectados en vez de solamente los clasificados como manzana por YOLO. Las métricas de los algoritmos entrenados con el propio dataset están de color azul y en negrita las mejores métricas sin tener en cuenta a los modelos entrenados con el dataset.

el **Dataset5** (ver figura 3.15), lo que tiene de particular es que la iluminación en cada imagen varía mucho, habiendo muchas sombras causadas por hojas. El **Dataset6.1** (figura 3.16) fue generado en un día nublado, teniendo una iluminación uniforme en las imágenes, además de contar con árboles menos frondosos y por lo tanto habiendo menos oclusión. La calidad de imagen en este dataset también es superior, además de que las manzanas contrastan más que en el **Dataset5**. Todos estos factores explican la disparidad de rendimientos entre estos datasets.

El modelo que logró los mejores resultados en los datasets de tracking es YOLO en su versión base con AP entre 30 y 40. Faster R-CNN entrenado con el **Dataset1** lo sigue con AP alrededor de 25, mientras que SAM y CLIP logra un AP cercano a 10 en este tipo de datasets. Los modelos base tuvieron resultados sostenidos en todos los tipos de datasets. El **Dataset1** produjo los mejores resultados al entrenar los modelos con él. Por lo tanto, el mejor tipo de conjunto de datos de entrenamiento para detección en árboles es uno con manzanas en contextos muy diversos. Los datasets de tracking, que tienen un contexto bastante fijo, no resultaron de mucha utilidad para entrenar. A lo mejor si se disponía de una cantidad muchísimo mayor de datos de este tipo los resultados mejoraban y se equiparaban más con el estado del arte, pero con la cantidad de datos manejada queda claro que un dataset diverso es el que logra mejores resultados.

Dataset2	Precisión	Recall	F1-score	AP
YOLO base	0.87	0.34	0.49	30.69
YOLO sin filtro	0.79	0.54	0.63	43.11
YOLO (**Dataset1**)	0.88	0.85	**0.87**	**77.39**
YOLO (**Dataset2**)	0.9	0.94	0.92	86.98
YOLO (**Dataset5**)	0	0	0	0
YOLO (**Dataset6.1**)	**0.89**	0.28	0.43	26.33
SAMCLIP	0.72	0.59	0.65	44.16
Faster R-CNN	0.5	0.67	0.57	34.44
Faster R-CNN (**Dataset1**)	0.66	**0.99**	0.79	63.10
Faster R-CNN (**Dataset2**)	0.03	0.92	0.06	4.25
Faster R-CNN (**Dataset5**)	0.03	0.69	0.06	3.77
Faster R-CNN (**Dataset6.1**)	0	0.07	0	0

Tabla 5.2: Métricas de detección obtenidas evaluando con el **Dataset2**. Entre paréntesis se indica con qué dataset fue entrenado el modelo. Las métricas de los algoritmos entrenados con el propio dataset están de color azul y en negrita las mejores métricas sin tener en cuenta a los modelos entrenados con el dataset.

Dataset5	Precisión	Recall	F1-score	AP
YOLO base	0.89	0.43	0.58	31.00
YOLO sin filtro	0.84	0.48	**0.62**	**34.15**
YOLO (**Dataset1**)	0	0	0	0
YOLO (**Dataset2**)	0	0	0	0
YOLO (**Dataset5**)	0	0	0	0
YOLO (**Dataset6.1**)	**0.98**	0.12	0.21	10.36
SAMCLIP	0.44	0.36	0.4	12.34
Faster R-CNN	0.45	0.5	0.47	22.67
Faster R-CNN (**Dataset1**)	0.98	0.31	0.47	26.50
Faster R-CNN (**Dataset2**)	0.29	0.63	0.4	8.30
Faster R-CNN (**Dataset5**)	0.19	0.42	0.27	1.15
Faster R-CNN (**Dataset6.1**)	0.32	**0.71**	0.44	13.66

Tabla 5.3: Métricas de detección obtenidas evaluando con el **Dataset5**. Entre paréntesis se indica con qué dataset fue entrenado el modelo. Las métricas de los algoritmos entrenados con el propio dataset están de color azul y en negrita las mejores métricas sin tener en cuenta a los modelos entrenados con el dataset.

Dataset6.1	Precisión	Recall	F1-score	AP
YOLO base	0.97	0.56	**0.71**	**42.87**
YOLO sin filtro	0.97	0.56	**0.71**	**42.91**
YOLO (**Dataset1**)	**0.98**	0.01	0.03	1.29
YOLO (**Dataset2**)	0.01	0	0	0
YOLO (**Dataset5**)	**0.98**	0.12	0.21	10.36
YOLO (**Dataset6.1**)	0.99	0.43	0.6	30.5
SAMCLIP	0.59	0.17	0.26	4.64
Faster R-CNN	0.32	0.85	0.47	22.67
Faster R-CNN (**Dataset1**)	**0.98**	0.31	0.47	26.50
Faster R-CNN (**Dataset2**)	0.17	0.76	0.28	1.83
Faster R-CNN (**Dataset5**)	0.2	**0.91**	0.33	6.90
Faster R-CNN (**Dataset6.1**)	0.15	0.72	0.26	13.35

Tabla 5.4: Métricas de detección obtenidas evaluando con el **Dataset6.1**. Entre paréntesis se indica con qué dataset fue entrenado el modelo. Las métricas de los algoritmos entrenados con el propio dataset están de color azul y en negrita las mejores métricas sin tener en cuenta a los modelos entrenados con el dataset.

	Tiempo promedio (s)
YOLO	0.10
SAMCLIP	0.85
Faster R-CNN	0.92

Tabla 5.5: Tiempos de detección promedio por imagen de los algoritmos de detección. Estos tiempos fueron obtenidos al evaluar con el **Dataset1**.

5.2. Clasificación

Para evaluar este tipo de algoritmos se desarrolló una clase análoga a la presentada en la sección de Detección llamada ClassifierEvaluator, la cual recibe de cada imagen la clase predicha y la real y va acumulando estos resultados. Al terminar de procesar el dataset, esta clase tiene una función para generar el reporte de métricas. Las métricas utilizadas fueron: tiempo promedio de clasificación por imagen, exactitud, precisión, recall y F1-score. Las métricas se obtuvieron tomando como clase positiva tanto a las manzanas enfermas como a las sanas para ver el rendimiento de los modelos en ambas clases. De igual forma que se mencionó para detección, esto permite una evaluación muy sencilla de algoritmos de clasificación.

Los datasets utilizados fueron: **Dataset3** y **Dataset4** (ver sección 3.2.5). En las tablas 5.7 y 5.8 se pueden ver los resultados obtenidos.

Una exactitud alta refiere a que la mayoría de las predicciones son correctas. Una precisión alta en cierta clase significa que cuando el modelo clasifica en esa clase es altamente probable que la clasificación sea correcta. Un recall alto en una clase se refiere a que la mayoría de imágenes de esa clase las predice correctamente.

La tabla 5.6 da cuenta de un mayor tiempo de clasificación para el modelo Transformer, el cual es alrededor de 3 veces más lento que CLIP.

En la tabla 5.7 se puede ver que el rendimiento del Transformer es perfecto en **Dataset3**. Esto se debe a que este modelo se entrenó con dicho dataset, aunque con una partición distinta a la utilizada para evaluación. Para este dataset, CLIP logró resultados muy balanceados ya que todas las métricas están en el entorno de 0.8.

Para el **Dataset4** (tabla 5.8) CLIP logra resultados muy similares que en el otro dataset, aunque con un recall inferior en manzanas enfermas, es decir, le costó más detectar a este tipo de manzanas aunque lo hace con una buena precisión (de 0.84). Para las manzanas sanas, el recall aumentó a 0.89 por lo que la mayoría de las manzanas sanas las clasifica correctamente. Por otro lado, el modelo Transformer tiene menor exactitud que CLIP. En la clase enferma, tiene una baja precisión y un recall muy cercano a 1, lo que da cuenta que predice muchas manzanas enfermas que no son, aunque las que sí son las predice casi todas bien. En las manzanas sanas ocurre lo opuesto, una precisión muy alta con un bajo recall. Esto da cuenta de que el Transformer tiene una tendencia a predecir con más frecuencia manzanas enfermas en este dataset.

En definitiva, los mejores resultados fueron logrados con CLIP, ya que en ambos datasets tuvo rendimientos muy similares. El modelo Transformer en el dataset en el que no fue entrenado tuvo un peor desempeño que CLIP. De todas formas, ambos métodos propuestos en este trabajo están por debajo de los algoritmos relevados del estado del arte que logran precisión y exactitud en el entorno de 0.94, aunque con datasets que no disponemos como para comparar.

	Tiempo promedio (s)
CLIP	0.09
Transformer	0.33

Tabla 5.6: Tiempos de clasificación promedio por imagen. Estos tiempos fueron obtenidos al evaluar con el **Dataset3**.

Dataset3		Clase enferma positiva			Clase sana positiva		
	Exactitud	Precisión	Recall	F1-score	Precisión	Recall	F1-score
CLIP	0.79	0.85	0.78	0.81	0.73	0.81	0.77
Transformer	1	1	1	1	1	1	1

Tabla 5.7: Métricas de calidad obtenidas evaluando con el **Dataset3**.

5.3. Tracking

La evaluación de tracking se realizó ejecutando el pipeline con los datasets de tracking, y para obtener las métricas se utilizó la librería TrackEval(*TrackEval*, 2020) que a partir de dos archivos en formato MOT Challenge (*Multiple Object Tracking Benchmark*, 2014) con los datos reales y los predichos de tracking, calcula todas las métricas: HOTA, MOTA, MOTP, IDF1 (ver anexo A para consultar sus definiciones). El pipeline por cada detección que trackea, la almacena en un archivo con formato MOT Challenge para facilitar la evaluación en tracking. Se utilizó en todas las pruebas de tracking a YOLO en su versión base como detector, ya que es un algoritmo que logró métricas muy buenas y equilibradas en todos los datasets.

En la tabla 5.9 se pueden ver los resultados de evaluar con el **Dataset5**. En lo que refiere a tiempos de detección, Norfair es el algoritmo más rápido por diferencia. Demoró en promedio 70ms, seguido por BoT-SORT con 349ms, mientras que el resto se encuentra por encima de 400ms. Fue muy dispar la cantidad de manzanas diferentes identificadas. Habiendo 1509 manzanas en total, HybridSORT trackeó 201 manzanas distintas, BoT-SORT 612 y otros encontraron más de las que había como StrongSORT que detectó 3733 manzanas. El algoritmo que se aproximó más fue DeepOCSORT que detectó 2204. StrongSORT es el algoritmo con mejor MOTA, que mide la exactitud teniendo en cuenta errores de falsos positivos, negativos y cambios de identificación. En esta métrica están muy parejos todos a excepción de BoT-SORT que logró un resultado inferior. En MOTP también están muy parejos a excepción de StrongSORT que es el peor en este rubro. Esta métrica mide la alineación de las detecciones correctas con

Dataset4		Clase enferma positiva			Clase sana positiva		
	Exactitud	Precisión	Recall	F1-score	Precisión	Recall	F1-score
CLIP	0.77	0.84	0.64	0.73	0.73	0.89	0.8
Transformer	0.69	0.62	0.98	0.75	0.94	0.44	0.6

Tabla 5.8: Métricas de calidad obtenidas evaluando con el **Dataset4**.

Dataset5	Tiempo prom.(ms)	Manzanas identificadas	HOTA	MOTA	MOTP	IDF1
StrongSORT	545	3733	30.7	**33.5**	59.0	**51.5**
BoT-SORT	349	612	30.1	19.0	**77.1**	32.4
HybridSORT	433	201	21.7	29.7	75.0	18.4
DeepOCSORT	467	**2204**	**35.1**	28.5	75.5	44.4
Norfair	**70**	737	35.0	27.0	71.9	50.0

Tabla 5.9: Métricas de tracking obtenidas evaluando con el **Dataset5**. El dataset contiene 1801 frames, con 1509 manzanas distintas y 82468 detecciones de manzanas, es decir, alrededor de 45 manzanas por frame.

las detecciones verdaderas. En IDF1 los mejores algoritmos son StrongSORT, Norfair y DeepOCSORT. Dicha métrica calcula el F1-score entre precisión y recall en la tarea de identificación (o dicho de otra forma, asociación de objetos entre frames).

En las tablas 5.10 y 5.11 se pueden ver las métricas al evaluar con los **Dataset6.1** y **Dataset6.2** respectivamente. Estos datasets son los generados en el campo del INIA, Las Brujas, por lo que los resultados en estos datasets serán los más fieles a la realidad cuando se testee el pipeline en campo. Estos datasets son más reducidos en cantidad de imágenes por frame, ya que tienen la mitad de manzanas por frame que el **Dataset5**. Sin embargo, esto no se vio reflejado en el tiempo promedio de tracking por imagen en ningún algoritmo salvo en Norfair, que el tiempo sí se redujo a la mitad. Para el resto de trackers, se mantuvo relativamente constante. En el resto de métricas, StrongSORT domina en todas para ambos datasets a excepción de MOTP. DeepOCSORT en estos datasets tiene métricas muy elevadas, cercanas a lo logrado por StrongSORT. Además, en los tres datasets predijo con bastante exactitud la cantidad real de manzanas. HybridSORT es el algoritmo que logró los peores resultados en los tres datasets. Trackea muy pocas manzanas y es el que tiene peor IDF1 y HOTA. Norfair se caracteriza por subestimar la cantidad de manzanas, además de tener HOTA y MOTA inferiores al resto de modelos en los dos datasets del INIA.

Los algoritmos que lograron mejores resultados fueron entonces StrongSORT y DeepOCSORT. De todas formas, todos los algoritmos tienen sus particularidades y puntos fuertes, además que el rendimiento de cada uno tuvo variaciones dependiendo del dataset.

5.4. Sistema

Como ya se ha mencionado varias veces en este trabajo, el pipeline combina la detección, la clasificación y el tracking de manzanas y fue pensado para que funcione en campos de manzanos, lo que quiere decir, que el tipo de imágenes que se quieren analizar son las que tienen manzanas en árboles, por lo que el tamaño que ocupan las manzanas en las imágenes es reducido. En esta sección se estudia el funcionamiento del sistema en su conjunto bajo las condiciones mencionadas.

Dataset6.1	Tiempo prom.(ms)	Manzanas identificadas	HOTA	MOTA	MOTP	IDF1
StrongSORT	678	**123**	**46.4**	**43.6**	79.3	**60.7**
BoT-SORT	430	59	39.6	30.1	**82.5**	46.5
HybridSORT	552	31	28.8	40.0	79.3	24.7
DeepOCSORT	434	110	42.7	37.9	80.5	54.8
Norfair	**19**	65	25.6	15.7	69.6	40.6

Tabla 5.10: Métricas de tracking obtenidas evaluando con el **Dataset6.1**. El dataset contiene 202 frames, con 139 manzanas distintas y 4509 detecciones de manzanas, es decir, alrededor de 22 manzanas por frame.

Dataset6.2	Tiempo prom.(ms)	Manzanas identificadas	HOTA	MOTA	MOTP	IDF1
StrongSORT	563	**172**	**26.8**	**28.9**	50.4	**48.6**
BoT-SORT	420	76	23.5	21.5	52.5	37.6
HybridSORT	522	43	19.0	27.0	51.1	23.8
DeepOCSORT	406	148	25.9	26.1	51.7	44.2
Norfair	**34**	70	20.0	6.1	**68.7**	28.1

Tabla 5.11: Métricas de tracking obtenidas evaluando con el **Dataset6.2**. El dataset contiene 254 frames, con 200 manzanas distintas y 5654 detecciones de manzanas, es decir, alrededor de 22 manzanas por frame.

Los algoritmos de detección ya fueron evaluados con el tipo de imágenes que se quiere que el sistema sea capaz de analizar (esto corresponde a la evaluación con los datasets de tracking), llegando a la conclusión que los modelos que mejor funcionan en este contexto son: YOLO base, Faster R-CNN base y Faster R-CNN (**Dataset1**).

De la misma forma, los algoritmos de tracking fueron evaluados y los que lograron mejores resultados fueron StrongSORT, DeepOCSORT y BoT-SORT.

Los algoritmos de calidad no fueron testeados aun con este tipo de imágenes, sino que se entrenó y evaluó con imágenes de manzanas en primer plano. Sin dudas que el disponer de muy pocos píxeles por manzana, generará una tasa de error mayor debido a que los defectos de las manzanas son menos visibles.

En esta sección se testeó el pipeline entero, con clasificación online para analizar el rendimiento de los algoritmos de calidad en este contexto, como también para evaluar al sistema en su conjunto.

En la tabla 5.12 están las combinaciones que fueron probadas, mientras que en la tabla 5.13 están los resultados con el **Dataset6.1**.

Analizando los casos 1 y 2, se puede ver que aproximadamente la mitad de las manzanas fueron detectadas como enfermas. En realidad, el dataset casi no contiene manzanas de este tipo, por lo que se busca que el pipeline clasifique la menor cantidad de manzanas posibles como enfermas. El objetivo de estos casos era comparar el uso de moda con promedio dejando fijo el resto de parámetros. Los resultados fueron idénticos.

En los casos 3 y 4 se comparó promedio y moda usando Faster R-CNN en

vez de YOLO. La diferencia entre manzanas enfermas es mínima. Lo que sí se puede ver es que Faster R-CNN detectó 7 veces más manzanas que YOLO. Los tiempos de detección, clasificación y tracking aumentaron considerablemente porque Faster R-CNN es mucho más lento. Al detectar más manzanas, se deben clasificar más imágenes y por eso el tiempo promedio por frame destinado a clasificación aumentó de 600ms a casi 3000ms.

Los casos 5 y 6 fueron pensados para evaluar Transformer tanto con moda como promedio. La cantidad de manzanas enfermas disminuyó casi 4 veces respecto a los casos 1 y 2. En el resto de combinaciones que se usa Transformer se puede ver que también clasifica proporcionalmente menos manzanas enfermas que CLIP, por ejemplo, entre los casos 7 y 8 Transformer clasifica la mitad de manzanas enfermas.

En los casos 7 y 8 se testea Faster R-CNN entrenado con **Dataset1**, que fue otro modelo con buenos resultados en detección. Este modelo fue el segundo que estuvo más cerca de predecir la cantidad de manzanas reales y es el que más cerca estuvo usando StrongSORT.

El resto de casos fueron pensados para testear DeepOCSORT y BoT-SORT. El caso 10 (DeepOCSORT) falla por una manzana, aunque no quiere decir que detectó correctamente a todas las manzanas menos a una, ya que puede haber distintos tipos de errores que al final den ese resultado tan próximo a la cantidad real.

Se empleó la tasa de frames por segundo (FPS) como indicador de rendimiento del pipeline, para evaluar la cantidad de frames que puede analizar por segundo. El pipeline en líneas generales está por debajo de un FPS de procesamiento, lo que dificulta que pueda ser usado en tiempo real ya que las cámaras hoy en día son capaces de grabar por lo menos en 30 FPS. Sin embargo, puede no ser completamente necesario analizar cada uno de los frames capturados, ya que si el robot no se desplazó lo suficiente, se terminaría analizando información muy similar. El modelo más rápido es con YOLO, Transformer y BoT-SORT, que logra 1.5 FPS. Seguramente usando CLIP sea aún más rápido ya que el modelo Transformer es mucho más lento, como se puede ver al comparar los resultados de los casos 1 y 5 (570ms de clasificación promedio contra 1899ms).

Se testeó el pipeline en modalidad offline (sin clasificación de manzanas a medida que procesa los frames) con las combinaciones 1 y 12 de la tabla 5.12, que fueron las más rápidas en términos de FPS. La configuración 1 logró 7.8 FPS de procesamiento, mientras que la 12 alcanzó 11.1 FPS. En esta modalidad, el pipeline podría llegar a funcionar en tiempo real. Esto da cuenta que el clasificar todas las manzanas en cada frame impacta significativamente en la velocidad del pipeline. Notar que esta diferencia depende de la cantidad de manazanas en promedio que tiene cada frame analizado.

Los mejores modelos de detección resultaron ser YOLO base y Faster R-CNN entrenado con **Dataset1**, ya que Faster R-CNN base sobreestima la cantidad de manzanas. En calidad, Transformer es el que menos manzanas enfermas clasifica. El método de clasificación (moda o promedio) prácticamente no influyó en los resultados.

Analizando las manzanas que son clasificadas como enfermas, se identificaron

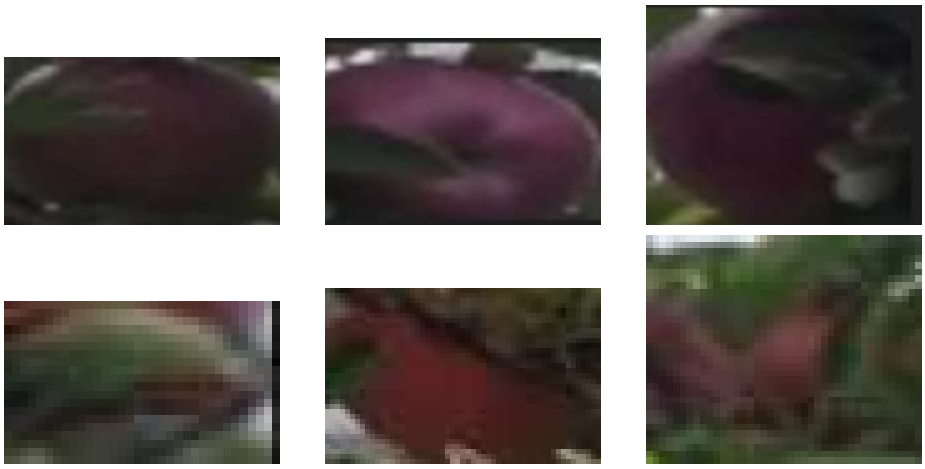

Figura 5.1: Ejemplos de imágenes clasificadas como enfermas por el pipeline.

varias causas que generan estas clasificaciones erróneas: manzanas con muy poca iluminación, manzanas ocluidas por hojas, manzanas muy pegadas entre sí y manzanas muy pequeñas (se tienen muy pocos pixeles de ellas). En la figura 5.1 se pueden ver ejemplos de esto.

Id	Detector	Clasificador	Tracker	Método clasificación
1	YOLO base	CLIP	StrongSORT	Promedio
2	YOLO base	CLIP	StrongSORT	Moda
3	Faster R-CNN base	CLIP	StrongSORT	Promedio
4	Faster R-CNN base	CLIP	StrongSORT	Moda
5	YOLO base	Transformer	StrongSORT	Promedio
6	YOLO base	Transformer	StrongSORT	Moda
7	Faster R-CNN (**Dataset1**)	CLIP	StrongSORT	Promedio
8	Faster R-CNN (**Dataset1**)	Transformer	StrongSORT	Promedio
9	Faster R-CNN base	Transformer	DeepOCSORT	Promedio
10	YOLO base	Transformer	DeepOCSORT	Promedio
11	Faster R-CNN base	Transformer	BoT-SORT	Promedio
12	YOLO base	Transformer	BoT-SORT	Promedio

Tabla 5.12: Configuraciones del pipeline que fueron testeadas.

Id	ME	MD	FPS	TD (ms)	TC (ms)	TT (ms)	TTot (s)
1	54	106	1.0	**54**	**570**	839	214
2	54	106	0.9	80	662	970	202
3	195	729	0.2	1003	2951	3710	952
4	190	729	0.2	991	2959	3721	951
5	16	106	0.4	84	1899	2210	463
6	16	106	0.4	79	1827	2212	442
7	51	124	0.3	1785	670	975	557
8	20	124	0.2	1699	2210	2513	851
9	58	656	0.1	959	4239	4869	1177
10	10	**138**	0.8	112	1055	1356	296
11	43	270	0.2	894	3248	3580	904
12	**7**	46	**1.5**	82	440	**584**	**134**

Tabla 5.13: Resultados del testing de las distintas configuraciones con **Dataset6.1** (tiene 139 manzanas). ME son las manzanas enfermas, MD las manzanas detectadas, FPS son los frames por segundo que el pipeline fue capaz de analizar, TD es el tiempo promedio de detección por frame, TC el tiempo promedio de clasificación por frame, TT el de tracking por frame y TTot es el tiempo total que se requirió para todo el dataset.

Capítulo 6

Conclusiones y Trabajo Futuro

Se desarrolló un pipeline de clasificación de manzanas en videos, que es capaz de adaptarse a cualquier algoritmo que pueda entrenarse a futuro, tanto de detección, clasificación como de tracking. Además, es capaz de integrarse a robots en el ambiente de ROS, detectar la ubicación de manzanas enfermas, teniendo la posibilidad de clasificarlas tanto de manera online como offline, y generando un mapa de calor para visualizar zonas con presencia de manzanas enfermas. Además, puede calcular métricas y producir los metadatos necesarios para su generación.

Además, se entrenaron y evaluaron distintos algoritmos del estado del arte utilizando diversos datasets. YOLOv8 en su versión base logró resultados cercanos al estado del arte para manzanas en primer plano, bajando considerablemente el recall para manzanas en árboles. Es decir, es muy preciso para detectar las manzanas pero hay muchas que no las detecta. Analiza muy rápidamente las imágenes, alrededor de nueve veces más rápido que SAM y CLIP y que Faster R-CNN. Tiene complicaciones en la clasificación de objetos, confunde manzanas con naranjas en determinados contextos. Esto también explica en buena medida el bajo recall que logra. Faster R-CNN no es muy preciso en sus detecciones pero su recall es alto. SAM y CLIP es el peor de los tres modelos, contrario a lo que se esperaba por lo que establecían en el artículo.

En lo que respecta a clasificación por calidad, CLIP fue el que logró mejores resultados en los datasets de calidad, aunque el modelo Transformer fue el que menos manzanas enfermas detectó incorrectamente en datasets de tracking. Es decir, con manzanas de tamaño reducido y en árboles, Transformer funciona mejor, aunque es más lento clasificando que CLIP.

Los algoritmos de tracking lograron un buen funcionamiento, en especial StrongSORT, BoT-SORT y DeepOCSORT. Estos algoritmos demostraron tener la capacidad de asociar eficazmente las detecciones a lo largo de los frames y de recuperarse rápidamente en caso de que el detector falle en detectar una man-

zana en un frame específico. Hay que tener en cuenta también que el desempeño de estos algoritmos está ligado al rendimiento de los algoritmos de detección, cuanto mejor sea el rendimiento del detector, mejor será el del tracker.

El pipeline desarrollado abre varias posibilidades sobre las que seguir trabajando en futuros proyectos.

En primer lugar, para poder poner en práctica el pipeline en un entorno real, se debe integrar a un robot con software que controle el movimiento del mismo. Para que el robot sea completamente autónomo, se debe desarollar un algoritmo que sea capaz de detectar el fin de una fila de árboles y pueda girar y comenzar a recorrer la siguiente hilera.

Una mejora que puede incluirse para perfeccionar la ubicación geográfica de las manzanas, es el uso de información de sensores de profundidad en el pipeline. Una alternativa a esto es realizar una reconstrucción 3D del entorno del robot para poder estimar estas ubicaciones. Esto también es de gran utilidad para poder determinar si una manzana se encuentra en la fila que se está analizando o en otra fila. Sabiendo esto, se puede evitar contar varias veces a las mismas manzanas ya que en ciertas ocasiones las imágenes del robot permiten ver la fila de atrás a la analizada.

Un aspecto en el que se intentó trabajar pero no se lograron buenos resultados, fue en la generación automática de un video con manzanas enfermas a partir de un video con manzanas en árboles, ya que no se contaba con videos ni datasets de tracking que contuvieran manzanas enfermas. La idea detrás de esto, es que cada detección de manzana en cada frame se sustituyera en el video, con una cierta probabilidad, por una manzana enferma tomada aleatoriamente de un dataset con manzanas enfermas. Las principales dificultades que se tuvieron fueron la diferencia en el brillo de las imágenes tomadas del dataset con la del video original, el ajuste de las dimensiones de las imágenes reemplazadas para que tuvieran un tamaño acorde a la distancia entre la cámara y la manzana. De todas formas, la generación de estos videos sintéticos con algún método más elaborado que logre resultados más realistas puede contribuir en la generación de datasets nuevos con distintas clases de manzanas.

Una línea sobre la que se puede trabajar es el uso de redes secuenciales para analizar múltiples imágenes de una misma manzana para determinar si la misma está enferma o no. Se debería entrenar a una red como LSTM o Transformer (o cualquiera de tipo encoder-decoder) y comparar los resultados con el cálculo del promedio (u otra operación) de probabilidades.

Los algoritmos de tracking durante este trabajo no se intentaron perfeccionar ni hacerles fine tuning. Partiendo de la base que la gran mayoría utilizan los filtros de Kalman para estimar el movimiento de los objetos, una forma de mejorar estas estimaciones podría ser utilizando información de un GPS diferencial (que tiene errores de centímetros o milímetros) o de odometría del robot para incluir esta información en la matriz de costos del método para tener mejores predicciones en base a información de otros sensores en vez de usar exclusivamente imágenes.

Es necesario generar conjuntos de datos de mayor tamaño tanto para detección, como para calidad, como para tracking. En el caso de detección, el dataset

debe ser diverso, con imágenes tanto en primer plano como en árboles, variando el fondo de la imagen, el tamaño que ocupan las manzanas en las imágenes y con distintas condiciones de iluminación para que haya manzanas con sombras y otras más iluminadas. En la evaluación de los modelos quedó en claro que este tipo de dataset es el que mejores resultados logra, incluso mejores que si se entrena con imágenes exclusivamente de manzanas en árboles ya que esto generó sobreajuste y poca capacidad de generalización en los modelos. Para los conjuntos de datos de calidad, se necesitan también imágenes de manzanas sanas y enfermas con tamaño reducido, con oclusión por hojas, ramas y otras manzanas y distintas condiciones de luz. El hecho de no disponer de imágenes de este tipo para entrenar causó que los modelos ante manzanas de muy reducido tamaño en las imágenes, ocluidas por hojas y con poca iluminación sean clasificadas como enfermas y este tipo de manzanas abundan en entornos reales.

La falta de conjuntos de datos de gran tamaño con las características ya mencionadas, produjo que los modelos fine-tuneados no tuvieran un desempeño muy superior a los modelos sin ajustar. En varios casos, los modelos sobreajustaron a los pocos y monótonos datos con los que se entrenaron.

Referencias

Alex Bewley, Z. G. (2016). Simple online and realtime tracking.

Alexey Dosovitskiy, L. B. (2021). An image is worth 16x16 words: Transformers for image recognition at scale.

Algoritmo húngaro. (2023). https://es.wikipedia.org/wiki/Algoritmo_h%C3%BAngaro. (Accessed: 2023-12-12)

Apple orchard production estimation using deep learning strategies: a comparison of tracking-by-detection algorithms. (2022). https://zenodo.org/records/7383338. (Accessed: 2024-02-09)

Ashish Vaswani, N. S. (2017). Attention is all you need.

Bargoti, S., y Underwood, J. (2017). Deep fruit detection in orchards.

Boxmot: pluggable sota tracking modules for segmentation, object detection and pose estimation models. (2020). https://github.com/mikel-brostrom/yolo_tracking. (Accessed: 2024-02-09)

Christian Szegedy, W. L. (2014). Going deeper with convolutions.

Convolutional neural network tutorial. (2023). https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network. (Accessed: 2023-05-08)

Curso de aprendizaje profundo para visión artificial - clase 1. (2022). https://eva.fing.edu.uy/pluginfile.php/212135/mod_resource/content/1/deepvision_LR.pdf. (Accessed: 2023-05-08)

Deep learning: Understanding the inception module. (2020). https://towardsdatascience.com/deep-learning-understand-the-inception-module-56146866e652. (Accessed: 2023-05-07)

Detectron2. (2024). https://ai.meta.com/tools/detectron2/.

deteksi-buah dataset. (2023). https://universe.roboflow.com/gunadarma-ypsh6/deteksi_buah. (Accessed: 2023-12-13)

Estado del arte. (2023). https://gitlab.fing.edu.uy/thomas.sheppard/proyecto-de-grado-calidad-de-manzanas-y-navegacion/-/blob/main/Estado_del_arte.pdf.

Fan, S., Li, J., Zhang, Y., Tian, X., Wang, Q., He, X., ... Huang, W. (2020). On line detection of defective apples using computer vision system combined with deep learning methods.

Fine-tune vit for image classification with transformers. (2024). https://huggingface.co/blog/fine-tune-vit.

Fu, L. (2020). Faster r-cnn-based apple detection in dense-foliage fruiting-wall trees using rgb and depth features for robotic harvesting.

Garderes-Gutiérrez. (2023). Reconocimiento y conteo de manzanas.

Ghimire, D., Kil, D., y heum Kim, S. (2022). A survey on efficient convolutional neural networks and hardware acceleration.

gmplot. (s.f.). https://pypi.org/project/gmplot/. (Accessed: 2023-11-17)

Hochreiter, S. (1997). Long short-term memory.

How to evaluate tracking with the hota metrics. (2021). https://autonomousvision.github.io/hota-metrics/#:~:text=HOTA%20(Higher%20Order%20Tracking%20Accuracy,MOTA%2C%20IDF1%20and%20Track%20mAP. (Accessed: 2024-02-17)

Introducing segment anything: Working toward the first foundation model for image segmentation. (2023). https://ai.facebook.com/blog/segment-anything-foundation-model-image-segmentation/. (Accessed: 2023-04-19)

Kaggle. (2023). https://www.kaggle.com/. (Accessed: 2023-12-16)

Kalmanfilter.net. (s.f.). https://www.kalmanfilter.net/default.aspx. (Accessed: 2023-11-17)

Kirillov, A., Mintun, E., y Ravi, N. (2023). Segment anything.

Li, F. (2021). Apple quality identification and classification by image processing based on convolutional neural networks.

Li, Y., Feng, X., Liu, Y., y Han, X. (2021). Apple quality identification and classification by image processing based on convolutional neural networks.

Maggiolino, G. (2023). Deep oc-sort: Multi-pedestrian tracking by adaptive re-identification.

Maheswari, P. (2021). Intelligent fruit yield estimation for orchards using deep learning based semantic segmentation techniques—a review. *Frontiers in Plant Science*.

Malek, S. (2019). Bioimage informatics.

Mean average precision (map): A complete guide. (2023). https://kili-technology.com/data-labeling/machine-learning/mean-average-precision-map-a-complete-guide. (Accessed: 2024-02-15)

Multiple object tracking benchmark. (2014). https://motchallenge.net/. (Accessed: 2024-02-04)

Nicolai Wojke, A. B. (2016). Simple online and realtime tracking with a deep association metric.

Nir Aharon, B.-Z. B., Roy Orfaig. (2022). Bot-sort: Robust associations multi-pedestrian tracking.

Norfair. (2020). https://github.com/tryolabs/norfair. (Accessed: 2024-02-09)

Object detection metrics. (2018). https://github.com/rafaelpadilla/Object-Detection-Metrics. (Accessed: 2024-02-08)

Object tracking. (s.f.). https://paperswithcode.com/task/object-tracking. (Accessed: 2023-11-17)

Object tracking state of the art 2022. (2022). https://medium.com/

@pedroazevedo6/object-tracking-state-of-the-art-2022-fe9457b77382. (Accessed: 2024-02-11)

Olafenwa, M. (2019). *Apple detection datasevt.* https://github.com/OlafenwaMoses/AppleDetection/releases/tag/v1. (Accessed: 2024-02-15)

Pepper noise. (2012). https://www.sciencedirect.com/topics/engineering/pepper-noise#:~:text=Salt%20and%20pepper%20noise%20refers,and%20pepper%E2%80%94on%20the%20image. (Accessed: 2023-12-27)

A perfect guide to understand encoder decoders in depth with visuals. (2023). https://medium.com/@ahmadsabry678/a-perfect-guide-to-understand-encoder-decoders-in-depth-with-visuals-30805c23659b#:~:text=It%20consists%20of%20two%20parts,%2C%20image%20captioning%2C%20among%20others. (Accessed: 2023-12-26)

Pip - transformers. (2024). https://pypi.org/project/transformers/.

Precision agriculture. (s.f.). https://www.sciencedirect.com/topics/earth-and-planetary-sciences/precision-agriculture. (Accessed: 2023-12-01)

Precision and recall — essential metrics for machine learning. (2023). https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/. (Accessed: 2024-02-15)

Priya, P. S., Jyoshna, N., Amaraneni, S., y Swamy, J. (2020). Real time fruits quality detection with the help of artificial intelligence.

Proyecto de grado - pipeline de detección y seguimiento de manzanas para geolocalización de anomalías. (2023). https://gitlab.fing.edu.uy/thomas.sheppard/proyecto-de-grado-calidad-de-manzanas-y-navegacion.

Radford, A., Kim, J. W., y Hallacy, C. (2021). Learning transferable visual models from natural language supervision.

Reddy, S. (2018). *Fruits fresh and rotten for classification.* https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification/data. (Accessed: 2023-12-16)

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2015). You only look once: Unified, real-time object detection.

Ren, S., He, K., Girshick, R., y Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks.

Resnet: Residual neural networks. (2023). https://databasecamp.de/en/ml/resnet-en#:~:text=Residual%20Neural%20Networks%2C%20or%20ResNets,characterized%20by%20a%20skip%20connection. (Accessed: 2024-02-04)

Ricardo Pereira, G. C. (2022). Sort and deep-sort based multi-object tracking for mobile robotics: Evaluation with new data association metrics.

Roboflow. (2023). https://roboflow.com/. (Accessed: 2023-12-14)

Ros introduction. (s.f.). http://wiki.ros.org/ROS/Introduction. (Accessed: 2023-12-02)

Ros - robot operating system. (s.f.). https://www.ros.org/. (Accessed: 2023-12-02)

Russakovsky, O. (2015). Imagenet large scale visual recognition challenge.

scikit-learn. (2007). https://scikit-learn.org/stable/. (Accessed: 2024-02-08)

Segmentación (procesamiento de imágenes). (2023). https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes). (Accessed: 2023-12-22)

Shaohua Wan, S. G. (2019). Faster r-cnn for multiclass fruit detection using a robotic vision system, computer networks.

State of the art - image classification on imagenet. (2024). https://paperswithcode.com/sota/image-classification-on-imagenet. (Accessed: 2024-03-25)

A step-by-step introduction to the basic object detection algorithms. (2018). https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/?utm_source=blog&utm_medium=image-segmentation-article. (Accessed: 2023-05-08)

Tanco, M. M. (2015). Reconocimiento y cuantificación de manzanas.

Tang, Z. (2019). Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification - supplementary.

Tian, Y., y Li, E. (2021). Diagnosis of typical apple diseases: A deep learning method based on multi-scale dense classification network.

Top 5 object tracking methods. (2021). https://medium.com/augmented-startups/top-5-object-tracking-methods-92f1643f8435. (Accessed: 2023-11-17)

Trackeval. (2020). https://github.com/JonathonLuiten/TrackEval. (Accessed: 2024-02-20)

ujjalfinal dataset. (2023). https://universe.roboflow.com/khec-txwam/ujjalfinal. (Accessed: 2023-12-14)

Ultralytics. (2024). https://www.ultralytics.com/es.

Understanding backpropagation algorithm. (2019). https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd. (Accessed: 2023-12-22)

Wan, S., y Goudos, S. (2019). Faster r-cnn for multi-class fruit detection using a robotic vision system.

What are neural networks? (s.f.). https://www.ibm.com/topics/neural-networks. (Accessed: 2023-11-23)

Winarno, E., Hadikurniawati, W., Nirwanto, A. A., y Abdullah, D. (2018). Multi-view faces detection using viola-jones method.

Yan, B. (2021). A real-time apple targets detection method for picking robot based on improved yolov5.

Yang, M. (2023). Hybrid-sort: Weak cues matter for online multi-object tracking.

Yunhao Du, Z. Z. (2022). Strongsort: Make deepsort great again.

Zhang, B., Gu, B., Tian, G., Zhou, J., Huang, J., y Xiong, Y. (2018). Challenges and solutions of optical-based nondestructive quality inspection for robotic fruit and vegetable grading systems: A technical review.

Zhang, B., Liu, L., Gu, B., Zhou, J., Huang, J., y Tian, G. (2018). From hyperspectral imaging to multispectral imaging: Portability and stability of his-mis algorithms for common defect detection.

Anexo A

Métricas para evaluación de algoritmos

En este anexo se detallarán las métricas empleadas durante el proceso experimental. Previo a su definición, resulta imprescindible clarificar los conceptos de verdadero positivo, verdadero negativo, falso positivo y falso negativo.

- Verdadero Positivo (VP): ocurre cuando un modelo clasifica correctamente un ejemplo como positivo cuando realmente lo es. Por ejemplo, en un sistema de detección de spam de correo electrónico, un verdadero positivo sería cuando el sistema clasifica correctamente un correo electrónico como spam cuando realmente es spam.
- Verdadero Negativo (VN): ocurre cuando un modelo clasifica correctamente un ejemplo como negativo cuando realmente lo es. Siguiendo el ejemplo del sistema de detección de spam, un verdadero negativo sería cuando el sistema clasifica correctamente un correo electrónico como no spam cuando realmente no lo es.
- Falso Positivo (FP): sucede cuando un modelo clasifica incorrectamente un ejemplo como positivo cuando en realidad es negativo. En el caso del sistema de detección de spam, un falso positivo sería cuando el sistema marca un correo electrónico legítimo como spam.
- Falso Negativo (FN): ocurre cuando un modelo clasifica incorrectamente un ejemplo como negativo cuando en realidad es positivo. Por ejemplo, en el contexto del sistema de detección de spam, un falso negativo sería cuando el sistema no detecta un correo electrónico de spam y lo clasifica como legítimo.

Las métricas utilizadas en la experimentación fueron las siguientes:

- Precisión: Se calcula como la proporción de verdaderos positivos (VP) sobre el total de predicciones positivas, es decir, la suma de verdaderos

positivos y falsos positivos (FP). La precisión se utiliza para responder a la pregunta: "¿De todas las instancias que el modelo clasificó como positivas, cuántas son realmente positivas?" Una alta precisión indica que el modelo tiene una baja tasa de falsos positivos.

- Exactitud: es una medida de la fracción de predicciones correctas realizadas por el modelo sobre el total de instancias. Se calcula como la proporción de predicciones correctas (verdaderos positivos y verdaderos negativos) sobre el total de predicciones (todos los positivos y negativos, verdaderos y falsos). La exactitud responde a la pregunta: "¿Cuántas de las predicciones totales del modelo son correctas?"

- Recall (o sensibilidad): mide la capacidad del modelo para encontrar todas las instancias positivas. Se calcula como la proporción de verdaderos positivos sobre el total de instancias que realmente son positivas, es decir, la suma de verdaderos positivos y falsos negativos. El recall responde a la pregunta: "¿De todas las instancias que son realmente positivas, cuántas identificó correctamente el modelo?"

- F1-Score: es una medida que combina precisión y recall en un solo número. Se calcula como la media armónica de precisión y recall. Es útil cuando hay un desequilibrio entre las clases en los datos. Un F1-Score alto indica un buen equilibrio entre precisión y recall (*Precision and Recall — Essential Metrics for Machine Learning*, 2023).

- Average Precision (AP): es el área bajo la curva de precisión-recall. La curva de precisión-recall es una función que muestra cómo cambia la precisión del modelo a medida que el umbral de clasificación se ajusta, en función del recall (ver figura A.1). Esta métrica encapsula el equilibrio entre las dos métricas. Este equilibrio es esencial para analizar de mejor manera el rendimiento del modelo, en comparación a analizar precisión y recall por separado (*Mean Average Precision (mAP): A Complete Guide*, 2023).

- Multiple Object Tracking Accuracy (MOTA): esta métrica junto con todas las que siguen son métricas de tracking. Se define de la siguiente forma:

  $MOTA = 1 - \frac{FN+FP+ID}{GT}$

  donde ID son todos los cambios de identificación que realizó el algoritmo de tracking y GT son todos los objetos reales.

  MOTA calcula entonces la exactitud de un algoritmo de tracking tomando en cuenta tres tipos de errores: FN, FP y re-identificaciones (*Object Tracking State of the Art 2022*, 2022).

- Multiple Object Tracking Precision (MOTP): Es el promedio de superposición de los objetos correctamente trackeados.

  Se calcula como sigue: $MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t M_i}$

donde M_t es la cantidad de matches que hubo en el frame t y d_t, i es la superposición entre la predicción del bounding box del objeto i con el bounding box real en el frame t. Es decir, MOTP es el cociente entre la suma de las superposiciones y todos los aciertos que hizo el modelo. Un MOTP alto significa que las predicciones del modelo son precisas y están bien alineadas con las ubicaciones reales de los objetos que se están rastreando (*Object Tracking State of the Art 2022*, 2022).

- IDF1: Es la métrica F1-score aplicada sobre las métricas precisión de identificación (IDP) y recall de identificación (IDR). Estas métricas se definen en base a identidades falsas negativas, identidades falsas positivas e identidades verdaderas positivas (Tang, 2019).

- Higher Order Tracking Accuracy (HOTA): Fue diseñada para superar muchas de las limitaciones de métricas anteriores como MOTA e IDF1. HOTA se puede pensar como una combinación de tres puntajes de IoU. Divide la tarea de evaluar el seguimiento en tres subtareas (detección, asociación y localización) y calcula un puntaje para cada una. Luego combina estos tres puntajes de IoU para cada subtarea en el puntaje final de HOTA. Localización le llaman al promedio entre todos los IoU de los verdaderos positivos en detección, es decir, es el MOTP. Le llaman puntaje de detección a la exactitud del modelo de detección, utilizando 0.5 como umbral de IoU. Asociación refiere a cuán bien el tracker asocia detecciones en los distintos frames con identificadores (*How to evaluate tracking with the HOTA metrics*, 2021).

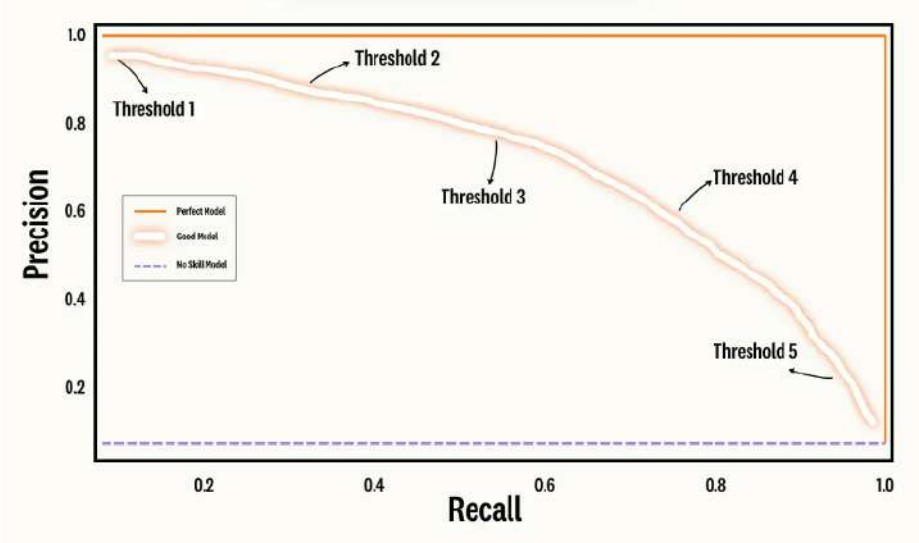

Figura A.1: Función de precisión-recall. Extraída de (*Mean Average Precision (mAP): A Complete Guide*, 2023).

Anexo B

Metodología para evaluar algoritmos de detección

A la hora de evaluar un algoritmo de aprendizaje automático, algunas de las métricas más usadas son: precisión, exactitud, sensibilidad, F1-score. Para poder calcular estas métricas se debe definir cuándo se está ante un caso de verdadero positivo, ante un falso positivo, ante un verdadero negativo o ante un falso negativo. Con esas cuatro clasificaciones se etiquetan a las predicciones hechas por un modelo según el valor de referencia que el dataset indica. En el caso de clasificación binaria de manzanas en sanas o enfermas, estas clasificaciones no necesitan aclaración alguna. Por ejemplo, si el modelo predijo que una manzana está enferma y el dataset dice lo mismo, se trata de un verdadero positivo; si el modelo predijo enferma y el dataset indica que está sana, se trata de un falso positivo.

Sin embargo, en el caso de la detección, como se predicen bounding boxes los cuales consisten en cuatro coordenadas y es muy difícil que se predigan exactamente, se deben definir las mencionadas clasificaciones para esta tarea. Esta metodología es la misma que la presentada en el artículo (Bargoti y Underwood, 2017).

Se considera que una predicción matchea con un bounding box del dataset cuando $IoU \geq threshold$, donde IoU significa Intersection over Union (Intersección dividido Unión) y refiere al cociente entre el área de intersección de dos bounding boxes dividido entre la unión entre estas dos áreas y el threshold es un umbral que se debe definir para determinar a partir de qué punto se considera que el modelo acertó en la predicción.

- Verdadero Positivo (VP): Se considera una predicción como correcta siempre matchee con un bounding box del dataset.

- Falso Positivo (FP): Todas las predicciones que no matchean con ningún bounding box del dataset.

- Falso Negativo (FN): Son todos los bounding boxes del dataset que no matchearon con ninguna predicción.
- Verdadero Negativo (VN): No tiene sentido en este dominio. Un caso positivo es un bounding box que se debería predecir, mientras que un caso negativo es un bounding box que no se debería predecir. Un verdadero negativo es entonces un bounding box que no se predijo y que no se debería haber predicho porque no hay ningún objeto ahí. Es por esto que no tiene sentido y por lo tanto no se utilizará para las métricas de detección.